National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Lab. of Thermal Turbomachines
Parallel CFD & Optimization Unit

Deep Neural Networks as Turbulence Model Surrogates in Aerodynamic Analysis and Optimization

Diploma Thesis

Ioannis Kordos

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2023

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to two individuals who were instrumental in the completion of my Diploma Thesis. First and foremost, I am deeply thankful for the guidance provided by Professor Kyriakos C. Giannakoglou. His expert knowledge, problem-solving skills, and unwavering support have been invaluable throughout my academic journey. It has been an honor to work under his supervision and I am grateful for the enriching experience he has provided.

Secondly, I would like to extend my heartfelt appreciation to Marina Kontou, whose assistance and insights have been invaluable in the successful completion of my thesis. Her expertise in the field of deep neural networks, coupled with her willingness to share her knowledge and offer constructive feedback, have been essential to my progress. I am incredibly grateful for her guidance and support and I will always cherish the knowledge and skills she imparted to me.

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Lab. of thermal Turbomachines
Parallel CFD & Optimization Unit

Deep Neural Networks as Turbulence Model Surrogates in Aerodynamic Analysis and Optimization

Diploma Thesis

Ioannis Kordos

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2023

The target of this Diploma Thesis is the use of DNNs as turbulence Model surrogates in aerodynamic analysis and optimization, in order to reduce the overall computational cost of this process.

DNNs are trained on a dataset with the aim of reproducing the turbulence viscosity field as accurately as possible. To train the DNN, the geometries are parameterized through volumetric morphing using NURBS lattices and using the Latin Hypercube Sampling (LHS) a set of new differentiated geometries is generated. The flow is computed by the RANS equations coupled with the turbulence model within PUMA software (PUMA-TM). In addition, particular attention is paid to the specification of the DNN architecture and the way of training the network.

The optimization cases used are the optimization of the airfoil shape NACA4318 with target to minimize the drag coefficient with constant lift coefficient, and the turbine blade LS89 with target to minimize the pressure total losses, keeping the flow exit angle close to its baseline value. The PUMA-TM model for the first case is the RANS equations coupled with the SA turbulence model, while for the second case it is the RANS equations coupled with the k-ω turbulence model assisted by the $\gamma - \tilde{Re}_{\theta t}$ transition model. The alternative models are the solver of the RANS equations coupled with the DNN of each case (PUMA-DNN). The MAEA (shape) optimizations are performed for both models as evaluation software. In addition, for the second case, the architecture of the DNN and the selection of appropriate inputs are optimized with a MAEA-based optimization and techniques like early stopping and dropout are applied during the training process. All optimizations with MAEA are implemented in the EASY software.

During the shape optimizations where the PUMA-DNN is the evaluation software, the best solutions of each generation are re-evaluated with the PUMA-TM and if necessary the DNN is retrained. At the end of the optimization, the results of

MAEA relying on PUMA-DNN and PUMA-TM evaluation software are compared in terms of their quality and the computational cost of the optimization.

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

Τα Βαθιά Νευρωνικά Δίκτυα ως Υποκατάστατα Μοντέλων Τύρβης στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση

Διπλωματική εργασία

Ιωάννης Κορδός

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2023

Στόχος της Διπλωματικής αυτής Εργασίας, είναι η χρήση Βαθιών Νευρωνικών Δικτύων (ΒΝΔ), ως υποκατάστατα μοντέλων τύρβης στην αεροδυναμική ανάλυση και βελτιστοποίηση, με στόχο τη μείωση του συνολικού υπολογιστικού κόστους της διαδικασίας αυτής.

Τα ΒΝΔ εκπαιδεύονται σε ένα σύνολο δεδομένων με στόχο να αναπαράγουν όσο το δυνατόν ακριβέστερα το πεδίο τυρβώδους συνεκτικότητας. Για να εκπαιδευτεί το ΒΝΔ, η αρχική γεωμετρία παραμετρικοποιείται μέσω ογκομετρικής μορφοποίησης με τεχνικές *NURBS* και με χρήση της τεχνικής δειγματοληψίας *Latin Hyperciube Sampling* (*LHS*), δημιουργείται ένα σύνολο νέων διαφοροποιημένων γεωμετριών. Οι μοντελοποιήσεις της ροής γίνονται με τον επιλύτη των *RANS* εξισώσεων και των μοντέλων τύρβης (ακριβές μοντέλο) στο λογισμικό *PUMA*. Επιπλέον, ιδιαίτερη σημασία δίνεται στον προσδιορισμό της βέλτιστης αρχιτεκτονικής του δικτύου καθώς και στον τρόπο εκπαίδευσης του.

Οι περιπτώσεις βελτιστοποίησης που χρησιμοποιούνται είναι η βελτιστοποίηση του σχήματος της αεροτομής $NACA4318$ με στόχο την ελαχιστοποίηση του συντελεστή αντίστασης και σταθερό συντελεστή άνωσης, και του πτερυγίου συμπιεστή $LS89$ με στόχο την ελαχιστοποίηση των απωλειών πίεσης και σταθερή γωνία εξόδου της ροής. Το ακριβές μοντέλο για την πρώτη περίπτωση, είναι ο επιλύτης των *RANS* με το μοντέλο τύρβης $Spalart-Allmaras$, ενώ για την δεύτερη είναι ο επιλύτης των *RANS* με το μοντέλο τύρβης k-ω και μετάβασης $\gamma-\widetilde{Re}_{\theta t}$. Τα εναλλακτικά μοντέλα, είναι ο επιλύτης των *RANS* εξισώσεων με το αντίστοιχο ΒΝΔ της κάθε περίπτωσης. Οι βελτιστοποιήσεις σχήματος μέσω εξελικτικού αλγορίθμου γίνονται και με τα δύο μοντέλα ως λογισμικό αξιολόγησης. Επιπλέον, η αρχιτεκτονική του ΒΝΔ και η επιλογή των κατάλληλων εισόδων της δεύτερης περίπτωσης βελτιστοποιούνται μέσω ενός εξελικτικού αλγορίθμου και, εν συνεχεία, εφαρμόζονται οι τεχνικές *early stopping* και

dropout κατα τη διαδικασία εκπαίδευσης. Όλες οι βελτιστοποιήσεις μέσω εξελικτικού αλγόριθμου υλοποιούνται στο λογισμικό *EASY*.

Κατα τη διάρκεια των βελτιστοποιήσεων σχήματος όπου τα εναλλακτικά μοντέλα αποτελούν το λογισμικό αξιολόγησης, οι βέλτιστες λύσεις κάθε γενιάς, επαναξιολογούνται με το ακριβές μοντέλο και όπου κρίνεται απαραίτητο το ΒΝΔ επανεκπαιδεύεται. Στο πέρας τον βελτιστοποιήσεων, τα αποτελέσματα των βελτιστοποιήσεων με τα δύο μοντέλα συγκρίνονται ως προς την ποιότητα τους και τον υπολογιστικό χρόνο της βελτιστοποίησης.

Nomenclature

NTUA	National Technical University of Athens
PCopt	Parallel CFD & Optimization unit
AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CFD	Computational Fluid Dynamics
EA	Evolutionary Algorithm
MAEA	Metamodel Assisted Evolutionary Algorithm
MAE	Mean Absolute Error
MSE	Mean Square Error
NURBS	Non-Uniform Rational B-Splines
LHS	Latin Hypercube Sampling
CD	Drag Coefficient
CL	Lift Coefficient
H	Heat Transfer Coefficient
$\mathrm{P}_{t,losses}$	Total Pressure Losses

Contents

Chapter 1

Artificial Intelligent - AI

1.1 Introduction to Artificial Intelligent - AI

There are several definitions for Artificial Intelligence (AI) and one of the most detailed and complete definition, is made by the online publication Quartz [1]:
"Artificial Intelligence is software or a computer program with a mechanism to learn. It then uses that knowledge to make a decision in a new situation, as humans do. The researchers building this software try to write code that can read images, text, video or audio and learn something from it. Once a machine has learned, that knowledge can be put to use elsewhere".

