National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Convolutional Neural Networks as Turbulence and Transition Closures in Aerodynamic Analysis and Shape Optimization

Diploma Thesis

Alexandros Karantonis

Supervisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Kyriakos Giannakoglou, for his invaluable guidance and mentorship throughout my Diploma Thesis journey. His extensive knowledge and insightful feedback have been instrumental in shaping the direction and quality of my work. I am truly fortunate to have had the opportunity to work under his supervision, and I am grateful for the enriching experience and knowledge I have gained.

I would also like to extend my sincere appreciation to Dr. Marina Kontou and Dr. Varvara Asouti, whose expertise, patience, availability and willingness to assist me whenever I encountered challenges or had questions were invaluable. Their support and collaboration significantly contributed to the successful completion of this thesis.

Furthermore, I am deeply grateful to my family and friends for their encouragement and understanding throughout my academic journey that ends with the completion of this thesis. Their steadfast belief in me and their constant support provided the foundation for me to persevere and overcome any obstacles that arose.

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Convolutional Neural Networks as Turbulence and Transition Closures in Aerodynamic Analysis and Shape Optimization

Diploma Thesis

Alexandros Karantonis

Supervisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

In this Diploma Thesis, Convolutional Neural Networks (CNNs) are used as surrogates for traditional turbulence and transition models in aerodynamic analysis and optimization, aiming to reduce the total computational cost. The CNNs use geometrical and flow data provided by the Reynolds-Averaged Navier-Stokes (RANS) equations, to predict the turbulent viscosity field μ_t, effectively closing the RANS without the use of differential turbulence and transition models.

To train the CNNs, a dataset is created by parameterizing the geometries using Non-Uniform Rational B-Splines (NURBS) and applying Latin Hypercube Sampling (LHS) to generate a sufficient number of new geometries, which serve as training patterns. These new geometries are then evaluated using the in-house GPU enabled flow solver PUMA, coupled with the turbulence and transition models that the CNNs aim to replace.

The use of CNNs, as opposed to Deep Neural Networks (DNNs), introduces the advantage of incorporating information from neighboring mesh elements when predicting the turbulent viscosity μ_t field. CNNs operate on entire fields of inputs rather than individual mesh elements, enabling inter-element communication, potentially leading to better predictions.

The two optimization cases performed involve the NLF0416 and the S8052 low-speed airfoils both of which exhibit transitional flow, demanding for the inclusion of a transition model in addition to the more common turbulence model. Due to the importance of transition on skin friction and drag, two separate optimization targets are set. The first target is the direct minimization of the drag coefficient C_D and the second is the maximization of the laminar area on the suction side of the airfoil by maximizing the distance between the leading edge and the transition point on the suction side TP_{SS}. This second goal serves as a proxy for reducing C_D, through a different approach.

For both optimizations, two evaluation software are used: PUMA-TM and PUMA-

CNN. PUMA-TM solves the RANS equations coupled with the one-equation Spalart-Allmaras turbulence model and the two-equation Smooth $\gamma - Re_{\theta,t}$ transition model. On the other hand PUMA-CNN solves the RANS equations and then uses the trained CNN to predict the μ_t field, closing the RANS.

While PUMA-TM incurs no capital cost, as no training is required, PUMA-CNN has a capital cost associated with the generation of training samples and the actual training of the CNN. However, PUMA-CNN benefits from a reduced per-evaluation cost due to not needing to solve the three additional differential equations.

The optimizations are performed by a Metamodel-Assisted Evolutionary Algorithm within the framework of EASY. The results are compared in terms of the quality of the final solution and computational cost. Additionally, it is examined whether maximizing TP_{SS} coincides with minimizing C_D.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης

Συνελικτικά Νευρωνικά Δίκτυα ως Υποκατάστατα Μοντέλων Τύρβης και Μετάβασης στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση Μορφής

Διπλωματική Εργασία

Αλέξανδρος Καραντώνης

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Σε αυτήν τη Διπλωματική Εργασία, εκπαιδεύονται και χρησιμοποιούνται Συνελικτικά Νευρωνικά Δίκτυα (ΣΝΔ) ως υποκατάστατα μοντέλων τύρβης και μετάβασης, στην αεροδυναμική ανάλυση και βελτιστοποίηση μορφής, με στόχο τη μείωση του συνολικού υπολογιστικού κόστους. Τα ΣΝΔ χρησιμοποιούν γεωμετρικά και ροϊκά δεδομένα προερχόμενα από την επίλυση των εξισώσεων RANS, ώστε να προβλέψουν την τυρβώδη συνεκτικότητα μ_t, κλείνοντας ουσιαστικά τις εξισώσεις RANS χωρίς τη χρήση διαφορικών μοντέλων τύρβης και μετάβασης.

Για την εκπαίδευση των ΣΝΔ, δημιουργείται μια βάση δεδομένων που περιλαμβάνει επαρκή αριθμό αεροδυναμικά αξιολογηθεισών γεωμετριών, που χρησιμεύουν ως πρότυπα εκπαίδευσης. Αρχικά, η βασική γεωμετρία παραμετροποιείται με χρήση NURBS και έπειτα πραγματοποιείται δειγματοληψία με τη μέθοδο LHS. Τέλος, αυτές οι νέες γεωμετρίες επιλύονται με τη χρήση του επιλύτη PUMA, σε συνδυασμό με τα μοντέλα τύρβης και μετάβασης που τα ΣΝΔ θα κληθούν να αντικαταστήσουν.

Η χρήση ΣΝΔ, έναντι Βαθιών Νευρωνικών Δικτύων (ΒΝΔ), εισάγει το πλεονέκτημα της ενσωμάτωσης πληροφοριών από γειτονικά κελιά κατά την πρόβλεψη του μ_t. Αυτό συμβαίνει διότι τα ΣΝΔ λειτουργούν χρησιμοποιώντας ολόκληρα τα πεδία εισόδου και όχι μεμονωμένα τα κελιά του πλέγματος, επιτρέποντας την επικοινωνία μεταξύ διαφορετικών κελιών, γεγονός που ενδεχομένως οδηγεί σε καλύτερες προβλέψεις.

Οι βελτιστοποιήσεις μορφής πραγματοποιούνται σε δύο αεροτομές χαμηλών ταχυτήτων, τη NLF0416 και τη S8052, που και οι δύο παρουσιάζουν μεταβατική ροή, επιτάσσοντας τη συμπερίληψη μοντέλου μετάβασης για την επίλυση της ροής. Λόγω της σημασίας της μετάβασης στην επιφανειακή τριβή και κατ' επέκταση στην οπισθέλκουσα, τίθενται δύο ξεχωριστοί στόχοι προς βελτιστοποίηση. Ο πρώτος στόχος ειναι η ελαχιστοποίηση του συντελεστή οπισθέλκουσας C_D και ο δεύτερος η μεγιστοποίηση της έκτασης της στρωτής περιοχής στην πλευρά υποπίεσης της αεροτομής, μέσω της μεγιστοποίησης της απόστασης του σημείου μετάβασης στην πλευρά αυτή από την ακμή πρόσπτωσης της

αεροτομής TP_{SS}. Αυτός ο δεύτερος στόχος αξιολογείται ως μια διαφορετική οπτική για τη μείωση της οπισθέλκουσας.

Για αμφότερες τις βελτιστοποιήσεις χρησιμοποιούνται δύο λογισμικά αξιολόγησης: το PUMA-TM και το PUMA-CNN. Το PUMA-TM επιλύει τις εξισώσεις RANS σε συνδυασμό με το μοντέλο τύρβης μιας εξίσωσης Spalart-Allmaras και το μοντέλο μετάβασης δύο εξισώσεων Smooth $\gamma - Re_{\theta,t}$. Από την άλλη πλευρά, το PUMA-CNN επιλύει τις εξισώσεις RANS και, στη συνέχεια, χρησιμοποιεί το εκπαιδευμένο ΣΝΔ (CNN) για να προβλέψει το πεδίο μ_t, κλείνοντας τις RANS.

Ενώ το λογισμικό PUMA-TM δεν έχει αρχικό κόστος, καθώς δεν απαιτεί εκπαίδευση, το PUMA-CNN έχει αρχικό κόστος που σχετίζεται με τη δημιουργία δειγμάτων εκπαίδευσης και την εκπαίδευση του ΣΝΔ πάνω σε αυτά. Ωστόσο, το PUMA-CNN επωφελείται από μειωμένο κόστος ανά αξιολόγηση, καθώς δεν χρειάζεται να λύσει τις τρεις επιπλέον διαφορικές εξισώσεις.

Οι βελτιστοποιήσεις πραγματοποιούνται με χρήση εξελικτικού αλγορίθμου υποβοηθούμενου από μεταμοντέλα μέσω του λογισμικού EASY. Τα αποτελέσματα συγκρίνονται ως προς την ποιότητα της τελικής λύσης και το υπολογιστικό κόστος. Επιπλέον, εξετάζεται αν η μεγιστοποίηση του TP_{SS} συμπίπτει με την ελαχιστοποίηση του C_D.

Nomenclature

NTUA	National Technical University of Athens
PCOpt	Parallel CFD & Optimization Unit
CFD	Computational Fluid Dynamics
AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
EA	Evolutionary Algorithm
MAEA	Metamodel-Assisted Evolutionary Algorithm
MAE	Mean Absolute Error
NURBS	Non-Uniform Rational B-Splines
LHS	Latin Hypercube Sampling
C_D	Drag Coefficient
C_L	Lift Coefficient
C_f	Friction Coefficient
TP_{SS}	Transition Point at the Suction Side
TP_{PS}	Transition Point at the Pressure Side

Contents

Chapter 1

Introduction

1.1 Laminar-to-Turbulent Transition

The transition from laminar to turbulent flow plays a crucial role in many fluid dynamic applications, ranging from flight aerodynamics [11] to heat transfer [41] processes. In aerodynamics, this phenomenon is of particular importance, as it has a significant impact on drag, since the type of flow, whether laminar or turbulent, affects the amount of skin friction at the airfoil surface.

Laminar flows exhibit a smooth, streamlined motion, resulting in lower skin friction and drag forces. In contrast, turbulent flows are characterized by irregular motions, leading to increased skin friction and higher drag. Therefore accurately predicting where and how transition takes place, allows for higher accuracy in drag calculations.

The transition from laminar to turbulent flow can be triggered by several mechanisms:

1. **Natural Transition**, involves Tollmien-Schlichting (T-S) waves, small two-dimensional instabilities within the boundary layer that can initiate the transition process [5]. T-S waves are amplified by viscous instabilities and can eventually break down into turbulent spots. As these spots move downstream, the grow and merge to create a fully-developed turbulent boundary layer [40].

2. **Bypass Transition**, occurs when high levels of free-stream turbulence are present. In this case, the turbulence in the free stream penetrate the, otherwise laminar, boundary layer, leading to the formation of turbulent spots earlier, bypassing the natural transition process [31]. The process of developing a turbulent boundary layer continues, following the principles of Natural Transition.

3. Finally, in **Separation-Induced Transition** the boundary layer separates from the surface due to an adverse pressure gradient or surface curvature [29]. The separated shear layer is highly unstable and can reattach as turbulent.

In the aerospace industry, the development of Natural Laminar Flow (NLF) wings, wings that maintain laminar flow over a significant portion of the wing surface, thereby delaying natural transition has seen increasing interest. The continuous rise in fuel costs, coupled with growing environmental awareness, has been driving the goal of reducing drag and thereby fuel consumption [7].

1.2 Artificial Intelligence

Although Artificial Intelligence (AI) has been around for a long time, it has recently resurfaced with a remarkable transformative potential across a wide range of scientific fields. This can be attributed to software progress with the inception of novel AI architectures that can describe the nature of specific problems and advanced deep learning frameworks (TensorFlow [1], PyTorch [32]) coupled with hardware advancements like the utilization of optimized Graphical Processing Units (GPUs) [44] and the development of specialized Tensor Processing Units (TPUs) providing exceptional computational power [20]. At the same time the sheer quantity of digital data available today, coupled with its accessibility and the enthusiasm of the Open Source community to innovate has further boosted this trend.

To define Artificial Intelligence one must first grasp the concept of intelligence itself. A simple yet compact definition is the ability to acquire, understand and use knowledge. So by definition AI is the effort to create artificial systems that can mimic this innate human ability. Data creation and processing encompass the acquisition of knowledge, while Machine Learning (ML) involves the discovery of patterns and understanding of information, with fast model deployment ultimately enabling the use of said knowledge to make an informed decision.

The goal of AI is to create intelligent agents. An agent is anything that perceives its environment through sensors and acts upon it through actuators [38]. Intelligence comes to fit between perception and action, to make sure the best course of action is taken.

1.3 Machine Learning

ML is the branch of AI that capacitates intelligent systems to learn from data and improve their score on specified metrics autonomously. It comprises of the algorithms that process the data, evaluate a performance metric and adapt the system to better perform based on the metric calculated. Based on the nature of

these algorithms ML can be categorized into 3 main sub-classes: supervised learning, unsupervised learning, and reinforcement learning.

1. **Supervised learning**: In supervised learning a labeled dataset is fed into the system. This means that each vector of input features (X) is associated with a corresponding output value vector (Y). The aim then is to create a system that recognizes the relationships between the input X that drives the output Y to have the known value. This system should then be able to generalize in unseen inputs, returning a decent prediction.
2. **Unsupervised learning**: In unsupervised learning the dataset is unlabeled. Therefore the focus is to discover underlying patterns and hidden structures within the data. It is unsupervised in the sense that the model is not forced to find specific connections but any structures deemed useful within the data. Unsupervised learning methods are often used in tasks like the detection of possibly false data (Anomaly Detection) or the highlighting of obsolete or irrelevant features (Principal Component Analysis).
3. **Reinforcement learning**: Reinforcement learning is basically the trial and error method of ML. In reinforcement learning the system is encouraged to learn in a game-like manner where good decisions are rewarded and bad decisions are penalized. Their interactive ever-learning nature makes them shine in dynamic environments where each outcome can also be used as feedback to optimize their performance over time.

1.4 AI in CFD

The integration of artificial intelligence, specifically convolutional neural networks (CNNs) and deep neural networks (DNNs), into computational fluid dynamics (CFD) has shown great potential in enhancing simulation efficiency and accuracy.

In [2], a CNN framework was proposed to efficiently predict the velocity and pressure fields around airfoils. The CNN was trained on Reynolds-Averaged Navier-Stokes (RANS) simulation data for various airfoil shapes, angles of attack, and Reynolds numbers. The network employed a shared encoder-decoder architecture with convolutional layers to map the geometry to the flow field outputs. The airfoil geometry was represented using a signed distance function on a Cartesian grid and was used as input to the CNN. The angle of attack and Reynolds number were then inserted into the latent space, and using transposed convolutions, the velocity and pressure fields were returned. The CNN could predict full flow fields orders of magnitude faster than RANS solvers, enabling rapid aerodynamic analysis.

In [22], a method to reduce the computational cost of RANS simulations by replacing the differential turbulence and transition model with DNNs was proposed. The DNN-based surrogate model used flow and geometrical data to estimate the turbu-

lent viscosity field required to close the RANS equations. This surrogate model was then used as the evaluation software in a metamodel-assisted evolutionary algorithm (MAEA)-based shape optimization problems, demonstrating significant reductions in computational costs.

In [12], a data-driven approach to improve turbulence and transition modeling for RANS simulations was presented. Because popular turbulence and transition models are typically calibrated based on a limited number of simple test cases, diminished accuracy is frequent when they are applied to complex flows deviating from the calibration cases. In this approach, case-specific high-fidelity data were used to infer functional corrections to account for the deficiencies in these turbulence and transition closure models. Machine learning techniques like neural networks and Gaussian processes were then applied to reconstruct the inferred corrections as functional forms based on local flow features. This enabled injecting the improved, data-driven model forms into RANS simulations to enhance their predictions.

1.5 Thesis Outline

This Diploma Thesis explores the integration of turbulence and transition modeling with Convolutional Neural Networks (CNNs), a subset of Deep Neural Networks, to replace traditional turbulence closure models and accelerate Computational Fluid Dynamics (CFD) simulations.

The chapters are outlined as follows:

- **Chapter 2**: This chapter discusses the fundamental concepts of Artificial Neural Networks (ANNs). It covers the structure of the artificial neuron, how neurons are combined to form layers, the activation functions that introduce non-linearities, and how successive layers create intelligent networks. The training process of ANNs is also discussed.
- **Chapter 3**: This chapter introduces Convolutional Neural Networks (CNNs) as a specialized subset of ANNs that excel in domains where spatial dependencies are crucial. It explains the convolutional layer, the building block of CNNs, and other components that differentiate CNNs from ANNs.
- **Chapter 4**: An introduction to Evolutionary Optimization is provided in this chapter. It describes the (μ, λ) evolutionary algorithm (EA) and its components, and how metamodels can assist in the optimization process, all within the EASY framework.
- **Chapter 5**: This chapter presents the application of CNNs to replace the one-equation Spalart-Allmaras (SA) turbulence model and the two-equation Smooth $\gamma - Re_{\theta,t}$ transition model in a Metamodel-Assisted Evolutionary Algorithm (MAEA)-based optimization of the NLF0416 isolated airfoil. Two

objective functions are used: minimization of the drag coefficient and maximization of the laminar area over the suction side. Two evaluation software are employed: PUMA-TM, which solves the Reynolds-Averaged Navier-Stokes (RANS) equations coupled with the SA turbulence model and the Smooth $\gamma - Re_{\theta,t}$ transition model, and PUMA-CNN, which solves the RANS equations and then uses CNNs to predict the μ_t field based on input fields from the RANS.

➢ **Chapter 6**: This chapter applies the same methodology as Chapter 5 to the S8052 isolated airfoil. By using a different geometry under different conditions and at a different angle of attack, this chapter aims to validate the methodology.

➢ **Chapter 7**: Conclusions are drawn and recommendations for future work are proposed.

Chapter 2

Artificial Neural Networks

2.1 Introduction

Artificial Neural Networks (ANNs) are ML components inspired by the structure of the biological neural networks in the brains of all living organisms. At the core of ANNs are neurons, gates that given inputs produce outputs. Stacking neurons together creates a neural layer, a cluster of non connected neurons. Layers containing different numbers of neurons are sequentially stacked to form an interconnected network.

The general principle is that any neuron of any layer is connected with all neurons of the previous layer, as it uses as input a learn-able linear combination of the outputs of all neurons from the preceding layer. This information flow allows them to capture complex relationships and establish important representations in the intermediate (hidden) layers.

The input layer has direct contact with the outside world, providing the data needed to perform the prediction. At each following layer each neuron receives a unique weighted sum of the outputs of the neurons from the previous layer and transforms it using an activation function.

At its essence a neural network operates by constructing linear combinations of inputs, via highly optimized, therefore fast, matrix multiplications and passing them through non-linear activation gates at each neuron, introducing the complex physics needed to describe complex problems.

A significant power of ANNs is the ease with which they can scale. Introducing more hidden layers or simply using richer in neuron layers provides with more learn-able parameters, amplifying the representative capabilities of the network. Such multi-

layer ANNs are called Deep Neural Networks (DNNs). DNNs often have millions of these parameters qualifying them for use in challenging tasks [46].

In this thesis, ANNs are used to substitute turbulence models in Computational Fluid Dynamics (CFD) simulations. They are trained to predict the turbulence viscosity μ_t, without solving the expensive differential equations used by turbulence and transition models.

2.2 Defining the artificial neuron

In nature the neuron is the building block of the nervous system. Its purpose is to transmit information via electrical signals throughout the body. Neurons consist of dendrites that propel signals from other neurons into a cell body that homes the nucleus, where the signals are processed and transformed into a single one that is carried on by the axon (Figure 2.1). Their ability to communicate with each other not only allows for information to flow throughout the body, but also for intricate signal transformations and combinations to take place, enabling the coordination of complex tasks at a lightning-fast pace.

Similarly the artificial neuron as first defined by McColough and Pitts in 1943 [30] is an element that receives multiple inputs and returns a single output.

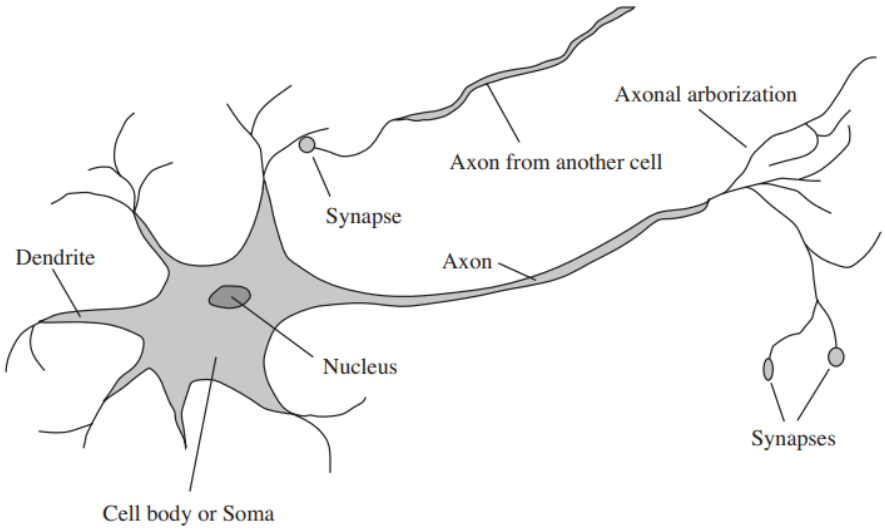


Figure 2.1: *A biological neuron. From [38].*

The input vector $x \in \mathbb{R}^n$ is multiplied by a weight vector $w \in \mathbb{R}^n$, containing learnable values that allow the neural net to adjust to the specifics of the problem it faces. This multiplication yields a value that is added to the bias b, returning $v \in \mathbb{R}$ that once passed through an activation function f produces the output of the neuron $y \in \mathbb{R}$. This artificial neuron model is illustrated on Figure 2.2.

$$\upsilon = \sum_{i=0}^{n} w_i \cdot x_i + b \tag{2.1}$$

$$y = f(\upsilon) \tag{2.2}$$

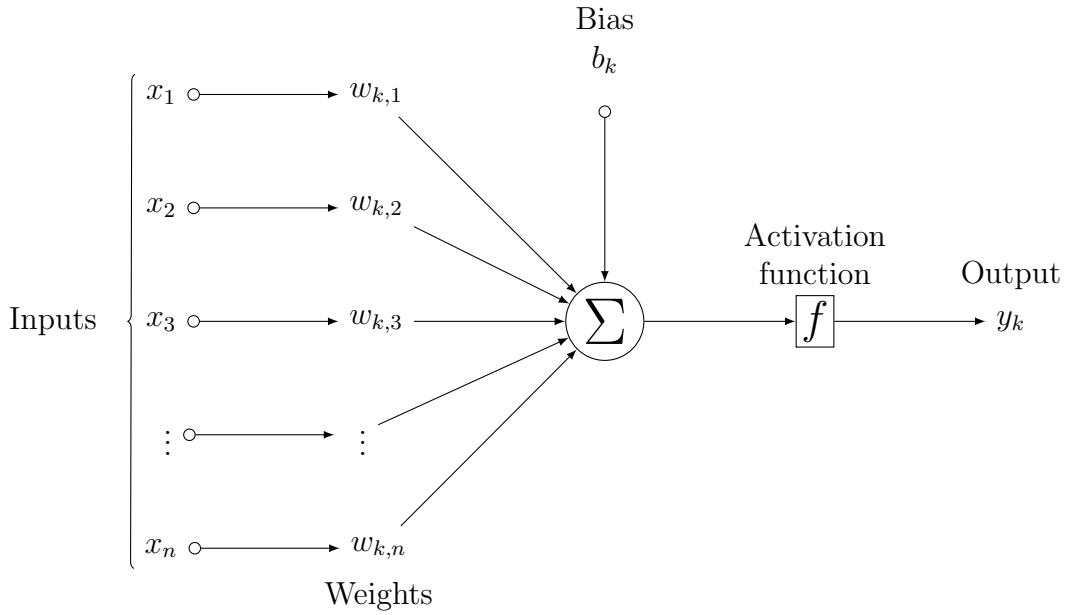


Figure 2.2: *Artificial neuron model.*

2.3 Selecting the activation function

The activation function f in large defines the neuron and inductively the neural network. A network where f is a linear function essentially declines into a linear regression model, where impractical constructions and reconstructions of linear combinations of the input parameters are performed at each layer. Furthermore adding more layers offers no performance boost as *the composition of two linear functions is a linear function itself.* This makes creating deeper networks obsolete and limits the ability of the network to model non-linear physics. Thus it is clear that a non-linear activation function is generally required.

A key characteristic of appropriate activation functions is their ability to limit the output magnitude. When the neural outputs are unbounded, instability or saturation during the training of the network might occur degrading the performance of the network. This happens because during the training process the weights gradients are heavily influenced by the neuron output values.

Table 2.1: *Basic Activation Functions.*

Activation Functions	
Rectified Linear Unit (ReLU)	$f(\upsilon) = max(\upsilon, 0)$
Hyperbolic Tangent (tanh)	$f(\upsilon) = \frac{e^{-u}-e^{u}}{e^{-u}+e^{u}}$
Sigmoid (σ)	$f(\upsilon) = \frac{1}{1+e^{-u}}$

Another aspect to be considered is the derivative of the activation function, as they play a huge role in the training process, by guiding parameter updates. In general, the derivative should be smooth to ensure stability and convergence.

Finally, there are particular cases, where the output of the network is a probabilistic quantity where the activation function must have an output range of 0 to 1.

Some of the most common activation functions are presented in Table 2.1.

Overall, selecting a suitable activation function is significant, as it affects the expressive power of the model, the range of the output variable and how the training unfolds.

2.4 Assembling the network

The neuron simple as it is, has little power by itself. But as often witnessed in nature, power is in numbers. Interconnected neurons exchange information and coordinate their computations to extract higher-level representations and capture non-linear physics. It is the collective intelligence of the many that enables the network to act smart, learning from the data and making sound predictions.

An ANN is constructed of interconnecting layers of neurons. Each neuron in a layer is connected to every neuron in the previous, allowing for flow of information. However, within each layer the neurons don't communicate with each other, acting independently based on the previous layer output and their respective weights. This happens primarily for the sake of simplicity and modularity, as it allows the engineer to view each layer as a separate entity and also the network can be described by means of simple matrix multiplications, facilitating the scaling of the system.

The connections between neurons, are basically weights that are multiplied to the output of the neuron it connects to, making the neuron at the receiving end to get a weighted sum of the outputs from the previous layer.

Expanding from Equations 2.1, 2.2 the activations of each layer can be calculated knowing the activations of the previous layer, and the weights and biases in between (chosen by the network) as presented in Figure 2.3.

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ w_{2,0} & w_{2,1} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \dots & w_{m,n} \end{pmatrix} \begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix} \right] \tag{2.3}$$

$$a^{(1)} = \sigma\left(\mathbf{W}^{(0)} a^{(0)} + \mathbf{b}^{(0)}\right)$$

Therefore the output of every layer k is:

$$a^{(k)} = \sigma\left(\mathbf{W}^{(k-1)} a^{(k-1)} + \mathbf{b}^{(k-1)}\right) \tag{2.4}$$

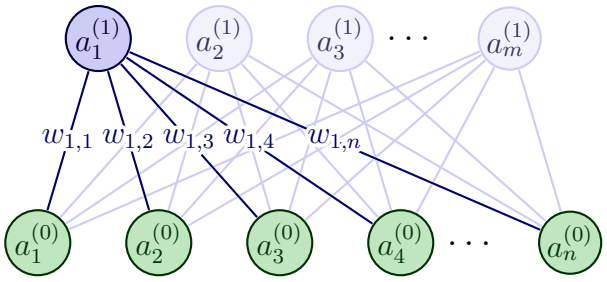


Figure 2.3: *Combining two layers.*

The names of the layers of a network, input, hidden and output reflect their respective roles.

1. **Input Layer**: The input layer is in charge of communicating with the external environment, receiving the decisive features and preparing to forward them to subsequent layers.
2. **Hidden Layers**: The hidden layers, remain invisible to the external environment as no external communication takes place. They are the powerhouse of the network where most computations take place and they are to thank for the feature extraction capabilities of the network.
3. **Output Layer**: The output layer receives the insights of the hidden layers and produces the output, returning it to the external environment

The difference between ANNs and DNNs is often vague. In general ANNs with multiple layers and nodes are called DNNs. In Figure 2.4 a small ANN is illustrated, while in Figure 2.5 a comparatively larger DNN is depicted.

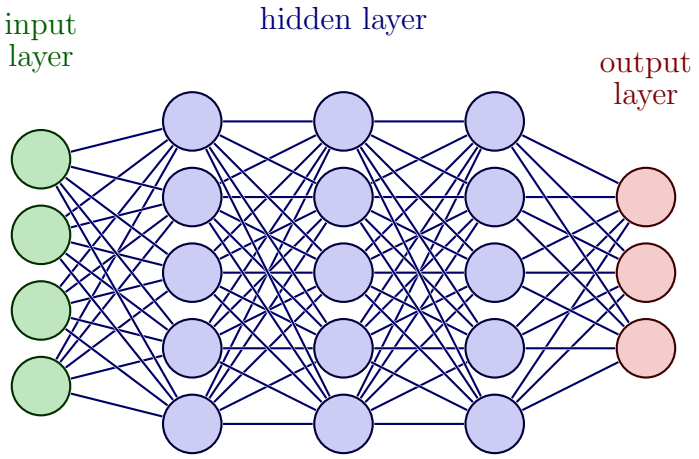


Figure 2.4: *A simple ANN.*

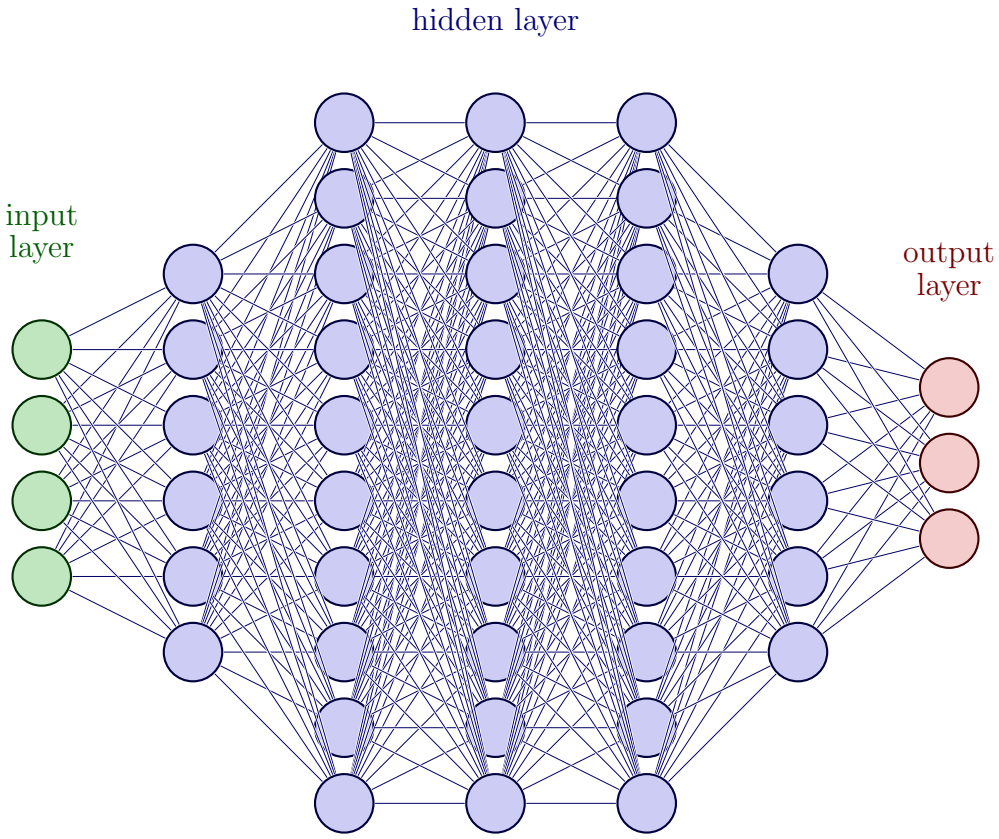


Figure 2.5: *A DNN.*

2.5 Training the network

Having constructed a network capable to represent complex physics and able to learn, the following step is to initiate the training process. To train the network a

training dataset must exist, where the values of the features X of the input layer and true values Y_t of the output layer are known. Based on X the DNN is to make a prediction Y_p that will be compared with the ground truth in order to make the network perform better.

To be most effective there are some essential concepts of the training method to be considered. First of all defining the appropriate loss function is crucial as it operates as a metric based on which the performance of the network is evaluated. Secondly, computing the gradients of the free parameters (weights) based on the error evaluated allows for weight updates in an informed direction. Lastly, an algorithm must be deployed to minimize the loss function by adjusting the weight values, in order to optimize the network for its assigned purpose.

2.5.1 Defining the loss function

Selecting the loss function used to train a network is problem-specific. The loss function offers a measure of how well the network performs and therefore guides the optimization process. It must be a metric that quantifies how close the network's predictions are in respect to the ground truth.

In general it is a function $L(Y_t, Y_p)$, where Y_t is the ground truth and Y_p the prediction, that returns bigger values depending on how far Y_t and Y_p are. In regression tasks usually the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) are used as they are a simple, interpretable quantity, the sum of the errors between the predicted and the true values passed through either the L1 or the L2 norm. Mean Absolute Percentage Error (MAPE) takes over when relative errors matter most.

1. **Mean Absolute Error (MAE)**: The average of the absolute differences between the predicted and actual values. Gives equal weight to all errors regardless of the magnitude.

$$L = \frac{1}{N} \sum_{i=1}^{N} \|Y_t - Y_p\| \tag{2.5}$$

2. **Mean Squared Error (MSE)**: The average of the squared differences between the predicted and actual values. Larger errors are amplified, as they are squared before averaging.

$$L = \frac{1}{N} \sum_{i=1}^{N} \|Y_t - Y_p\|^2 \tag{2.6}$$

3. **Mean Absolute Percentage Error (MAPE)**: The average of the absolute differences between the predicted and actual values relative to the actual values. Tends to infinity when actual values tend to 0.

$$L = \frac{1}{N} \sum_{i=1}^{N} \left\| \frac{Y_t - Y_p}{Y_t} \right\| \tag{2.7}$$

2.5.2 Calculating the gradients

A DNN can be thought of as a continuous differentiable function, where the input passes through multiple transformative layers to produce an output. With that in mind, basic calculus rules apply.

The chain rule, a well known calculus technique, allows for the computation of the derivative of a composition of functions. Starting from the output layer and moving backwards, using the chain rule, the gradient of the loss function with respect to each layer's output can be computed.

By iteratively applying the chain rule and calculating gradients layer by layer in a backward fashion, we can assume the direction in which weight adjustment should improve performance. This process, known as backpropagation [37], enables the network to learn from the training data and improve its performance over time.

2.5.3 Minimizing the loss function

Having the gradients for the free parameters, the weights and the biases, an algorithm that updates them must be deployed.

In gradient-based optimization problems it is common to use variants of the gradient descent [36]. Gradient descent is the foundational optimization algorithm used for minimizing the loss function in most machine learning tasks. It involves changing the weights in the direction of steepest descent, the direction opposite to the gradient, at a specified step and recalculating the loss and gradients.

It operates on an iterative manner adjusting the model's parameters multiple times to reach the minimum of the loss function. The start is made by an initialization of the free parameters and convergence is agreed upon specified criteria.

In standard gradient descend the entire training set is used to compute the mean gradient of the loss function at each iteration. This ensures a more precise approximation of the gradient, but is slow since a pass of the whole dataset is required before performing the weight update. At the same time, due to having a deterministic objective function, getting stuck at local minima is all too common.

$$w_{n+1} = w_n + \eta_n \cdot \nabla_w L(w(n)) \tag{2.8}$$

where η the learning rate during the n-th step of the algorithm and $\nabla_w L(w(n))$ the newly calculated loss gradient in respect to the old weights.

Stochastic (mini-batch) gradient descent (SGD) on the other hand uses randomly selected batches, subsets of training data to calculate the gradients and perform the weight update. In time all samples will be included the same amount in the optimization process so no bias is induced. It is called stochastic as the batch loss function becomes probabilistic, since for any different set of samples a different value and its gradients will be calculated, even if the weights remain the same.

Using a mini-batch allows for weight updates to take place faster. During an epoch, a full pass of the training set, the gradients are not calculated once but more often, therefore the optimization is faster and less computationally expensive. Adding a stochastic component to the equation, acts as noise and enables escaping local minima, as even if the objective function of a batch $B1$ reaches a local minimum, the same objective function for the next batch $B2$ will have non-zero gradients and therefore escape the local minimum pitfall.

Adjusting the batch size acts as a lever between stability of the gradient descend (full pass before updates) and the efficacy of stochastic gradient descent (updates after every sample).

$$w_{n+1} = w_n + \eta_n \cdot \nabla_w L_B(w(n)) \tag{2.9}$$

where η the learning rate during the n-th step of the algorithm and $\nabla_w L_B(w(n))$ the newly calculated loss gradient in respect to the old weights for batch B.

In this Thesis the Adaptive Moment Estimation (ADAM) will be used, a variant of SGD that includes first order momentum calculations [21].

ADAM comes with the best of both worlds. It comes with the stability of momentum methods, that take into account past gradients influence on the current step and adaptive learning rate methods that dynamically adjust the step size of each parameter individually based on its own gradients' history.

ADAM is known for its efficiency and is widely applied due to its adaptive nature, which makes it suitable for a wide range of problems. It typically converges faster than traditional gradient descent and is user-friendly, requiring minimal hyperparameter tuning. Table 2.2 presents the basic hyperparameters of the training

Table 2.2: *Parameters of the training algorithm.*

Parameters	
Learning rate	Base step size for the gradient descent
Epochs	Number of full iterations over the training set
Batch size	Number of samples seen per weight update
Validation split	Subset of the training set used for model validation

algorithms.

2.6 Ensuring generalization

When training a network the aim should never be solely to perform well over the training set but having adequate performance over the test set as well. The network is not asked to memorize what it has been presented with, but to make informed predictions over unseen data. The ability of the network to perform well on unseen data is called generalization and it remains a fundamental challenge in training ANNs. Thus many strategies have been developed that can aid in generalization.

2.6.1 Overfitting

A common pitfall, that hinders generalization, when training a network is overfitting.

Overfitting occurs when a model is fit beyond necessity. An overfit model performs very well during training but when asked to predict on unseen data it underperforms. An overfit network has learnt the inherent noise in the data and doesn't actually understand the underlying physics of the problem. Therefore it can't perform that well on unseen data as it takes into account noise that shouldn't be present.

Overfitting appears in networks that:

1. Are strong enough to learn more than they should
2. Are trained long enough to learn more than they should

3. Are trained in a static manner that gives them the confidence to learn more than they should

2.6.2 Small models

An effective approach with a proven impact on aiding generalization is to use smaller models. Often the simplest explanations that are good enough are explain a phenomenon must be preferred. In the context of ANNs smaller, simpler models are preferred over more complex models if they perform nearly as well on the given task.

Small models have fewer parameters and are less prone to overfitting during training. Because they have less learning capacity the are unable to fit all the noise present in the entire training set to improve their score further and therefore they settle with capturing the real underlying physics.

Other than aiding in generalization, being less complex allows for faster training and inference times. At the same time the low computational cost comes in pair with a lower memory footprint.

2.6.3 Dropout

Dropout is a regularization technique used to improve generalization in DNNs.

A dropout layer has neurons that can randomly be deactivated during training. One can choose the fraction of neurons that will be off at any training step. By doing this the network is discouraged from relying too heavily on specific neurons, encouraging lower weight values and a more uniform distribution accross the many neurons and creating a more robust network. It is important to note that during inference all the nodes are on and no dropout is in place, thereby all the predictive power of the network is unleashed.

2.6.4 Pruning

Pruning is a common technique with the purpose of simplifying and optimizing a DNN. It involves transforming a fully connected model into a sparsely connected one, by selectively removing connections from an already trained network. It is based upon the fact that biological brains are highly sparse [15].

With pruning a fully connected model is transformed into a sparsely connected one. It involves selectively removing connections from an already trained network to improve its efficiency, to reduce its memory footprint and accelerate inference time. Its motivation lays in the fact that many times DNNs appear to be over-parameterized, meaning that they contain more parameters than needed to describe the problem at hand. This can result in increased training times and slower inference

speeds, but also in overfitting. Pruning addresses these issues by identifying and eliminating redundant components of the network.

Most methods are usually magnitude based. Individual weights in a pre-trained network are ranked based on their magnitude and the lowest-ranking weights are removed. This is based on the assumption that the connections with smallest magnitudes contribute the least to the prediction of the network. In this method a threshold value is selected, below which all weights are pruned.

In other specific cases though, structured pruning techniques are applied, sometimes even prior to training the network [6].

Chapter 3

Convolutional Neural Networks

3.1 Introduction

Convolutional Neural Networks are deep learning architectures inspired by the human visual cortex. They were first used in the realm of image analysis [26], but their specified processes can be applied to any field where spatial dependencies are of high relevance in the input data.

CNNs are a subclass of DNNs, and as such, they are similar in a fundamental manner. Where in DNNs there are layers of nodes, in CNNs, there are layers of matrices (Figure 3.1). In both of them, layers receive a linear combination of the previous layer output and then apply a non-linear transformation through an activation function. It is this hierarchical structure that enables both of them to capture intricate patterns.

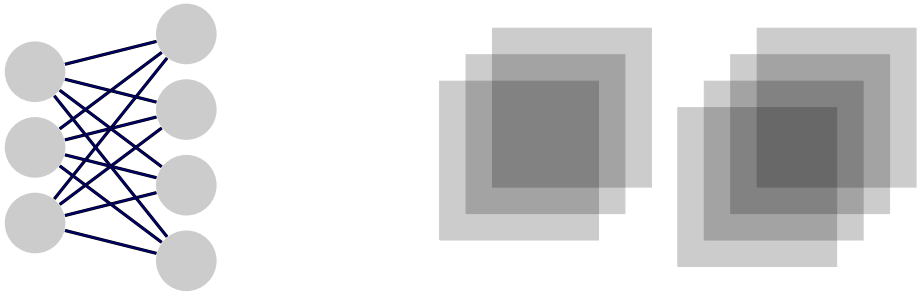


Figure 3.1: *DNNs operate using nodes and CNNs operate using channels. A DNN layer takes 3 nodes and returns 4 nodes (left), while a CNN layer takes 3 channels and returns 4 channels (right).*

Unlike traditional DNNs that fully connect each neuron from one layer to the next,

CNNs use convolutional filters, basically kernels of weights, that scan the input spatially, capturing local patterns in small receptive fields. By leveraging parameter sharing in the convolutional layers, CNNs significantly reduce the number of learnable parameters, making them computationally efficient [25]. Reducing the number of learnable parameters introduces a built-in inductive bias that can guide the CNN to learn a better approximation and generalize better. This bias is primarily characterized by the assumption of spatial and translation invariance. This means that a certain pattern (kernel) learned in one region of the input is likely to be relevant and useful in other regions as well. By encoding this prior knowledge into the model architecture, CNNs have been proven superior in many tasks such as image recognition, object detection, and semantic segmentation [26, 25, 27].

In this thesis, Convolutional Neural Networks (CNNs) serve as replacements for traditional turbulence models in Computational Fluid Dynamics (CFD)-based aerodynamic optimizations. Their purpose is to predict the turbulence viscosity (μ_t) field without the need to solve the computationally expensive differential equations utilized by conventional turbulence and transition models. Unlike Deep Neural Network (DNN) approaches [23], which solely rely on information from individual mesh elements, CNNs integrate flow and geometrical data from neighboring mesh elements, aiming to enhance prediction accuracy.

3.2 Basics of the Architecture

To gain a first understanding of the CNN architecture, the image processing paradigm is ideal as it is intuitive and commonplace.

Consider an RGB image with $H \times W$ pixels. The image configuration is represented by three $H \times W$ matrices, one for each of the three channels R (red), G (green), and B (blue). The values range between 0 and 255, reflecting the intensity of the basic colors in each pixel. In short, the form of the input data consists of tensors, multidimensional matrices with a shape of (H, W, C), where H is the height of the image, W the width of the image, and C the number of channels in the input field (typically 3 in the realm of images).

Let's take a small 100x100 multicolor image and try to process it through an ANN. The input tensor has a size of $100 \times 100 \times 3$, therefore an input layer of 30 thousand nodes would be required. Even an excessive 10-fold reduction to the next layer, would require a second layer of 3 thousand nodes, resulting in 9 million free parameters in just the first 2 layers. That is a prohibitive number for most DNNs, not to mention the impossibility of creating a $30k \times 3k$ matrix of unique elements. Therefore, another approach must be investigated.

Two crucial concepts, related to the statistical properties of an image's pixel values can be used to our advantage.

1. **Locality of Pixel Dependencies:**
    - The locality of pixel dependencies underscores that the relationship between pixels is typically stronger for nearby pixels than for those farther apart.
    - This implies that examining relationships within small local areas is more valuable for understanding image patterns.
    - Therefore connections between far apart pixels are not required.
2. **Stationarity of Statistics:**
    - Stationarity of statistics in the context of images suggests that certain statistical properties, such as mean, variance, and correlation, remain relatively constant across different regions of an image.
    - Because of that features learned in one part of an image can be applicable in other parts [25].
    - Therefore no different parameters are required to treat different parts of one image.

Convolutional Layers, layers performing convolutional operations, fully exploit both of these concepts, as they operate on tensors using kernels, small windows that indicate the neighborhood of a given pixel. These kernels are used as filters sliding through matrices to produce new matrices, where each new value is a linear combination of the neighboring values. In the context of image processing, Convolutional Layers can detect simple features like edges, corners, or textures by convolving over the picture values. The deeper in the network, the more sophisticated the features that can be detected. Usually, when referring to a Convolutional Layer, one includes the convolution operation that, using the layer's kernel, performs a linear combination, as well as the activation layer that applies the activation function inserting the non-linear element in the network.

Most CNNs are employed with some kind of downsampling operators, usually Pooling Layers. These aim to reduce the spatial dimensions of the feature maps by considering regions and selecting representative values for each one. This not only reduces the computational burden of the network but also encourages the network to focus on more high-level features, filtering out minor details and aiding in generalization. Understanding that there are 4 red pixels in a fine-grained region is no more helpful than knowing 1 red pixel is in its equivalent downsampled region. At the same time, since many CNNs only require a low-dimension output (classification tasks), downsampling allows for a smooth transition.

Occasionally, CNNs may use upsampling operators to invert the dimensionality reduction realized by Pooling Layers. These are often used when the network is required to output higher-dimension outputs (depth maps, segmentation maps, etc.), therefore restoring the spatial dimensions is necessary [4].

3.3 Convolutional Layers

Figure 3.2 presents an example CNN. Similar to a DNN, the input layer contains the inputs that are successively passed through the hidden layers to reach the output layer. In this illustration, each square represents a matrix A, and each connecting line represents a convolution C using a trainable kernel K.

Each matrix in a layer is produced as follows: a 2D convolution is performed on each matrix of the previous layer, and the resulting matrices are summed element-wise. The resulting matrix is then passed through an activation function f, similar to the process in DNNs.

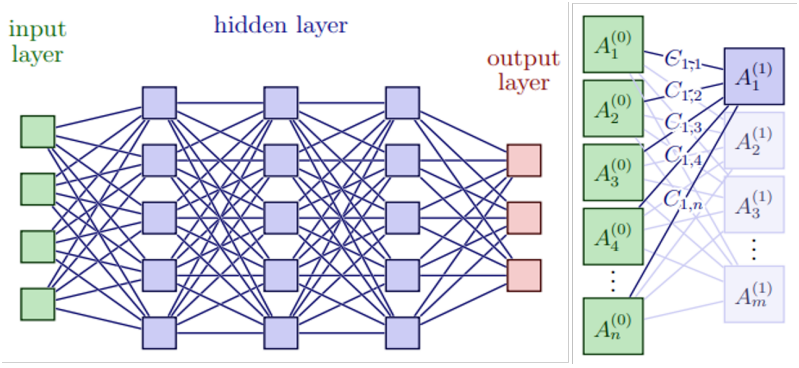


Figure 3.2: *An example of a Convolutional Neural Network (CNN). Each square represents a matrix A, and each connecting line represents a convolution C using a trainable kernel K.*

3.3.1 The convolution

The convolution is a fundamental mathematical operation that can be applied in matrices.

It involves sliding a filter, the kernel, over the input matrix in a systematic manner and computing the element-wise dot product between that and the region of the input it overshadows. Performing this process with the same kernel across all locations in the input matrix results in a feature map that highlights the presence of specific patterns. The size of the filter kernel is usually much smaller than the input as it is used to extract localized information. The type of patterns to be highlighted are determined by the values of the kernel. For example:

- A $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$ kernel is used to detect horizontal edges and a

- A $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ kernel is used to detect vertical edges in an image.

In mathematical terms the convolution performed on an input tensor I of size (H,W,C), where H,W are the spatial dimensions of the input and C the channels, by a kernel K of size (H_k,W_k,C) produces the a 2-dimensional feature map F as follows:

$$F(i,j) = \sum_{c=1}^{C} \left(\sum_{h=-h_k}^{h_k} \left(\sum_{w=-w_k}^{w_k} (I(i+h, j+w, c) \cdot K(h,w,c)) \right) \right) \tag{3.1}$$

for $i \in [h_k, H - h_k]$ and $j \in [w_k, W - w_k]$, with $h_k = \frac{H_k - 1}{2}$ and $w_k = \frac{W_k - 1}{2}$

It is interesting that the number of elements in the convolutional kernel is $H_k \times W_k \times C$, independent of the input's spatial dimensions W, H.

In Figure 3.3 a convolution is performed on a 2D matrix. In a similar manner the convolution is performed in 3D.

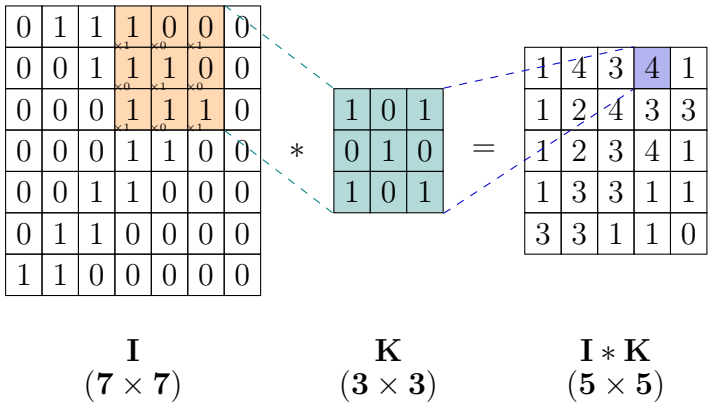


Figure 3.3: *The Convolution Operation on a 2D matrix. From [34].*

The convolution operator needs access to h_k cells below and above and w_k left and right of the cells to be convolved. Therefore in a plain convolution the shape of the matrix is set to be reduced by $(H_k - 1)$ and $(W_k - 1)$ in height and in width respectively as to accommodate for the convolution of the boundary cells. This is often disorienting when successive convolutions take place, so padding techniques are employed to counteract, as discussed in a following section.

In a convolutional layer, convolutions are performed multiple times using the input to create multiple feature maps. That is because alternate representations are needed to grant the necessary representational power to the network. The number of feature maps, or the size of the output channel axis is a defined parameter similar to the number of nodes in the hidden layers of a DNN. This number determines the number of convolutions to take place, or the number of filter kernels to be used therefore it

can be referred to as filters. So at each convolutional layer the number of weights incorporated is:

$$N = H_k \times W_k \times C_i \times C_o \tag{3.2}$$

where (H_k, W_k) the kernel shape and C_i, C_o the depth of the input and output tensor respectively.

This behavior is presented in Figure 3.4 where kernels of different colors have the same shape, but use different weights to produce different feature maps.

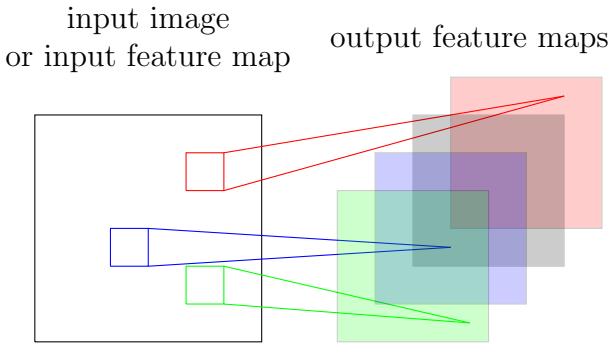


Figure 3.4: *Multiple convolutions in a Convolutional Layer create different feature maps. From [34].*

3.3.2 The kernel weights

In a Convolution Layer the weights of the kernels performing convolutions are learnable parameters. That is because we require the network to be intelligent enough to find the kernels that detect the most representative features and the patterns that create the most informative maps in regards to the task at hand.

In the same way that weights in a DNN allow the network to learn, in a CNN it is the kernel weights that possess the required neuroplasticity.

The convolution operation in CNNs leverages the concept of parameter sharing. The same filter weights are used across the entire input space along the spatial dimensions H, W. This parameter sharing reduces the number of learnable parameters, making the network computationally efficient and allowing it to generalize better. Using the same kernel to slide across the entire matrix, also introduces the invaluable property of spatial invariance, where patterns are recognized irrespective of their exact location in the input, a property crucial for image recognition tasks.

3.3.3 Strides and padding

In practice, the convolution operation can be customized by introducing stride and padding parameters.

Padding, as mentioned before, is of high importance as it allows to preserve spatial dimensions after convolution. If no padding is added to the input, the convolutional kernels are only applied to positions where the kernel can fit entirely around the pixel. This results in a boundary of dead pixels after each convolution, therefore decreasing the spatial dimensions of the feature maps, as illustrated in Figure 3.5. This reduction may lead to the loss of valuable information, especially when there is such in the edges of the input.

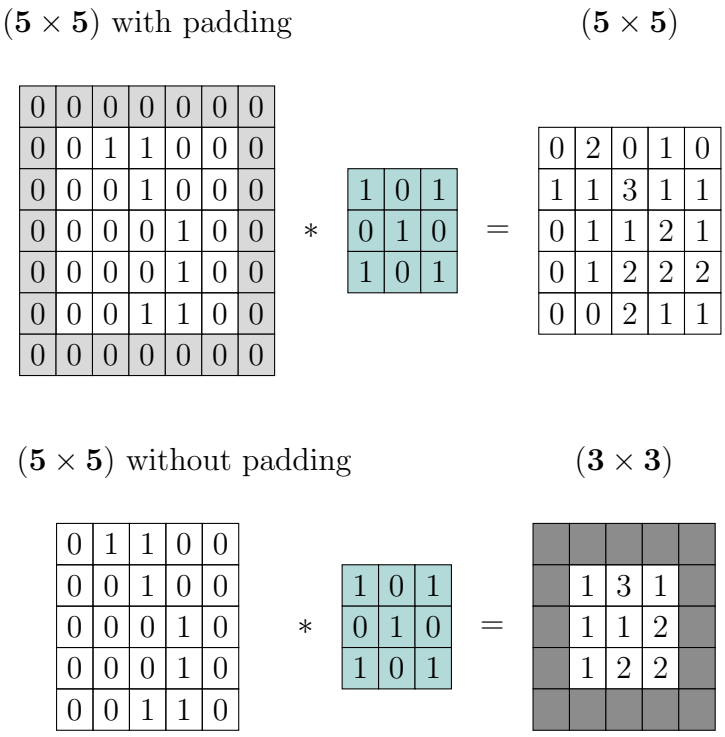


Figure 3.5: *Using padding retains spatial dimension. Convolution without prior padding results in lower spatial dimensions, as a boundary of dead pixels appears.*

Furthermore, when no padding is applied, after each convolution the output map's dimensions are determined by the size of the convolutional kernel. Therefore it is harder to keep track of the feature maps dimensions, especially when multiple layers are stacked, needlessly hindering the design process.

Padding is applied in a way that the feature map after the convolution retains the dimensions of the input. For a unit-stride convolution, the padding size can be determined by just the kernel size as $(\frac{H_k-1}{2}, \frac{W_k-1}{2})$.

There are three main categories of padding:

1. **Zero Padding**: Zeros are added around the input feature map.

2. **Mirror Padding**: The mirrored reflection in respect to the edges is added around the input feature map.

3. **Replicate Padding**: The border pixels are duplicated to provide the padding.

Stride determines the step size at which the filter moves over the input. A larger stride results in smaller feature maps with less overlap, as each pixel belongs in a smaller number of neighborhoods. Larger strides also reduce the number of computations proportionally, enabling faster training and inference times.

Finally, non-unit strides reduce the spatial dimensions of the feature maps, effectively downsampling the data. It should be considered that the number of weights is not affected by the stride size, therefore the representative power of the network is not affected.

Figure 3.6 showcases how different stride convolutions scan through a matrix.

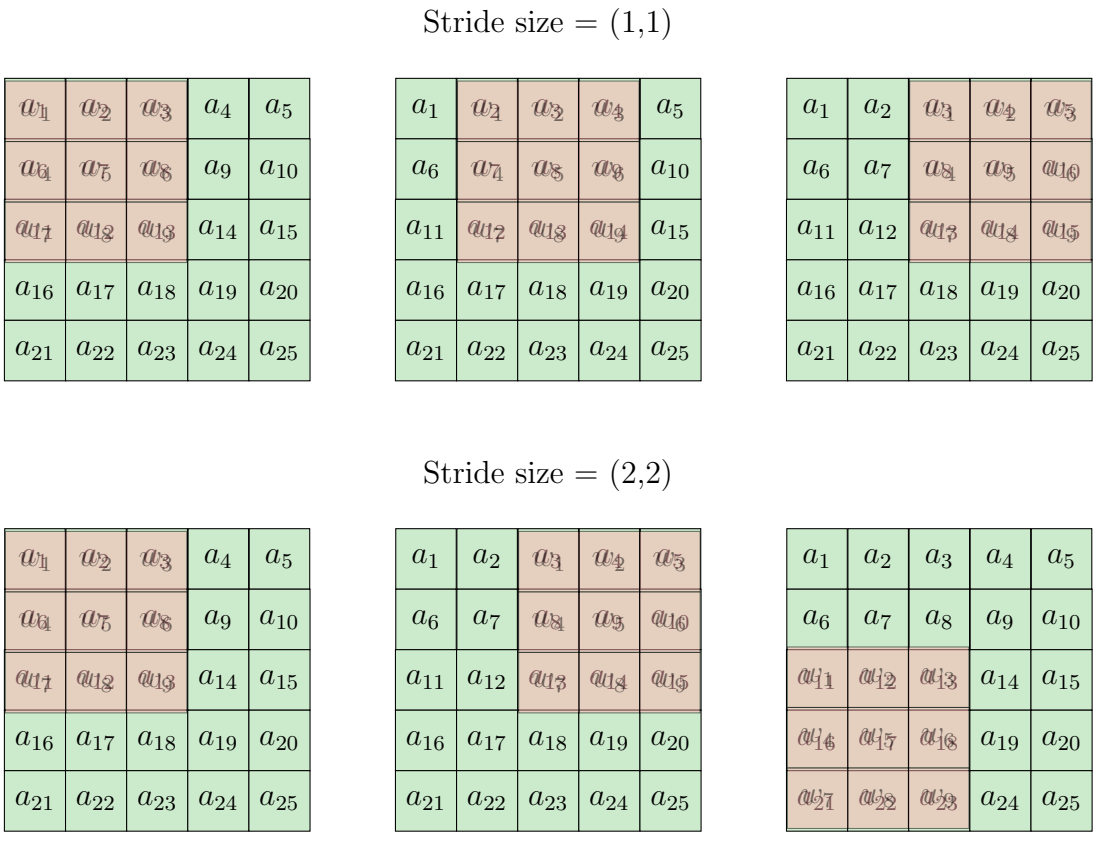


Figure 3.6: *Showcasing how stride works.*

3.3.4 Stacking convolutional layers

Stacking convolutional layers by itself yields no benefit. As the convolution is a linear operation, a combination of convolutions is still a convolution. Therefore, only a linear combination of the input tensor can be produced this way.

The non-linearity is introduced by applying an activation function to each pixel following each convolutional layer. The activation functions are no different than the once used in DNNs.

With appropriate non-linearities included, stacking convolutional layers offers an impactful approach to extracting meaningful features from the input. In one sense more layers mean more kernel weights, therefore more learning capacity. In another, the sequencing allows for deeper and deeper representation to take place.

Moreover the accumulation of layers expands the network's receptive field. With every convolution the neighborhood of a given pixel is broadened. As a pixel is influenced by its neighboring pixels defined by the kernel, it then becomes part of a larger neighborhood that affects subsequent pixels through subsequent convolutions (Figure 3.7). This progressive widening of the field of view enhances the network's perception and enables it to detect large scale patterns.

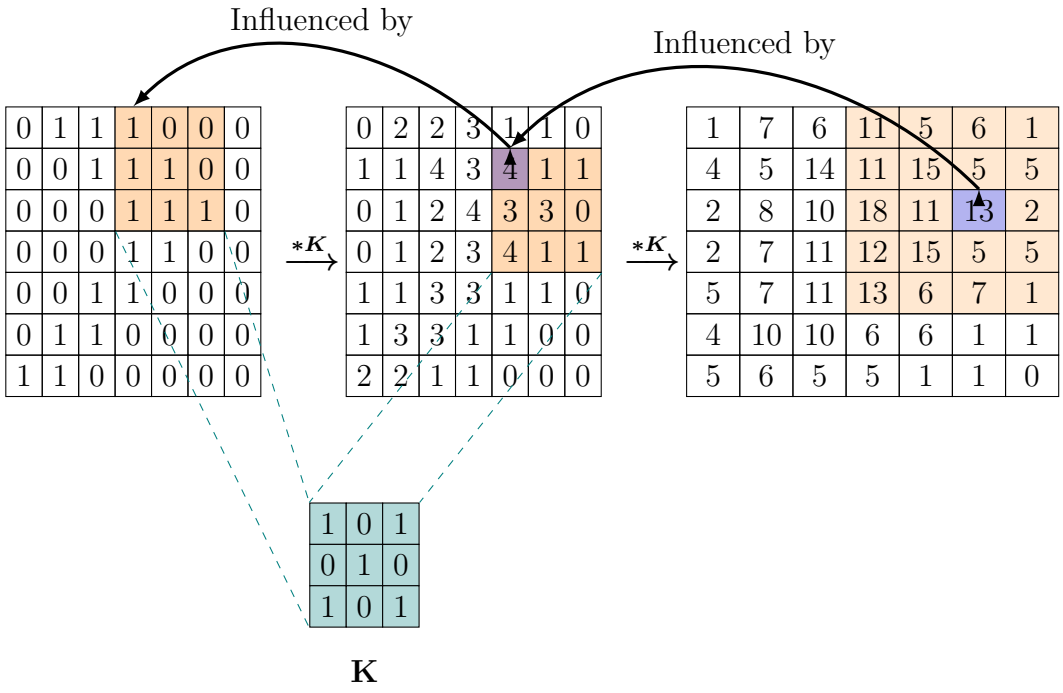


Figure 3.7: *Stacking convolutions increases the field of view.*

3.3.5 Comparison to a Fully Connected Layer

A convolutional layer can be thought of as a heavily pruned fully connected layer. Consider a feature map A of shape (4,4) that is convolved into feature map B of shape (4,4) using a (3,3) kernel K.

During convolutions, different cells (corner, edge or central) use a different number of cells from the previous layer to perform a crossproduct as seen in Figure 3.8. The 4 corner pixels of B are connected with 4 of A's pixels, the 8 edge pixels are connected with 6 of A's pixels and the 4 middle pixels are using the entire kernel being connected with 9 pixels from A. Therefore the total number of connection N_C are:

$$N_C = \overbrace{4 * 9}^{mid} + \overbrace{8 * 6}^{edge} + \overbrace{4 * 4}^{corner} = 100 \tag{3.3}$$

But thanks to parameter sharing the number of parameters is even less, just the number of kernel elements, in this case 9.

$$N_W = 3 * 3 = 9 \tag{3.4}$$

To fully connect these two feature maps, first they would be flattened and then each of B's 16 elements would be connected to each of A's 16 elements, with distinct weights acting as connections. So both the number of connections and the number of weights would be 256, as depicted in Figure 3.9.

Therefore even in this simple example by leveraging the convolution operator there is a 60% reduction in connections and a 95% reduction in free parameters. Reduction in connections acts as pruning, allowing for a more coherent flow of information and aiding in realising better representations. Having less free parameters, in some sense reduces the design space, the total possible weight combinations and simplifies the optimization performed by stochastic gradient descent.

3.4 Downsampling and Upsampling

Downsampling and upsampling techniques provide CNNs with the ability to manage spatial dimensions effectively, allowing for the modification of granularity and enabling the network to perceive both local details and global context despite the local nature of convolutional operators.

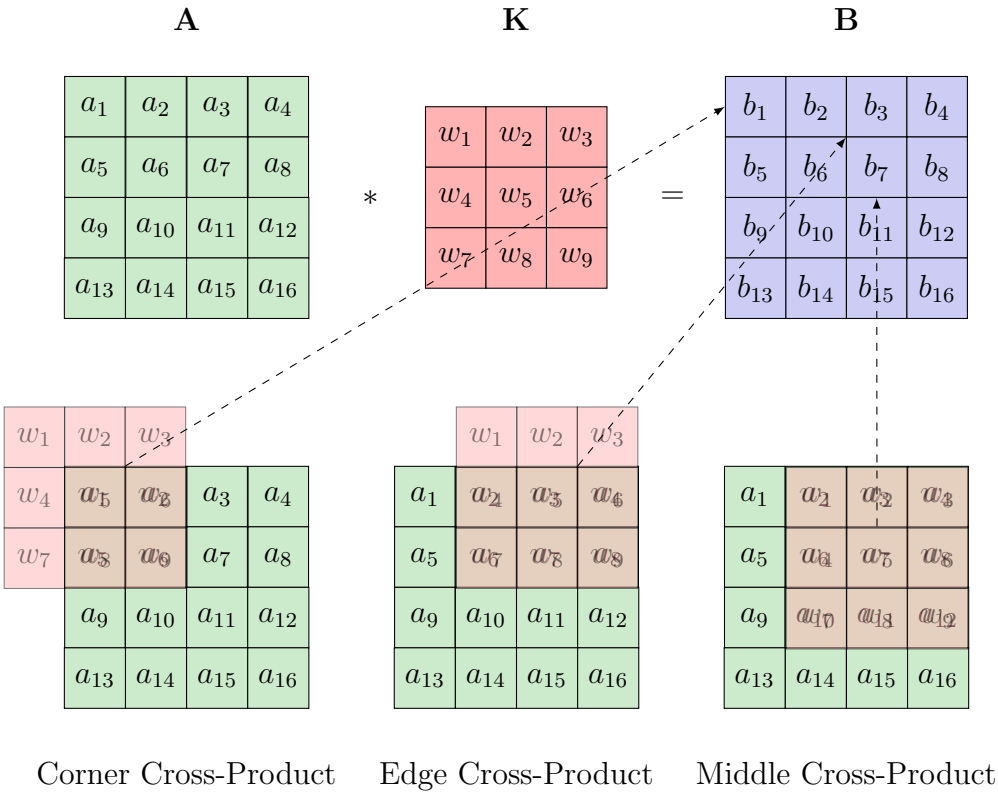


Figure 3.8: *Performing cross-products on different cells.*

3.4.1 Downsampling

Downsampling is an operation that summarizes over a spatial neighborhood [39]. It is often executed through Pooling Layers that trim the spatial dimensions of feature maps by selecting representative values from local regions. The region is defined by a pooling kernel that works in a way similar to the convolutional kernel. The strides define how much the kernel moves and the method of selecting the representative values is user-defined.

1. **Max Pooling** chooses the largest value in the region as a representative value to further in the next layer (Figure 3.10).

2. **Average Pooling** takes the mean of the values within the region, so that each pixel contributes to the following layer. (Figure 3.11).

In Figure 3.12 a pooling operation over multiple feature maps is presented. Square regions in the original matrices are mapped to a single point in the downsampled matrices.

Downsampling offers a dual advantage – it reduces computational load and improves generalization.

$\mathbf{N}_C = 256$ connections

$\mathbf{N}_W = 256$ discrete weight values

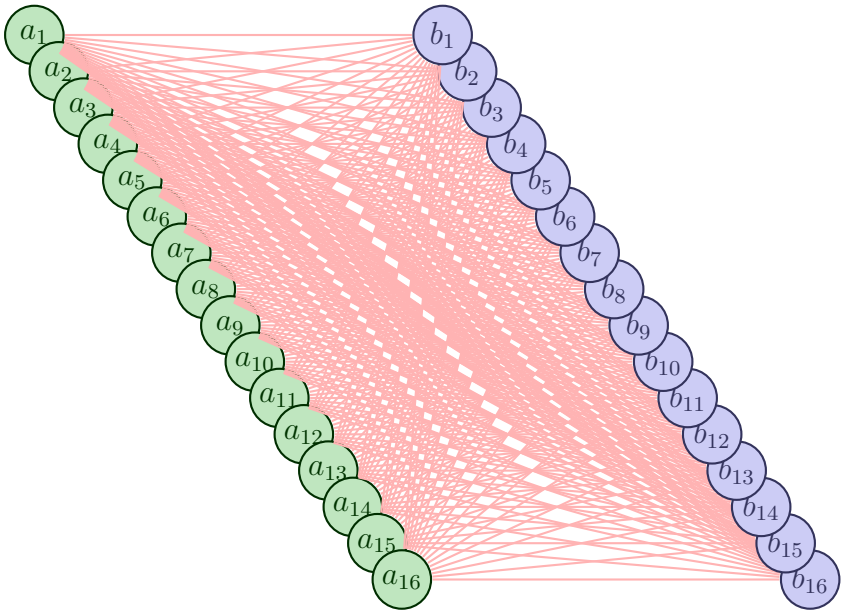


Figure 3.9: *Fully connecting feature maps.*

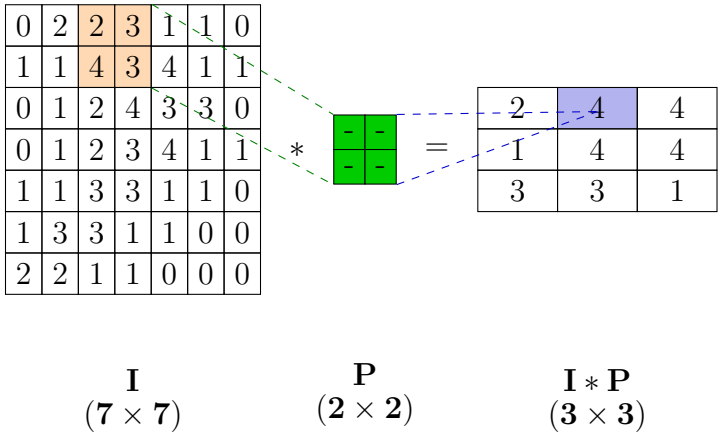


Figure 3.10: *Max Pooling.*

By trimming the spatial dimensions the convolutional kernels are required to scan through a smaller input, therefore reducing the number of calculations required. As the sum-product is performed at each pixel, reducing the number of pixels to half reduces the number of calculations to half.