In simple terms, AI refers to a machine's ability to utilize algorithms that can learn data patterns and make decisions like humans. These algorithms can process large amounts of data and make decisions based on statistical models, reducing the number of errors compared to humans who are prone to cognitive biases [2].

In the late 1990s and early 2000s, researchers in AI faced an obstacle in their quest to develop machines with intelligence. They realized that a thorough understanding of intelligence was crucial, but there remained no clear definition of it. This issue slowed the progress of AI. To overcome this, researchers changed their approach and focused on creating a system that could grow its intelligence [3], rather than trying to build intelligence from scratch. This approach established the new subfield of AI called Machine Learning (ML).

1.2 Machine Learning

ML is a subfield of AI that enables a machine to improve its performance on tasks automatically through experience. The algorithms used in ML examine data, learn from it, and then make predictions or classifications about new data. This process allows the machine to continually evolve its understanding and decision-making skills, without the need for specific programming.

Types of machine learning [4] (see Figure 1.1)

- **Supervised**
  Supervised ML in the context of Computational Fluid Dynamics (CFD) refers to the process of training a model on a dataset of CFD simulations, so that the model can learn to predict the output (e.g. pressure, velocity, temperature) given a set of input parameters (e.g. geometry, boundary conditions, fluid properties). This type of ML is used in this Diploma Thesis for predicting the turbulence viscosity field (output) with geometrical and flow data as inputs.

- **Unsupervised**
  Unsupervised ML in the context of CFD refers to the use of ML algorithms that do not require responses for training. In this context, the algorithms analyze the fluid dynamics simulations to identify patterns and relationships in the data, without the need for prior knowledge or the responses. The goal is to uncover hidden structure and correlations in the fluid flow patterns and to gain insights into the underlying physical processes that govern fluid dynamics.

- **Reinforcement**
  Reinforcement ML in the context of CFD refers to the use of ML algorithms that use trial-and-error learning to make decisions about how to control a fluid system. The goal of reinforcement learning in CFD is to optimize fluid behavior by learning from interactions with the fluid environment and making decisions based on the consequences of previous actions. Reinforcement learning algorithms typically involve an agent that makes decisions in response to the state of the fluid system, and a reward signal that provides feedback about the quality of the decisions.

In the early 2010s, several developments shaped the future of ML. With advancing technology, more computing power became accessible, allowing for the evaluation of more complex ML models. Additionally, declining costs for data processing and storage meant more data became available for ML systems. At the same time, advancements in our understanding of the human brain paved the way for researchers to develop new ML algorithms based on this knowledge. These advancements led to the rise of a new area of ML known as Deep Learning.

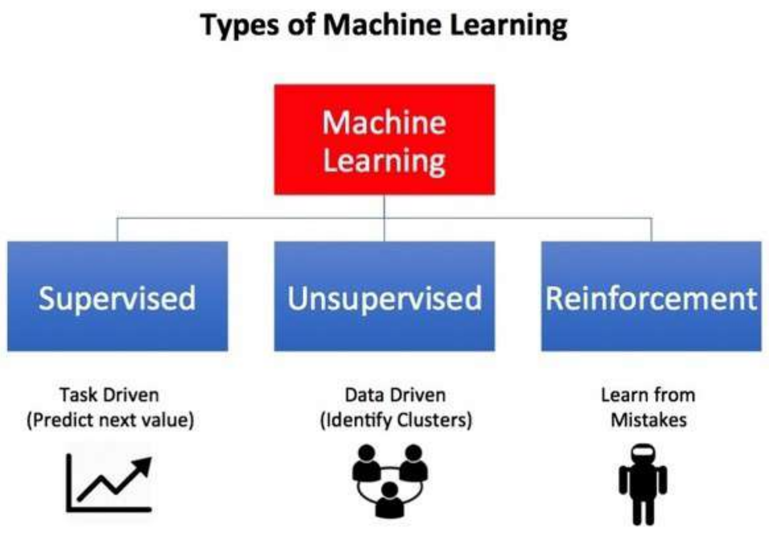

Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven (Predict next value)
Data Driven (Identify Clusters)
Learn from Mistakes

Figure 1.1: *Types of machine learning. From: [5]*

1.3 Deep Learning

Deep learning (DL), a subfield of ML, is an area of AI that is rapidly growing. It is utilized to tackle complex problems that were once deemed impossible to solve and involves working with vast amounts of data. The key to DL lies in the use of artificial neural networks, that are trained on large amounts of data to learn patterns and make decisions or predictions based on that data. Despite its potential for solving challenging issues, implementing DL requires access to a substantial dataset and significant computational resources. These requirements make it necessary for organizations to invest in both hardware and data infrastructure to support their DL initiatives.

The way that AI, ML, and DL relate to each other, is showing in Venn diagram of Figure 1.2.

1.4 AI in CFD

Computational Fluid Dynamics (CFD) is a field that combines mathematics, fluid mechanics, and computer science, using numerical methods to solve the equations of fluid mechanics to simulate fluid flow and heat/mass transfer. Despite its effectiveness, CFD faces challenges, including increasing complexity of physical models, difficulty in generating an appropriate mesh for complex geometries, and high post-processing time for complex flow data. To address these issues, many researchers are exploring the use of AI algorithms within CFD.

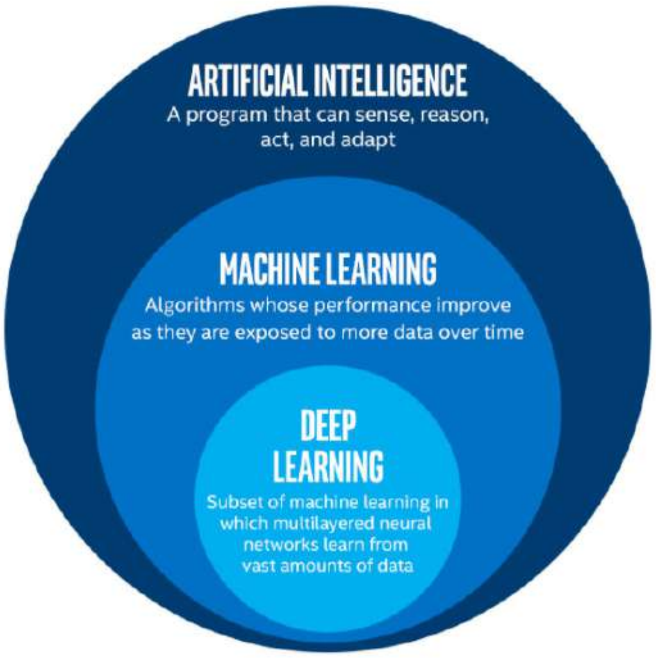

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 1.2: *Artificial Intelligence, Machine Learning, and Deep Learning. From: [6].*

AI algorithms have become increasingly popular for solving problems in the field of CFD, due to their accuracy and efficiency. There are three main AI-CFD coupling models : data-driven, physical, and hybrid. The data-driven model, which is used in this Diploma Thesis, relies on a large database to directly input CFD data into a ML model to make parameters predictions. This method is fast and has low error but can make it difficult to understand the underlying physical process. An example of data-driven model, is a study by Kontou M et al. (2022) [1], where a DNN-based surrogate for turbulence and transition closure of the Reynolds-Averaged Navier-Stokes (RANS) equations was presented. The DNN hyper-parameters were optimized, and the model was trained using geometrical and flow data to predict turbulence viscosity field. Then, the RANS coupled with the DNN and it was used as evaluation software in a MAEA-based (shape) optimization, reducing the optimization turnaround time.

The physical model in AI-CFD coupling utilizes CFD data to enhance the accuracy of fluid mechanics models. This approach requires fewer data than the data-driven model, as it takes into account physical mechanisms. In a study by Tracey B. et al. (2015) [8], a neural network was developed based on supervised learning algorithms to optimize the source term of Spalart-Allmaras (SA) models. The authors obtained data by running the SA model solver for a variety of flow scenarios, and then trained the neural network to replace the source term. The resulting model was integrated into the SA solver, and the coupled solver was able to predict a range of flow situations.

The hybrid model combines both the data-driven and physical models, resulting in high precision. Although the underlying relationship of data remains challenging to explain, this approach provides a new means to examine mechanisms. A study by Maulik et al [9], the authors explored the use of AI algorithms in Large Eddy Simulation (LES) and the classification of spatio-temporally dynamic turbulence model. The grid nodes were classified, and for a hybrid closure, the turbulence model was combined with the predictions of a neural network that used conditional probability.The results showed that the proposed methods were robust and stable, and could potentially be used to take advantage of the strengths of different models for predicting complex flow phenomena.

When it comes to combining CFD with AI models, there are several fundamental questions that need answers. Can machine learning be successful with CFD data? Will the algorithm offer stability comparable to a manual model? Will the learned model accurately predict results for unseen flows? These questions are explored through this Diploma Thesis.

1.5 Thesis Outline

Following the introduction, the chapters composing this thesis are presented:

- **Chapter 2:** A gentle introduction is provided to the functioning of Deep Neural Networks (DNNs). The model of the neuron, the training process and the fundamental components of neural networks are described. The early stopping and dropout generalization techniques are presented, as well.
- **Chapter 3:** The mathematical foundation of the RANS equations, Spalart-Allmaras (SA) and k-ω turbulence models, and $\gamma - \tilde{Re}_{\theta t}$ transitional model is presented. The system of equations is numerically solved in PUMA software which is also discussed.
- **Chapter 4:** A gentle introduction to Optimization using Evolutionary Algorithms (EA) is given. The Metamodel-Assisted Evolutionary Algorithms (MAEA) are also introduced. All EA or MAEA based optimizations were performed using the EASY software, which is also presented.
- **Chapter 5:** The DNNs are exploited for predicting the turbulence viscosity field (μ_t) in order to replace the SA turbulence model without transition in a MAEA based optimization of NACA4318 airfoil case. All the necessary steps to create and train the DNN are presented. Two MAEA based optimizations carried out, one using the RANS coupled with the SA turbulence model within PUMA, and another using the RANS coupled with the DNN within PUMA, in order to compare and evaluate the results.

- **Chapter 6:** The DNNs are exploited for predicting the turbulence viscosity field in order to replace the k-ω turbulence model assisted by the $\gamma - \tilde{Re}_{\theta t}$ transitional model in a MAEA based optimization of LS89 turbine blade case. All the necessary steps to create and train the DNN are presented. Two MAEA based optimizations carried out, one using the RANS coupled with the k-ω turbulence and $\gamma - \tilde{Re}_{\theta t}$ transitional model within PUMA, and another using the RANS coupled with the DNN within PUMA, in order to compare and evaluate the results.

Chapter 2

Deep Neural Network - DNN

2.1 Introduction

Artificial neural networks (ANNs), also known as "neural networks," are nonlinear mapping systems that were inspired by the difference in computation between the human brain and conventional digital computers. The brain, a highly complex and parallel information-processing system, composed of neurons, can perform tasks such as pattern recognition, perception, and motor control much faster than the fastest digital computer [10]. The nonlinear and parallel nature of the brain motivated the development of ANNs.

A brain neuron is comprised of three main parts: a dendritic tree that receives signals from other neurons, a cell body that processes the signals and produces a response, and an axon that transmits the response to other neurons. Each neuron's response is a non-linear function of its inputs and state and is mainly determined by the strength of its input connections. A single neuron can receive anywhere from a few hundreds to tens of thousands of inputs and has multiple branches on its axon. The concept here is that large systems of simple units connected in the right way can exhibit complex and interesting behaviors. A simple brain neuron is shown in Figure 2.1.

An ANN is a model designed to resemble the structure and function of a biological neuron. It is composed of a large number of simple processing units, known as neurons, that are interconnected via weighted connections. Each neuron takes input from multiple other neurons, calculates an output value using the inputs and stored information, and then passes that output on as input to other neurons.

The simplest structure of an ANN (see Figure 2.2), consists of three layers: input, hidden, and output. The input layer provides input to every unit in the next layer, the hidden layer, which does not have direct connections to the outside world. The output layer receives inputs from the hidden layer and produces the final output of the network. The network has only forward connections, with no sideways or backward connections. The goal of an ANN is to be trained on input patterns (e.g. current stock market prices) to generate correct output patterns (e.g. next year's stock market prices).

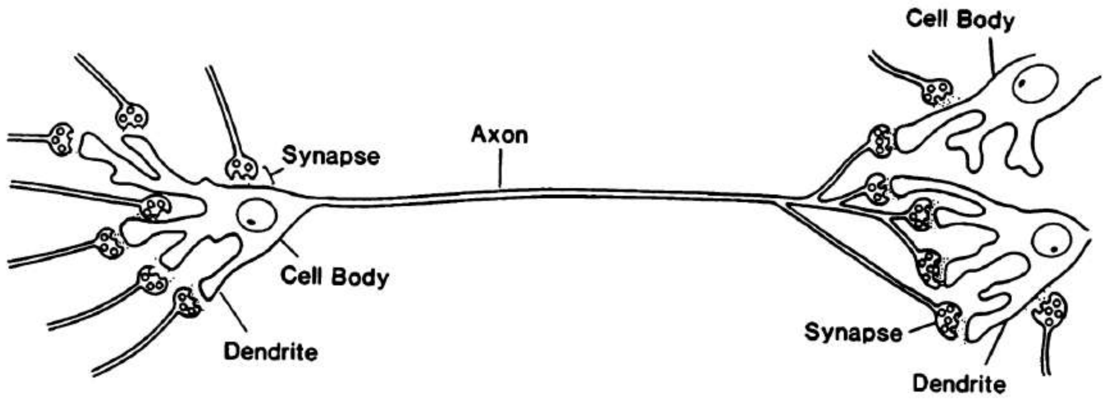


Figure 2.1: *A schematic diagram of a "typical" vertebrate neuron. The electrical signals flow into the dendrites and out through the axon. Thus, in this diagram, the information flows from left to right. From: [11]*