By selecting representative values from local regions it encourages translation invariance. It ensures that the network recognizes features irrespective of their precise positions in the input, as the appearance of said features becomes more important

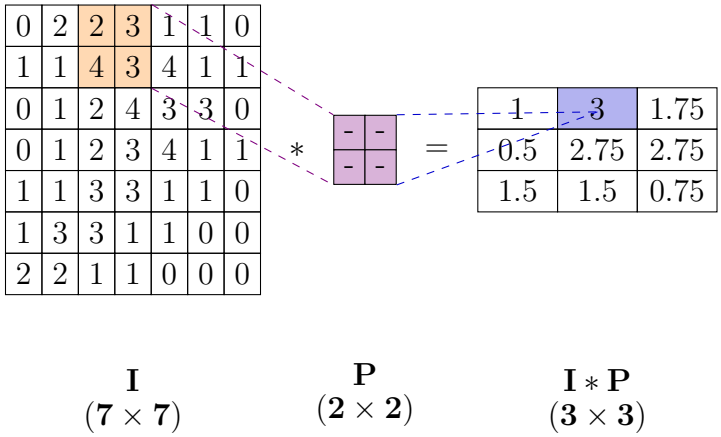


Figure 3.11: *Average Pooling.*

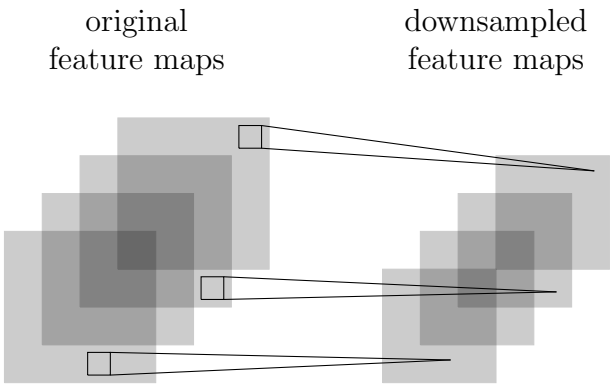


Figure 3.12: *Pooling multiple feature maps. From [45].*

than the exact pixel region where they appear.

3.4.2 Upsampling

Upsampling techniques are used to reverse the effects of downsampling and restore lost spatial information. This is of high importance in tasks where high-dimension outputs are required. In image segmentation [35] and generation [49] the network should reinstate the initial spatial dimensions as a per pixel prediction is required. In super-resolution tasks [10] the CNN is asked to produce an even larger image than the input it received so by definition upsampling is required.

Upsampling methods, like transposed convolution and bi-linear interpolation, expand the dimensions of feature maps, allowing the network to generate outputs with higher resolution.

1. **Bi-linear interpolation** is the most straightforward of the two. It involves inserting new pixels with values assigned by considering the weighted average

of the neighboring pixels in the original feature map. It is computationally efficient and produces smooth feature maps, but since it contains no learn-able parameters, it struggles where sharp non-linear transitions take place.

2. **Transposed Convolution**, also known as deconvolution, is the main learn-able approach to upsampling feature maps. They allow the network to learn the best way to upsample rendering them superior in tasks requiring precise spatial information.

In Figure 3.13 an upsampling operation over multiple feature maps is presented. Small regions in the original matrices are mapped to larger regions in the upsampled matrices.

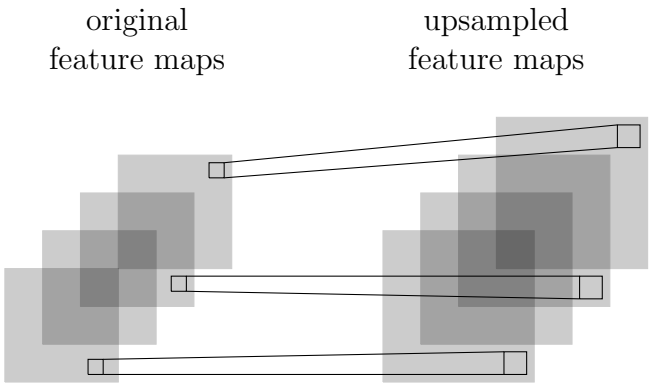


Figure 3.13: *Upsampling multiple feature maps.*

3.4.3 Skip connections

Upsampling operations are often accompanied with skip connections, connections that allow the flow of information from the primitive layers into later layers of the network without any alteration, addressing some of the more common obstacles faced during training.

Skip connections are basically shortcuts within the network. Unlike traditional feed-forward network that use a strictly sequential flow of information, skip connections connect non-adjacent layers skipping the layers in between (Figure 3.14).

The main motivation for their design was to address the vanishing gradient problem in very deep networks, where weight gradients start to get incredibly small during backpropagation, leading to slow or even stagnant convergence. Skip connections, provide a shortcut for the gradients to flow to these more primitive layers they connect to, limiting the vanishing gradients and facilitating faster training.

At the same time skip connections encourage feature reuse by enabling earlier layers to contribute to the final prediction. In a way it allows the network to choose whether to use lower or higher level features aiding in generalization.

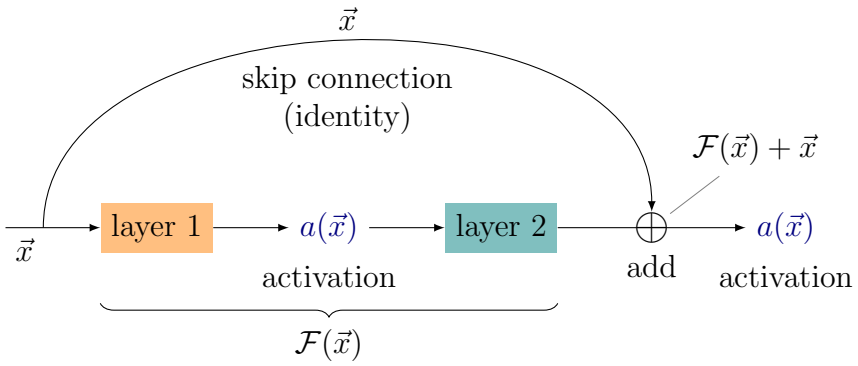


Figure 3.14: *A simple skip connection. From [34].*

Residual networks [19], the pioneering architectures that included skip connections use them, inside the residual block, a block of convolutional layers were the input is also added to the last layers output.

U-Net [35] is a popular architecture for tasks like image segmentation and medical image analysis. It employs skip connections to combine low-level feature maps from encoder layers with high-level feature maps from decoder layers, enabling precise segmentation.

3.5 Training

The training of a CNN is identical to that of a DNN. The same loss functions can be used and the network can be considered as a differentiable function. Utilizing the chain rule the gradients are calculated and using the common gradient-based optimization algorithms the best weights can be achieved.

Chapter 4

Evolutionary Optimization

4.1 Evolutionary algorithms

Evolutionary Algorithms (EAs) constitute a class of optimization algorithms inspired by the mechanisms of biological evolution. These algorithms are particularly effective in solving complex optimization problems where traditional methods may struggle.

The fundamental concept involves mimicking the process of natural selection [8] to iteratively improve a population of candidate solutions over generations. Starting with an initial population, randomly or otherwise selected, all individuals are evaluated based on a pre-defined objective function. "Survival of the fittest" is ingrained in the process, as the best-performing individuals are selected to reproduce. Reproduction, executed through probabilistic recombination of parents, employs user-defined functions to generate the next generation. Any offspring might be born with a mutation, introducing diversity to the algorithm. This process is repeated until either a certain number of generations is reached, or the best-performing individual is deemed good enough.

The main advantages of Evolutionary Algorithms are presented below:

1. **Global Optimization:** Evolutionary Algorithms (EAs), utilizing global populations, are capable of exploring the solution space entirely [13], therefore easily avoiding being trapped in local optima. This wide perspective enhances the likelihood of finding the best solution across the entire search space.

2. **Adaptability:** Evolutionary Algorithms exhibit a high degree of adaptability, as they do not require access to the source code, making them particularly well-suited for dynamic and changing problem environments [24].

3. **Parallelism:** EAs can be easily parallelized since they operate with populations, where each member can be independently calculated [14]. This allows for the exploration of multiple solution paths concurrently, making them a perfect fit for modern parallel computing architectures, tackling large-scale optimization problems.

4. **Versatility:** One of the notable strengths of EAs lies in their versatility. These algorithms are based on low-level math operations and that makes them applicable to a wide range of problems and domains [48]. That adaptability makes them a go-to choice for various applications.

5. **No Need for Derivatives:** Finally, unlike most traditional optimization methods that rely on derivatives of the objective function to determine the direction of parameter updates, EAs operate without requiring derivatives, using collective intelligence. This characteristic makes them applicable to problems with non-differentiable or discontinuous functions or simply functions hard to differentiate, expanding their utility to a broader class of problems. [13]

4.2 The Evolutionary Algorithm SYstem Software - EASY

The Evolutionary Algorithms SYstem (EASY) software [18], developed by PCOpt/NTUA, serves as a versatile optimization tool for solving single or multi-objective problems, including constraints. Utilizing a (μ, λ) EA approach, EASY incorporates both stochastic and deterministic optimization methods, supports various optimization schemes and includes distributed, asynchronous, and hierarchical Evolutionary Algorithms , while granting users a high degree of control through tunable parameters.

EASY introduces the integration of low-cost surrogate evaluation models, such as RBF Networks, functioning as online-trained metamodels, as well as, the use of external off-line trained metamodels establishing the framework for Metamodel-Assisted Evolutionary Algorithm (MAEA) optimization.

4.3 Components of an Evolutionary Algorithm (EA)

Evolutionary Algorithms (EAs) have a complex architecture, consisting of several different components that grant them the power to tackle complex optimization problems. Understanding these components is essential for understanding the mechanics of EAs and why they are able to improve solutions over generations.

Encoding

Encoding plays a fundamental role in Evolutionary Algorithms (EAs) as it defines the language through which the algorithm communicates and represents potential solutions within the population. In essence, encoding is the method by which candidate solutions are translated into a format that the EA can manipulate.

Most common encoding methods are **Binary**, where the genome is represented as a string of binary digits, and **Real** where vectors of real numbers represent individuals. **Binary-Gray** on the other hand, is a variant of binary encoding where adjacent numbers in the encoding scheme differ by only one bit. This can help in preventing large changes in the phenotype when small changes occur in the genotype.

In this thesis, Real Encoding will be used.

Population

A population in an EA is any set of candidate solutions to the optimization problem that are evaluated using the objective function [13]. In the (μ, λ) EA, during each generation, the parent population μ is created by the offspring population λ by the selection mechanism and in turn produces new offspring through means of crossover and mutation. An initial offspring population is required, usually created by randomly sampling the design space.

In this Thesis, the parent and the offspring population will contain $\mu = 10$ and $\lambda = 24$ individuals respectively.

Objective Function

The objective function F, is the metric that evaluates the performance of each individual solution. Minimizing this objective function acts as the goal, driving the algorithm toward solutions that fulfill the optimization criteria.

Selection Mechanism

The selection mechanism is inspired by nature's "survival of the fittest." It determines the individuals from the current population that will be chosen to reproduce and contribute to the next generation. Choosing the ratio between parents and offspring determines the selective pressure inside a population.

Selection methods include, rank-based approaches, where only the top N candidates are allowed to reproduce and probabilistic approaches, where all members of the population are given a chance to reproduce, but probability is assigned proportionally to the fitness of each individual.

In this Thesis, tournament selection will be used, with a tournament size of three. For each of the μ parents to be selected, three of the λ offspring will be randomly chosen to enter a tournament where the fittest of the three will be promoted to the parent population.

Crossover

Crossover entails the creation of new individuals by combining genetic material from selected parents. The crossover mechanism controls how the genetic information of two or more parents is split and recombined to produce offspring. This process aims to create more fit individuals by combinining beneficial traits from well-performing candidate solutions.

In this Thesis, every offspring is produced from 3 parents, using the Simulated Binary Crossover [9].

Mutation

Mutation is the random alteration of the genetic information of an individual. It is an element that introduces new characteristics to the population, ensures diversity and facilitates escaping local optima. At the same time, introducing mutations disrupts the deterministic nature of selecting the initial population. Regardless of the initialization, the introduction of mutations ensures that the algorithm is potent to explore the entire design space.

The mutation rate determines the probability of a mutation occurring in an individual. Adjusting it, tunes the balance between deterministic and stochastic behavior, explorative and exploitative nature of the EA.

In this Thesis the initial mutation rate is 5% and it is increased when 10 consecutive idle generations occur.

Termination Criteria

Termination criteria determine when the population should stop evolving. Common criteria include reaching a specified number of generations or calls to the evaluation software, often based on available computational resources. Terminating when reaching a specified number of idle generations or when the population diversity drops under a given threshold [13], works to avoid incurring further computational costs when additional improvements become negligible. Finally, termination based on achieving a satisfactory fitness level might be employed when a "good enough" solution is sufficient.

4.4 Structure of an Evolutionary Algorithm (EA)

The structure of an Evolutionary Algorithm follows a series of steps, guiding the optimization process. Within the (μ, λ) EA three populations coexist in any given generation g. The population of λ offspring $S^{g,\lambda}$, the population of μ parents $S^{g,\mu}$ and the population of elite individuals $S^{g,e}$, where the e best solutions are stored. The list below outlines the key steps in the structure of a (μ, λ) EA [17]:

1. **Initialization:** Generate an initial population of offspring $S^{0,\lambda}$ (randomly or using a predefined method) and assess the fitness of each offspring $F(S^{0,\lambda})$, according to the objective function.
2. **Termination Criteria:** Define criteria to determine when the algorithm should stop evolving and repeat steps 3 to 7 until these criteria are met.
3. **Elite Selection:** Renew the elite population $S^{g,e+1}$, with the best canditate solutions from $S^{g,\lambda} \cup S^{g,e}$. If no offspring fitter than the elites is presented, the elite population remains the same.
4. **Elitism:** Replace candidates of $S^{g,\lambda}$ with a number of elites from $S^{g,e}$.
5. **Parent Selection:** Choose individuals from $S^{g,\lambda}$ to serve as the parents for the next generation in $S^{g+1,\mu}$. Selection methods may include probabilistic approaches or rank-based methods.
6. **Recombination and Mutation:** Form the new generation of offspring $S^{g,\lambda}$, by combining genetic material from members of $S^{g+1,\mu}$ and randomly mutating some of the produced offspring.
7. **Evaluation:** Evaluate the fitness of the new offspring population. $F(S^{g+1,\lambda})$

Figure 4.1 illustrates the basic structure of a (μ, λ) EA.

4.5 Metamodel-Assisted Evolutionary Algorithms (MAEA)

Metamodel-Assisted Evolutionary Algorithms (MAEA) represent a subset of Evolutionary Algorithms (EAs) that harness metamodels to speed up optimization processes. Metamodels, being regressors, can approximate the objective function for any candidate solution directly from its genome, without calling the often costly Problem Specific Model (PSM). This addresses the challenge of EAs requiring a large number of software evaluations to initiate improvements, as most of them can be performed by the metamodels.

The metamodels can be trained either offline, using examples of candidate genomes and the corresponding objective function values, or online using candidates from previous generations evaluated by the PSM.

EASY [18] uses online-trained Radial Basis Function (RBF) metamodels. During a MAEA optimization, the first couple generations are all evaluated using the PSM, to form an initial database for the RBF to train on. Then the RBFs approximate the objective function of new candidate solutions and only the most promising are accurately evaluated using the PSM, at the same time enriching the database the

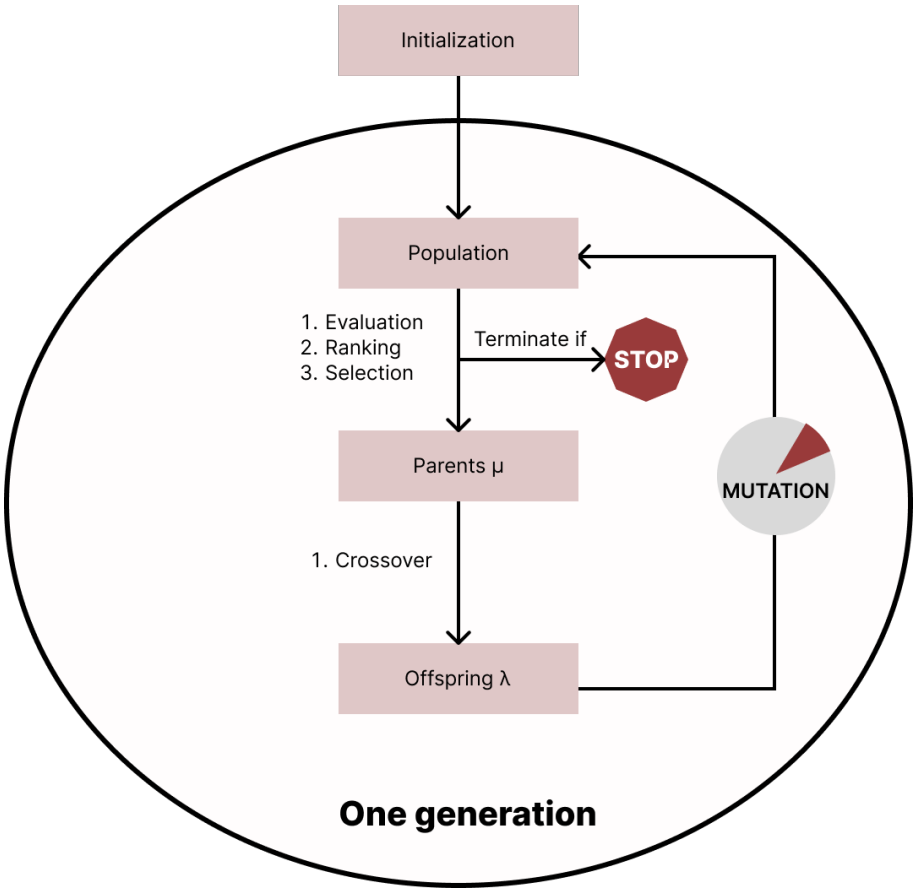


Figure 4.1: *Basic Structure of a (μ, λ) Evolutionary Algorithm.*

RBFs are trained on. As the optimization progresses and the distribution of candidate solutions evolves, the RBF metamodel can adapt by refining its approximation of the objective function within regions of interest.

Chapter 5

The NLF0416 Isolated Airfoil

This chapter revolves around the shape optimization of the NLF0416 isolated airfoil, with the primary objective being to minimize drag. To achieve this, the geometry is parameterized using volumetric Non-Uniform Rational B-Splines (NURBS), and the optimization process is conducted using the EASY software of the PCOpt/NTUA.

The uniqueness of this study lies in the adoption of two different evaluation software:

1. **PUMA-TM** uses PUMA [3], the in-house GPU accelerated flow analysis software, to solve the Reynolds-Averaged Navier-Stokes (RANS) equations using vertex-centered finite volumes, where the turbulent viscosity μ_t is computed by solving numerically the one-equation Spalart-Allmaras (SA) turbulence model [43] and the two-equation Smooth $\gamma - Re_{\theta,t}$ transition model [33].

2. **PUMA-CNN** uses PUMA to solve the RANS equations, but μ_t is computed directly from a CNN using geometrical and flow data as inputs.

Given that the NLF0416 is designed to retain laminar flow as much as possible over its surface and the effect of transition on the performance characteristics, two distinct optimizations are conducted with different objectives each. The first aims to directly minimize drag using it in the objective function. In the second, the distance between the leading edge and the point of transition over the suction side, is used as the objective function. Maximizing this quantity aims to extend the laminar area, anticipating a reduction of drag in the process.

The purpose of using these two objective functions is to assess whether they will converge on the same or similar airfoil shapes.

5.1 The NLF0416 Airfoil Case

The NLF0416 airfoil belongs to the NASA Low-Speed Family of airfoils and was developed as a part of their research into aerodynamic designs. It is applicable in the subsonic range of speeds. The NLF designation stands for "Natural Laminar Flow", revealing its aim of retaining laminar flow as much as possible over the airfoil.

The profile of the NLF0416 can be seen in Figure 5.1. A 705×97 C-type mesh was generated in a radius of more than 1000 chords around the airfoil to accurately model the air body. The PUMA code, however, uses the above structured grid as an unstructured one.

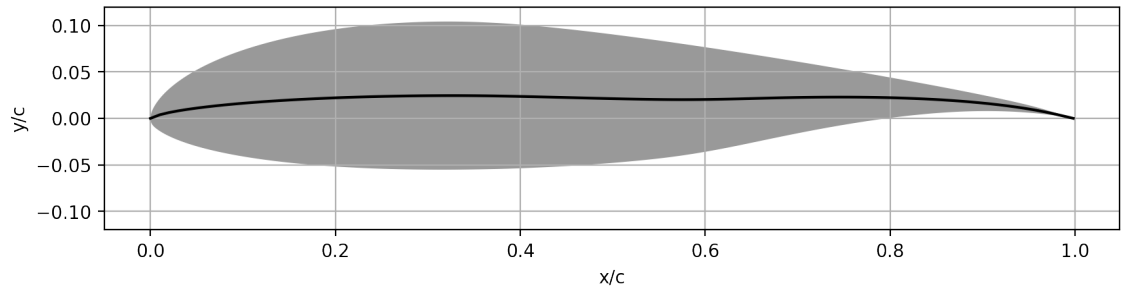


Figure 5.1: *The NLF0416 Airfoil.*

For this case, the airfoil is simulated under the conditions shown in Table 5.1. To study the effect of the angle of attack on the transition from laminar to turbulent flow, the simulations were conducted for $\alpha = 0.02°$ and $\alpha = 2°$. The RANS equations are solved, and turbulence is modeled using the Spalart-Allmaras (SA) model coupled with Piotrowski & Zingg's Smooth $\gamma - Re_{\theta,t}$ transition model.

The resulting aerodynamic coefficients after convergence are presented in Table 5.2. The drag coefficient (C_D) is broken down to its two components, the friction drag coefficient $C_{D,f}$ and the pressure drag coefficient $C_{D,P}$. $C_{D,f}$ accounts for the shear stresses acting on the surface of the airfoil, due to the viscosity of the fluid, while $C_{D,P}$ results from the different static pressures around the blade. Notably, $C_{D,f}$ constitutes almost 80% of the total C_D for both angles, highlighting the significant impact of transition delay on drag reduction.

Table 5.1: *NLF0416 Case Conditions.*

Flow Conditions	
Chord length c	0.609 m
Mach number M_∞	0.1
Reynolds number Re_∞	$3.997 \cdot 10^6$
Turbulence Intensity TI_∞	0.0015

Table 5.2: *Aerodynamic Coefficients for the Baseline Geometry.*

Angle of Attack α_∞	**0.02°**	**2°**
Drag Coefficient C_D	$5.30 \cdot 10^{-3}$	$5.80 \cdot 10^{-3}$
Friction Drag Coefficient $C_{D,f}$	$4.15 \cdot 10^{-3}$	$4.31 \cdot 10^{-3}$
Pressure Drag Coefficient $C_{D,P}$	$1.15 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$
Lift Coefficient C_L	$4.88 \cdot 10^{-1}$	$7.21 \cdot 10^{-1}$
Moment Coefficient C_M	$1.08 \cdot 10^{-1}$	$1.10 \cdot 10^{-1}$

The pressure and friction distributions along the airfoil blade for $\alpha = 0.02°$ and $\alpha = 2°$ are presented in Figures 5.2 and 5.3, respectively.

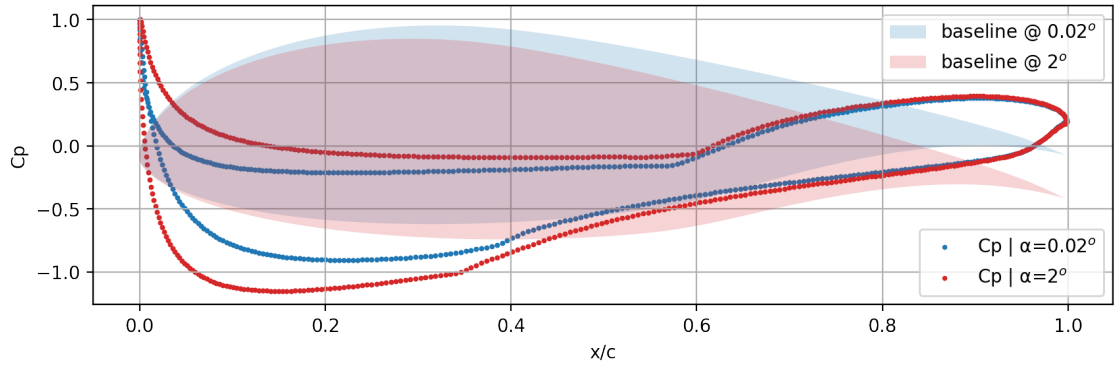


Figure 5.2: *Pressure distribution along the baseline geometry. C_p starts at 1 at the leading edge, where stagnation occurs, and decreases further along.*

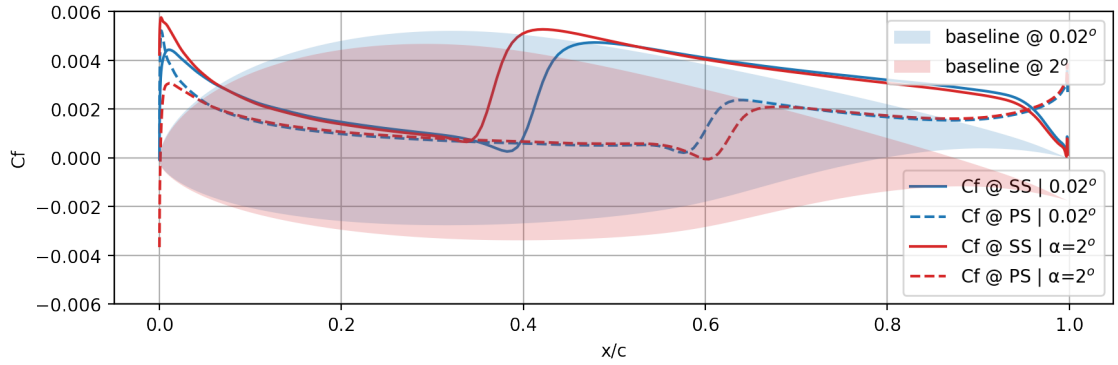


Figure 5.3: *Friction distribution along the baseline geometry. As the angle of attack increases, transition begins earlier on the suction side and later on the pressure side.*

In the C_p graph, throughout the chord length, the C_p on the pressure side remains higher than on the suction side, generating lift. For $\alpha = 2°$ (red), the gap between C_p on the pressure side and C_p on the suction side is larger, resulting in increased lift.

In the C_f graph, sudden jumps indicate the transition from laminar to turbulent flow. For $\alpha = 0.02°$ transition occurs at approximately 40% of the chord on the suction side and at approximately 60% on the pressure side. For $\alpha = 2°$ transition begins earlier on the suction side and later on the pressure side.

The friction coefficient in the turbulent regions differs significantly between the two sides, being nearly twice as high on the suction side, reaching 0.005, compared to 0.0025 on the pressure side post-transition. Thus, delaying transition on the suction side is considered favorable, as not only more room for delay exist, since transition occurs earlier, but a transition delay on the suction side results in a more substantial reduction in total friction than an equal one in the pressure side.

From this point forward, the analysis focuses on the airfoil at an angle of $\alpha = 0.02°$.

5.2 Shape Parameterization

The parameterization of the airfoil shape has a dual significance. Initially, it serves as the introduction of the design variables within stochastic optimization via EASY. Subsequently, the same design variables can form the basis for the creation of a database (DB_{CNN}) used to train the CNN to predict the turbulence field μ_t, replacing the numerical solution of the turbulence and transition equations of the RANS.

To achieve airfoil parameterization, volumetric Non-Uniform Rational B-Splines (NURBs) were used. Specifically, the box depicted in Figure 5.4 is employed. Red points are free-to-move, while blue points are fixed. The free points are allowed a 10% range of displacement along the vertical and an 8% range of displacement along the parallel direction in respect to the chord. Thus, 30 design variables are utilized, two for each of the 15 control points.

To create the training database, the Latin Hypercube Sampling (LHS) is used. Using LHS the design space is explored to generate 80 different geometries. The corresponding flow fields are computed using PUMA to solve the RANS, the SA and the $\gamma - Re_{\theta,t}$ transition model, to form DB_{CNN}. The 80 airfoils, depicted in Figure 5.5, surround the baseline geometry denoted in red.

Having access to DB_{CNN} provides insights into the design space of the optimization algorithm. Figure 5.6 displays the values of the six quantities of interest for all geometries within the database. The values are sorted from small to large, with the corresponding value for the baseline geometry denoted in red, and some of the constraints later imposed on the optimization filled in gray.

As about 20 out of the 80 airfoils have a lower C_D than the baseline, it appears that although the baseline is relatively well-optimized concerning drag, room for improvement still exists. In regards to C_L and C_M, the baseline falls right in the

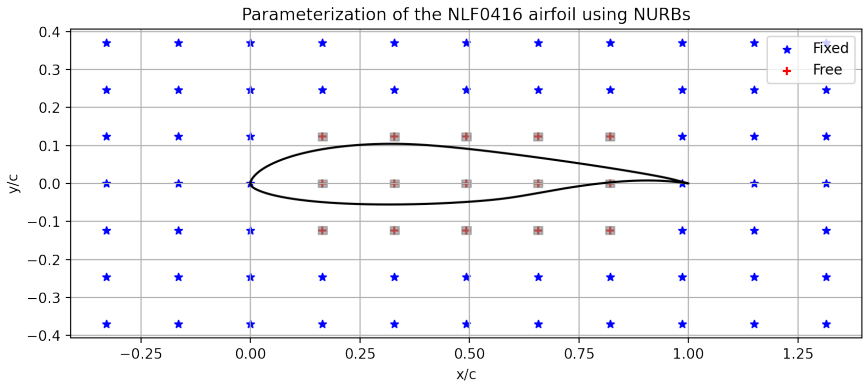


Figure 5.4: *Parameterization of the NLF0416 airfoil. The free-to-move control points are marked in red and are allowed to move ±10% vertically and ±8% horizontally, within the gray boxes.*

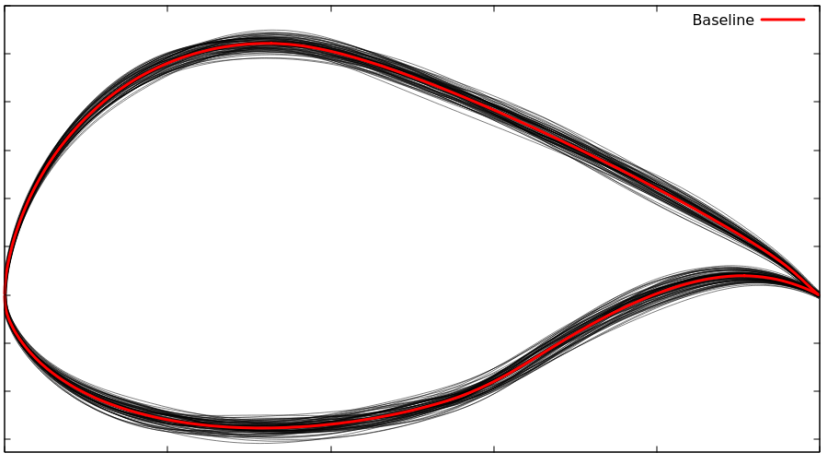


Figure 5.5: *The 80 airfoil profiles from DB_{CNN} (in black) surrounding the baseline geometry (in red). The parameterization can create an airfoil profile roughly within the area in black. This plot is not in scale.*

middle, with approximately 40 blades having higher and 40 blades having lower values. Regarding the transition point on the pressure side, it is evident that prior optimization has taken place, as fewer than 10 of the 80 geometries transition later and even then, only slightly. For the suction side, less than half of the geometries transition later, indicating a potential increase. Regarding the area of the blades, the baseline occupies a central position, as anticipated, given its role as the baseline of the NURBs lattice used to generate the other geometries. Simultaneously, the smaller blade is no more than 5% smaller than the baseline, affirming that the parameterization can only produce blades within the area constraints.

The possible inputs to the CNN include geometrical information, nodal coordinates $x_k = (x, y)$ and wall distances W_d as well as flow data, the density field ρ, the

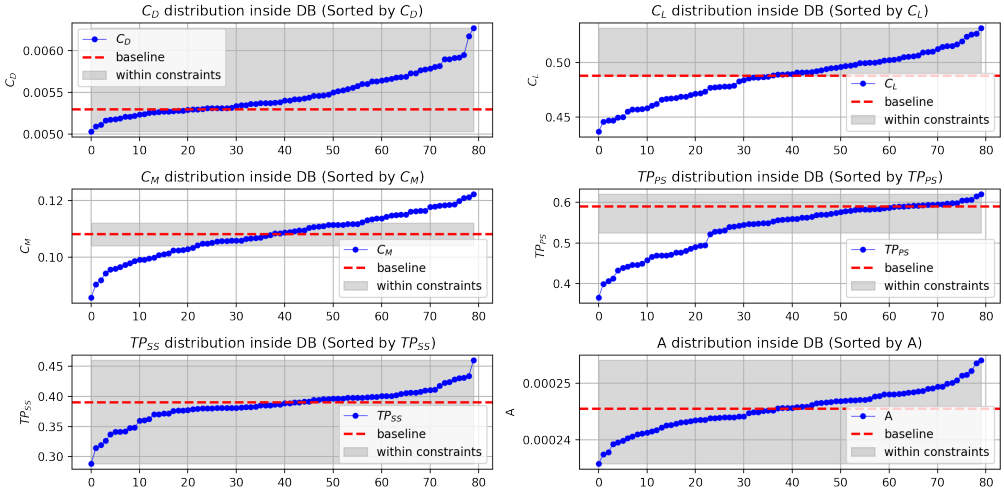


Figure 5.6: *Distribution of quantities of interest in DB_{CNN}. For each plot the database is sorted based on the specific quantity plotted. As a result, there is no direct correspondence between the plots based on the geometry number.*

pressure field P, the velocity vector field $u_k = (u, v)$. The strain rate S and the vorticity magnitude Ω are also included, acknowledging their significance in turbulence modeling. While these last two quantities, related to the gradients of the flow data, can be directed produced from the flow fields, including them aims to provide valuable intel to the CNN, to understand the underlying physics faster. It should be emphasized that the CNN operates on entire fields, using the complete flow and geometrical fields across the entire grid as inputs and producing the μ_t field as output. This approach contrasts with DNNs that operate on an element-wise basis, receiving values for individual nodes and returning the corresponding μ_t value for each node separately.

Table 5.3: *Pool of inputs and output of the CNN.*

Inputs	Output
•Nodal coordinates x_k	•Turbulent Viscosity μ_t
•Density ρ	
•Pressure P	
•Velocity vector u_k	
•Vorticity Ω	
•Wall distance W_d	
•Strain Rate S	

It is a standard practice to scale the input and output quantities before providing them to any artificial network. Given that different input parameters operate at different scales, combining them effectively can prove challenging unless scaling precedes. Scaling also plays a role in safeguarding against issues like vanishing or exploding gradients during the training phase.

Min-Max scaling (Equation 5.1) was chosen to scale the inputs and the output, as it is simple, efficient and preserves the original relative relationships, unlike standardization which distorts the inputs. This scaling method linearly adjusts data to a range of 0 to 1, in contrast to standardization, which transforms data into a normal distribution centered around 0 with a standard deviation of 1.

$$x = \frac{X - Q_{min}}{Q_{max} - Q_{min}} \tag{5.1}$$

Here, Q_{min} and Q_{max} represent the minimum and maximum values of the input fields Q present on DB_{CNN}.

5.3 CNN configuration

The CNN was developed using python's TensorFlow [1], a tensor manipulation framework with a focus in Deep Learning. The CNN will be trained to predict the μ_t field, based on the available from the solution of the RANS flow fields and the geometrical data.

Due to its numerous parameters the number of distinct CNN architectures is practically limitless, therefore, to navigate the realm of possible architectures, certain assumptions were made:

1. The number of channels at any hidden layer should be a power of 2.
2. The number of channels at a given hidden layer is either double (during expansion) or half (during compression) that of the previous layer.
3. All convolutions are performed using a uniform kernel size.
4. All layers follow the same principal format (Conv-ReLU-BatchNorm)

5.3.1 Preprocessing data from PUMA

As previously mentioned, the mesh utilized in this study adopts a C-type structured configuration, comprising quadrilateral elements. A structured mesh is *logically rectangular* [47], exhibiting a grid-like pattern and can be conveniently represented through a (i, j) annotation (or (i, j, k) in three dimensions). Although PUMA treats the mesh as unstructured, the structured representation is required for the CNN to operate, as it can then consider the mesh as an image, where each node corresponds to a pixel. The mesh is formed by 97 iso-η lines along the stream-wise direction and 705 iso-ξ lines along the span-wise direction. Therefore the input tensor for the CNN takes the form of a (97, 705, c) tensor, where c is the number of utilized flow fields.