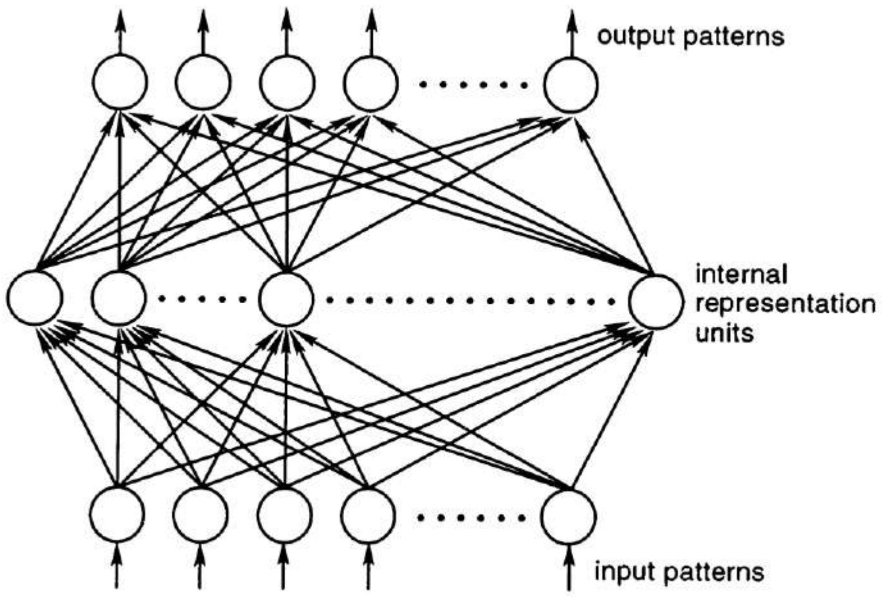


Figure 2.2: *A simple multilayer network. Each unit connects to all units in the layer above it. There are no sideways connections, or back connections. The "internal representation units" are often referred to as the "hidden units". From: [11]*

2.2 Artificial Neuron Model

An artificial neuron is a fundamental building block of a neural network and plays a crucial role in processing information. A neuron's structure, is shown in Figure 2.3, consists of three main components.

- A set of synapses is defined, each with a weight. The input signal x_j at synapse j, connected to neuron k, is multiplied by the weight w_{kj} of the synapse.
- An adder u_k that sums the input signals, each multiplied by its corresponding synaptic strength.
- An activation function $\phi(\cdot)$ is used to restrict the magnitude of the output produced by a neuron.
- The bias b_k of k neuron.

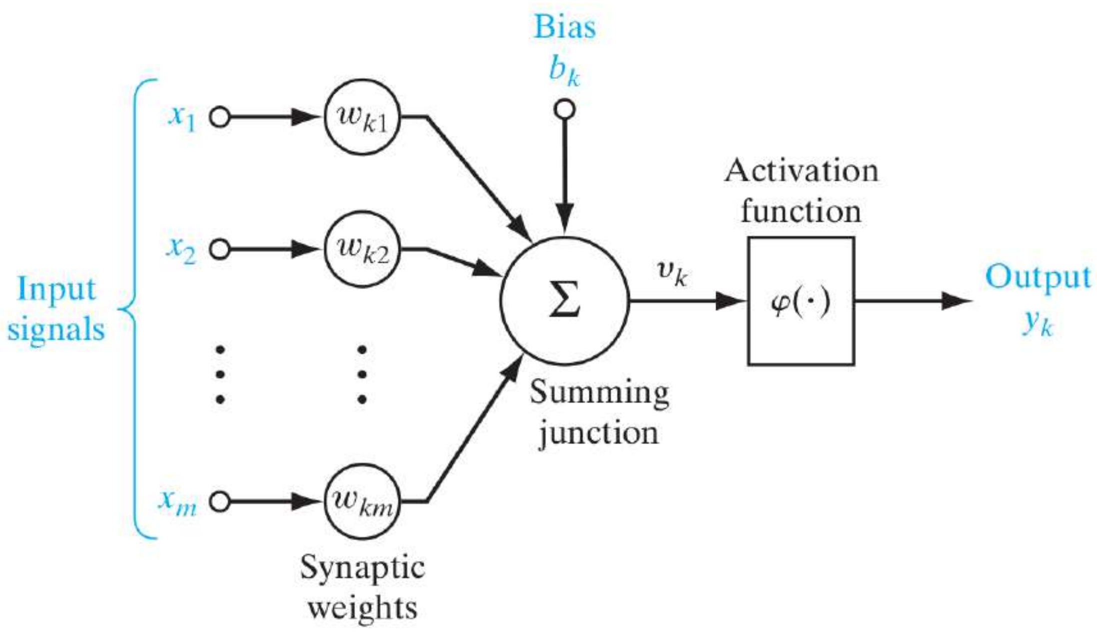


Figure 2.3: *An artificial neuron model. From: [10]*

The processing units (neurons), have responses like:

$$y_k = \phi(v_k) \tag{2.1}$$

where $v_k = u_k + b_k$, and

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.2}$$

where $x_1, x_2, ..., x_m$ are the input signals and $w_{k1}, w_{k2}, ..., w_{km}$ are the respective synaptic weights of neuron k. Here, the unit computes a weighted linear combination of its inputs and passes this through the nonlinearity $\phi(\cdot)$ to produce a scalar output y_k. This nonlinearity allows for a greater range of problem-solving capabilities compared to linear systems, as $\phi(\cdot)$ is a bounded, non-decreasing nonlinear function. Some basic activation functions are listed in Table 2.1.

Some Activation Functions	
Sigmoid	$$\phi(\upsilon_k) = \frac{1}{1+e^{-\upsilon_k}} \tag{2.3}$$
Hyperbolic Tangent (tanh)	$$\phi(\upsilon_k) = \frac{e^{\upsilon_k} - e^{-\upsilon_k}}{e^{\upsilon_k} + e^{-\upsilon_k}} \tag{2.4}$$
Rectified Linear Unit (ReLU)	$$\phi(\upsilon_k) = max(\upsilon_k, 0) = \begin{cases} \upsilon_k, & \text{if } \upsilon_k \geq 0 \\ 0, & \text{if } \upsilon_k < 0 \end{cases} \tag{2.5}$$

Table 2.1

By itself, a single processing element is not very powerful. The strength of an artificial neural network lies in the combination of many simple processing elements, known as neurons. By varying the connections and connection weights, the network can be trained to perform complex functions. Research has demonstrated that a large network with the right structure and weights can accurately mimic any function within certain constraints [13].

2.3 Training Process

In the previous section, the general idea and structure of neural networks have been presented. To gain deeper insight, the training process of neural networks, composed of three key concepts - loss functions, back-propagation, and optimization - requires further examination.

2.3.1 Loss function

The loss function in neural networks is a crucial component of the training process. It calculates the discrepancy between the expected or target output and the actual output generated by the model. The value obtained from the loss function is then used to compute the gradients that adjust the weights in the network, ultimately helping to improve its accuracy. In this section, we will be exploring two common types of loss functions, however, it is worth noting that there are numerous other options available. The choice of loss function depends on the specific requirements of the problem and can greatly impact the performance of the model. In this Diploma Thesis the type of loss function was chosen is the Mean Absolute Error (MAE).

Mean Absolute Error (MAE) [14], also called L1 loss and is used for regression problems. It measures the average magnitude of absolute differences between N predicted vectors $S = \{y_1, y_2, ..., y_N\}$ and N actual observations $S^* = \{y_1^*, y_2^*, ..., y_N^*\}$ and the corresponding loss function is defined as:

$$\frac{1}{N}\sum_{\imath=1}^{N} ||y_\imath - y_\imath^*||_1 \tag{2.6}$$

where $||\cdot||_1$ denotes L_1 norm.

MAE is robust to outliers and is useful in situations where the distribution has many modes and accurate predictions at a specific mode are desired.

Mean Square Error (MSE) [14] . MSE also called L2 loss and it used also for regrassion problems. MSE denotes a quadratic scoring rule that measures the average magnitude of N predicted vectors $S = \{y_1, y_2, ..., y_N\}$ and N actual observations $S^* = \{y_1^*, y_2^*, ..., y_N^*\}$. The corresponding loss function is shown as:

$$L_{MSE}(S, S^*) = \frac{1}{N}\sum_{\imath=1}^{N} ||y_\imath - y_\imath^*||_2^2 \tag{2.7}$$

where $||\cdot||_2$ denotes L_2 norm.

MSE is sensitive towards outliers and it is used when it's desirable to penalize significantly (quadratically) more large errors than small ones.

The total error of the neural network is just the sum of the individual pattern errors from the training set

$$L = \sum_p L_p \tag{2.8}$$

where p indexes the patterns in the training set and L_p is the corresponding loss function at p pattern which can be calculated as described before.

2.3.2 Back-Propagation: The derivative calculation

After obtaining the outputs and computing the error, the next step is to calculate the derivative of the error with respect to the weights, in order to minimize the error using an optimization algorithm. Back Propagation [12] is an algorithm used in training neural networks that computes these derivatives by iteratively processing the information in reverse order, starting from the output layer and proceeding to the input layer.

First, we note that the total error of the neural network is just the sum of the individual pattern errors from the training set, so the total derivative is just the sum of the per-pattern derivatives,

$$\frac{\partial L}{\partial w_{kj}} = \sum_{p} \frac{\partial L_p}{\partial w_{kj}} \tag{2.9}$$

where p indexes the patterns of the training set and L_p is the corresponding loss function at p pattern of the training set.

The derivative can be written as:

$$\frac{\partial L_p}{\partial w_{kj}} = \sum_{k} \frac{\partial L_p}{\partial \upsilon_k} \frac{\partial \upsilon_k}{\partial w_{kj}} \tag{2.10}$$

where υ_k and w_{kj} explained in section 2.2.

The success of the back-propagation algorithm in finding derivatives is attributed to its organized and systematic decomposition of steps. It's important to note that the next steps of the derivative calculation process will differ based on the type of loss function used. However, it's not necessary to analyze the process in great detail as the objective of this explanation is to provide a general understanding of the training process, rather than exploring each and every variation of the algorithm.

2.3.3 Optimization Algorithm

The next step after obtaining the gradients is to use optimization algorithms to minimize the error, as defined by the loss function, by altering the values of the synaptic weights. These algorithms work in an iterative manner, meaning they repeat a sequence of steps to find the optimal solution. They initiate with an initial approximation of the weights, represented by $w_{kj}(n = 0)$, and then iteratively update this approximation using a specific equation until a convergence is reached. The equation form is:

$$w_{kj}(n + 1) = w_{kj}(n) + \eta p(n) \tag{2.11}$$

The update of the synaptic weights in the training process of a neural network is based on a search direction vector $p(n)$ and a positive scalar learning rate η, which determines the step length. The choice of $p(n)$ distinguishes various optimization algorithms, as does the selection of η.

Gradient Descent

The first method for training a Back Propagation network was the gradient algorithm [12]. This method is based on the principle of taking successive steps in the direction opposite to the gradient, or a close approximation, of the loss function at the current point. This direction represents the steepest descent and helps the algorithm to converge to an optimal solution. The gradient algorithm proceeds as follows: given the current weight matrix values $W(n)$, it takes a step in the direction of the negative gradient of the loss function at that point, updating the weight matrix to $W(n+1)$. This process continues until the algorithm converges to a local minimum of the loss function. The gradient algorithm proceeds as follows:

$$W(n+1) = W(n) - \eta_n \nabla_w L(W(n)) \tag{2.12}$$

where η_n is the step size at the n-th iteration and the $L(W(n))$ is the loss function at the n-th iteration. The term $\nabla_w L(W(n))$ has been already calculated from back-propagation algorithm.

The gradient method is the first proposed training algorithm, it is simple to implement, but has the following disadvantages:

- slow convergence
- finds only a local extremum.

The limitations of the gradient algorithm in training Back Propagation networks have been addressed by its modifications, one of which is the ADAM algorithm, which was used in this Diploma Thesis. This method, introduced in [15], is known for its efficiency and low memory requirements in optimizing the model. Unlike traditional optimization methods, ADAM requires only first-order gradient estimates, making it highly advantageous. The algorithm personalizes the learning rate for each parameter in the model by utilizing estimates of the first and second moments of the gradients. The name "ADAM" is an acronym for Adaptive Moment Estimation, reflecting its capability to dynamically adapt the learning rate based on the data.

2.4 Components of the Learning Algorithm

Training a deep learning neural network model using stochastic gradient descent with backpropagation involves choosing a number of components and hyperparameters, which are:

- Network Topology
- Loss Function

- Weight Initialization
- Batch Size
- Learning Rate
- Epochs
- Data Preparation

Some are more crucial than others and some selections can lead to sensible defaults for other elements. Loss function has already been discussed, the rest will be analyzed.

The capacity of a neural network is an important factor in determining the range of functions it is capable of representing. This capacity is determined by the number of nodes in the hidden layer, with a larger number of nodes providing a greater degree of versatility, but also making the model more difficult to train due to an increased number of parameters that need to be learned. While it is possible for a single hidden layer with a sufficient number of nodes to theoretically approximate any mapping function, this is not a practical solution in real-world applications. Instead, incorporating multiple layers with fewer nodes each increases the capacity of the model while simplifying the optimization process.

- **Network Topology.** The number of nodes (or equivalent) in the hidden layers and the number of hidden layers in the network.

The search or optimization process in the training of a neural network requires a starting point to initiate the model updates. The initial model parameters, also known as weights, define the starting point. It is crucial to choose an appropriate starting point as the optimization algorithm is sensitive to it due to the non-convex error surface. To overcome this, small random values are commonly used as the initial model weights, however, different weight initialization methods can also be utilized to determine the scale and distribution of these values. The choice of activation function can also have an impact on the weight initialization method used.

- **Weight Initialization.** The procedure by which the initial small random values are assigned to model weights at the beginning of the training process.

The process of updating the model involves using a selection of examples from the training dataset to compute the loss. For smaller datasets, it may be appropriate to use all examples in the calculation. However, for problems where the data changes frequently or is streamed, using a single example may be more appropriate. There is also a combination approach where a certain number of examples can be selected to estimate the error gradient. This selection of examples is referred to as the batch size.

- **Batch Size.** The number of examples used to estimate the loss gradient before updating the model parameters.

Once the error gradient has been estimated, the next step is to calculate the derivative of the activation function and use it to update the model parameters. However, there are several factors that can add statistical noise to both the training dataset and the estimated error gradient. Additionally, the depth of the model (number of layers) and the separate updating of model parameters can make it challenging to determine the optimal amount of change for each parameter to effectively move the model down the error surface. To address these challenges, only a small portion of the weight update is performed in each iteration. The size of this update is determined by a hyperparameter known as the learning rate, which controls the speed of learning for the model on the training dataset.

- **Learning Rate.** The amount that each model parameter is updated per iteration of the learning algorithm.

The training of a neural network involves repeating the process of updating the model parameters many times until an optimal or satisfactory set is found. The iteration of the process is determined by the number of passes through the complete training dataset, after which the training stops. This is known as the number of training epochs, a hyperparameter that is closely linked to the choice of learning rate and batch size. In some cases, when utilizing certain regularization methods, this hyperparameter can be set to a high value and its impact can be minimized.