Figure 5.7 illustrates an example C-type mesh surrounding the NLF0416 airfoil and the equivalent rectangular mesh in the $\xi - \eta$ coordinate system. Green highlights the split-line, blue represents the outflow, gray corresponds to the airfoil, and red signifies the far-boundary. Iso-ξ lines are dashed, and iso-η lines are continuous.

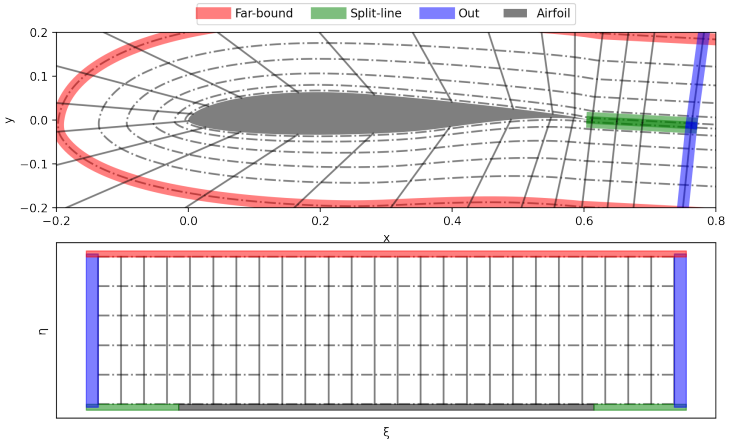


Figure 5.7: *C-type mesh around the NLF0416 airfoil in the physical $(x - y)$ and the computational $(\xi - \eta)$ system. Green: split-line, Blue: outflow, Gray: airfoil, Red: far-boundary. Dashed lines: Iso-ξ, Continuous lines: Iso-η.*

This transformation converts every node in the physical $x - y$ coordinate system to another in the computational $\xi - \eta$ system and is applicable to any flow field to create the equivalent rectangular representation. Figure 5.8 displays the pressure field in both domains. In the $\xi - \eta$ system, moving from left to right, increased pressure occurs at the trailing edge of the pressure side, with a slightly decreased pressure following on the rest of the pressure side. The maximum pressure occurs at the leading edge, while the minimum pressure is observed over the entire suction side, followed by pressure recovery afterward.

The same transformation is applied to the μ_t field as seen in Figure 5.9.

Transforming all nine potential input fields and the output field, as illustrated in Figure 5.10, clarifies the CNN's objective: to predict the turbulent viscosity across the entire mesh at once, leveraging the nine available flow and geometrical fields.

5.3.2 Kernel size comparison

This analysis investigates the impact of varied kernel sizes—1, 3, 5, and 7—within two consistent CNN architectures of different depth. Figure 5.11 illustrates one of them, reaching a depth of 128 channels on the fourth hidden layer. The core elements of the CNN (activation functions, batch normalization etc.) remain uniform, while the training method remains consistent, to ensure a fair comparison.

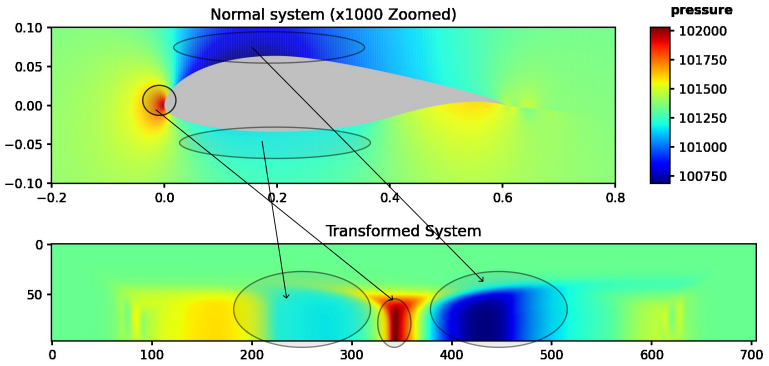


Figure 5.8: *Pressure field transformation between physical and computational domains.*

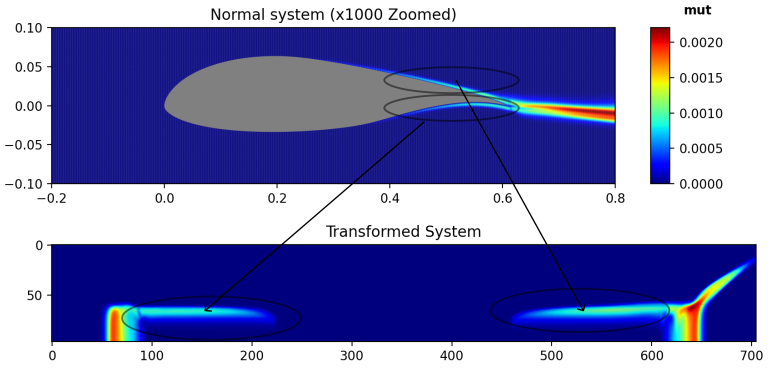


Figure 5.9: *Turbulent viscosity field transformation between physical and computational domains.*

Figure 5.12 illustrates how kernel sizes of 1, 3 and 5 operate on a matrix. A kernel with a size of 1 considers only the corresponding input value, not allowing inter-pixel communication. Kernels of size 3 and 5 incorporate neighboring input values in a square region around the reference element. For a 3×3 kernel, the central element and its 8 nearest neighbors contribute to the output, while a 5×5 kernel involves the central element and its 24 nearest neighboring elements.

The use of a 1×1 kernel presents a unique case, as it does not allow for inter-pixel communication, essentially functioning like a DNN being applied to each mesh element separately. This equivalent DNN, containing nodes instead of channels and receiving/providing values instead of fields, with the same architecture is also tested.

Figure 5.13 illustrates the training progress for the different model configurations. The increase in kernel size from 1 to 3 exhibits notable reductions in both training

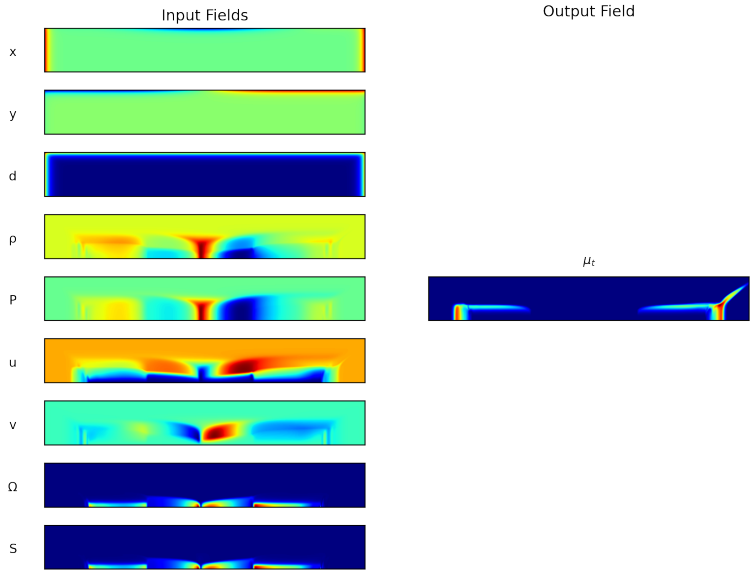


Figure 5.10: *Input and output fields for the CNN.*

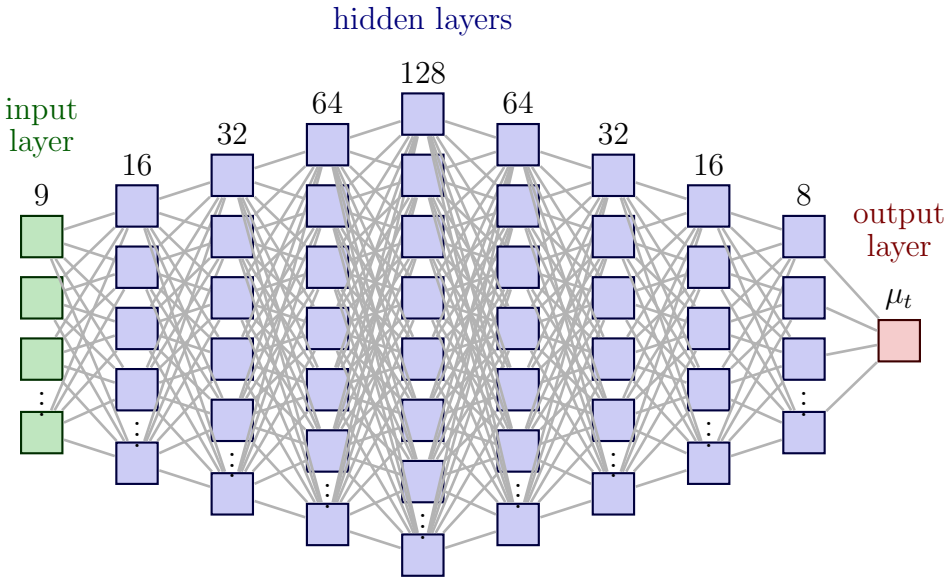


Figure 5.11: *One of the standardized architectures used for all 4 configurations. Studying the influence of a certain parameter in isolation from all others is crucial in research and development.*

and validation losses. However, further increases to 5 and 7 yield marginal improvements in training loss, but noisier validation losses. Additionally, the deeper network model demonstrates slightly smaller losses overall.

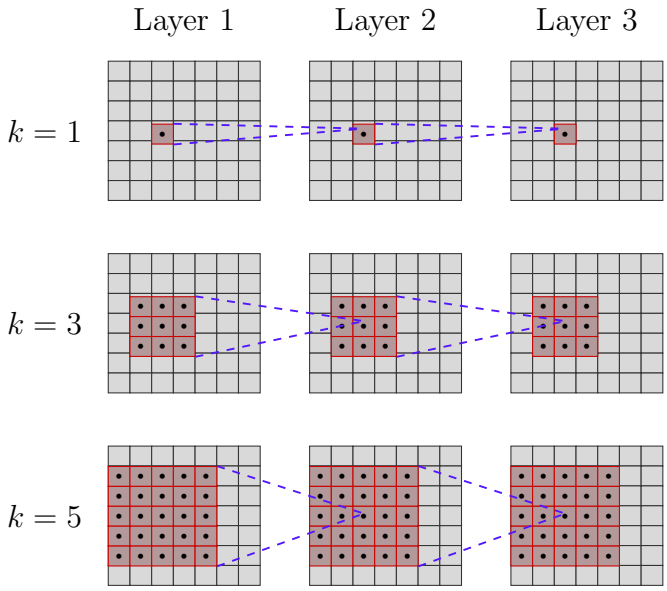


Figure 5.12: *Kernel sizes of 1, 3 and 5 operating on a matrix. Each element in a layer is computed by applying the kernel to the corresponding region around it, (red area) in the previous layer.*

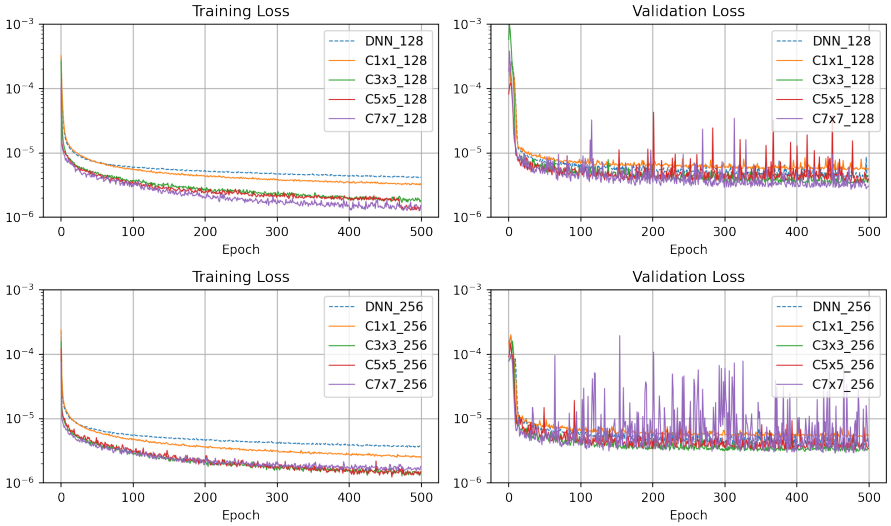


Figure 5.13: *Training losses across kernel sizes. Increases from 1 to 3 kernels show notable decreases in losses, but further increases to 5 and 7 exhibit minor improvements.*

Comparing the DNN and the CNN using 1×1 kernels, it is evident that they follow similar but not identical loss curves. Given their shared architecture, the differences observed can be attributed to variations in initialization, batch partitioning, and other related training nuances.

Figure 5.14 presents the relationship between free weights and training times across various configurations. The number of parameters grows quadratically concerning kernel size, while training time exhibits a more exponential increase.

Notably, the CNN with a 1×1 kernel and the DNN exhibit different training times, with the former being trained approximately 30% faster. This can only be attributed to inherent differences in computational efficiency, including diverse performance characteristics in matrix multiplication and GPU utilization, between the DNN and the CNN architecture.

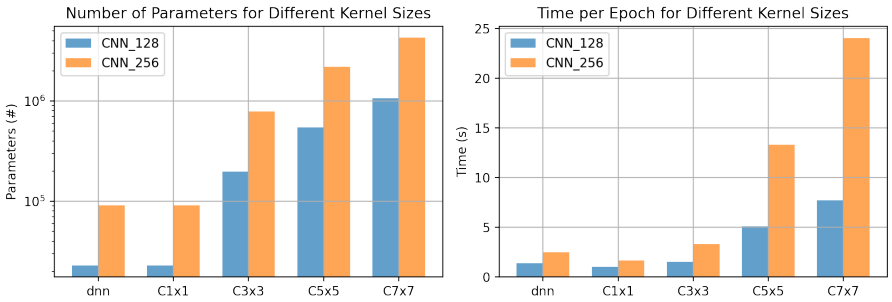


Figure 5.14: *Number of free weights and training times. Number of free weights grows quadratically with kernel size, while training time exhibits an exponential increase. The CNN with a 1×1 kernel requires 30% less training time compared to the equipotent DNN.*

Table 5.4 provides detailed metrics, showcasing the resulting losses, the free weights, and the training times for the CNN models utilizing different kernel sizes.

Table 5.4: *Losses, number of free weights and training times for CNNs using different kernel sizes.*

Model	**CNN_{128}**			**CNN_{256}**		
Kernel Size	**Losses**	**Free Weights**	**Time per epoch**	**Losses**	**Free Weights**	**Time per epoch**
1	3.18/5.47	23k	0.98	2.49/5.22	91k	1.62
3	1.70/3.32	198k	1.48	1.29/3.06	789k	3.30
5	1.27/3.33	546k	5.10	1.24/3.31	2.2m	13.3
7	1.27/2.71	1,07m	7.69	1.55/2.91	4.3m	24.0
Model	**DNN_{128}**			**DNN_{256}**		
	4.13/4.10	23k	1.34	3.58/3.47	91k	2.45
	Corresponds to Kernel Size 1 in CNN					

5.3.3 Final Architecture

Based on the insights from subsection 5.3.2 the kernel size of the CNN is 3. This combined with the assumptions made earlier, narrowed down the possible architectures. Through trial and error the final architecture was derived, as described in Table 5.5.

Table 5.5: *Training and Architecture Parameters.*

Architecture	
Inputs	$x, y, \rho, u, v, \Omega, S, W_d$
Hidden layers	12
Channels per layer	16 - 32 - 64 - 128 - 256 - 512 - 256 - 128 - 64 - 32 - 16 - 8
Kernels	3×3
Activation, Padding, BatchNorm	ReLU, Zero, Yes
Training Parameters	
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	1e-4
Batch Size	Dynamic (1 to 5)
Validation Split	10%

The CNN was trained using the 80 geometries in DB_{CNN} in three steps, incorporating progressively larger batch sizes. As suggested by recent studies [42], an increased batch size can have a similar effect to lower learning rates, potentially aiding in the model's generalization capabilities.

In Figure 5.15, the progression of the training loss across the three training steps is visually depicted. The first 500 epochs utilized a batch size of 1, followed by the next 500 epochs with a batch size of 3, and the final 300 epochs with a batch size of 5. Larger batch sizes were unattainable due to memory limitations. The observation reveals that after increasing the batch size, the gap between validation and training losses decreased. The transition from a batch size of 3 to 5 did not bring about dramatic changes, but the curves became less noisy.

Evaluating CNN performance traditionally involves comparing its output with actual values over a test set. However, in this scenario, accurately predicting only the μ_t field isn't sufficient. With the CNN integrating with PUMA for shape optimization, ensuring the generalization performance of the entire system (PUMA-CNN), that solves the RANS using PUMA and produces μ_t using the trained CNN, becomes crucial.

Evaluating the performance on the baseline geometry is critical, as it not only represents the midpoint of the Latin Hypercube Sampling, but also serves in the initialization of the optimization algorithm. PUMA-CNN predicted a C_D 5.4% lower

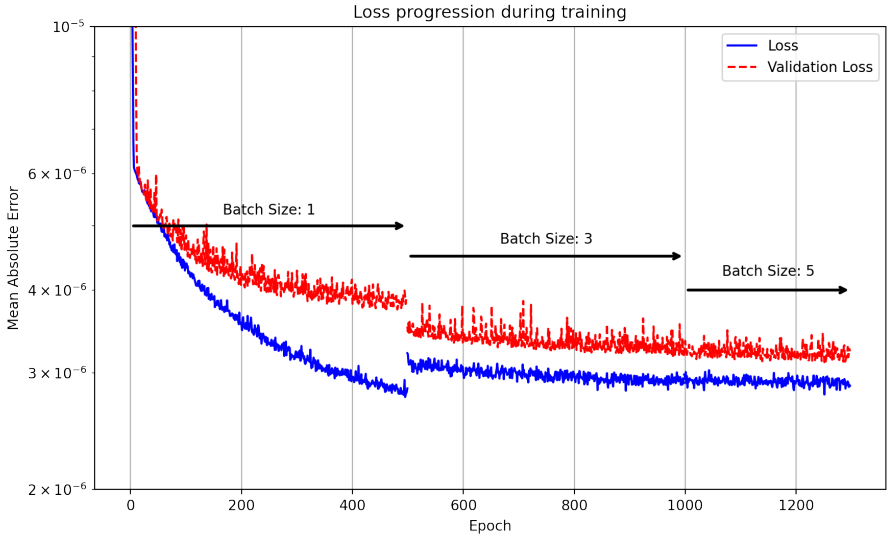


Figure 5.15: *Progression of training loss across three training steps with increasing batch sizes. The training was performed on a GeForce RTX 2070 GPU with the total cost amounting to approximately 3.5h.*

than the actual and C_L and C_M 1.4% and 1.1% higher. The actual coefficients and the errors can be seen on Table 5.6.

Table 5.6: *Comparison of Drag, Lift, and Moment coefficients: PUMA-CNN vs PUMA-TM.*

Metric	PUMA-CNN	PUMA-TM	Relative Error
Drag	0.00503	0.00532	-5.4%
Lift	0.49537	0.48821	$+1.4\%$
Moment	0.10947	0.10833	$+1.1\%$

Figure 5.16 illustrates the μ_t field surrounding the baseline airfoil, as computed by both PUMA-TM and PUMA-CNN, along with the errors between the two evaluations. No errors are apparent on the suction side, while some error appears on the pressure side. The most significant differences accumulate in the wake region behind the airfoil.

Recognizing that performance on a single geometry might not suffice, a set of 9 additional geometries was generated similarly to how DB_{CNN} was formed. For these geometries, the prediction errors in C_D and the discrepancies between produced and actual C_f curves were assessed. In Figure 5.17 the actual and predicted values of drag are compared for the 9 geometries. PUMA-CNN consistently underestimates drag by an average of 5.5%.

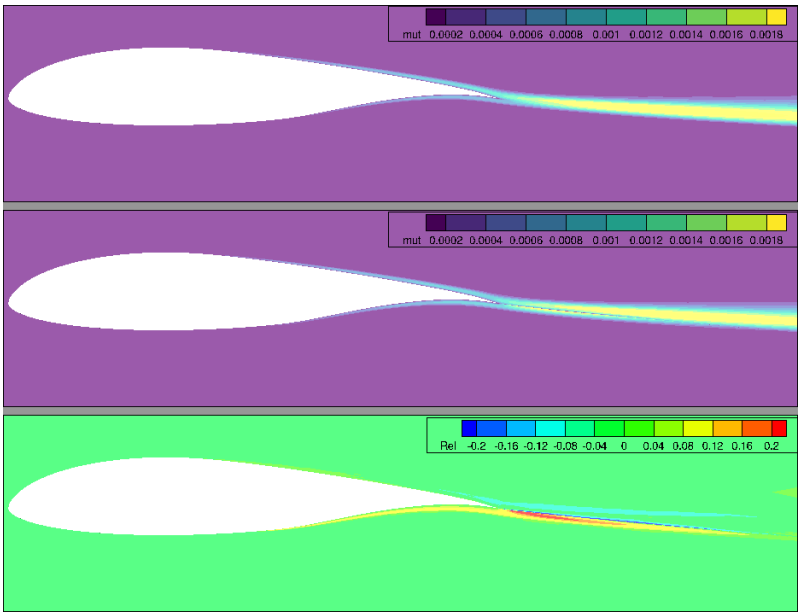


Figure 5.16: *Comparison of μ_t field between PUMA-TM (top) and PUMA-CNN (middle), highlighting the differences (bottom).*

In evolutionary optimization, the ranking of solutions is more important than the actual values, thus showing how well PUMA-CNN ranks the potential solutions is equally or more important. Figure 5.18 presents the ranking comparison between PUMA-TM and PUMA-CNN. Figures 5.19 and 5.20 depict the C_f distributions produced by PUMA-TM (in red) and PUMA-CNN (in blue).

For the suction side (Figure 5.19), the point of transition is accurately predicted in all but one geometry. Before transition, in the laminar area, the error between predicted and actual C_f is minimal, while after transition PUMA-CNN struggles more as it produces an oscillating curve, a token of the more complex turbulent physics.

For the pressure side (Figure 5.20), although transition is predicted earlier using the CNN, both in the laminar and in the turbulent area the C_f values are accurately predicted by the CNN.

5.4 Shape Optimization

The shape optimization utilized MAEA-based optimization via EASY. Two optimizations with distinct objectives were conducted: one aimed at minimizing drag, while the other focused on maximizing the laminar area, by maximizing the distance between the leading edge and the transition point on the suction side (TP_{SS}). The latter served as a surrogate objective for the evolutionary algorithm, acting as a proxy for the primary goal of reducing drag.

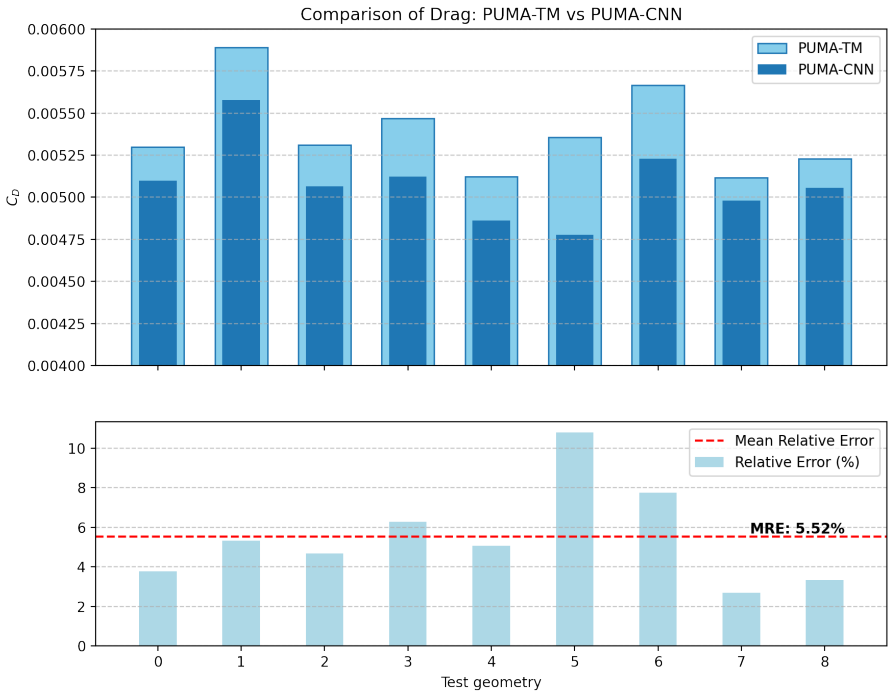


Figure 5.17: *PUMA-CNN underestimates the drag for all test geometries. The Mean Relative Error is 5.52% over the 9 geometries.*

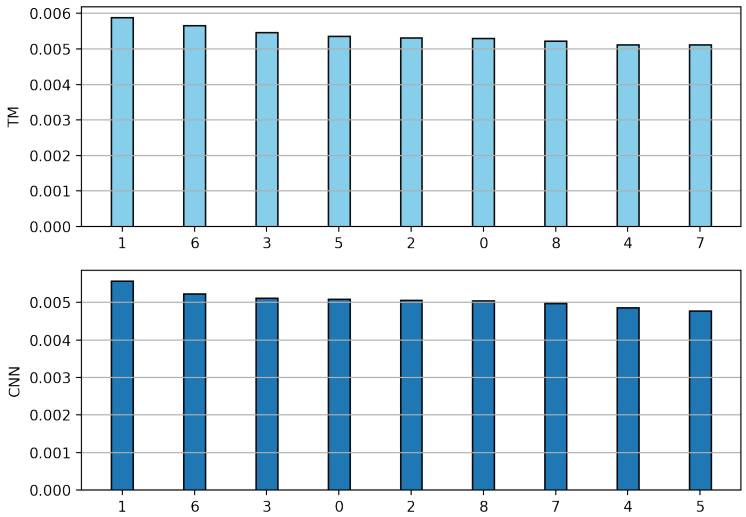


Figure 5.18: *Comparison of solution rankings between PUMA-TM and PUMA-CNN.*

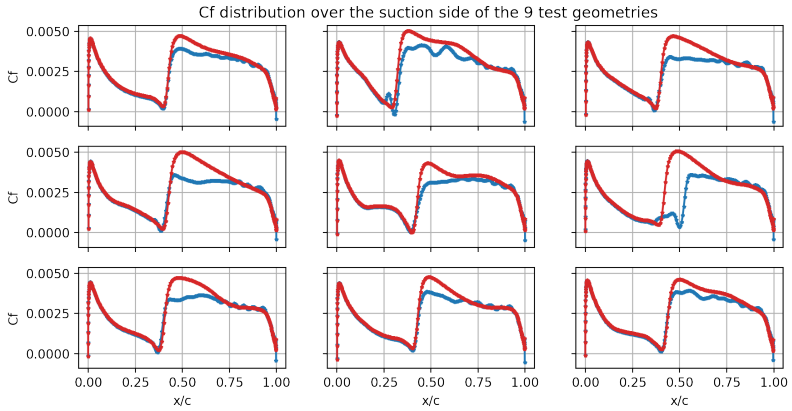


Figure 5.19: *Comparison of C_f over the suction side for the 9 test geometries. PUMA-CNN in blue vs PUMA-TM in red.*

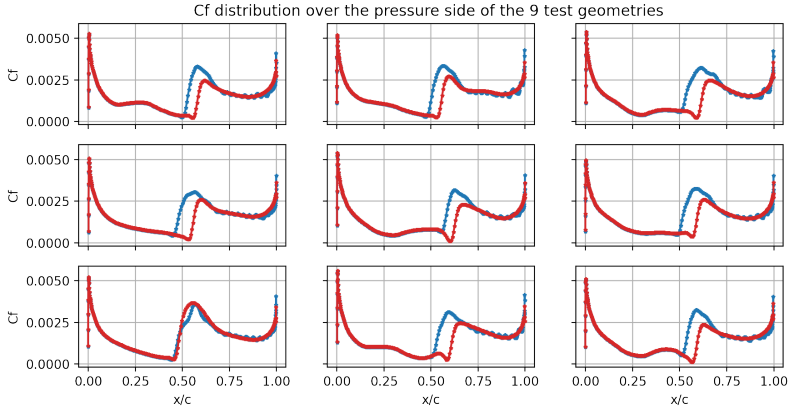


Figure 5.20: *Comparison of C_f over the pressure side for the 9 test geometries. PUMA-CNN in blue vs PUMA-TM in red.*

Each of the two optimizations will be performed using, two different evaluation software, the original PUMA-TM and the AI-driven PUMA-CNN, which has a lower computational cost. So, in total, the 4 optimization schemes of Table 5.7 are conducted.

Table 5.7: *The 4 optimization schemes.*

Objective	**Using TM**	**Using CNN**
Min. Drag	PUMA-TM-DRAG	PUMA-CNN-DRAG
Max. Lam. Area	PUMA-TM-TRANS	PUMA-CNN-TRANS

The same settings are imposed on EASY for all 4 schemes as described in Table 5.8. As for the constraints, retaining approximately the moment and at least the lift of the reference airfoil was imposed; next to them, an additional inequality constraint

was to not reduce the area of the airfoil below 90%.

As presented in Section 5.2, in the 80 patterns the CNN was trained on, approximately 25% of them displayed a lower C_D value than the baseline. However, when considering all the imposed constraints (moment, lift, and area), only 4 of the training patterns (5% of the total) actually outperformed the baseline, and these improvements were marginal, with C_D values no more than 2% lower than the reference. Therefore, employing an EA to optimize the airfoil becomes necessary to achieve substantial improvement.

Table 5.8: *Settings and Constraints for the evolutionary algorithm.*

Settings	
Coding	Real
Parents	10
Offspring	24
Number of Elites	5
Principal Component Analysis (PCA)	Yes
Meta-Model Assistance — Inexact Pre-Evaluation	
Type	RBF_IFs
Minimum DB entries to start	50
Minimum not failed DB entries	30
Patterns for training	$35 - 55$
Exact evals. per generation	$2 - 3$
Extrapolate prediction	No
Non-dimensionalize prediction	Yes
Idle generations for IPE pause	5
Constraints	
$C_M \geq C_{M,\text{base}} - 0.04$, $C_M \leq C_{M,\text{base}} + 0.04$	
$C_L \geq C_{L,\text{base}}$	
$A \geq 0.9 \cdot A_{\text{base}}$	
TP-Specific Constraints	
$TP_{PS} \geq 0.32m$	

Objective Function	
max. TP_{SS}	to maximize the laminar area
min. C_D	to minimize drag

5.4.1 Optimization with the PUMA-TM software

The PUMA-TM software handles the complete set of RANS equations: the 4 mean flow equations, 1 turbulence equation, and 2 transition model equations. Each evaluation using PUMA-TM is referred to as a Time Unit (TU). Both optimization

schemes, PUMA-TM-DRAG, and PUMA-TM-TRANS terminate after reaching 500 TUs.

Figure 5.21 illustrates the evolution of various quantities across the two optimization schemes. Blue indicates optimization aimed at delaying transition, while red indicates optimization focusing on drag reduction. All data points respect the set of constraints. Both optimizations achieve a substantial reduction in drag, with PUMA-TM-DRAG exhibiting a slight advantage. Interestingly, despite no incentive for PUMA-TM-DRAG to increase TP_{SS}, it is increased by the same amount as the transition-focused PUMA-TM-TRANS.

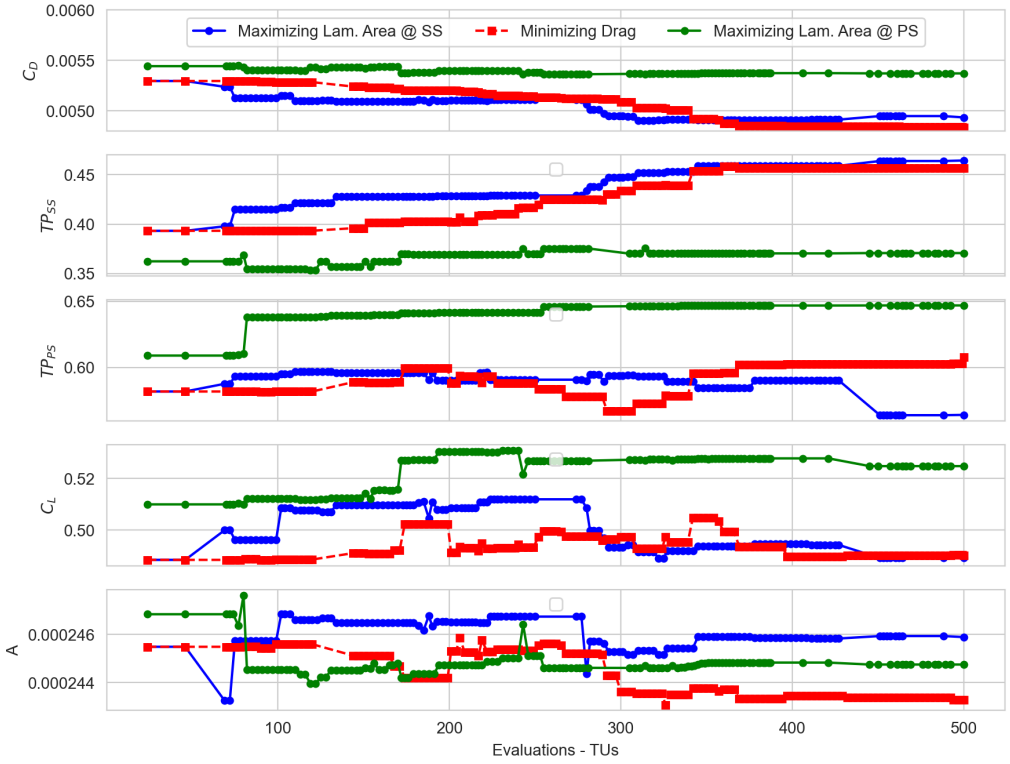


Figure 5.21: *Evolution of relevant metrics throughout the optimization using PUMA-TM. Drag is reduced and transition is delayed on the suction side. Lift and area respect the set of constraints. An optimization aimed at delaying transition on the pressure side (green) shows decreased performance.*

In the same plot, an optimization aimed at delaying transition on the pressure side, instead of the suction side, is also included in green. This optimization achieved no decrease in C_D despite increasing the laminar area on the pressure side by approximately 5% of the chord. This is because the transition point on the suction side moved closer to the leading edge, so any decrease in $C_{D,f}$ on the pressure side was offset by an increase in $C_{D,f}$ on the suction side.

Elitism within the evolutionary algorithm enforces a monotonic evolution of the goal quantity. In the PUMA-TM-DRAG scheme, a consistent decrease in C_D is

anticipated, while the PUMA-TM-TRANS scheme is expected to exhibit a constant increase in TP_{SS}. Despite these expectations, both C_D and TP_{SS} evolve almost monotonically for both schemes. Reductions in drag appear to be accompanied by delayed transition.

Exploring these behaviors involved generating normalized plots for C_D and TP_{SS}, where both quantities decreases, as $-TP_{SS}$ is plotted. Figure 5.22 illustrates the hand to hand evolution of the two quantities.

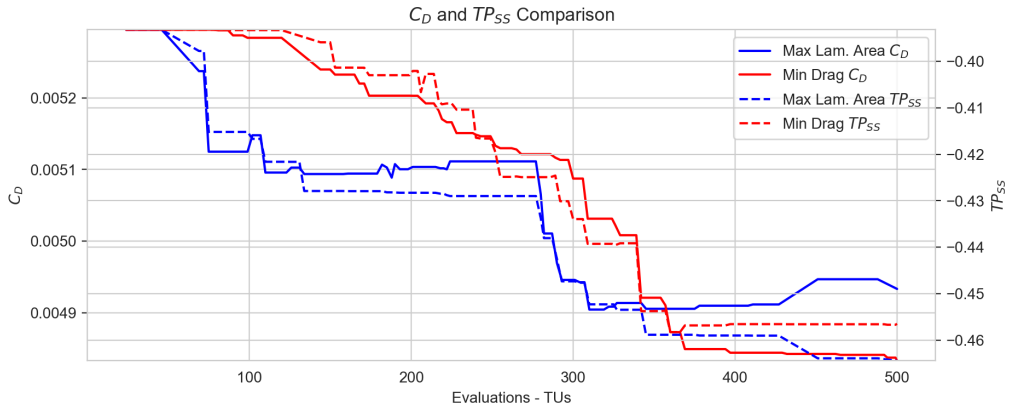


Figure 5.22: *Progression of C_D and TP_{SS}.*

The scatter plot of C_D against TP_{SS}, as depicted in Figure 5.23, was created, and linear regression was performed. The results highlight a linear relationship ($R^2 > 0.9$) between the two quantities for both optimization schemes. This encourages the idea that TP_{SS} can indeed work as a proxy to reduce drag.