- **Epochs.** The number of complete passes through the training dataset before the training process is terminated.

The significance of the data used in the learning process of the mapping function is often overlooked. The scale of the target variable plays a crucial role in determining the choice of activation function for the output layer of the network. Moreover, the scale of the input variables greatly impacts the scale of the weights in the input and the initial hidden layers, which in turn affects the stability of the learning process. The consideration of the scale and structure of the data used for learning is referred to as data preparation.

- **Data Preparation.** The schemes used to prepare the data prior to modeling in order to ensure that it is suitable for the problem and for developing a stable model.

The process of training deep learning neural networks involves several components and hyperparameters that must be carefully adjusted to achieve optimal performance. Despite the numerous available extensions to the learning algorithm, configuring a network for a specific problem remains a challenging task, with no universally agreed-upon set of rules for optimal network configuration.

2.5 Generalization in Neural Networks

The ultimate goal of training a neural network is to produce a final model that can perform effectively on both the training data, as well as any new data that it will be applied to for predictions. It is crucial that the model can accurately learn from the known examples and apply that learning to future examples it has not seen before. To measure the generalization performance of a model, techniques such as train/test split or k-fold cross-validation [16] can be utilized. However, striking the balance between learning and generalizing can be challenging, as too little learning will result in underfitting, causing poor performance on both the training and new data, while too much learning will result in overfitting, leading to exceptional performance on the training data, but poor performance on new data. In both scenarios, the model has failed to generalize properly. The type of model fit are the followings [17]:

- **Underfit Model.** A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on new patters.
- **Overfit Model.** A model that learns the training dataset too well, performing well on the training dataset but does not perform well on new patterns.
- **Good Fit Model.** A model that suitably learns the training dataset and generalizes well to new patterns.

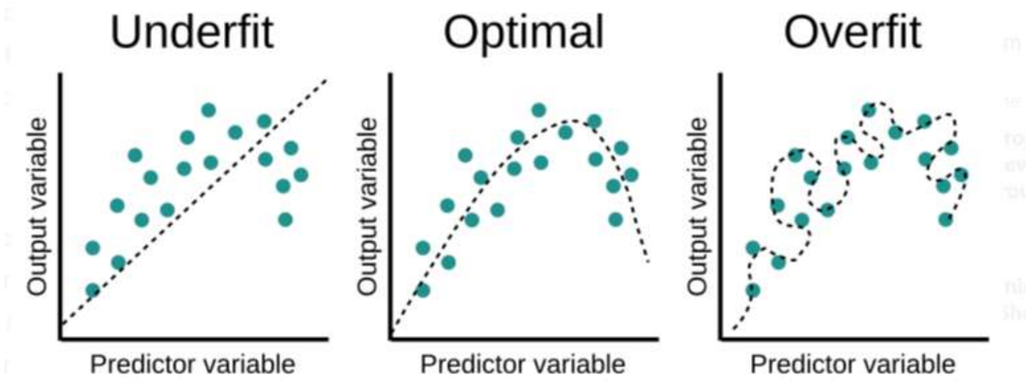


Figure 2.4: *Different types of fitting. From: [18]*

When it comes to evaluating the fit of a model, it is important to consider the bias-variance trade-off. A model that is underfitted has high bias and low variance, meaning that it cannot effectively learn the problem at hand regardless of the data it is trained on. On the other hand, an overfitted model has low bias and high variance, as it has learned the training data too well and its performance can vary greatly with new, unseen examples or even with small variations in the training data.

To address underfitting, one can increase the capacity of the model by changing its structure, such as by adding more layers or nodes to existing layers. Overfitting, on the other hand, is often indicated by monitoring the model's performance on both a training dataset and a validation dataset.

2.5.1 Reduce Overfitting by Constraining Complexity

There are two ways to approach an overfit model [17]:

- Reduce overfitting by training the network on more examples.
- Reduce overfitting by changing the complexity of the network.

The advantage of utilizing very deep neural networks is that their performance increases as the size of the dataset grows. There will come a point, when the model has been exposed to an almost infinite number of examples, where its performance will reach a plateau based on the network's capacity for learning. Overfitting the training dataset may occur if the network has the ability to do so, which is a result of having a large capacity. Decreasing the network's capacity can reduce the likelihood of overfitting and can be done by either modifying the structure of the network in terms of the number of nodes and layers or by changing the parameters, specifically the weights, which contribute to the network's complexity. Therefore, it's possible to reduce the complexity of a neural network to reduce overfitting in one of two ways [17]:

- Change network complexity by changing the network structure (number of weights).
- Change network complexity by changing the network parameters (values of weights).

In the case of neural networks, the complexity can be varied by changing the number of adaptive parameters in the network. This is called structural stabilization. The second principal approach to controlling the complexity of a model is through the use of regularization which involves the addition of a penalty term to the loss function.

In this Diploma Thesis, efforts were made to mitigate overfitting in neural networks by adjusting their complexity. This was achieved by reducing the number of units and weights within the network. By doing so, the complexity of the network was reduced, leading to a decreased likelihood of overfitting.

2.5.2 Regularization Methods for Neural Networks

Below is a list of two common regularization methods that were used in this Diploma Thesis.

- **Dropout:** Probabilistically remove inputs during training.

- **Early Stopping:** Monitor model performance on a validation set and stop training when performance degrades.

Decouple Layers with Dropout

DNNs can be susceptible to overfitting a training dataset with limited examples. In order to mitigate this problem, multiple models with different configurations can be combined into ensembles. However, this solution requires a significant computational expense in terms of training and maintaining multiple models. Alternatively, dropout is a cost-efficient and effective regularization technique that can be used to reduce overfitting and improve generalization error in DNNs. In dropout, random nodes are dropped out during training (Figure 2.5), effectively simulating an ensemble of different network architectures. This regularization method was introduced by [19] and has been proven to be an effective solution for reducing overfitting in various types of deep neural networks.

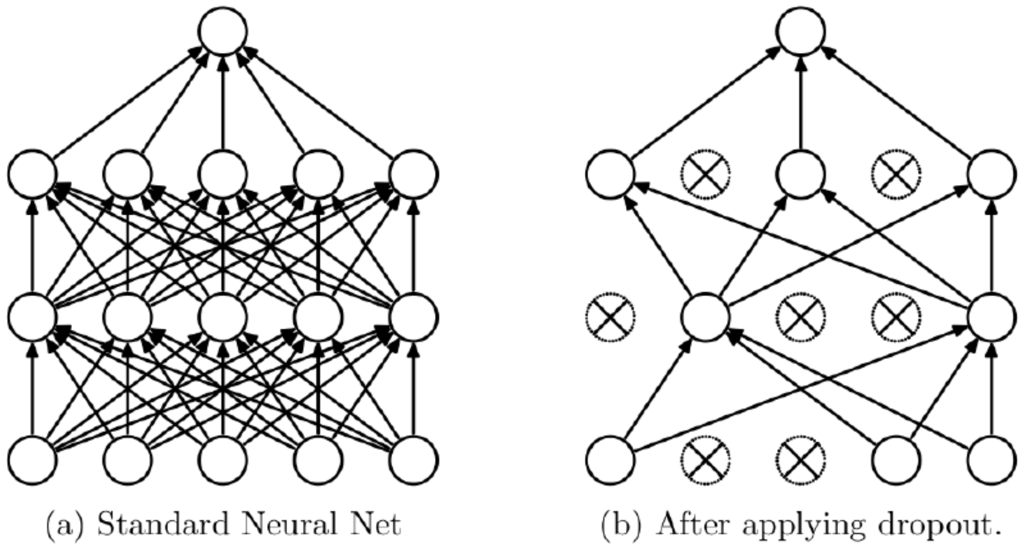


Figure 2.5: *Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. From: [19]*

The Dropout technique is a well-established regularization strategy that aims to prevent overfitting of DNNs by approximating the training process of multiple neural networks with different architectures. During the training phase, a random number of node outputs are temporarily ignored, making the current layer appear and behave as if it has a different number of nodes and connections to the previous layer. This leads to each layer being updated with a different configuration of nodes during training, effectively making the process more noisy and forcing nodes to adapt to changing conditions and assume varying levels of responsibility for inputs. This leads to a more robust model, as the layers are less likely to co-adapt to correct mistakes made by previous layers.