Figure 5.24 shows the C_f distribution for the best solutions compared to the baseline. Both optimized geometries experience delayed transition on the suction side. However, on the pressure side, only PUMA-TM-DRAG achieves a later transition. It's worth reminding that the transition point on the pressure side holds less influence on drag, as explained earlier.

Figure 5.25 displays the profiles of the three airfoils. PUMA-TM-TRANS barely alters the pressure side, retaining it nearly identical to the baseline, aligning with its objective to delay transition solely on the suction side. Conversely, PUMA-TM-DRAG appears to refine the airfoil by thinning the pressure side. Both schemes expand the suction side, particularly in the middle section.

5.4.2 Optimization with the PUMA-CNN software

The purpose of implementing PUMA-CNN is to reduce the computational cost. Unlike PUMA-TM, which requires solving three additional differential equations (one for turbulence and two for transition modeling), PUMA-CNN utilizes a convolutional neural network (CNN) to calculate turbulence μ_t.

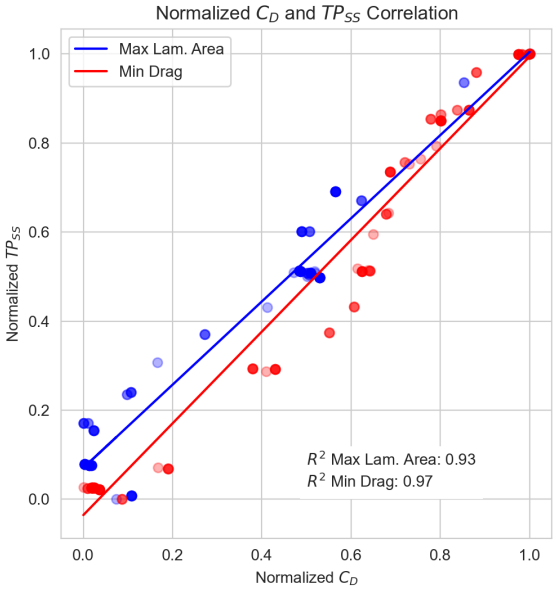


Figure 5.23: *Correlation between C_D and TP_{SS} values for all generations. A correlation coefficient above 0.9 proves a linear relationship between the two variables across both schemes.*

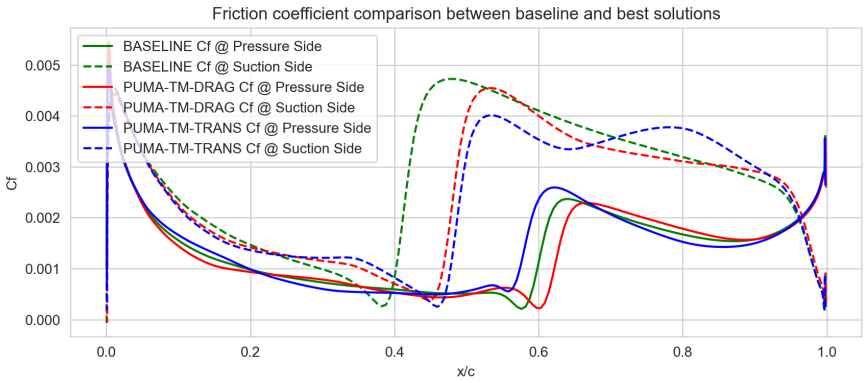


Figure 5.24: *Friction coefficient along the baseline and the optimized blades. Transition is delayed significantly in the suction side of both optimized blades. The pressure side of the blade produced by PUMA-TM-TRANS, transitions earlier than the baseline and the blade produced by PUMA-TM-DRAG.*

PUMA-CNN demonstrates computational efficiency by completing an evaluation in approximately 5.5 minutes, which is only 63% of the required by PUMA-TM 9 minutes. This aligns with the expected improvement, given the reduction in the number of differential equations solved, from 7 to 4 ($4/7 = 57\%$).

Certainly, this efficiency gain comes with costs. The CNN underwent training on a dataset consisting of 80 airfoils and was further validated with an additional 10, all solved by PUMA-TM. The training of the network took a total of 3 hours and

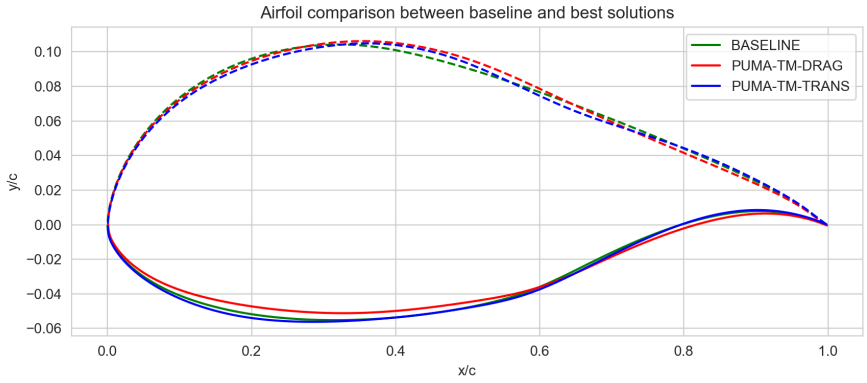


Figure 5.25: *The profile of the optimized blades in comparison to the baseline. Both optimized blades have a "bloated" suction side. PUMA-TM-DRAG thinned the pressure side, while PUMA-TM-TRANS barely changed it at all. This plot is not in scale.*

18 minutes, equating to 22 PUMA-TM evaluations. Therefore any optimization using PUMA-CNN starts with a capital cost of 112 TUs. Table 5.9 summarizes the computational costs of the two softwares.

Table 5.9: *Computational costs for PUMA-TM and PUMA-CNN.*

Software	Cost for DB	Cost to train	Cost per run
PUMA-TM	0	0	1 TU = 8m 55s
PUMA-CNN	90TUs	3h 18m	5m 36s
CNN/CFD	90TUs	22TUs	0.63TUs

Maximizing Laminar Area

PUMA-CNN-TRANS is called to delay transition on the suction side, maximizing the laminar area. The scheme ran for 500 evaluations using PUMA-CNN, reaching a cost of 427TUs, including the training cost settled at 112 TUs.

The optimization progress is presented in Figure 5.26, in comparison with the equivalent optimization using PUMA-TM. During the optimization, four solutions of the AI driven PUMA-CNN were reevaluated using PUMA-TM, to guarantee that no large errors appear. The drag was underestimated by approximately 5% throughout the evolution, as anticipated based on validation results. TP_{SS} was predicted accurately for all but the final solution, where PUMA-TM unveiled an even later transition.

The optimized airfoil created by PUMA-CNN outperformed that of PUMA-TM and was reached 15% faster, utilizing 75 fewer TUs, over the course of the optimization.

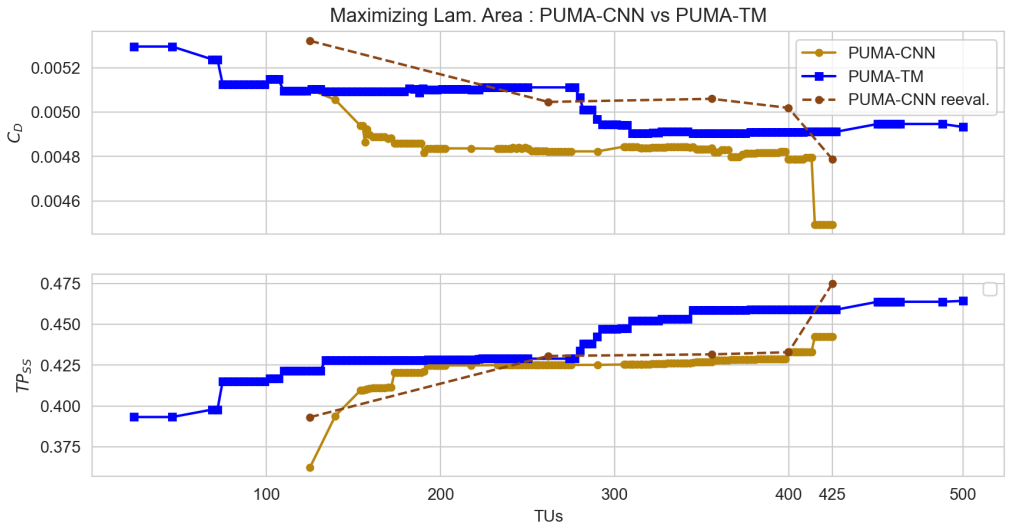


Figure 5.26: *Optimization progress for maximizing laminar area using PUMA-CNN-TRANS compared to PUMA-TM-TRANS. PUMA-CNN-TRANS achieved later transition (goal) and lower drag coefficient. All data points respect the set of constraints.*

Minimizing Drag

PUMA-CNN-DRAG in employed to minimize C_D, in contrast to PUMA-CNN-TRANS that was used to maximize TP_{SS}. The scheme run for 500 evaluations using PUMA-CNN, to cost a total of 427TUs.

The optimization progress (if no retraining was planned) is presented in Figure 5.27. Unfortunately, the prediction errors of PUMA-CNN grew significantly, reaching 10%. Despite that, thanks to the regular re-evaluations of the best-so-far solutions, the increase in errors was detected as early as 6 generations in, at approximately 170 TUs (including training costs). Therefore, the optimization was halted, and retraining was performed. An optimization branch where no retraining is performed is also presented, to highlight the profit gained from halting, retraining and restarting the optimization. This branch, as presented in Figure 5.27 reaches an optimized airfoil with a C_D of 0.00502, 5.2% lower than the baseline.

The objective of the retraining procedure starting at 170 TUs is to enhance the performance of PUMA-CNN over the five best candidate geometries.The original CNN tested over these five best canditate geometries (at 170 TUs) exhibited a Mean Absolute Relative Error of 10.9%.

To assess whether it was just a matter of insufficient training on the original dataset, the CNN was initially retrained for an additional 300 epochs using the original training method over DB_{CNN} without including the five new geometries, something that did not bring any improvement. Afterwards, an alternative approach was tested, where the CNN was retrained on the original dataset for 300 epochs, but using MAPE as the loss function over MAE. This approach, led to a decrease in the av-

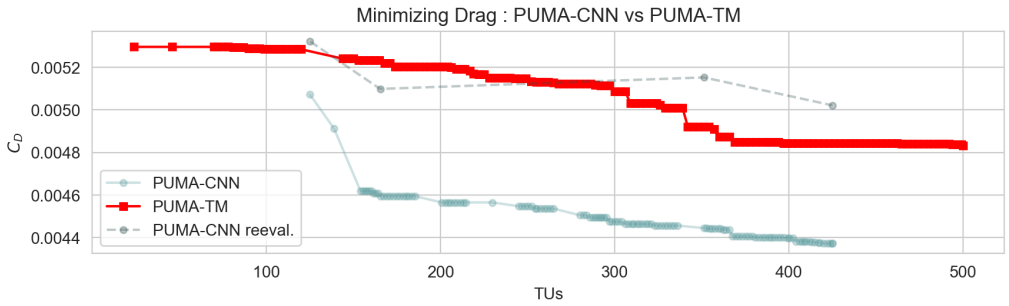


Figure 5.27: *Optimization progress for minimizing drag using PUMA-CNN-DRAG compared to PUMA-TM-DRAG. All data points respect the set of constraints.*

erage error over five geometries to 3.7%, despite not conducting actual training on them.

Following this, a second retraining was performed, this time including the five new geometries, using MAPE as the loss function. Unlike many approaches that rely on database enrichment for retraining, here the network was trained solely on the new geometries, to limit computational cost. To ensure that the network does not deviate significantly from the weights that initially showed good performance over the original database, a very low training learning rate (1e-7), compared to the original (1e-4) was used. 1500 epochs were performed and the error over the new geometries was reduced to 2.1%.

The error for each geometry, for the original and the models produced after the first and second stages of the retraining are presented in Figure 5.28.

The total cost of retraining amounts to just less than 10TUs. The cost of generating the five geometries, was not considered, as they were produced during the optimization, making the solved flow fields readily available.

In Figure 5.29, the optimization progress, with and without retraining is presented, along with the PUMA-TM-DRAG scheme. The results highlight the improvement achieved through the retraining process, as the retrained branch finds a solution on par with the one PUMA-TM reached.

5.4.3 Comparison of the 4 optimization schemes

This section goes into a detailed analysis of the performance exhibited by the baseline and the five optimized airfoils. The key metrics include the two objectives, C_D and TP_{SS}.

In Figure 5.30, the C_D and TP_{SS} values for each optimized airfoil are presented in comparison to the baseline results. Reductions in drag, ranging from 5.2% to 9.6%, and delays in transition by 10% to 21%, are observed. PUMA-CNN-TRANS

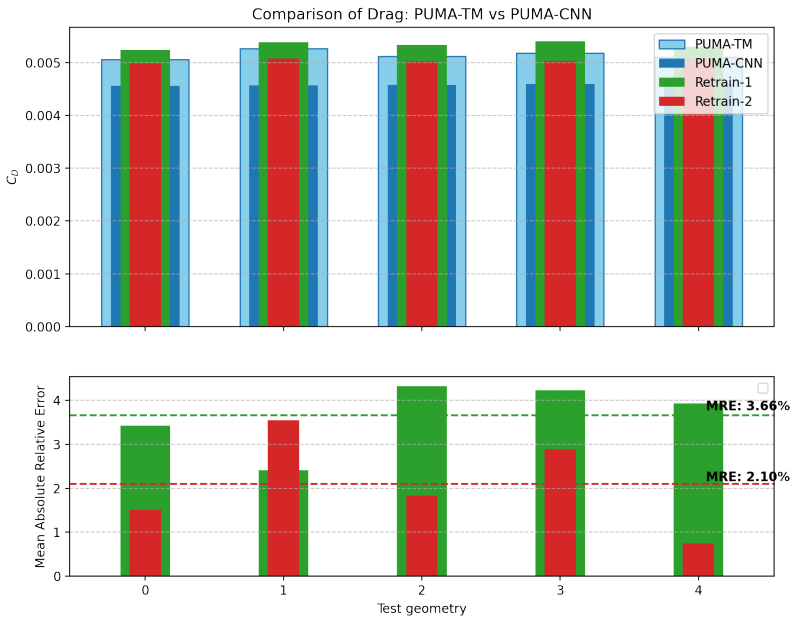


Figure 5.28: *Error comparison for each geometry during the retraining stages.*

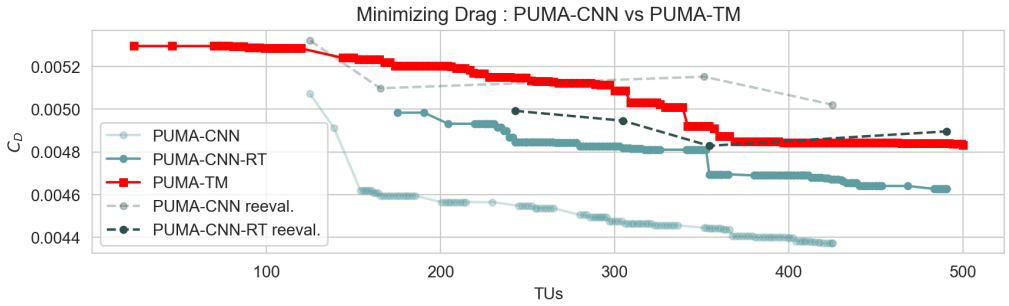


Figure 5.29: *Optimization progress for minimizing drag using PUMA-CNN-DRAG-Retrained compared PUMA-CNN-DRAG and PUMA-TM-DRAG.*

emerges as the top-performing scheme, both in minimizing drag and maximizing laminar area. The comparison is more than fair, as PUMA-CNN-TRANS required 427 TUs to finish the optimization, in comparison to 490 TUs for PUMA-CNN-DRAG and 500 TUs for the PUMA-TM optimizations.

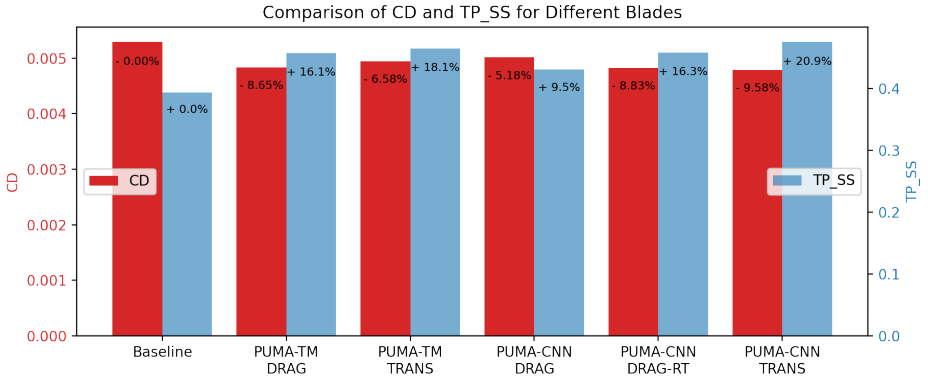


Figure 5.30: *Comparison of C_D and TP_{SS} for each optimized airfoil relative to the baseline. Significant reductions in drag and delays in transition are evident. PUMA-CNN-TRANS demonstrates superior performance.*

In Figure 5.31 the drag components are deconstructed in bar charts. All optimization schemes successfully reduced both friction and pressure drag. Friction Drag witnessed reductions of up to $0.4 \cdot 10^{-3}$ for PUMA-CNN-TRANS and pressure drag was reduced by a maximum of $0.14 \cdot 10^{-3}$ for PUMA-TM-DRAG. This chart also highlights the big percentage of friction drag in the total drag (around 80%), and therefore the importance of delaying transition to reduce drag.

Figure 5.32 depicts the best airfoil, generated by optimization using PUMA-CNN with the objective of maximizing laminar area, and the baseline airfoil. The C_f curves for both the pressure and suction sides are also presented. The optimized airfoil appears thicker after the midpoint on both sides, and the start of the suction side is noticeably thinner. Additionally, transition is delayed on both sides, with a more pronounced delay on the suction side, aligning with the optimization objective.

Figure 5.33 presents the intermittency γ contours for the baseline (top) and the optimized airfoil (bottom). Yellow represents a γ value near 1, indicating turbulent flow, while purple and blue hues indicate laminar and transitional flow. Yellow regions dominate the contour, indicating that the flow is turbulent everywhere except in the boundary layer just above the airfoil, where the flow remains laminar before transition. It can be observed that the optimized geometry achieves a later transition compared to the baseline geometry, as was the goal of the optimization.

In Figure 5.34, the pressure contours are shown. The pressure distribution just above the suction side has changed, with the area of minimum pressure decreasing in height but increasing in length. The pressure distribution on the pressure side and at the leading and trailing edges remains largely unchanged.

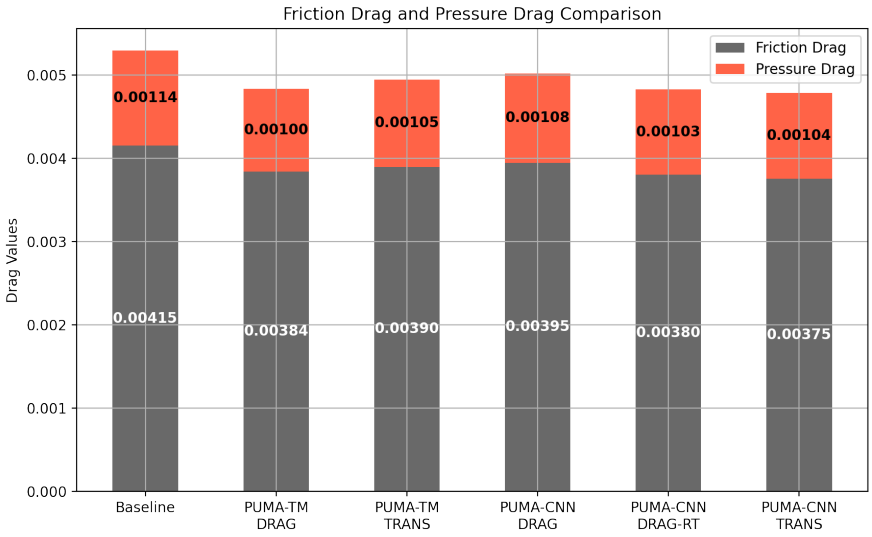


Figure 5.31: *Breakdown of drag components for all optimization schemes. Both friction and pressure drag were reduced in all schemes.*

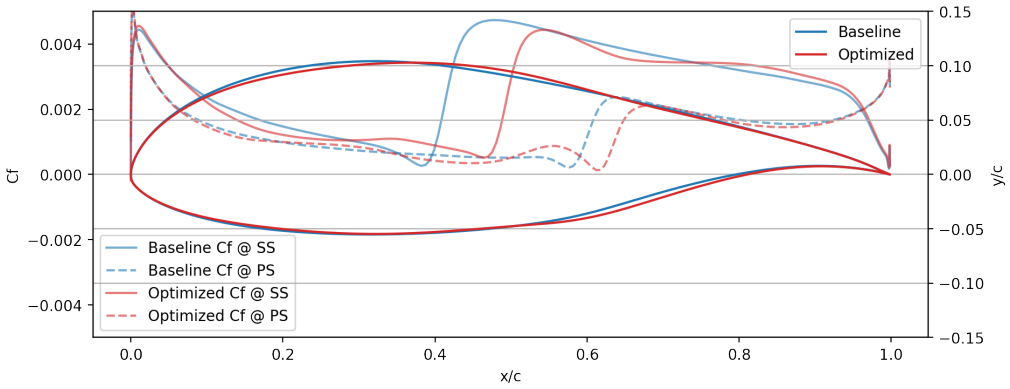


Figure 5.32: *Comparison between the geometries and the C_f curves of the best and the original airfoils. The optimized geometry appears thicker after the midpoint on both sides and thinner at the start of the suction side. Transition is delayed on both sides, more notably on the suction side.*

5.5 Optimization with different RNG

5.5.1 PUMA-TM

To assess whether maximizing the laminar area can reliably surrogate minimizing drag, the optimization experiments were repeated using different random number generator (RNG) seeds for the EASY algorithm. The original optimization with

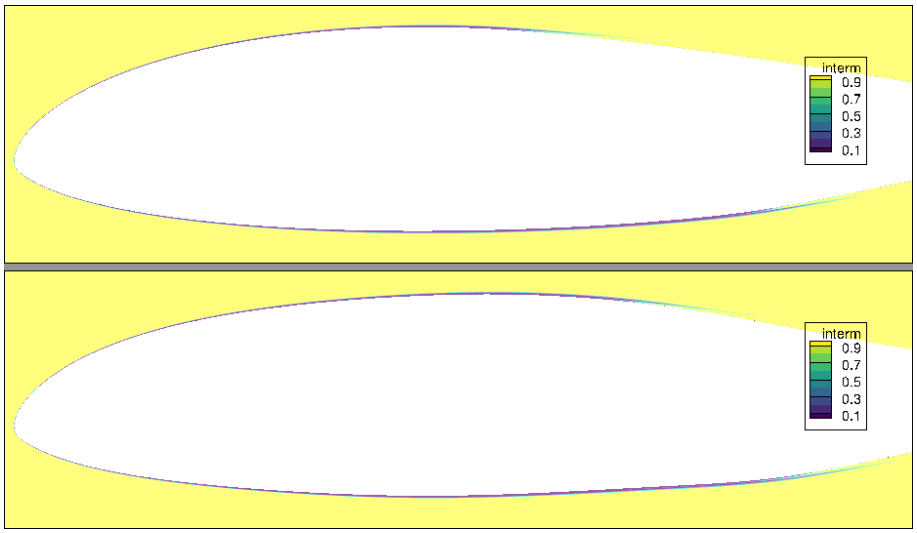


Figure 5.33: *Intermittency γ contours for the baseline (top) and optimized (bottom) airfoils. Yellow ($\gamma \approx 1$) indicates turbulent flow, while purple and blue ($\gamma < 1$) indicate laminar and transitional flow. The optimized geometry shows a delayed transition compared to the baseline, on the suction side.*

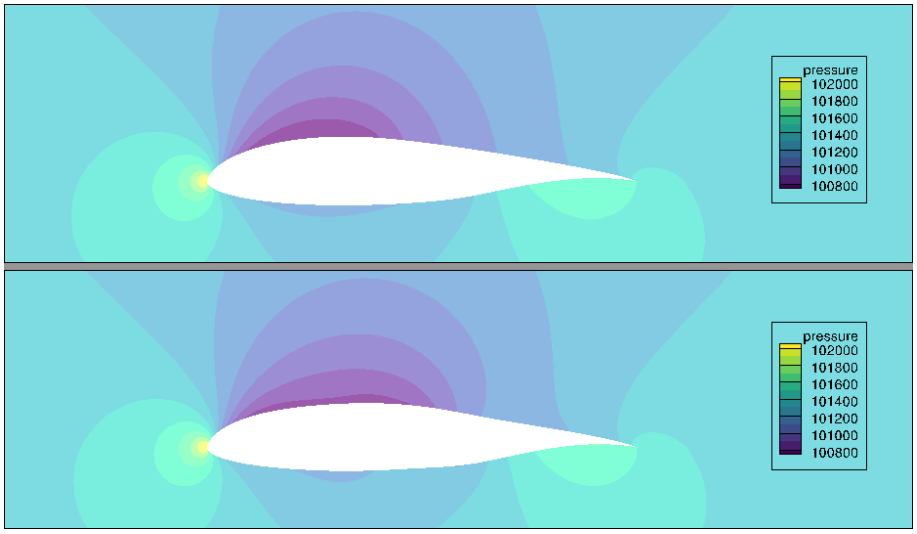


Figure 5.34: *Pressure contours for the baseline (top) and optimized (bottom) airfoils. The suction side shows a change in pressure distribution, with a decreased height and increased length of the minimum pressure area. The pressure side and the leading and trailing edges remain largely unchanged.*

RNG1 was compared against two additional runs, RNG2 and RNG3 using both the $|\mathbf{max}.\ \mathbf{TP_{SS}}|$ and the $|\mathbf{min}.\ \mathbf{C_D}|$ objective functions.

Figure 5.35 illustrates the progress of drag (C_D) and transition point (TP_{SS}) for the three RNGs. The blue line represents the original RNG1, while the red and green lines correspond to RNG2 and RNG3, respectively. The top part of the Figure shows

the evolution of C_D, while the bottom part illustrates TP_{SS}. The left side contains the $|\mathbf{max.\ TP_{SS}}|$ schemes, while the right side the $|\mathbf{min.\ C_D}|$ schemes.

It is evident that RNG1 yielded sub-optimal solutions for both goals, drag and transition, as both RNG2 and RNG3 outperformed it. Comparing based on C_D, setting the objective function to $|\mathbf{max.\ TP_{SS}}|$ proved superior. Although it reached a slightly worse solution in the RNG1 run, for RNG2, it achieved a significantly better solution and performed equally well for RNG3.

Interestingly, the evolution of TP_{SS} using the drag-focused scheme for all RNGs is almost monotonic. The clear elitism in the evolution of C_D is also evident in TP_{SS}, confirming that drag reduction is typically accompanied by transition delay.

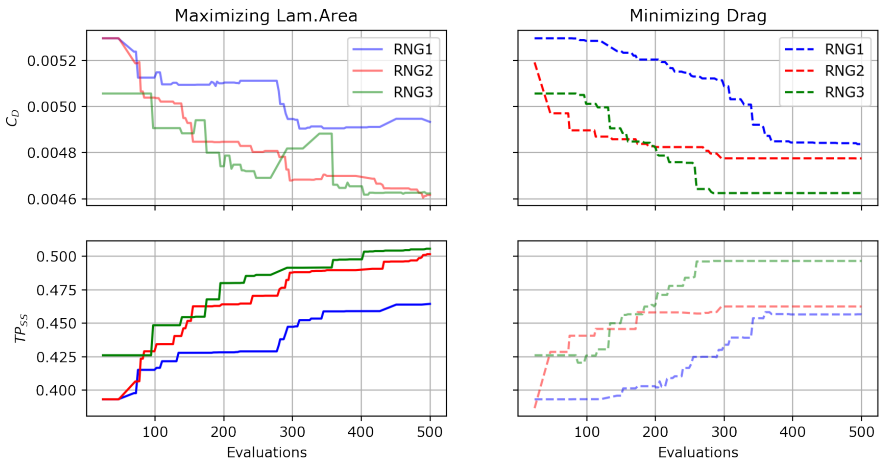


Figure 5.35: *Evolution of C_D and TP_{SS} for different RNGs. Using $|\mathbf{max.\ TP_{SS}}|$ as the objective function (left) leads to better solutions than using $|\mathbf{min.\ C_D}|$ (right). Final C_D (top) is lower for RNG2, RNG3 and slightly worse for RNG1. As for TP_{SS} (bottom) using it as the goal leads to a later transition for all 3 RNG runs.*

In Figure 5.36, the same optimizations are presented separately based on the RNG run. The top part shows C_D, and the bottom part shows TP_{SS}. The plots are arranged from left to right, representing RNG1 to RNG3. Continuous lines denote $|\mathbf{max.\ TP_{SS}}|$, while dashed lines represent $|\mathbf{min.\ C_D}|$. For RNG1 the best solution for both C_D and TP_{SS} in the initial population coincides, so the same second generation is produced. The elite on the second generation is also shared, but for the third generation a better individual is produced in the $|\mathbf{max.\ TP_{SS}}|$ scheme. For the RNG2, even from the random initialization the elites differ. Interestingly in RNG3, the elite selection returns the same individual for the 8 first generations, and only after 87 evaluations the evolution plots start to diverge, as an elite that is not best in both C_D and TP_{SS} is produced.

In Figure 5.37 the correlation between C_D and TP_{SS} is further validated. Both quantities are normalized using the minimum and maximum values observed in all 3

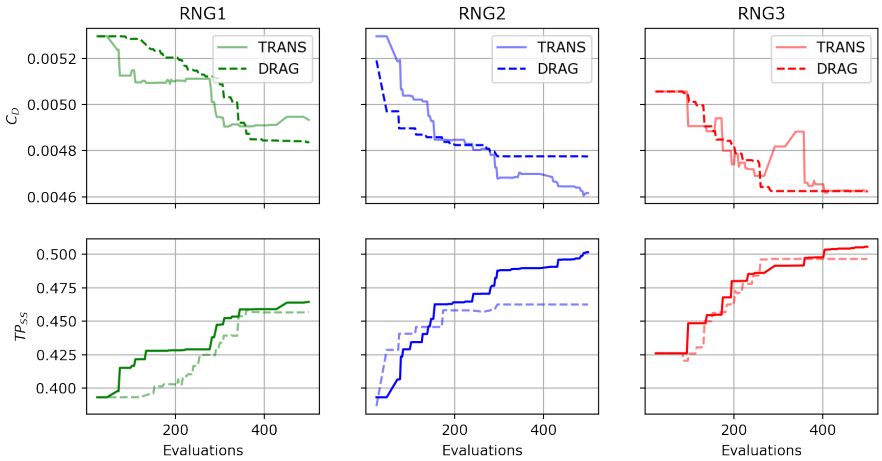


Figure 5.36: *Optimization results for different RNGs. Maximizing TP_{SS} (continuous) proves superior to minimizing C_D (dashed). For RNG1 (left)* $|\mathbf{min.\,C_D}|$ *achieves an 8.7% reduction compared to 7.4% from* $|\mathbf{max.\,TP_{SS}}|$*. For RNG2 (middle)* $|\mathbf{max.\,TP_{SS}}|$ *outperforms, reaching a 13% reduction compared to 9.6%. For RNG3 (right) both schemes produce a 12.5% reduction.*

RNG runs. The x axis represents the normalized C_D and the y axis the normalized TP_{SS}. The data points for the elites of all generations are linearly fitted. For all six optimizations strong linear behavior is observed. The $|\mathbf{min.\,C_D}|$ oriented optimizations (bottom) present a higher correlation than the $|\mathbf{max.\,TP_{SS}}|$ runs (top).

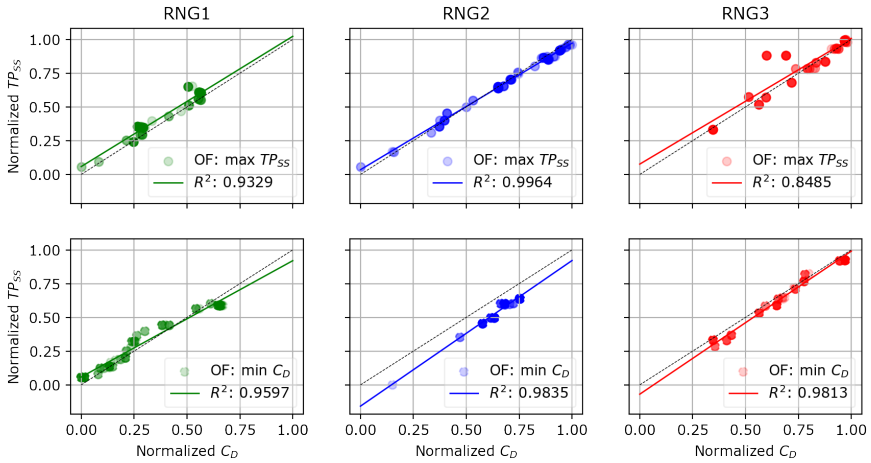


Figure 5.37: *Scatter plot showing the correlation between normalized C_D and TP_{SS} for different RNGs and objective functions. Top:* $|\mathbf{max.\,TP_{SS}}|$*, Bottom:* $|\mathbf{min.\,C_D}|$*.*

Additionally, the resulting flow coefficients and transition points for the three RNGs and both schemes are summarized in Table 5.10.

Table 5.10: *Flow coefficients and points of transition for best solutions for all different RNGs and objective functions.*

RNG	Goal	C_D	C_L	C_M	TP_{SS}	TP_{PS}
1	$\uparrow \mathbf{TP_{SS}}$	4.90	4.91	1.06	0.452	0.593
1	$\downarrow \mathbf{C_D}$	4.83	4.90	1.04	0.456	0.608
2	$\uparrow \mathbf{TP_{SS}}$	4.60	4.99	1.09	0.501	0.623
2	$\downarrow \mathbf{C_D}$	4.78	4.96	1.04	0.462	0.612
3	$\uparrow \mathbf{TP_{SS}}$	4.62	4.94	1.10	0.503	0.609
3	$\downarrow \mathbf{C_D}$	4.63	4.91	1.04	0.496	0.614

In conclusion, this analysis suggests that using transition delay as a surrogate goal for drag reduction, within a shape optimization, can lead to equal or even superior solutions in terms of drag minimization.

5.5.2 PUMA-CNN

To ensure that PUMA-CNN can reliably perform as an evaluation tool for an optimization using EASY, the same numerical experiments were repeated using PUMA-CNN.

In Figure 5.38, the three optimizations with C_D as the objective functions are presented, along with the reevaluations using PUMA-TM. In Figure 5.39, the three optimizations with TP_{SS} as the objective are presented, along with the reevaluations using PUMA-TM. Finally, in Table 5.11, the resulting C_D and TP_{SS} values are presented for all optimizations, with the three different RNG values, using both PUMA-TM and PUMA-CNN as the evaluation software and with the goal function being either C_D or TP_{SS}.

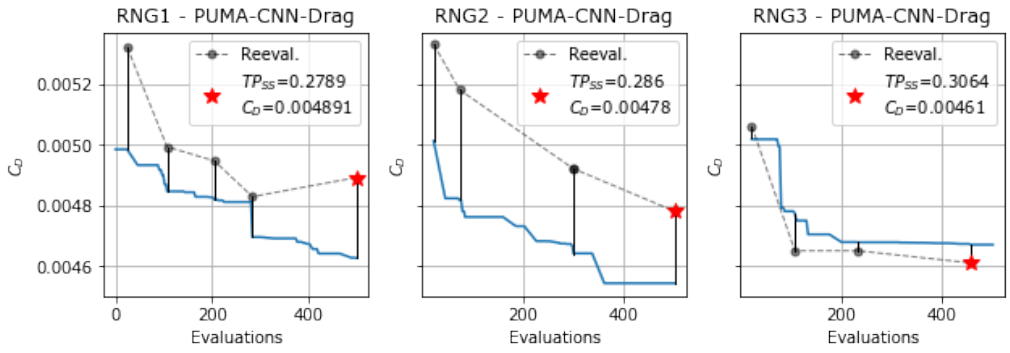


Figure 5.38: *Comparative analysis of optimizations with C_D as the objective function, re-evaluated using PUMA-TM.*

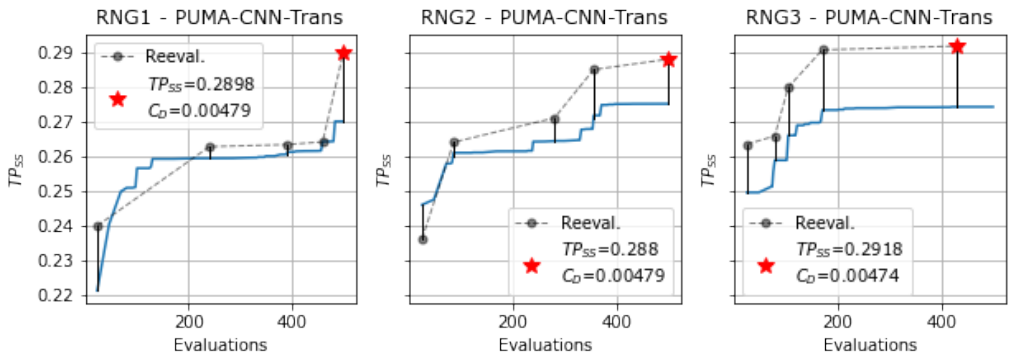


Figure 5.39: *Comparative analysis of optimizations with TP_{SS} as the objective function, re-evaluated using PUMA-TM.*

Table 5.11: *Flow coefficients and points of transition for best solutions for all different RNGs and objective functions, using both PUMA-TM and PUMA-CNN.*

		PUMA-TM		PUMA-CNN	
RNG	Goal	C_D	TP_{SS}	C_D	TP_{SS}
1	$\uparrow \mathbf{TP_{SS}}$	4.90	0.452	4.79	0.475
1	$\downarrow \mathbf{C_D}$	4.83	0.456	4.89	0.457
2	$\uparrow \mathbf{TP_{SS}}$	4.60	0.501	4.79	0.472
2	$\downarrow \mathbf{C_D}$	4.78	0.462	4.78	0.469
3	$\uparrow \mathbf{TP_{SS}}$	4.62	0.503	4.74	0.478
3	$\downarrow \mathbf{C_D}$	4.63	0.496	4.61	0.502

5.6 Conclusions

The findings of this analysis can be summarized in the following.

- CNNs can be used as turbulence closures to replace the Spalart-Allmaras turbulence and the $\gamma - Re_{\theta,t}$ transition model effectively, reducing the evaluation time by approximately 40%, during a shape optimization of the NLF0416 airfoil. In cases where no retraining was required, PUMA-CNN completed 500 evaluations in approximately 425 TUs which is 15% less than the budget of 500 evaluations allowed for PUMA-TM-based optimizations, including the costs of initial training and database generation. Notably, the best-performing airfoil (9.6% drag reduction) was produced by a PUMA-CNN-based optimization, demonstrating that this method can yield superior results at a lower computational cost.
- CNNs incorporating information from neighboring mesh elements showed improved accuracy in predicting μ_t compared to DNNs that work in an element-wise manner. The use of 3×3 kernels in CNNs enabled effective inter-element communication, halving the training loss compared to equivalent DNNs while

only marginally increasing training time.

- Larger kernel sizes (5×5 and 7×7) do not provide the expected performance gains relative to 3×3 kernels. The disproportionate increase in training time outweighs minor improvements in accuracy, making 3×3 kernels the best choice.
- Implementing regular re-evaluations using PUMA-TM during PUMA-CNN-driven optimizations enables early detection of increasing errors, allowing for targeted retraining on best-so-far solutions. This restores the CNN performance allowing the optimization to proceed effectively.
- When using PUMA-TM across three RNG seeds, having TP_{SS} as the objective function consistently yielded equal or superior results in terms of both C_D and TP_{SS}. This suggests that maximizing TP_{SS} can serve not only as an effective surrogate for minimizing C_D, but potentially as a superior objective function, capable of discovering even better solutions.

Chapter 6

The S8052 Isolated Airfoil

6.1 The S8052 Airfoil Case

The S8052 is an airfoil, developed and tested by the University of Illinois at Urbana Champaign Low-Speed Airfoil Tests (UIUC LSATs) team, as part of their endeavor to achieve performance improvements in airfoils used on powered remote control aircraft [28].

The profile of the S8052 can be seen in Figure 6.1, along with the camber line, at an angle of 2^o. The airfoil is modelled using a 705×97 C-type mesh and the flow is solved by the PUMA code, that handles the structured grid as an unstructured one, solving the RANS and 3 additional differential equations from the Spalart-Allmaras turbulence model and the Piottrowski&Zingg's Smooth $\gamma - Re_{\theta,t}$ transition model [33].

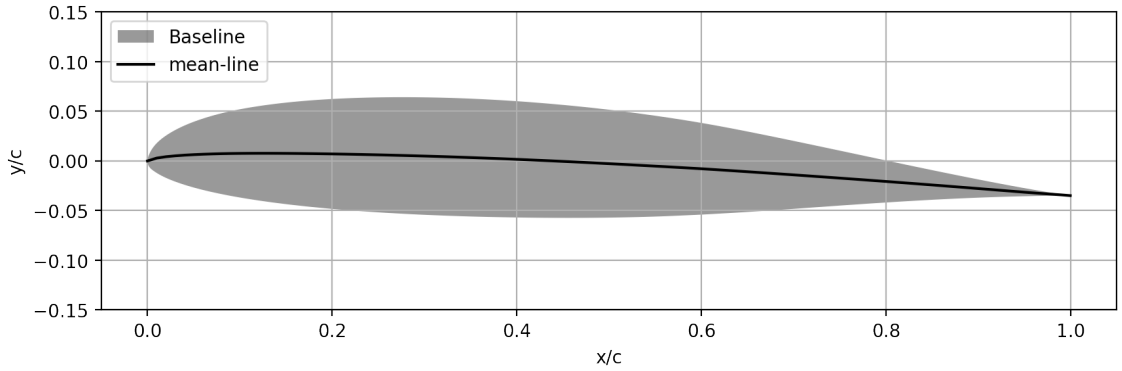


Figure 6.1: *The S8052 Airfoil.*

For this case, the airfoil is simulated under the conditions shown in Table 6.1. The

initial simulations are conducted for the airfoil on two different angles of attack, $\alpha = 0°$ and $\alpha = 2°$, and the resulting coefficients after convergence are presented in Table 6.2.

The friction drag coefficient $C_{D,f}$, the drag component accounting for the shear stresses on the surface of the airfoil, is approximately 80% of the total drag coefficient C_D, highlighting the expected benefit of delaying transition.

Table 6.1: *S8052 Case Conditions.*

Flow Conditions	
Chord length c	1 m
Mach number M_∞	0.1
Reynolds number Re_∞	$2.29 \cdot 10^6$
Turbulence Intensity TI	0.005

Table 6.2: *Aerodynamic Coefficients for the Baseline Geometry.*

Angle of Attack α_∞	**0°**	**2°**
Drag Coefficient C_D	$4.76 \cdot 10^{-3}$	$5.37 \cdot 10^{-3}$
Friction Drag Coefficient $C_{D,f}$	$3.98 \cdot 10^{-3}$	$4.28 \cdot 10^{-3}$
Pressure Drag Coefficient $C_{D,P}$	$0.78 \cdot 10^{-3}$	$1.09 \cdot 10^{-3}$
Lift Coefficient C_L	$1.72 \cdot 10^{-1}$	$3.99 \cdot 10^{-1}$
Moment Coefficient C_M	$3.01 \cdot 10^{-2}$	$3.10 \cdot 10^{-2}$

The friction distributions along the airfoil blade for $\alpha = 0°$ and $\alpha = 2°$ are presented in Figure 6.2. For $\alpha = 0°$ transition occurs at approximately 60% of the chord both on the suction side and on the pressure side. Increasing the angle to 2^o results in earlier transition on the suction side (approximately 40%) and later transition on the pressure side (approximately 70%).

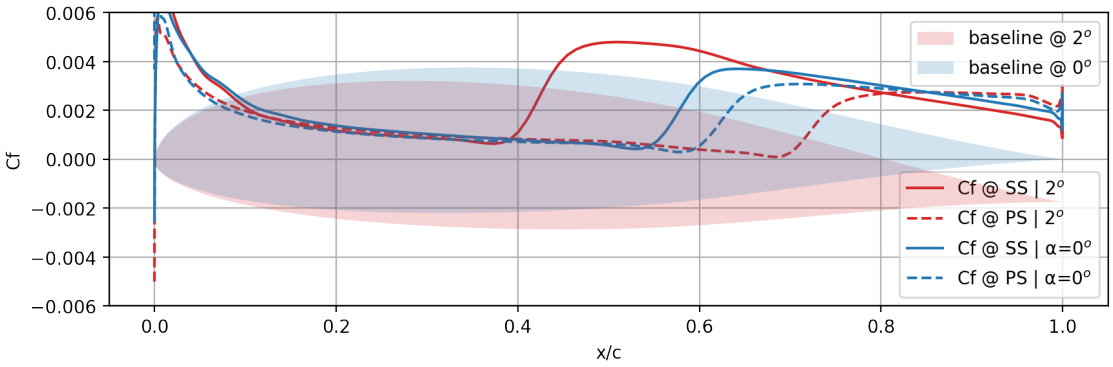


Figure 6.2: *Friction distribution along the baseline geometry. As the angle of attack increases, transition begins earlier on the suction side and later on the pressure side.*

From this point forward, the analysis focuses on the airfoil at an angle of $\alpha = 2°$.

6.2 Shape Parameterization

NURBS were utilized for airfoil parameterization, employing the box shown in Figure 6.3. Red points are movable, while blue points remain fixed. Movable points have a 10% range vertically and horizontally, resulting in 24 design variables, two for each of the 12 control points.

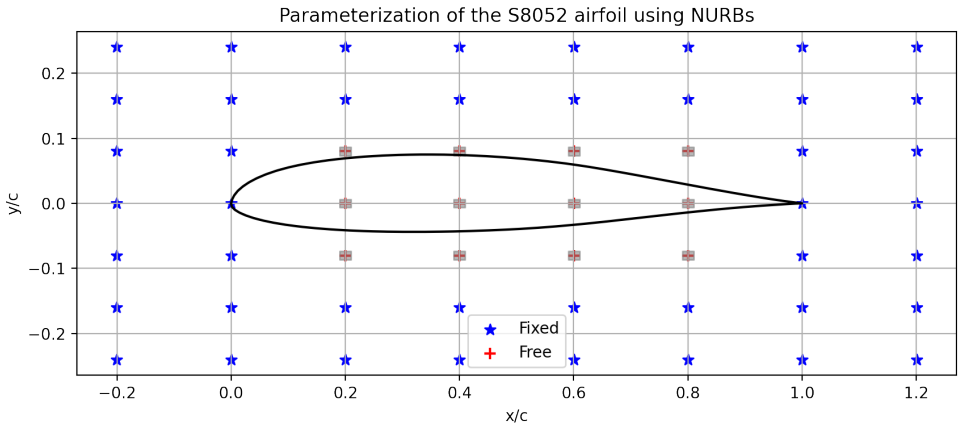


Figure 6.3: *Parameterization of the S8052 airfoil. The free-to-move control points are marked in red and are allowed to move* $\pm 10\%$ *vertically and horizontally. This plot is not in scale.*

Other than introducing the design variables for the optimization, the parameterization allow for the creation of the training database for the CNN. LHS explores the design space and produces 50 unique geometries as displayed in Figure 6.4 surrounding the baseline geometry highlighted in red. PUMA is then employed to compute the corresponding flow fields, solving the RANS, the Spalart-Allmaras turbulence model, and the $\gamma - Re_{\theta,t}$ transition model, forming DB_{CNN}.

The geometrical and the flow data that are available as CNN inputs are the same as in the previous case, as presented in Table 5.3. The selected inputs are scaled before feeding them to the network, to safeguard against challenges arising from the presence of different scales within the input data.

6.3 CNN Comparison

In this section, starting with a CNN architecture using exclusively size 1 kernels (C_0 in Figure 6.5), the effect of increasing the kernel size of each layer to 3 will be investigated. For C_0, no layer uses a size 3 kernel, meaning no information about neighboring elements is utilized for prediction. In C_1, only the first layer uses a 3×3

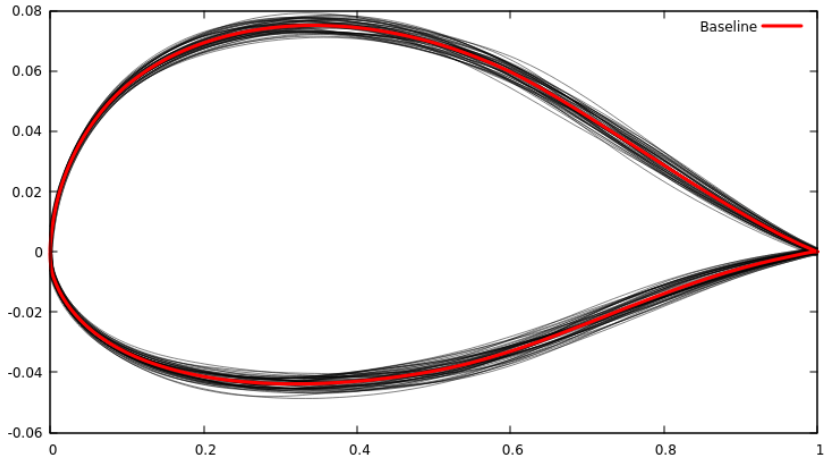


Figure 6.4: *The 50 airfoil profiles from DB_{CNN} (in black) surrounding the baseline geometry (in red). The parameterization can create an airfoil profile roughly within the area in black. This plot is not in scale.*

kernel, with the rest retaining a size of 1. In C_2 and C_3, the first 2 and the first 3 convolutional layers respectively employ a 3×3 kernel.

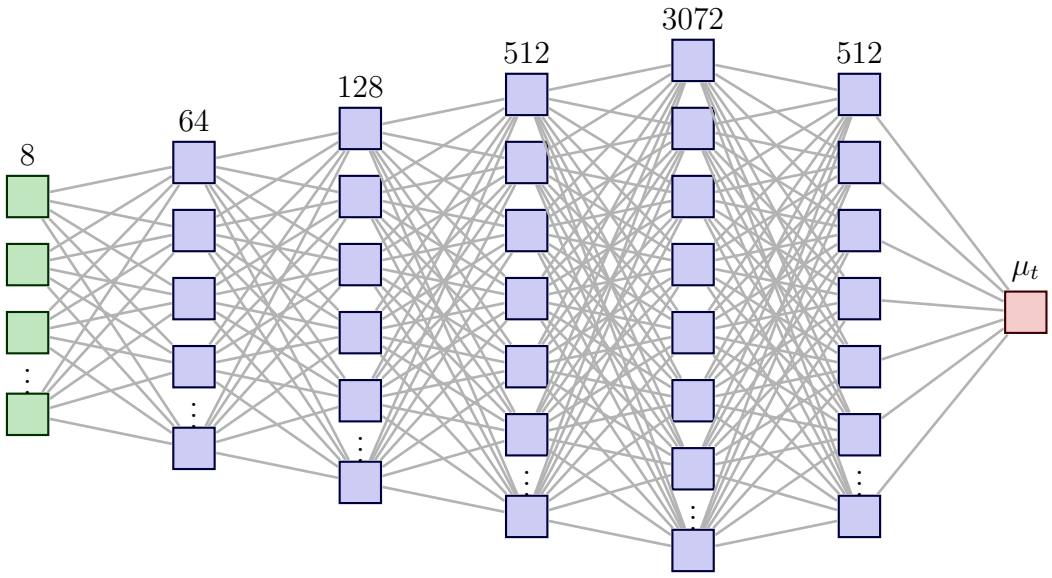
The CNNs use 8 input fields to produce one output, the μ_t field. The input fields are: the nodal coordinates x_k and the wall distance W_d, the pressure field P, the velocity vector field u_k, the strain rate S and the vorticity Ω. All four CNNs were trained using the same configuration on the same 50 training geometries and were validated on the 10 validation geometries. Figure 6.6 illustrates the training and validation losses for the four networks.

C_0 achieves a training loss of $1.1 \cdot 10^{-5}$ and a validation loss of $1.5 \cdot 10^{-5}$. On the other hand, C_1, C_2, and C_3 all reach a training loss of approximately $0.6 \cdot 10^{-5}$, while the validation curve oscillates around $1 \cdot 10^{-5}$, with no clear winner among them.

It is evident that using a non-unit kernel has a significant impact even from the first layer. The networks that contain at least one layer using a 3×3 kernel, account for neighboring elements in their predictions, leading to better performance over both training and validation geometries.

The fact that even C_1, a network that includes a 3×3 kernel in just first layer, displays such an immense improvement, suggests that this improvement stems from the inherent physics of utilizing neighboring information rather than an increase in representational power coming from a minimal increase in free weights-parameters.

Moreover, using a non-unit kernel in all subsequent layers can be ineffective, considering increases in training and inference costs.



C_0	k=1	k=1	k=1	k=1	k=1	k=1
C_1	k=3	k=1	k=1	k=1	k=1	k=1
C_2	k=3	k=3	k=1	k=1	k=1	k=1
C_3	k=3	k=3	k=3	k=1	k=1	k=1

Figure 6.5: *CNN architectures used for comparison. C_0 utilizes only 1×1 kernels, while C_1, C_2, and C_3 progressively incorporate 3×3 kernels in the initial layers.*

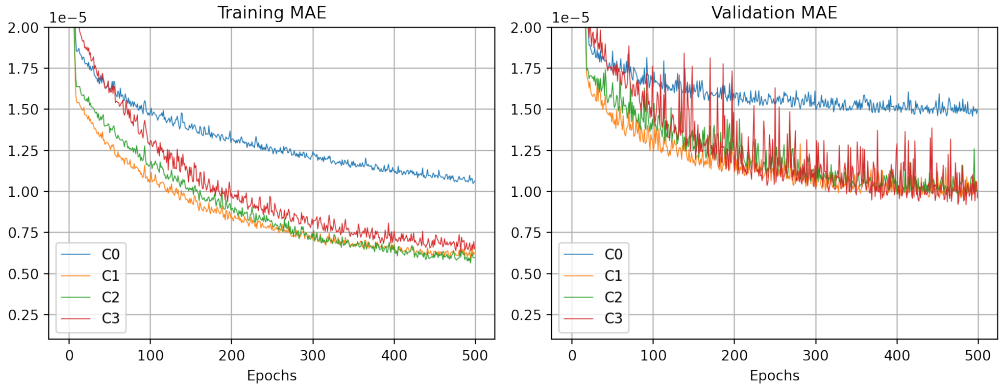


Figure 6.6: *Training and validation losses for the four CNN architectures.*

6.4 CNN Validation

From the four architectures mentioned in the previous section, C_3 will be used in coordination with PUMA to replace the turbulence and transition models, as it achieved the lowest validation loss.

As the CNN predicts the μ_t field, and not the relative to the optimization metrics, its performance over the μ_t predictions is not as important as the performance of the whole PUMA-CNN system, in predicting TP_{SS} and C_D.

Comparing the computed by PUMA-CNN baseline flow fields, to the corresponding of PUMA-TM is of high importance, as the baseline represents a midpoint of the design space and also an initial solution for the optimization algorithm.

In Figure 6.7, the μ_t fields surrounding the baseline computed by PUMA-TM and PUMA-CNN are depicted. Small errors are apparent on the suction side and the pressure side where separation occurs, while larger errors in the wake region.

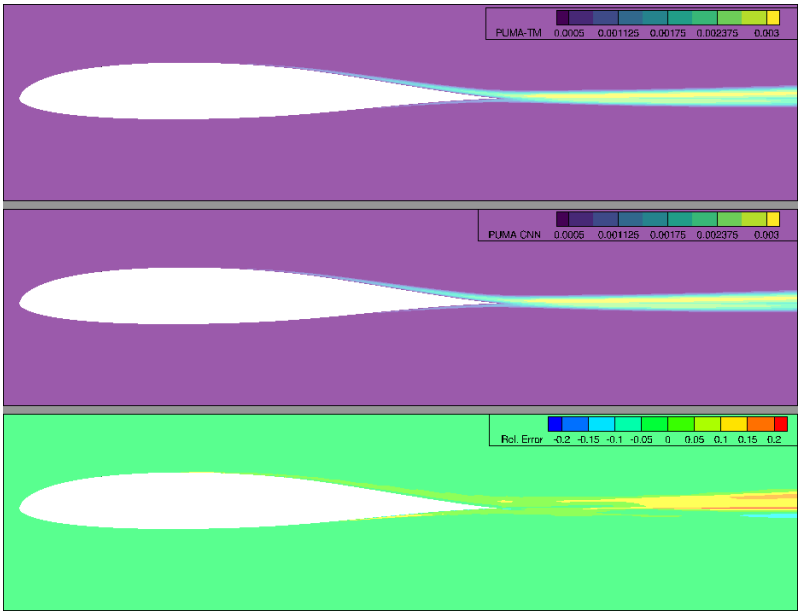


Figure 6.7: *Comparison of μ_t field between PUMA-TM (top) and PUMA-CNN (middle), highlighting the differences (bottom).*

As for the flow coefficients, PUMA-CNN overestimates C_D by 0.5% and underestimates C_L and C_M by 1% and 1.2% respectively, as shown on Table 6.3.

Table 6.3: *Comparison of Drag, Lift, and Moment coefficients for the baseline: PUMA-CNN vs PUMA-TM.*

Metric	PUMA-CNN	PUMA-TM	Relative Error
Drag	0.00539	0.00537	+0.5%
Lift	0.39543	0.39949	−1.0%
Moment	0.03067	0.03104	−1.2%

To validate the performance of $PUMA-CNN$ in predicting C_D and generating accurate C_f curves, PUMA-CNN was run on the same 10 validation geometries used during the CNN's training.

In Figure 6.8 the actual and predicted values of drag are compared for the 10 geometries. PUMA-CNN consistently underestimates drag by an average of 3.8%.

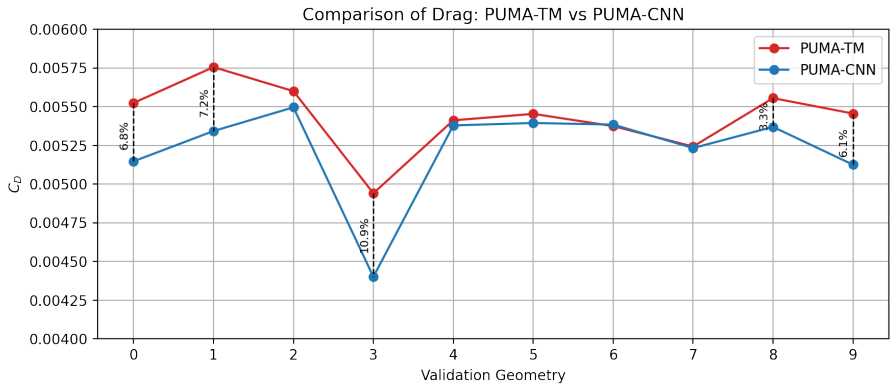


Figure 6.8: *PUMA-CNN underestimates the drag for all test geometries. The Mean Relative Error is 3.8% over the 10 geometries.*

6.5 Shape Optimization

To optimize the airfoil, two MAEA-based optimizations were conducted, with distinct objectives: minimizing drag and maximizing laminar area, each using two evaluation software: the original PUMA-TM and the cost-effective PUMA-CNN. Thus, a total of four optimization schemes, as shown in Table 6.4, were executed.

Table 6.4: *The 4 optimization schemes.*

Objective	Using TM	Using CNN
Min. Drag	PUMA-TM-DRAG	PUMA-CNN-DRAG
Max. Lam. Area	PUMA-TM-TRANS	PUMA-CNN-TRANS

All four schemes had the same settings imposed on EASY, as outlined in Table 6.5. Constraints included maintaining the original moment and initial lift, as well as avoiding excessive reduction in airfoil area.

6.5.1 Optimization with the PUMA-TM software

The PUMA-TM software handles the complete set of RANS equations: the 4 mean flow equations, 1 turbulence equation, and 2 transition model equations. Each evaluation using PUMA-TM is referred to as a Time Unit (TU). Both optimization schemes, PUMA-TM-DRAG, and PUMA-TM-TRANS terminate after reaching 500 TUs.

Table 6.5: *Settings and Constraints for the evolutionary algorithm.*

Settings	
Coding	Real
Parents	10
Offspring	24
Number of Elites	5
Principal Component Analysis (PCA)	Yes
Meta-Model Assistance — Inexact Pre-Evaluation	
Type	RBF_IFs
Minimum DB entries to start	50
Minimum not failed DB entries	30
Patterns for training	$35 - 55$
Exact evals. per generation	$2 - 3$
Extrapolate prediction	No
Non-dimensionalize prediction	Yes
Idle generations for IPE pause	5

Constraints
$C_M \geq C_{M,\text{base}} - 0.0007$, $C_M \leq C_{M,\text{base}} + 0.0007$
$C_L \geq C_{L,\text{base}}$
$A \geq 0.9 \cdot A_{\text{base}}$

Objective Function	
max. TP_{SS}	to maximize the laminar area
min. C_D	to minimize drag

In Figure 6.9, the evolution of various quantities across the two schemes is depicted. Blue indicates optimization aimed at delaying transition, while red indicates optimization focusing on drag reduction.

Both optimizations achieved a substantial drag reduction. The best solution from PUMA-TM-DRAG has a C_D equal to $4.79 \cdot 10^{-3}$, while that of PUMA-TM-TRANS reaches $4.80 \cdot 10^{-3}$. Therefore a reduction of approximately 11% was made from both optimizations.

The transition point on the suction side moved from 0.39m on the baseline to 0.49m on PUMA-TM-TRANS's best and 0.47 on PUMA-TM-DRAG's best.

Drag reduction appears to coincide with delayed transition, as highlighted in Figure 6.10. In Figure 6.11, the values of C_D and TP_{SS} for all 500 individuals within the two optimizations are scattered and linear regression is performed. The results highlight a linear relationship between C_D and TP_{SS} encouraging the idea that the decreasing TP_{SS} can work as a proxy for increasing C_D.

Figure 6.12 shows the C_f distribution for the best solutions compared to the baseline.

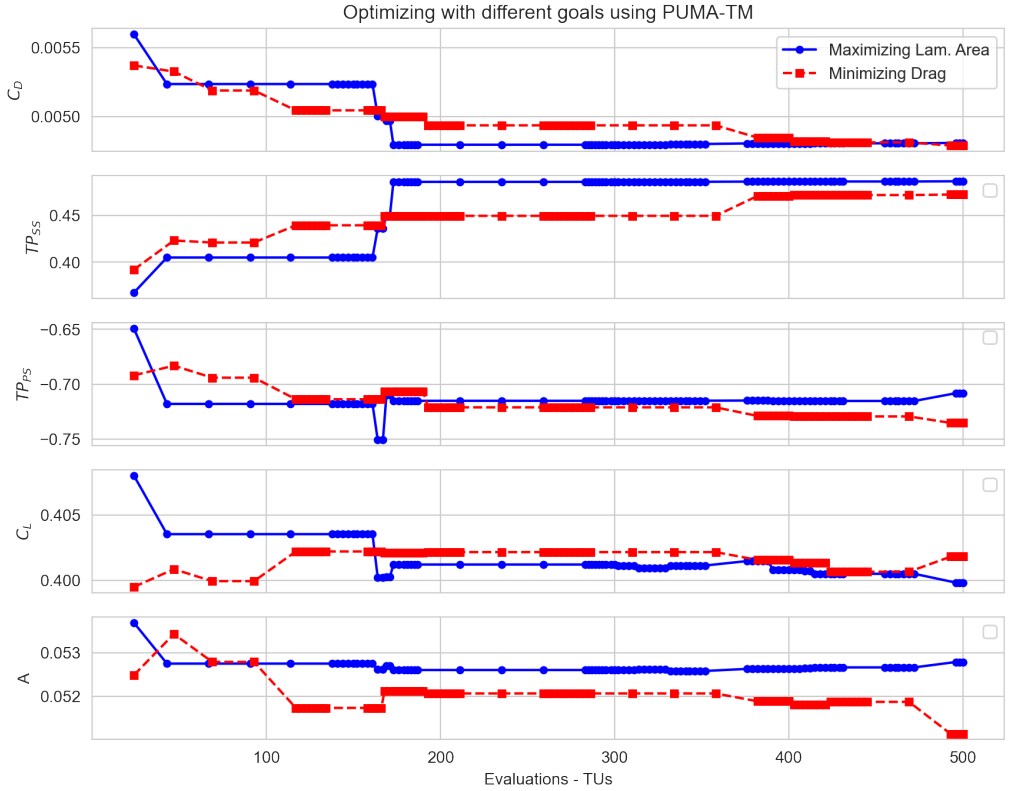


Figure 6.9: *Evolution of relevant metrics throughout the optimization using PUMA-TM. Drag is reduced and transition is delayed on the suction side. Lift and area respect the set of constraints.*

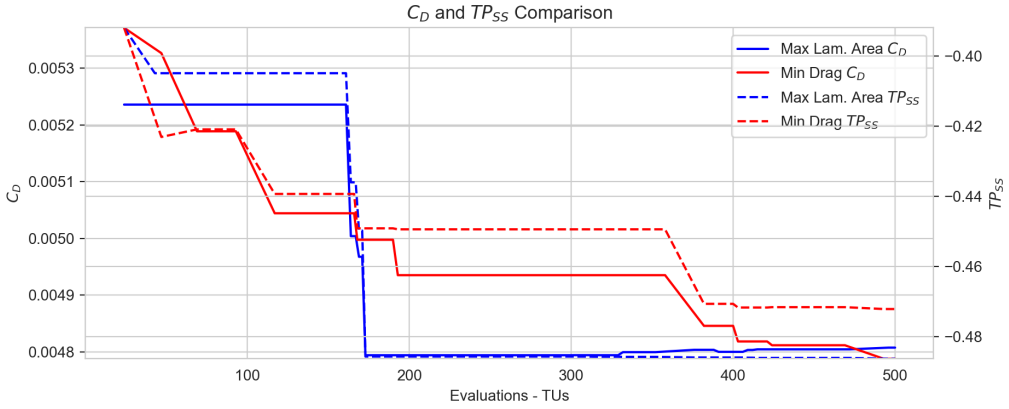


Figure 6.10: *Progression of C_D and TP_{SS}.*

Both schemes achieve a substantial delay in transition for the suction side, while on the pressure side, the transition point is shifted less by PUMA-TM-TRANS and more by PUMA-TM-DRAG.

Figure 6.13 displays the profiles of the three airfoils. The optimized airfoils generated

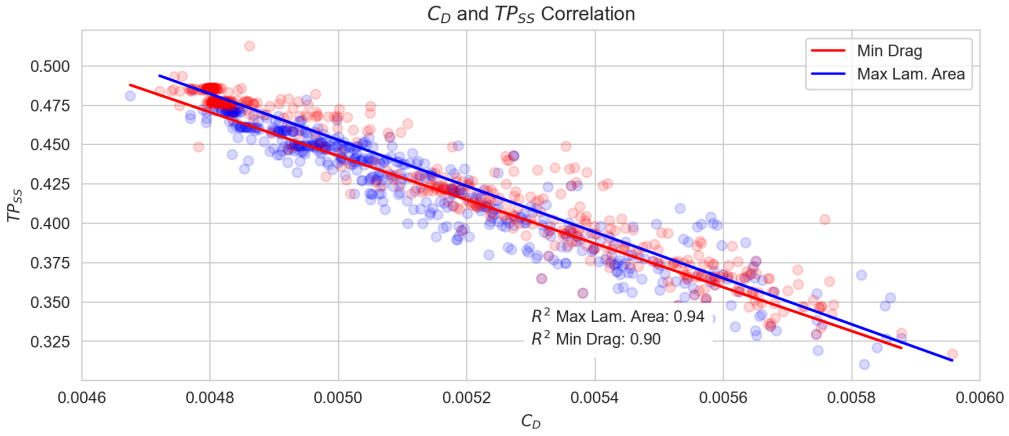


Figure 6.11: *Correlation between C_D and TP_{SS} values for all 500 individuals.*

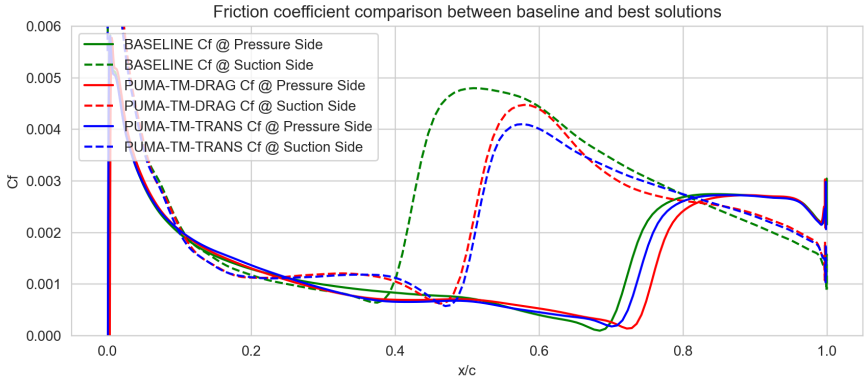


Figure 6.12: *Friction coefficient along the baseline and the optimized blades. Transition is delayed in both sides of the two optimized airfoils.*

by PUMA-TM-DRAG and PUMA-TM-TRANS exhibit remarkably similar suction sides, with a thicker profile in the middle section and a thinner profile towards the end, effectively delaying transition. PUMA-TM-DRAG deliberately thins the pressure side to reduce drag, while PUMA-TM-TRANS, lacking the incentive to modify the pressure side, barely does so.

6.5.2 Optimization with the PUMA-CNN software

Implementing PUMA-CNN aims to reduce computational costs. Unlike PUMA-TM, which involves solving three additional differential equations (one for turbulence and two for transition modeling), PUMA-CNN utilizes a CNN to compute turbulence μ_t.

PUMA-CNN is computationally efficient, completing evaluations in around 6 minutes, 29% faster than PUMA-TM, which takes 8.5 minutes. However, the CNN un-

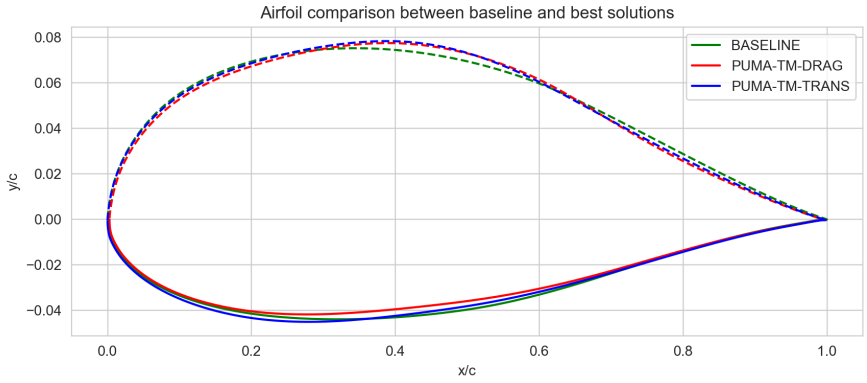


Figure 6.13: *Comparison of the optimized airfoil profiles with the baseline. Both optimized airfoils exhibit a suction side that is thicker in the middle and thinner towards the end. While PUMA-TM-DRAG thinned the pressure side, PUMA-TM-TRANS largely preserved the original shape. This plot is not in scale.*

derwent training on a dataset of 50 airfoils and was validated with an additional 10, all solved by PUMA-TM. Training the network took 2 hours and 5 minutes, equivalent to 14 PUMA-TM evaluations. Therefore, any optimization using PUMA-CNN incurs a capital cost of 74 TUs. Table 6.6 summarizes the computational costs for the two softwares.