Model Description [19]

This section outlines the dropout neural network model. The model consists of L hidden layers and uses the index l, ranging from 1 to L, to identify the hidden layers. The vector of inputs into layer l is denoted as $z^{(l)}$, while the vector of outputs from layer l is denoted as $y^{(l)}$ ($y^{(0)} = x$ represents the input). Additionally, the weights and biases at layer l are represented by $W^{(l)}$ and $b^{(l)}$, respectively. The feed-forward operation of a standard neural network (as shown in (Figure 2.6a)) can be expressed as follows (for all values of l ranging from 0 to L-1, and for any given hidden unit i):

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)}$$

$$y_i^{(l+1)} = \phi\left(z_i^{(l+1)}\right)$$

where ϕ is any activation function. With dropout, the feed-forward operation becomes (Figure 2.6b)

$$r_j^{(l)} \sim Bernoulli(p)$$

$$\tilde{y}^{(l)} = r^{(l)} * y^{(l)}$$

$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}$$

$$y_i^{(l+1)} = \phi(z_i^{(l+1)})$$

The model uses the symbol $*$ to represent element-wise product. For each layer l, the vector $r^{(l)}$ consists of independent Bernoulli random variables [1], each with a probability of p of being equal to 1. The vector is sampled and multiplied element-wise with the outputs of layer l, represented by $y^{(l)}$, to create the thinned outputs $\tilde{y}^{(l)}$. These thinned outputs are then used as inputs to the following layer. This process is repeated for every layer. The result is the selection of a sub-network from the larger network. During the learning process, the derivatives of the loss function are backpropagated through this sub-network. At the time of testing, the weights are scaled as $W_{test}^{(l)} = pW^{(l)}$, as shown in Figure 2.7.

[1] Independent Bernoulli random variables are random variables with two possible outcomes: success (represented by 1) and failure (represented by 0). The probability of success is represented by the parameter p, and the probability of failure is represented by (1-p). Each Bernoulli random variable is independent of each other, meaning the outcome of one does not affect the outcome of another.

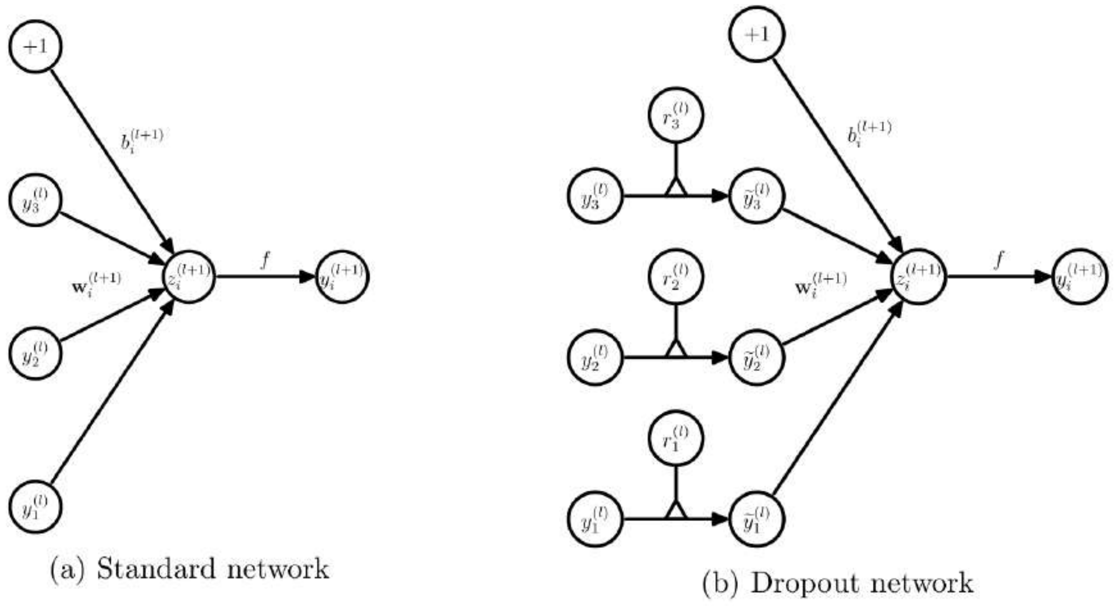


Figure 2.6: *Comparison of the basic operations of a standard and dropout network. From: [19]*

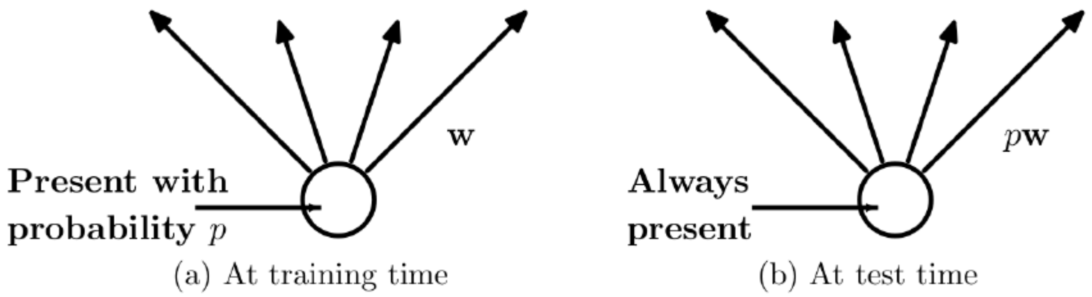


Figure 2.7: *Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights w. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time. From: [19]*

Chapter 3

Aerodynamic Analysis in Turbulent Flows

3.1 RANS equations

Turbulence is characterized by an unsteady flow pattern, with motion that is irregular and unpredictable. This type of flow is prevalent in nature and plays a crucial role in engineering, as many flows in industrial settings are turbulent. The turbulence in the flow causes significant variations in key flow properties, such as velocity, pressure, temperature, and even density. These fluctuations can occur at frequencies as high as 10 Hz, making it computationally and memory-intensive to solve the Navier-Stokes equations using traditional methods, which would require an extremely dense mesh. To address this issue, the Reynolds decomposition is used to separate the velocity component, u, into its average value and fluctuation component. Thus the RANS equations are an average from the time-dependent Navier-Stokes equations. The RANS equations for steady state flows of compressible fluids are written as follows:

$$R_n^{MF} = \frac{\partial f_{nj}^{inv}}{\partial x_j} - \frac{\partial f_{nj}^{vis}}{\partial x_j} = 0 \tag{3.1}$$