Table 6.6: *Computational costs for PUMA-TM and PUMA-CNN.*

Software	Cost for DB	Cost to train	Cost per run
PUMA-TM	0	0	1 TU = 8.5m
PUMA-CNN	60TUs	2h 5m	6m
CNN/CFD	60TUs	14TUs	0.71TUs

Minimizing Drag

PUMA-CNN-DRAG is employed to minimize C_D, running for 500 evaluations and costing a total of 429 TUs. Figure 6.14 shows the optimization progress compared to the equivalent optimization using PUMA-TM. Throughout the process, several PUMA-CNN solutions were reevaluated using PUMA-TM to ensure accuracy. The drag was underestimated by approximately 6%.

PUMA-CNN-DRAG's final solution was reached in 429 TUs and resulted in an 8.8% reduction in drag, from $5.37 \cdot 10^{-3}$ to $4.90 \cdot 10^{-3}$, while PUMA-TM-DRAG achieved a 10.8% reduction.

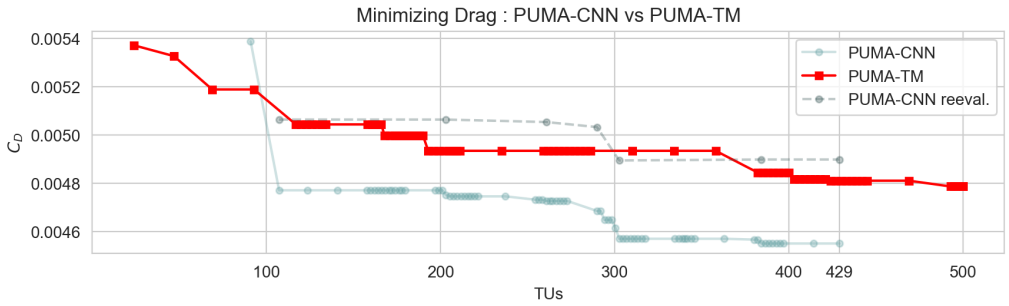


Figure 6.14: *Optimization progress for minimizing drag using PUMA-CNN-DRAG compared to PUMA-TM-DRAG. All data points respect the set of constraints.*

Maximizing Laminar Area

Because the performance of PUMA-CNN in predicting TP_{SS} was deemed inadequate for an optimization where TP_{SS} is the objective, DB_{CNN} was enriched with an additional 30 geometries and the CNN was retrained. Retraining the network and generating the new geometries cost an additional 40 TUs. The retrained network coupled with PUMA (PUMA-CNN-RT) managed to predict TP_{SS} accurately, with an average error of 3%, as depicted in Figure 6.15.

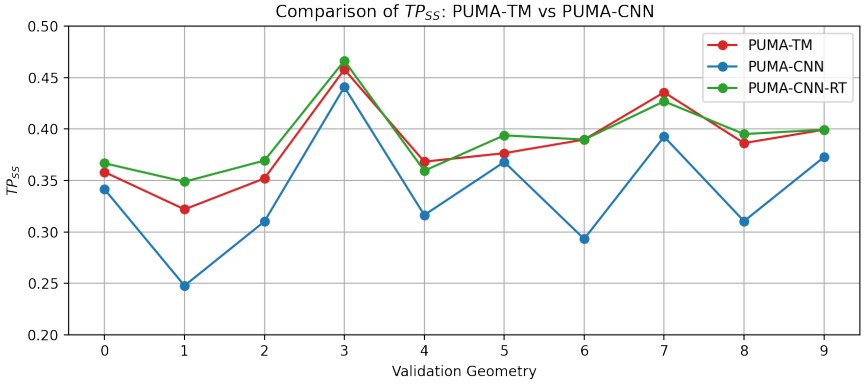


Figure 6.15: *Accuracy of TP_{SS} predictions by PUMA-CNN and by the retrained PUMA-CNN-RT. All data points respect the set of constraints.*

PUMA-CNN-TRANS aims to delay the transition on the suction side, maximizing the laminar area. Running the optimization for 500 evaluations amounts to a cost of 469 TUs, accounting a training cost of 114 TUs.

As displayed in Figure 6.16, the algorithm reached the same C_D as PUMA-TM-TRANS but higher TP_{SS}. Transition on suction side occurs on approximately 0.5m, 25% later than on the baseline geometry. Throughout the optimization, C_D was constantly underestimated and TP_{SS} slightly overestimated.

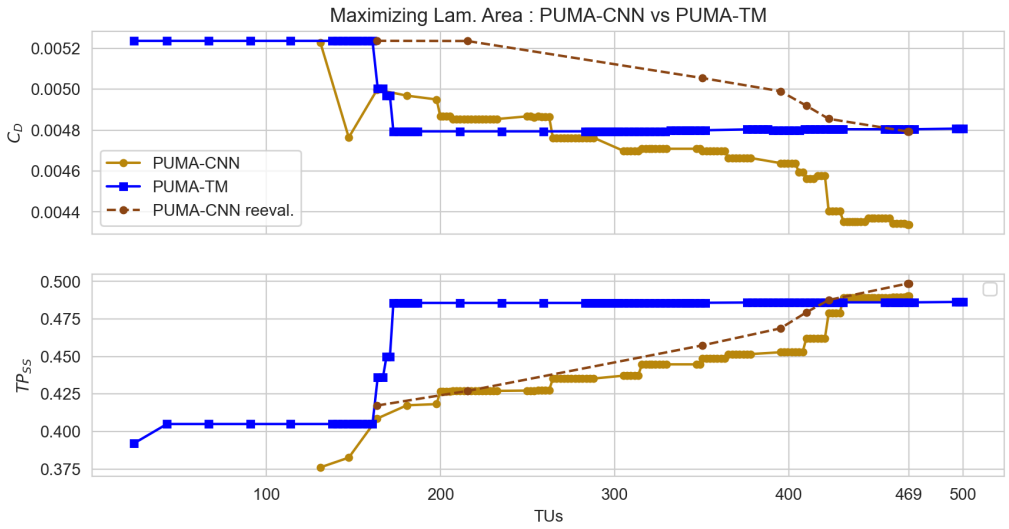


Figure 6.16: *Optimization progress for maximizing laminar area using PUMA-CNN-TRANS compared to PUMA-TM-TRANS. PUMA-CNN-TRANS achieved later transition (goal) and lower drag coefficient.*

6.5.3 Comparison of the 4 optimization schemes

In this section the baseline airfoil and the four best solutions from the four optimization schemes are compared.

In Figure 6.17, the C_D and TP_{SS} values for each optimized airfoil are presented in comparison to the baseline results. Drag was reduced by 8.8% to 10.9% and transition was delayed by 19% to 27%.

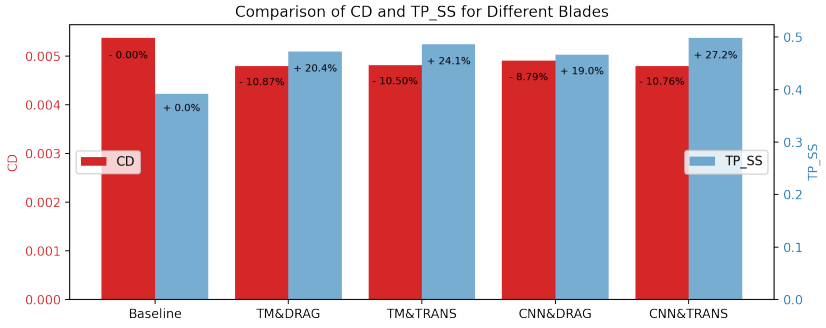


Figure 6.17: *Comparison of C_D and TP_{SS} for each optimized airfoil relative to the baseline. Significant reductions in drag and delays in transition are evident.*

In Figure 6.18 the two drag components are plotted. All optimization schemes successfully reduced both friction and pressure drag. Pressure drag was reduced from $1.09 \cdot 10^{-3}$ for the baseline up to about $0.95 \cdot 10^{-3}$ for all optimized airfoils. Further decreases in total C_D stemmed from decreases in the friction component,

as $C_{D,f}$ in the optimized airfoils ranges from $3.83 \cdot 10^{-3}$ to $3.94 \cdot 10^{-3}$, compared to $4.28 \cdot 10^{-3}$ for the baseline geometry.

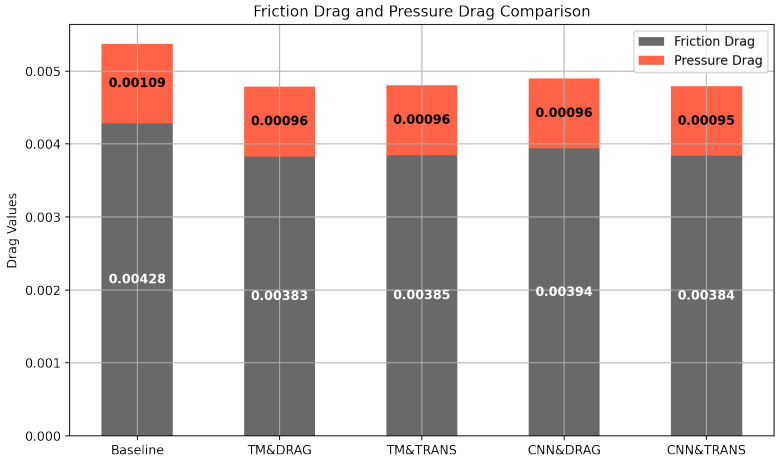


Figure 6.18: *Breakdown of drag components for all optimization schemes. Both friction and pressure drag were reduced in all schemes.*

The best airfoil in terms of C_D was generated in an optimization using C_D as the objective function, with PUMA-TM being the evaluation software. The geometry, and the C_f curves compared to the baseline are presented in Figure 6.19

Along the suction side, the optimized airfoil is thinner at the beginning, gets thicker after 40% of the length and thins down again at the end. As for the pressure side, the start and the end of the remain unchanged with the middle getting thinner. Transition is delayed on both sides, with a more pronounced delay on the suction side, where the largest gain in terms of friction drag is made.

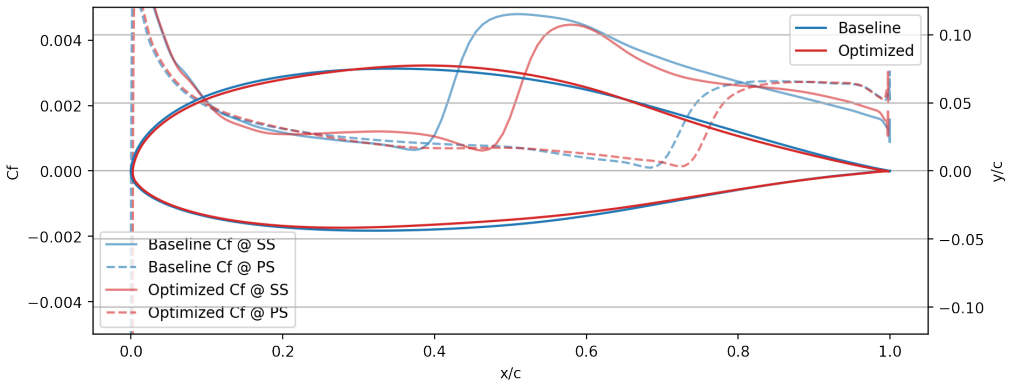


Figure 6.19: *Comparison between the geometries and the C_f curves of the best and the original airfoils.*

Figure 6.20 presents the intermittency γ contours for the baseline (top) and the

optimized airfoil (bottom). Yellow represents a γ value near 1, indicating turbulent flow, while purple and blue hues indicate laminar and transitional flow. Yellow regions dominate the contour, indicating that the flow is turbulent everywhere except in the boundary layer just above the airfoil, where the flow remains laminar before transition. It can be observed that the optimized geometry achieves a later transition on both sides as the hues remain darker for longer over the airfoil.

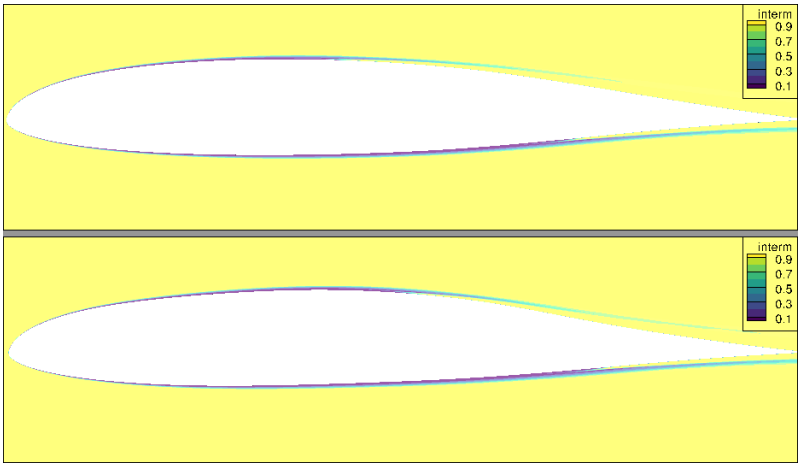


Figure 6.20: *Intermittency γ contours for the baseline (top) and optimized (bottom) airfoils. The optimized geometry retains laminarity (blue hues) for longer, especially on the suction side.*

In Figure 6.21, the pressure contours are shown. The pressure distribution just above the suction and the pressure side has changed, with the area of minimum pressure decreasing in height and increasing in length.

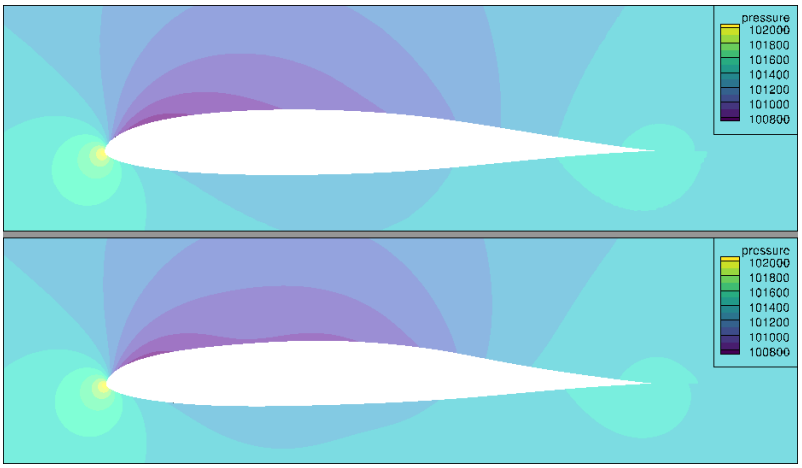


Figure 6.21: *Pressure contours for the baseline (top) and optimized (bottom) airfoils.*

6.6 Conclusions

The findings of this analysis can be summarized in the following.

- As observed in the NLF0416 case, CNNs can effectively surrogate the Spalart-Allmaras turbulence and the $\gamma - Re_{\theta,t}$ transition model in a MAEA-based shape optimization. The evaluation cost was reduced by approximately 30% for the S8052 case, compared to 40% for the NLF0416 case. This difference is due to the distinct CNN architectures used, resulting in varying inference costs.
- The CNN, initially trained on 50 samples, demonstrated good performance in predicting C_D and was successfully used for optimization with C_D as the objective function. However, to achieve comparable performance in TP_{SS} predictions for an optimization targeting TP_{SS}, the CNN required additional training on an enriched database of 80 samples. Despite this, both PUMA-CNN optimizations remained cost-effective: the C_D-focused optimization cost 429 TUs, while the TP_{SS}-focused optimization cost 469 TUs, both below the 500 TU budget allowed for the PUMA-TM-driven optimizations.
- Unlike the NLF0416 case, PUMA-TM with C_D as the objective produced the best airfoil in terms of C_D. However, PUMA-CNN optimization with TP_{SS} as the objective achieved a nearly equivalent solution (10.8% vs 10.9% C_D reduction) in 6% less time.
- Consistent with NLF0416 findings, maximizing laminar area by delaying transition proved an effective surrogate for minimizing drag. A strong correlation between C_D and TP_{SS} was demonstrated, validating this approach across different airfoil types.
- Similar to the first case, using 3×3 kernels in the CNN layers provides a significant performance boost compared to using only 1×1 kernels. However, in contrast to the previous case, this study examined whether all layers require 3×3 kernels. It was found that using 3×3 kernels only in the first few layers of the network provides the same performance improvement as using them in later layers as well, suggesting that early capture of local spatial correlations is sufficient.

Chapter 7

Conclusions

7.1 Overview

In this Diploma Thesis, a novel approach was explored to reduce the computational cost associated with aerodynamic analysis and shape optimization by employing Convolutional Neural Networks (CNNs) as surrogate models for traditional turbulence and transition models.

The research focused on two low-speed airfoils, the NLF0416 and the S8052, both exhibiting transitional flow. The two airfoils were subject to a MAEA-based optimization using EASY. Two optimization targets were considered: minimizing the drag coefficient (C_D) and maximizing the laminar area on the suction side of the airfoil by delaying the transition point (TP_{SS}). Two evaluation software packages were utilized: PUMA-TM, which solved the RANS equations coupled with the one-equation Spalart-Allmaras turbulence model and the two-equation Smooth $\gamma - Re_{\theta,t}$ transition model, and the proposed PUMA-CNN, which employed a trained CNN to predict the turbulent viscosity field μ_t, thereby closing the RANS.

The CNNs used by PUMA-CNN were trained to predict μ_t, using geometrical and flow fields obtained from the RANS. Training was performed on datasets created by parameterizing the baseline geometries of the two airfoils, generating new geometries through LHS and computing the actual flow fields using PUMA-TM. The CNN of the first case was trained on 80 geometries, while the CNN of the second was trained on 50.

During the research about the CNN architecture, for the NLF0416 case, the effect of the kernel size of the convolutional layers was investigated. A CNN using 1×1 filters proved inferior to CNNs using 3×3 filters or larger. This was attributed to the lack of inter-element communication in 1×1 CNNs, which essentially function

similarly to DNNs treating each mesh element separately. On the other hand CNNs using any non-unit filter incorporate neighboring mesh elements to make predictions and achieve better accuracy. Filters larger than 3×3, showed minimal decreases in accuracy but high increases in computational cost, leading to the decision to use 3×3 kernels. For the S8052 case, it was determined that not all convolutional layers need 3×3 filters, as having just the first few layers with 3×3 filters proved sufficient.

As for the optimizations, both goals - minimizing C_D and maximizing TP_{SS} - achieved equivalent decreases in C_D showing that delaying transition can indeed serve as a proxy to decreasing drag. The PUMA-CNN evaluation software lead to equally good solutions in comparison to PUMA-TM in less time. Even in cases were retraining was required due to exploding errors in PUMA-CNN predictions, the optimization cost (including retraining) did not exceed the 500 Time Units required for PUMA-TM. In both cases it was proven that the CNN can indeed substitute the traditional turbulence and transition models to accelerate a shape optimization.

7.2 Conclusions

Based on the works performed during this Diploma Thesis, the following conclusions were drawn.

- CNNs demonstrated their capability to effectively surrogate the Spalart-Allmaras turbulence and the $\gamma - Re_{\theta,t}$ transition models in MAEA-based shape optimization for both airfoils. The evaluation cost reduction varied between cases, with approximately 40% for NLF0416 and 30% for S8052. This difference can be attribute to the distinct CNN architectures employed, resulting in different inference costs.

- PUMA-CNN consistently outperformed PUMA-TM in terms of computational efficiency, even when the training and database creation costs are included. For the NLF0416 case, PUMA-CNN completed the optimization, in the case where no retraining was required, in approximately 425 TUs, 15% less than the PUMA-TM budget of 500 TUs. In the case were retraining was performed the optimization was still finished before 500 TUs. Similarly, for the S8052 case, both C_D-focused and TP_{SS}-focused optimizations (429 TUs and 469 TUs respectively) remained under the evaluation budget of the PUMA-TM optimizations.

- Both cases demonstrated a strong correlation between C_D and TP_{SS}, validating the approach of maximizing TP_{SS} as an effective surrogate for minimizing C_D across different airfoil types. For the NLF0416, optimizations with TP_{SS} as the objective consistently yielded equal or superior results in terms of both C_D and TP_{SS}. In the S8052 case, while PUMA-TM with C_D as the objective produced the best airfoil, PUMA-CNN with TP_{SS} as the objective achieved a

nearly equivalent solution in less time. These findings indicate that optimizing for TP_{SS} may not only effectively substitute for C_D minimization, but could also be a preferable objective, capable of discovering superior designs.

- By incorporating information from neighboring mesh elements, CNNs exhibited improved accuracy in predicting μ_t compared to Deep Neural Networks (DNNs). This advantage stems from the nature of CNNs that allow inter-element communication and therefore include information of neighboring mesh elements to make the μ_t field prediction.
- The CNN architectures employed in this research exhibited optimal performance when utilizing 3×3 convolutional kernels only in the initial layers. Extending the use of larger kernels throughout the entire network did not yield further improvements, suggesting that the observed benefits arise from capturing local spatial dependencies rather than an increase in representational power due to additional trainable parameters.

7.3 Future Work

Based on the findings of this Diploma Thesis, the following future works and improvements are proposed:

- **Extension to 3D Flows:** The application of Convolutional Neural Networks (CNNs) can be extended to three-dimensional (3D) flow problems. CNN architectures employed in video processing, which leverage 3D convolutions, enable inter-pixel and inter-frame communication. These 3D convolutions can be adapted to consider 3D neighborhoods of mesh elements for predicting the turbulent viscosity field (μ_t). The anticipated speed up is expected to be more pronounced in 3D flows due to their inherent computational complexity and higher per-evaluation cost.
- **Adaptation to Unstructured Meshes:** CNNs are currently limited by their requirement for a regular structure, typically found in structured meshes. To overcome this limitation, Convolutional Graph Neural Networks (CGNNs) can be employed. CGNNs perform convolutions using graph convolutional filters, which can be applied to any structure that can be represented as a graph [16]. This capability allows for the extension of the proposed approach to unstructured grids, expanding its applicability to a wider range of problems and mesh types.

Bibliography

[1] Abadi, M., et al.: Tensorflow: A system for large-scale machine learning. CoRR **abs/1605.08695** (2016)

[2] Afshar, Y., Bhatnagar, S., Pan, S., Duraisamy, K., Kaushik, S.: Prediction of aerodynamic flow fields using convolutional neural networks. Computational Mechanics **64**, 525 – 545 (2019)

[3] Asouti, V., Trompoukis, X., Kampolis, I., Giannakoglou, K.: Unsteady cfd computations using vertex-centered finite volumes for unstructured grids on graphics processing units. International Journal for Numerical Methods in Fluids **67**(2), 232–246 (2011)

[4] Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(12), 2481–2495 (2017)

[5] Baines, P., Majumdar, S., Mitsudera, H.: The mechanics of the tollmien-schlichting wave. Journal of Fluid Mechanics **312**, 107–124 (1996)

[6] Bar, N., Giryes, R.: Pruning at initialization - a sketching perspective. ArXiv **abs/2305.17559** (2023)

[7] Campbell, R., Lynde, M.: Natural laminar flow design for wings with moderate sweep. 34th AIAA Applied Aerodynamics Conference (2016)

[8] Darwin, C.: On the Origin of Species by Means of Natural Selection. Murray (1859)

[9] Deb, K., Agrawal, R.: Simulated binary crossover for continuous search space. Complex Syst. (1995)

[10] Dong, C., Loy, C., He, K., Tang, X.: Image super-resolution using deep convolutional networks. CoRR **abs/1501.00092** (2015)

[11] Du, Y., Gao, Z., Wang, C., Huang, Q.: Boundary-layer transition of advanced fighter wings at high-speed cruise conditions. Chinese Journal of Aeronautics **32**(4), 799–814 (2019)

[12] Duraisamy, K., Zhang, Z., Singh, A.: New approaches in turbulence and transition modeling using data-driven techniques. 53rd AIAA Aerospace Sciences Meeting (2015)

[13] Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Natural Computing Series, Springer (2003)

[14] Fogel, D.: Practical advantages of evolutionary computation. Applications of Soft Computing pp. 14–22 (1997)

[15] Friston, K.: Hierarchical models in the brain. PLOS Computational Biology **4**(11), 1–24 (2008)

[16] Gama, F., Marques, A., Leus, G., Ribeiro, A.: Convolutional graph neural networks. 2019 53rd Asilomar Conference on Signals, Systems, and Computers pp. 452–456 (2019)

[17] Giannakoglou, K.: Optimization Methods in Aerodynamics. NTUA (2006), `http://velos0.ltt.mech.ntua.gr/kgianna/optim/distr/Book-Optim-Giannakoglou.pdf`

[18] Giannakoglou, K.: EASY, The Evolutionary Algorithm SYstem, v2.0 (2008), `http://velos0.ltt.mech.ntua.gr/EASY/`

[19] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015)

[20] Jouppi, N., Kurian, G., Li, S., Ma, P., Nagarajan, R., Nai, L., Patil, N., Subramanian, S., Swing, A., Towles, B., Young, C., Zhou, X., Zhou, Z., Patterson, D.: Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. Proceedings of the 50th Annual International Symposium on Computer Architecture (2023)

[21] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings (2015)

[22] Kontou, M.: The Continuous Adjoint Method with Consistent Discretization Schemes for Transitional Flows and the Use of Deep Neural Networks in Shape Optimization in Fluid Mechanics. Ph.D. thesis, Laboratory of Thermal Turbomachines, N.T.U.A., Athens (2023)

[23] Kontou, M., Asouti, V., Giannakoglou, K.: Dnn surrogates for turbulence closure in cfd-based shape optimization. Applied Soft Computing **134**, 110013 (2023)

[24] Kordon, A., Kalos, A., Castillo, F., Jordaan, E., Smits, G., Kotanchek, M.: Competitive advantages of evolutionary computation for industrial applications. 2005 IEEE Congress on Evolutionary Computation **1**, 166–173 Vol.1 (2005)

[25] Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems (NIPS) **25**, 1097–1105 (2012)

[26] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

[27] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 3431–3440 (2015)

[28] Lyon, C., Broeren, A., Giguère, P., Gopalarathnam, A., Selig, M.: Summary of Low Speed Airfoil Data: Volume 3. SoarTech Publications (1997)

[29] Mayle, R.: The 1991 igti scholar lecture: The role of laminar-turbulent transition in gas turbine engines. Journal of Turbomachinery **113**(4), 509–536 (1991)

[30] McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**, 115–133 (1943)

[31] Morkovin, M.: On the Many Faces of Transition. Springer US (1969)

[32] Paszke, A., et al.: Pytorch: An imperative style, high-performance deep learning library. CoRR **abs/1912.01703** (2019)

[33] Piotrowski, M., Zingg, D.: Smooth local correlation-based transition model for the spalart–allmaras turbulence model. AIAA Journal **59**(2), 474–492 (2021)

[34] Riebesell, J., Bringuier, S.: Collection of standalone tikz images (2020)

[35] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 pp. 234–241 (2015)

[36] Ruder, S.: An overview of gradient descent optimization algorithms. CoRR **abs/1609.04747** (2016)

[37] Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)

[38] Russell, S., Norvig, P., Davis, E.: Artificial Intelligence: A Modern Approach. Prentice Hall (2010)

[39] Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. Artificial Neural Networks – ICANN 2010 pp. 92–101 (2010)

[40] Schlichting, H.: Boundary-Layer Theory. Springer (1979)

[41] Schook, R., De Lange, H., Van Steenhoven, A.: Heat transfer measurements in transitional boundary layers. International journal of heat and mass transfer **44**(5), 1019–1030 (2001)

[42] Smith, S., Kindermans, P., Le, Q.: Don't decay the learning rate, increase the batch size. CoRR **abs/1711.00489** (2017)

[43] Spalart, P., Allmaras, S.: A one-equation turbulence model for aerodynamic flows. 30th Aerospace Sciences Meeting and Exhibit (1992)

[44] Steinkraus, D., Buck, I., Simard, P.: Using gpus for machine learning algorithms. Eighth International Conference on Document Analysis and Recognition (ICDAR'05) pp. 1115–1120 Vol. 2 (2005)

[45] Stutz, D.: Collection of latex resources and examples. `https://github.com/davidstutz/latex-resources`

[46] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. CoRR **abs/1409.4842** (2014)

[47] Thompson, J., Soni, B., Weatherill, N.: Handbook of Grid Generation. CRC Press (1998)

[48] Vikhar, P.: Evolutionary algorithms: A critical review and its future prospects. 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC) pp. 261–265 (2016)

[49] Zhao, C., Sun, Q., Zhang, C., Tang, Y., Qian, F.: Monocular depth estimation based on deep learning: An overview. Science China Technological Sciences **63**(9), 1612–1627 (2020)

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Συνελικτικά Νευρωνικά Δίκτυα ως Υποκατάστατα Μοντέλων Τύρβης και Μετάβασης στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση Μορφής

Διπλωματική Εργασία - Εκτενής Περίληψη στα Ελληνικά

Αλέξανδρος Καραντώνης

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Εισαγωγή

Ο πρωταρχικός στόχος αυτής της Διπλωματικής Εργασίας είναι να διερευνηθεί η δυνατότητα χρήσης Συνελικτικών Νευρωνικών Δικτύων (ΣΝΔ) ως υποκατάστατων του μοντέλου τύρβης μιας εξίσωσης Spalart-Allmaras [43] και του μοντέλου μετάβασης δύο εξισώσεων $\gamma - Re_{\theta,t}$ [33], στην αεροδυναμική ανάλυση και βελτιστοποίηση μορφής.

Τα ΣΝΔ εκπαιδεύονται σε μια βάση δεδομένων με στόχο την ακριβή πρόβλεψη του πεδίου της τυρβώδους συνεκτικότητας, χρησιμοποιώντας ως εισόδους γεωμετρικά και ροϊκά πεδία από τις εξισώσεις μέσης ροής που λύνει το οικείο λογισμικό Υπολογιστικής Ρευστοδυναμικής PUMA [3]. Η αντικατάσταση των διαφορικών μοντέλων τύρβης και μετάβασης από τα ΣΝΔ περιορίζει τις προς επίλυση διαφορικές εξισώσεις στις τέσσερις εξισώσεις της μέσης ροής, οδηγώντας σε σημαντική μείωση του υπολογιστικού κόστους ανά ρευστοδυναμική ανάλυση.

Η μεθοδολογία εφαρμόζεται στη βελτιστοποίηση δύο μεμονωμένων αεροτομών: της NLF0416 και της S8052, με στόχο την ελαχιστοποίηση της οπισθέλκουσας C_D. Καθώς η ροή και στις δύο περιπτώσεις είναι μεταβατική, σημαντικό μέρος του C_D είναι η συνιστώσα τριβής $C_{D,f}$ που είναι υψηλότερη όταν η ροή είναι τυρβώδης. Για τον λόγο αυτό, θεωρώντας το σημείο μετάβασης από στρωτή σε τυρβώδη ροή εξίσου σημαντικό χαρακτηριστικό με το C_D, πραγματοποιούνται παράλληλες βελτιστοποιήσεις με στόχο τη μεγιστοποίηση της τετμημένης του σημείου μετάβασης στην πλευρά υποπίεσης TP_{SS}. Αυτή η προσέγγιση επιτρέπει τη μελέτη του κατά πόσον οι δύο διαφορετικοί στόχοι βελτιστοποίησης οδηγούν σε ίδιες ή παρόμοιες λύσεις, κάτι που αποτελεί έναν δευτερεύοντα στόχο της Διπλωματικής Εργασίας.

Τεχνητά Νευρωνικά Δίκτυα

Η Τεχνητή Νοημοσύνη (ΤΝ) και η Μηχανική Μάθηση (ΜΜ) αποτελούν σήμερα δύο από τους ταχύτερα αναπτυσσόμενους τομείς της επιστήμης και της τεχνολογίας, με εφαρμογές σε πολλούς κλάδους, συμπεριλαμβανομένης της μηχανικής. Η σημασία τους έγκειται στην ικανότητά τους να επεξεργάζονται τεράστιους όγκους δεδομένων, να αναγνωρίζουν πολύπλοκα πρότυπα και να παρέχουν λύσεις σε προβλήματα που παραδοσιακά απαιτούσαν ανθρώπινη νοημοσύνη. Η ΜΜ, ως υποπεδίο της ΤΝ, επικεντρώνεται στην ανάπτυξη αλγορίθμων και μοντέλων που επιτρέπουν στους υπολογιστές να μαθαίνουν από δεδομένα ώστε να εκτελούν συγκεκριμένες εργασίες, χωρίς να προγραμματίζονται ρητά για τον σκοπό αυτό.

Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) αποτελούν μια κατηγορία αλγορίθμων ΜΜ, εμπνευσμένη από τη δομή και λειτουργία των βιολογικών νευρωνικών δικτύων. Αποτελούνται από διασυνδεδεμένους κόμβους (νευρώνες) οργανωμένους σε επίπεδα: το επίπεδο εισόδου, ένα ή περισσότερα κρυφά επίπεδα και το επίπεδο εξόδου. Κάθε νευρώνας λαμβάνει σταθμισμένες τις τιμές όλων των νευρώνων του προηγούμενου επιπέδου, τις επεξεργάζεται μέσω μιας συνάρτησης ενεργοποίησης και παράγει μια έξοδο. Η εκπαίδευση των ΤΝΔ περιλαμβάνει την προσαρμογή των βαρών των συννάψεων μεταξύ των νευρώνων, ώστε το δίκτυο να μπορεί να αποδίδει στο ζητούμενο πρόβλημα.

Τα ΒΝΔ έχουν εφαρμοστεί ευρέως σε διάφορους τομείς, συμπεριλαμβανομένης της Υπολογιστικής Ρευστοδυναμικής [22], λόγω της ικανότητάς τους να μοντελοποιούν πολύπλοκες μη-γραμμικές σχέσεις.

Συνελικτικά Νευρωνικά Δίκτυα

Τα Συνελικτικά Νευρωνικά Δίκτυα (ΣΝΔ) αποτελούν μια εξειδικευμένη κατηγορία ΤΝΔ, σχεδιασμένη για την αποτελεσματική επεξεργασία δεδομένων με ισχυρές χωρικές εξαρτήσεις, όπως εικόνες ή πεδία ροής. Βασική δομική μονάδα των ΣΝΔ είναι το συνελικτικό επίπεδο, το οποίο εφαρμόζει την πράξη της συνέλιξης στα δεδομένα εισόδου. Η συνέλιξη πραγματοποιείται με τη χρήση πυρήνων που ολισθαίνουν πάνω στα δεδομένα εισόδου, εξάγοντας τοπικά χαρακτηριστικά. Αυτή η διαδικασία επιτρέπει στα ΣΝΔ να αναγνωρίζουν και να διατηρούν τη χωρική συσχέτιση των δεδομένων, καθώς κάθε νευρώνας συνδέεται μόνο με μια τοπική περιοχή της εισόδου. Η ιδιότητα αυτή καθιστά τα ΣΝΔ ιδιαίτερα αποτελεσματικά στην επεξεργασία εικόνων και την ανάλυση χωρικά συσχετισμένων δεδομένων.

Στην εργασία αυτή, τα ΣΝΔ χρησιμοποιούνται για την πρόβλεψη του πεδίου της τυρβώδους συνεκτικότητας. Βασική διαφορά με τις αρχιτεκτονικές ΤΝΔ είναι ότι το ΣΝΔ λαμβάνει ως εισόδους ολόκληρα τα πεδία ροής και καλείται μία φορά για να προβλέψει, ολόκληρο πάλι, το πεδίο μ_t. Αυτή η καθολική επεξεργασία των πεδίων επιτρέπει την ενσωμάτωση πληροφορίας ροής και γεωμετρίας από γειτονικά κελιά του υπολογιστικού πλέγματος, στοχεύοντας στη βελτίωση της ακρίβειας πρόβλεψης.

Εξελικτικοί Αλγόριθμοι

Οι Εξελικτικοί Αλγόριθμοι (ΕΑ) αποτελούν μια οικογένεια στοχαστικών μεθόδων βελτιστοποίησης, εμπνευσμένων από τις αρχές της βιολογικής εξέλιξης. Βασίζονται στην ιδέα της φυσικής επιλογής, όπου οι καλύτερες λύσεις "επιβιώνουν" και αναπαράγονται, ενώ οι λιγότερο κατάλληλες απορρίπτονται. Οι ΕΑ λειτουργούν με πληθυσμούς πιθανών λύσεων, οι οποίες εξελίσσονται μέσω επαναλαμβανόμενων κύκλων επιλογής, διασταύρωσης και μετάλλαξης. Ξεκινώντας με έναν αρχικό πληθυσμό, οι λύσεις αξιολογούνται βάσει μιας συνάρτησης-στόχου, και οι καλύτερες επιλέγονται για αναπαραγωγή, δημιουργώντας νέες γενιές λύσεων.

Τα βασικά πλεονεκτήματα των ΕΑ περιλαμβάνουν την απλότητα των μαθηματικών τους, τη δυνατότητα παράλληλης επεξεργασίας διαφόρων λύσεων, την προσαρμοστικότητά τους σε διαφορετικά προβλήματα και τη βεβαιότητα ότι, χάρη στο στοχαστικό τους χαρακτήρα, έστω και μετά από πολλές αξιολογήσεις αναμένεται να εντοπίσουν σίγουρα τη βέλτιστη λύση. Στο πλαίσιο της αεροδυναμικής βελτιστοποίησης, οι ΕΑ χρησιμοποιούνται αποτελεσματικά για την αναζήτηση βέλτιστων γεωμετριών που ικανοποιούν συγκεκριμένα κριτήρια απόδοσης.

Χρήση ΣΝΔ ως Υποκατάστατα Μοντέλων Τύρβης και Μετάβασης στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση της Μεμονωμένης Αεροτομής NLF0416

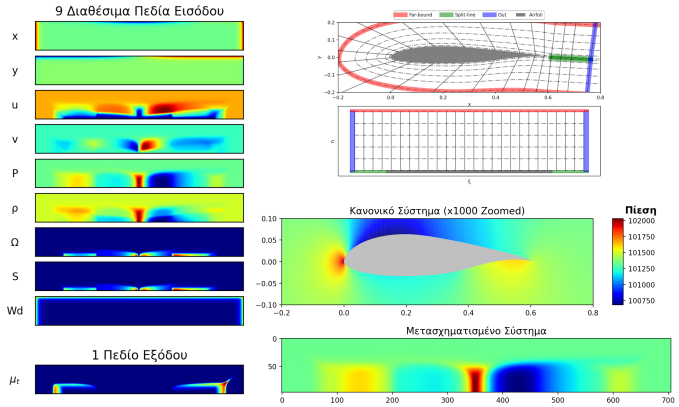
Η αεροτομή NLF0416 αναλύεται αρχικά υπό συνθήκες ροής: $M_\infty = 0.1$, $Re_\infty =$

4×10^6, $TI_\infty = 0.0015$ και σε γωνίες πρόσπτωσης $0.02°$ και $2°$. Αυτό επιτρέπει τη μελέτη της επίδρασης της γωνίας πρόσπτωσης στον συντελεστή τριβής C_f και στο φαινόμενο της μετάβασης. Παρατηρείται ότι με την αύξηση της γωνίας πρόσπτωσης, η μετάβαση επιταχύνεται στην πλευρά υποπίεσης και καθυστερεί στην πλευρά πίεσης. Η ανάλυση στη συνέχεια εστιάζεται στη αεροτομή σε γωνία πρόσπτωσης $0.02°$.

Η γεωμετρία της αεροτομής παραμετροποιείται χρησιμοποιώντας ογκομετρικές NURBS με 15 ελεύθερα σημεία ελέγχου, τα οποία μπορούν να κινηθούν οριζόντια και κατακόρυφα. Αυτό οδηγεί σε 30 μεταβλητές σχεδιασμού, οι οποίες αξιοποιούνται τόσο για τη δημιουργία της βάσης δεδομένων (ΒΔ) εκπαίδευσης του ΣΝΔ, όσο και ως μεταβλητές σχεδιασμού για τη βελτιστοποίηση που ακολουθεί.

Η δημιουργία της ΒΔ πραγματοποιείται μέσω της μεθόδου Latin Hypercube Sampling (LHS), παράγοντας 80 νέες γεωμετρίες. Αυτές επιλύονται από το λογισμικό PUMA, το οποίο συνδυάζει τις εξισώσεις μέσης ροής με το μοντέλο τύρβης SA και το μοντέλο μετάβασης $\gamma - Re_{\theta,t}$. Το ΣΝΔ εκπαιδεύεται να προβλέπει το πεδίο τυρβώδους συνεκτικότητας μ_t, χρησιμοποιώντας ως εισόδους γεωμετρικά δεδομένα (x, y, W_d) και ροϊκές ποσότητες $(\rho, p, u, v, \Omega, S)$ που προκύπτουν από την επίλυση των εξισώσεων μέσης ροής.

Αν και ο PUMA χειρίζεται το πλέγμα ως μη-δομημένο, το ΣΝΔ απαιτεί δομημένη τοπολογία για τη λειτουργία του. Το πλέγμα είναι δομημένο και αποτελείται από 97 ίσο-η και 705 ίσο-ξ πλεγματικές γραμμές. Συνεπώς, το ΣΝΔ λειτουργεί με πεδία που αντιστοιχούν στο ισοδύναμο ορθογωνικό πλέγμα στο υπολογιστικό σύστημα συντεταγμένων ξ-η, όπως αυτό προκύπτει από το φυσικό σύστημα x-y μέσω του γνωστού μετασχηματισμού (Σχήμα 7.1).



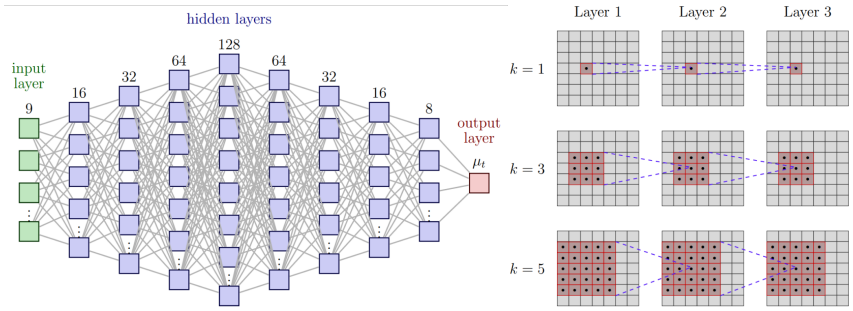
Σχήμα 7.1: *Τα πεδία εισόδου και εξόδου για το ΣΝΔ (αριστερά) στο υπολογιστικό σύστημα ξ-η, όπως προκύπτουν από το μετασχηματισμό (δεξιά).*

Το μέγεθος του πυρήνα συνέλιξης σε ένα ΣΝΔ αποτελεί καίρια παράμετρο, καθώς καθορίζει το εύρος της γειτονιάς που συνεισφέρει στην πρόβλεψη του μ_t για κάθε κελί, αλλά και τον αριθμό των εκπαιδεύσιμων βαρών του δικτύου. Για τη διερεύνηση της επίδρασης του μεγέθους του πυρήνα, πραγματοποιείται σύγκριση δικτύων με ταυτόσημη αρχιτεκτονική, εκπαιδευμένων στις ίδιες 80 γεωμετρίες και υπό τις ίδιες ρυθμίσεις

εκπαίδευσης. Η μοναδική διαφοροποίηση μεταξύ των δικτύων έγκειται στο μέγεθος του πυρήνα, με εξεταζόμενες διαστάσεις 1×1, 3×3, 5×5 και 7×7.

Ιδιαίτερη περίπτωση αποτελεί το ΣΝΔ που χρησιμοποιεί αποκλειστικά συνελίξεις με πυρήνα 1×1. Αυτή η διαμόρφωση δεν ενσωματώνει πληροφορίες από γειτονικά κελιά, λειτουργώντας κατ' ουσίαν όπως ένα ΒΝΔ ίδιας αρχιτεκτονικής που επεξεργάζεται τιμές για κάθε πλεγματικό στοιχείο μεμονωμένα και προβλέπει το μ_t. Για λόγους πληρότητας, το ισοδύναμο ΒΝΔ συμπεριλήφθηκε επίσης στη σύγκριση.

Το Σχήμα 7.2 απεικονίζει την αρχιτεκτονική των συγκρινόμενων δικτύων και παρουσιάζει πως το μέγεθος του πυρήνα επηρεάζει τη διαδικασία της συνέλιξης.

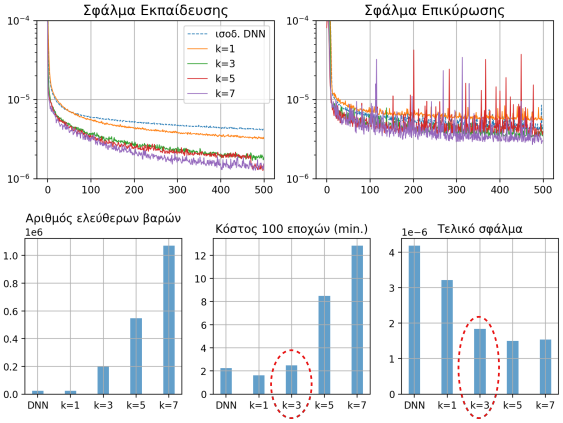


Σχήμα 7.2: *Αρχιτεκτονική δικτύων που συμμετείχαν στη σύγκριση (αριστερά). Πως λειτουργούν διαφορετικοί πυρήνες συνέλιξεις (δεξιά).*

Στο Σχήμα 7.3 φαίνεται η πορεία εκπαίδευσης των πέντε διαφορετικών δικτύων. Η ανάλυση αναδεικνύει ότι το δίκτυο με πυρήνα $k = 3$ παρουσιάζει σημαντικά μικρότερο σφάλμα συγκριτικά με αυτό που χρησιμοποιεί πυρήνα $k = 1$ και το αντίστοιχο DNN, γεγονός που υποδεικνύει τη σημασία της ενσωμάτωσης πληροφοριών από γειτονικά κελιά στην πρόβλεψη του μ_t. Ωστόσο, η χρήση μεγαλύτερων πυρήνων δεν επιφέρει ανάλογη μείωση του σφάλματος που να δικαιολογεί το αυξημένο υπολογιστικό κόστος κατά την εκπαίδευση. Συνεπώς, λαμβάνοντας υπόψη τη βέλτιστη ισορροπία μεταξύ ακρίβειας πρόβλεψης και αποδοτικότητας, επιλέχθηκε το μέγεθος πυρήνα $k = 3$ για την τελική αρχιτεκτονική του ΣΝΔ.

Το ΣΝΔ που τελικά επιλέχθηκε έχει την αρχιτεκτονική που περιγράφεται στον Πίνακα 7.1. Η εκπαίδευσή του πραγματοποιείται στη ΒΔ των 80 επιλυμένων γεωμετριών, χρησιμοποιώντας τον αλγόριθμο βελτιστοποίησης Adam με ρυθμό εκπαίδευσης 10^{-4} και ως συνάρτηση απώλειας το μέγιστο απόλυτο σφάλμα.

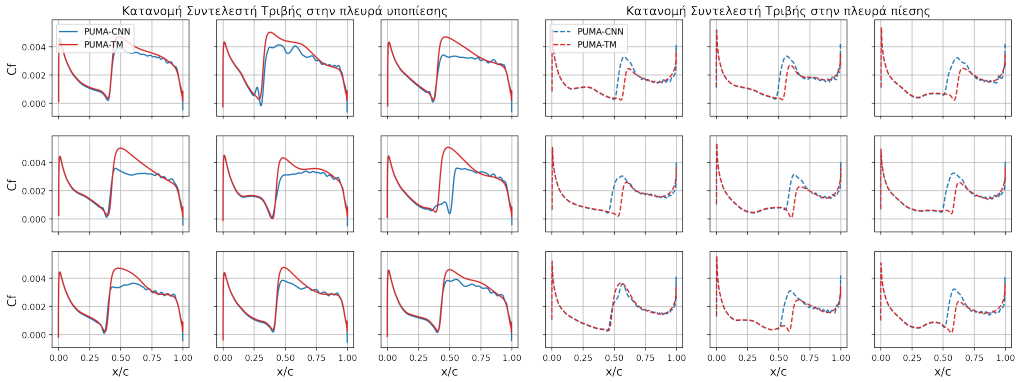
Η αξιολόγηση της απόδοσης του ΣΝΔ δεν περιορίζεται μόνο στο σφάλμα πρόβλεψης του μ_t. Κρισιμότερη είναι η επίδοσή του όταν ενσωματώνεται στο σύστημα PUMA-CNN, το οποίο επιλύει τις εξισώσεις μέσης ροής χρησιμοποιώντας το ΣΝΔ ως υποκατάστατο μοντέλο τύρβης και μετάβασης. Το Σχήμα 7.4 παρουσιάζει τις καμπύλες C_f για 9 άγνωστες γεωμετρίες, συγκρίνοντας τα αποτελέσματα του PUMA-TM και του PUMA-CNN. Στην πλευρά υποπίεσης, το σημείο μετάβασης προβλέπεται με ακρίβεια σε 8 από τις 9 περιπτώσεις. Ωστόσο, στην πλευρά πίεσης, το PUMA-CNN τείνει να προβλέπει πρωιμότερη μετάβαση στις περισσότερες περιπτώσεις.



Σχήμα 7.3: *Πορεία σφαλμάτων εκπαίδευσης, αριθμός ελεύθερων βαρών, κόστος εκπαίδευσης και τελικό σφάλμα.*

Πίνακας 7.1: *Αρχιτεκτονική ΣΝΔ*

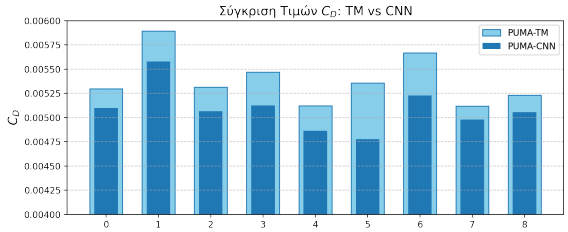
Είσοδοι	$x, y, \rho, u, v, \Omega, S, W_d$
Κρυφά επίπεδα	12
Κανάλια ανά επίπεδο	16 - 32 - 64 - 128 - 256 - 512 - 256 - 128 - 64 - 32 - 16 - 8
Πυρήνας Συνέλιξης	3×3
Συνάρτηση Ενεργοποίησης	ReLU



Σχήμα 7.4: *Κατανομές συντελεστή τριβής στις γεωμετρίες επικύρωσης για PUMA-TM (κόκκινο) και PUMA-CNN (μπλέ)*

Όσον αφορά το C_D το PUMA-CNN σταθερά υποεκτιμά την τιμή του όπως φαίνεται στο Σχήμα 7.5, με μέση τιμή σφάλματος 5.5%.

Αρχικά, διεξάγονται τρεις βελτιστοποιήσεις χρησιμοποιώντας το λογισμικό αξιολόγησης PUMA-TM, με τρεις διαφορετικούς στόχους: ελαχιστοποίηση του συντελεστή οπισθέλκουσας (min. C_D), μεγιστοποίηση του σημείου μετάβασης στην πλευρά υποπίεσης (max. TP_{SS}), και μεγιστοποίηση του σημείου μετάβασης στην πλευρά πίεσης (max.

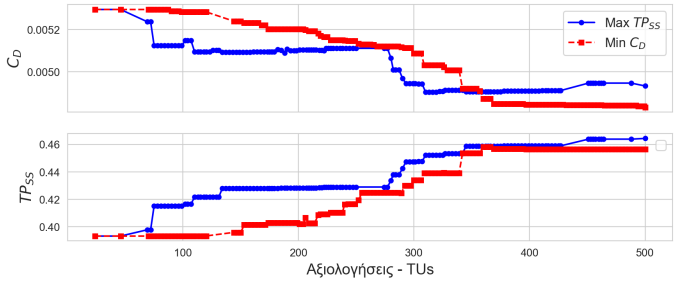


Σχήμα 7.5: *Προβλέψεις C_D για PUMA-TM και PUMA-CNN.*

TP_{PS}). Η τελευταία βελτιστοποίηση, αν και πραγματοποιήθηκε για λόγους πληρότητας, δεν επέφερε μείωση στο C_D και δεν θα αναλυθεί περαιτέρω στην περίληψη.

Όλες οι βελτιστοποιήσεις πραγματοποιούνται με εξελικτικούς αλγόριθμους υποβοηθούμενους από μεταμοντέλα μέσω του EASY. Οι περιορισμοί που τίθενται είναι: διατήρηση ή αύξηση του συντελεστή άνωσης (C_L), διατήρηση σχεδόν σταθερού συντελεστή ροπής (C_M), και διατήρηση του εμβαδού (A) τουλάχιστον στο 90% της αρχικής τιμής.

Το Σχήμα 7.6 απεικονίζει την πορεία των δύο κύριων βελτιστοποιήσεων. Η μπλε γραμμή αντιπροσωπεύει τη βελτιστοποίηση με στόχο max. TP_{SS}, ενώ η κόκκινη αυτή με στόχο min. C_D. Το άνω διάγραμμα δείχνει την εξέλιξη του C_D, ενώ το κάτω την εξέλιξη του TP_{SS}. Παρατηρείται ότι και οι δύο ποσότητες μεταβάλλονται με παρόμοιο τρόπο, ενισχύοντας την υπόθεση ότι η καθυστέρηση της μετάβασης μπορεί να οδηγήσει σε μείωση της οπισθέλκουσας. Συγκεκριμένα, η βελτιστοποίηση max. TP_{SS} επέφερε μείωση του C_D κατά 6.6% και καθυστέρηση της μετάβασης στην πλευρά υποπίεσης κατά 18.1%. Αντίστοιχα, η βελτιστοποίηση min. C_D πέτυχε μείωση του C_D κατά 8.7% και καθυστέρηση της μετάβασης στην πλευρά υποπίεσης κατά 16.1%.

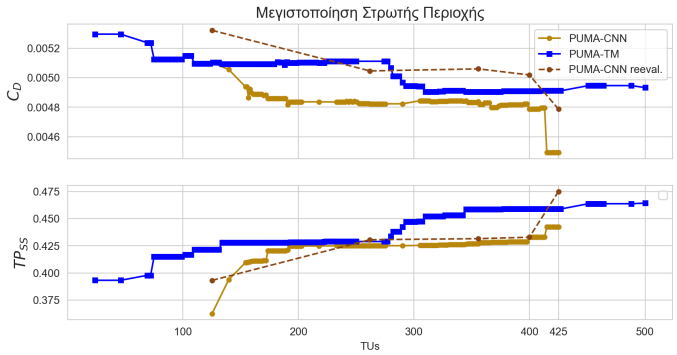


Σχήμα 7.6: *Πορεία βελτιστοποίησης με στόχο το C_D (κόκκινο) και με στόχο το TP_{SS} (μπλε). Εξέλιξη C_D πάνω και TP_{SS} κάτω για τις δύο βελτιστοποιήσεις.*

Οι ίδιες βελτιστοποιήσεις επαναλαμβάνονται χρησιμοποιώντας το λογισμικό PUMA-CNN, το οποίο αξιοποιεί ΣΝΔ αντί των συμβατικών μοντέλων τύρβης και μετάβασης. Μία αξιολόγηση με το PUMA-CNN απαιτεί 0.63 χρονικές μονάδες (όπου 1 χρονική μονάδα αντιστοιχεί σε μία αξιολόγηση του PUMA-TM). Ωστόσο, σε κάθε βελτιστοποίηση με το PUMA-CNN προστίθεται ένα αρχικό κόστος 112 χρονικών μονάδων, για την εκπαίδευση του ΣΝΔ και τη δημιουργία της βάσης δεδομένων.

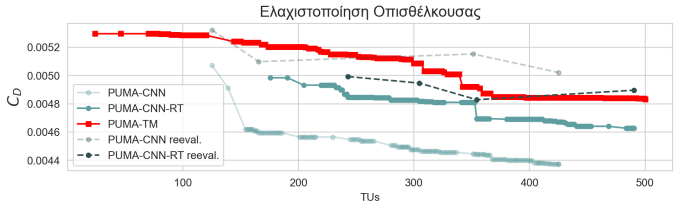
Το Σχήμα 7.7 απεικονίζει τη βελτιστοποίηση με στόχο τη μεγιστοποίηση του TP_{SS}.

Παρατηρείται ότι το PUMA-CNN επιτυγχάνει καλύτερη λύση σε σύγκριση με το PUMA-TM, τόσο ως προς το TP_{SS} όσο και ως προς το C_D. Οι τακτικές επαναξιολογήσεις (απεικονίζονται με διακεκομμένη γραμμή) επιβεβαιώνουν την εγκυρότητα των αποτελεσμάτων του PUMA-CNN. Τελικά, επιτυγχάνεται μείωση του C_D κατά 9.6% σε 425 μονάδες χρόνου, που είναι 15% ταχύτερα σε σύγκριση με το PUMA-TM.



Σχήμα 7.7: *Μεγιστοποίηση TP_{SS} με PUMA-CNN και PUMA-TM.*

Το Σχήμα 7.8 παρουσιάζει τη βελτιστοποίηση με στόχο την ελαχιστοποίηση του C_D. Κατά την πρώτη επαναξιολόγηση, διαπιστώθηκε αύξηση του σφάλματος του PUMA-CNN στην πρόβλεψη του C_D, οδηγώντας σε επανεκπαίδευση του CNN. Το σχήμα απεικονίζει δύο σενάρια βελτιστοποίησης με το PUMA-CNN: με επανεκπαίδευση (σκούρες αποχρώσεις) και χωρίς (ανοιχτές αποχρώσεις). Ακόμη και χωρίς επανεκπαίδευση, επιτυγχάνεται μείωση του C_D, ωστόσο, το σενάριο με επανεκπαίδευση αποδίδει καλύτερα αποτελέσματα, καθώς το σφάλμα μειώνεται σημαντικά και η βελτιστοποίηση οδηγεί σε λύση εφάμιλλη αυτής του PUMA-TM σε περίπου ίδιο χρόνο.



Σχήμα 7.8: *Ελαχιστοποίηση C_D με PUMA-CNN και PUMA-TM.*

Δεδομένης της στοχαστικότητας των εξελικτικών αλγορίθμων όλα τα πειράματα επαναλαμβάνονται για άλλες δύο (τρεις σύνολο) διαφορετικές αρχικοποιήσεις της γεννήτριας τυχαίων αριθμών. Τελικά επαληθεύεται πως οι βελτιστοποιήσεις με στόχο το TP_{SS} οδηγούν συστηματικά σε εφάμιλλα ή και καλύτερα αποτελέσματα όσον αφορά το C_D, συγκριτικά με τις βελτιστοποιήσεις που στοχεύουν απευθείας στο C_D. Επιπλέον, το PUMA-CNN αποδεικνύεται αξιόπιστο ως λογισμικό αξιολόγησης. Σε όλες τις περιπτώσεις, επιτυγχάνει ικανοποιητική μείωση του C_D και αύξηση του TP_{SS}, με τα αποτελέσματα να επικυρώνονται μέσω επαναξιολογήσεων από τον PUMA-TM.

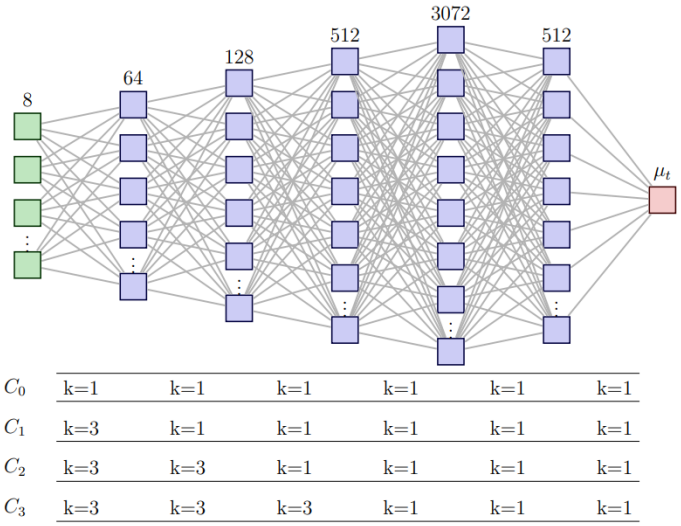
Χρήση ΣΝΔ ως Υποκατάστατα Μοντέλων Τύρβης και Μετάβασης

στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση της Μεμονωμένης Αεροτομής S8052

Η αεροτομή S8052 μελετάται στις ακόλουθες συνθήκες ροής: $M_\infty = 0.1$, $Re_\infty = 2.3 \times 10^6$, $TI_\infty = 0.005$ και σε γωνία πρόσπτωσης 2°.

Η γεωμετρία της αεροτομής παραμετροποιείται χρησιμοποιώντας ογκομετρικές NURBS με 12 ελεύθερα σημεία ελέγχου, τα οποία μπορούν να μετακινηθούν οριζόντια και κατακόρυφα. Αυτό οδηγεί σε 24 μεταβλητές σχεδιασμού. Με τη μέθοδο δειγματοληψίας Latin Hypercube Sampling (LHS) δημιουργούνται 50 γεωμετρίες, οι οποίες θα χρησιμοποιηθούν για την εκπαίδευση του ΣΝΔ. Οι διαθέσιμες είσοδοι για το ΣΝΔ και η μορφή που δέχεται (ορθογωνικά πεδία στο μετασχηματισμένο σύστημα ξ-η) παραμένουν ίδιες με την περίπτωση της αεροτομής NLF0416.

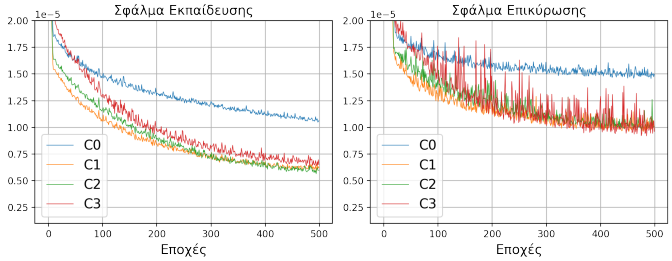
Με βάση τα συμπεράσματα από την προηγούμενη μελέτη, όπου διαπιστώθηκε ότι το βέλτιστο μέγεθος πυρήνα είναι 3, στην παρούσα ανάλυση διερευνάται κατά πόσον είναι απαραίτητο όλα τα συνελικτικά επίπεδα να πραγματοποιούν συνελίξεις με πυρήνες 3×3. Εξετάζονται τέσσερις αρχιτεκτονικές (C_0, C_1, C_2, C_3) όπως φαίνονται στο Σχήμα 7.9, ξεκινώντας από ένα δίκτυο που χρησιμοποιεί αποκλειστικά πυρήνες 1×1 (C_0) και σταδιακά αυξάνοντας τον αριθμό των επιπέδων με πυρήνες 3×3. Τα ΣΝΔ χρησιμοποιούν 8 πεδία εισόδου, τα : $x, y, u, v, p, S, \Omega, W_d$, για να παράγουν το πεδίο μ_t.



Σχήμα 7.9: *ΣΝΔ προς σύγκριση. Το C_0 χρησιμοποιεί μόνο 1×1 πυρήνες, ενώ τα C_1, C_2, C_3 προοδευτικά εισάγουν πυρήνες 3×3.*

Τα αποτελέσματα δείχνουν ότι η χρήση μη μοναδιαίου πυρήνα έχει σημαντική θετική επίδραση ακόμη και από το πρώτο επίπεδο. Όπως φαίνεται από την πορεία της εκπαίδευσης στο Σχήμα 7.10, τα δίκτυα που περιέχουν τουλάχιστον ένα επίπεδο με πυρήνα 3×3 παρουσιάζουν καλύτερη απόδοση τόσο στις γεωμετρίες εκπαίδευσης όσο και στις γεωμετρίες επικύρωσης. Η σημαντική βελτίωση ακόμη και για το C_1 υπ-

οδεικνύει ότι αυτή προέρχεται κυρίως από την εγγενή φυσική της χρήσης γειτονικών πληροφοριών.



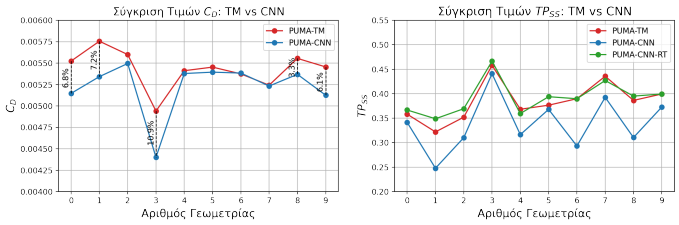
Σχήμα 7.10: *Πορεία εκπαίδευσης των ΣΝΔ.*

Από τα 4 δίκτυα, για τη συνέχεια επιλέχθηκε το C_3 καθώς εμφάνισε το μικρότερο σφάλμα επικύρωσης, υποδεικνύοντας την καλύτερη ικανότητα γενίκευσης σε νέα δεδομένα. Αυτό συνεργάζεται με τον PUMA (λογισμικό αξιολόγησης PUMA-CNN) και πραγματοποιεί κάθε αξιολόγηση σε 0.7 χρονικές μονάδες.

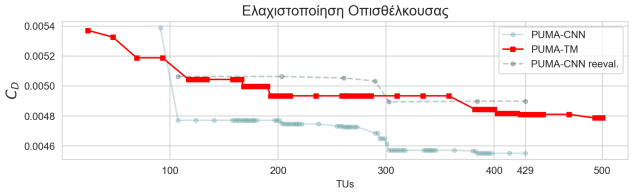
Στη συνέχεια πραγματοποιούνται οι 4 βελτιστοποιήσεις για τους δύο στόχους: min. C_D και max. TP_{SS}, και τα δύο λογισμικά αξιολόγησης: το PUMA-CNN και το PUMA-TM. Το σύστημα PUMA-CNN εξετάζεται ως προς την πρόβλεψη των C_D και TP_{SS} πάνω σε 10 γεωμετρίες επικύρωσης, όπως φαίνεται στο Σχήμα 7.11. Η απόδοση όσον αφορά το C_D είναι εξαιρετική (μέσο σφάλμα 3.8%) οπότε η βελτιστοποίηση με στόχο το C_D μπορεί να πραγματοποιηθεί. Επειδή όμως η απόδοση στην πρόβλεψη του TP_{SS} δεν είναι εξίσου καλή η βάση δεδομένων εμπλουτίζεται με 30 επιπλέον γεωμετρίες και το δίκτυο επανεκπαιδεύεται με κόστος 40 χρονικών μονάδων, οπότε και επιτυγχάνει εξίσου καλή απόδοση (μέσο σφάλμα 3%). Η βελτίωση αυτή είναι κρίσιμη για την αξιοπιστία της βελτιστοποίησης με στόχο τη μεγιστοποίηση της στρωτής περιοχής.

Στο Σχήμα 7.12 παρουσιάζεται η βελτιστοποίηση με στόχο min. C_D για τα δύο λογισμικά αξιολόγησης. Και τα δύο λογισμικά επιτυγχάνουν σημαντική μείωση του C_D, με το PUMA-CNN να τερματίζει σε 429 χρονικές μονάδες, 14% γρηγορότερα από το PUMA-TM, και να βρίσκει σχεδόν ισάξια λύση.

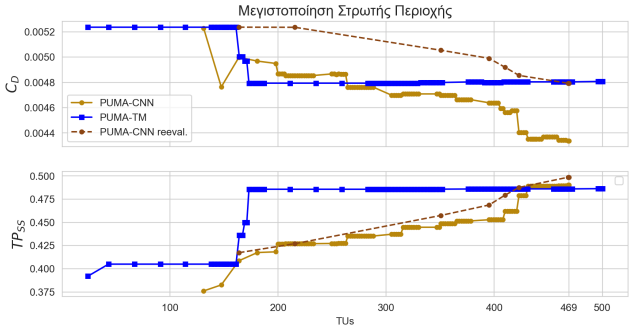
Στο Σχήμα 7.13 παρουσιάζεται η βελτιστοποίηση με στόχο max. TP_{SS} για τα δύο λογισμικά αξιολόγησης. Τα δύο λογισμικά καταλήγουν σχεδόν στην ίδια λύση, τόσο ως προς το C_D και ως προς το TP_{SS}. Το PUMA-CNN τελειώνει σε 469 χρονικές μονάδες,



Σχήμα 7.11: *Σφάλματα C_D και TP_{SS} στις 10 γεωμετρίες επικύρωσης*



Σχήμα 7.12: *Πορεία βελτιστοποιήσεων με στόχο min. C_D για τα δύο λογισμικά αξιολόγησης.*



Σχήμα 7.13: *Πορεία βελτιστοποιήσεων με στόχο max. TP_{SS} για τα δύο λογισμικά αξιολόγησης.*

6% γρηγορότερα από το PUMA-TM. Σε σχέση με τις λύσεις των βελτιστοποιήσεων με στόχο min. C_D, οι λύσεις με στόχο max. TP_{SS} δεν υστερούν ως προς το C_D. Αυτό υποδεικνύει ότι και εδώ, η μεγιστοποίηση της στρωτής περιοχής φαίνεται να είναι ένας αποτελεσματικός εναλλακτικός στόχος για τη μείωση της οπισθέλκουσας.

Συμπεράσματα

Τα ΣΝΔ απέδειξαν την ικανότητά τους να υποκαθιστούν αποτελεσματικά τα μοντέλα τύρβης SA και μετάβασης $\gamma - Re_{\theta,t}$ στη βελτιστοποίηση μορφής αεροτομών. Η μείωση του κόστους αξιολόγησης κυμάνθηκε μεταξύ 30% και 40%, ανάλογα με την περίπτωση και την αρχιτεκτονική ΣΝΔ που χρησιμοποιήθηκε. Το PUMA-CNN ξεπέρασε σταθερά το PUMA-TM σε υπολογιστική αποδοτικότητα, ακόμη και όταν συμπεριλήφθηκαν τα κόστη εκπαίδευσης, ολοκληρώνοντας τις βελτιστοποιήσεις σε λιγότερο χρόνο.

Και στις δύο περιπτώσεις αεροτομών παρατηρήθηκε ισχυρή συσχέτιση μεταξύ C_D και TP_{SS}, επικυρώνοντας την προσέγγιση της μεγιστοποίησης του TP_{SS} ως υποκατάστατο για την ελαχιστοποίηση του C_D. Σε αρκετές περιπτώσεις, η βελτιστοποίηση με στόχο το TP_{SS} οδήγησε σε ισοδύναμα ή ακόμη και καλύτερα αποτελέσματα σε σύγκριση με τη βελτιστοποίηση με στόχο το C_D, υποδεικνύοντας ότι η μεγιστοποίηση της στρωτής περιοχής ενδεχομένως είναι ένας προτιμότερος στόχος βελτιστοποίησης.

Τα ΣΝΔ έδειξαν βελτιωμένη ακρίβεια στην πρόβλεψη του μ_t σε σύγκριση με τα ΒΝΔ, λόγω της ικανότητάς τους να ενσωματώνουν πληροφορίες από γειτονικά στοιχεία του πλέγματος. Η βέλτιστη απόδοση επιτεύχθηκε με τη χρήση πυρήνων συνέλιξης 3×3 μόνο στα αρχικά επίπεδα του δικτύου, υποδηλώνοντας ότι τα οφέλη προέρχονται

κυρίως από την καταγραφή τοπικών χωρικών εξαρτήσεων παρά από την αύξηση της αναπαραστατικής ισχύος λόγω επιπλέον εκπαιδεύσιμων παραμέτρων.