where $n = 1,..,5$ and $j = 1,..3$. $f_j^{inv} = [\rho u_j \;\; \rho u_j u_1 + p\delta_{1j} \;\; \rho u_j u_2 + p\delta_{2j} \;\; \rho u_j u_3 + p\delta_{3j} \;\; \rho u_j h_t]^T$ are the inviscid fluxes and $f_j^{vis} = [0 \;\; \tau_{1j} \;\; \tau_{2j} \;\; \tau_{3j} \;\; u_k\tau_{kj} + q_j]^T$ the viscous/turbulent ones. ρ, p, u_j,h_t and δ_{jm} stand for the fluid's density, pressure, velocity components, total enthalpy and the Kronecker symbol, respectively. Based on the Boussinesq assumption, the components of the stress tensor are:

$$\tau_{jm} = (\mu + \mu_t)\left(\frac{\partial u_j}{\partial x_m} + \frac{\partial u_m}{\partial x_j} - \frac{2}{3}\delta_{jm}\frac{\partial u_k}{\partial x_k}\right) - \frac{2}{3}\delta_{jm}\rho k \tag{3.2}$$

where μ is the bulk and μ_t the turbulence viscosity (with the last term of Equation 3.2 being used only with the k-ω SST model) and q_j is the heat flux. k is turbulent kinetic energy (TKE), $k = \frac{1}{2}\overline{u_j' u_m'}$, which can be computed by using the k-ω SST model. The velocity variables and pressure result from the solution of the mean flow equations, and μ_t results from the selected turbulent model.

3.2 The Spalart-Allmaras (SA) Turbulence Model

The SA is a one-equation turbulence model [2], that solves a modeled transport equation for the turbulence field $\tilde{\nu}$. The one-equation model for steady state, compressible flow is given by the following equation:

$$\begin{aligned} R^{SA} = & \frac{\partial(\rho\tilde{\nu}u_k^R)}{\partial x_j} - \frac{\rho}{Re_0\sigma}\left\{\frac{\partial}{\partial x_j}\left[(\nu+\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_k}\right] + c_{b2}\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k}\right\} \\ & - \rho c_{b1}\tilde{S}\tilde{\nu} + \frac{\rho}{Re_0}c_{w1}f_w\left(\frac{\tilde{\nu}}{\Delta}\right)^2 \end{aligned} \tag{3.3}$$

where Δ stands for the distance of each point within the flow domain from the closest wall boundary. The turbulent eddy viscosity is computed from:

$$\mu_t = \rho\tilde{\nu}f_{v1} \tag{3.4}$$

Equation 3.3 is supplemented by the following relations and constants [2]:

$$\chi = \frac{\tilde{\nu}}{\nu} \qquad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}$$

$$f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}} \qquad f_w = g\left(\frac{1+c_{w3}^6}{g^6 + c_{w3}^6}\right)^{1/6}$$

$$\Omega = \sqrt{\epsilon_{jkm}\epsilon_{jqr}\frac{\partial u_m^A}{\partial x_k}\frac{\partial u_r^A}{\partial x_q}} \qquad \tilde{S} = \Omega + \frac{\tilde{\nu}f_{v2}}{Re_0\kappa^2\Delta^2}$$

$$g = r + c_{w2}(r^6 - r) \qquad r = min\left(\frac{\tilde{\nu}}{Re_0 \tilde{S} \kappa^2 \Delta^2}, 10\right)$$

$$\tilde{\mu} = \rho\tilde{\nu} \qquad \nu = \frac{\mu}{\rho}$$

- $c_{b1} = 0.1355$
- $c_{v1} = 7.1$
- $c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}$
- $\sigma = 2/3$
- $\kappa = 0.41$

- $c_{b2} = 0.622$
- $c_{w2} = 0.3$
- $c_{w3} = 2$
- $c_{t3} = 1.2$
- $c_{t4} = 0.5$

where Ω is the magnitude of the vorticity of the flow.

3.3 The k-ω Turbulence Model

The "Standard" Menter SST Two-Equation Model [3] (written in conservation form) for steady state and compressible flows is given by the following:

$$\frac{\partial(\rho u_j k)}{\partial x_j} = P - D + \frac{\partial}{\partial x_j}\left((\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_j}\right) \tag{3.5}$$

$$\frac{\partial(\rho u_j \omega)}{\partial x_j} = \frac{\gamma}{\nu_t}P - \beta\rho\omega^2 + \frac{\partial}{\partial x_j}\left((\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right) + 2(1 - F_1)\frac{\rho\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{3.6}$$

where:

$$P = \tau_{ij}\frac{\partial u_i}{\partial x_j} \tag{3.7}$$

$$D = \beta^* \rho\omega k \tag{3.8}$$

and the turbulent eddy viscosity is computed from:

$$\mu_t = \frac{\rho \alpha_1 k}{max(\alpha_1 \omega, \Omega F_2)} \tag{3.9}$$

Each of the constants is a blend of an inner (1) and outer (2) constant, blended via:

$$\phi = F_1\phi_1 + (1 - F_1)\phi_2 \tag{3.10}$$

where ϕ_1 represents constant 1 and ϕ_2 represents constant 2.
Additional functions are given by:

$$F_1 = tanh(arg_1^4) \tag{3.11}$$

$$arg_1 = min\left(max\left(\frac{\sqrt{k}}{\beta^*\omega\Delta}, \frac{500\nu}{\Delta^2\omega}\right), \frac{4\rho\sigma_{\omega 2}k}{CD_{k\omega}\Delta^2}\right) \tag{3.12}$$

$$CD_{k\omega} = max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-20}\right) \tag{3.13}$$

$$F_2 = tanh(arg_2^2) \tag{3.14}$$

$$arg_2 = max\left(2\frac{\sqrt{k}}{\beta^*\omega\Delta}, \frac{500\nu}{\Delta^2\omega}\right) \tag{3.15}$$

The constants are:

- $\gamma_1 = \frac{\beta_1}{\beta^*} - \frac{\sigma_{\omega 1}k^2}{\sqrt{\beta^*}}$
- $\sigma_{k1} = 0.85$
- $\sigma_{\omega 1} = 0.5$
- $\alpha_1 = 0.31$
- $\beta_1 = 0.075$
- $\gamma_2 = \frac{\beta_2}{\beta^*} - \frac{\sigma_{\omega 2}k^2}{\sqrt{\beta^*}}$
- $\sigma_{k2} = 1.0$
- $\beta_2 = 0.0828$
- $\sigma_{\omega 2} = 0.856$
- $\beta^* = 0.09$
- $\kappa = 0.41$

3.4 The $\gamma - \tilde{Re}_{\theta t}$ Transitional Model

The model is based on two transport equations, one for the intermittency and the other for a transition onset criterion in terms of the momentum-thickness Reynolds number [4]. When the flow is transitional, the k-ω SST turbulence model is coupled with the $\gamma - \tilde{Re}_{\theta t}$ transition model.
The transport equation for the intermittency γ reads

$$R^\gamma = \frac{\partial(\rho u_j \gamma)}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\left(\mu + \frac{\mu_t}{\sigma_f}\right)\frac{\partial \gamma}{\partial x_j}\right) - P_\gamma + E_\gamma = 0 \tag{3.16}$$

The transition sources are defined as follows:

$$P_\gamma = F_{length} c_{\alpha 1} \rho S [\gamma F_{onset}]^{0.5} (1 - c_{e1}\gamma) \tag{3.17}$$

where F_{length} is an empirical correlation that controls the length of the transition region and F_{onset} controls the transition onset location. The destruction/relaminarization source term is defined as follows:

$$E_\gamma = c_{\alpha 2} \rho \Omega \gamma F_{turb} (c_{e2}\gamma - 1) \tag{3.18}$$

Transition onset is controlled by the following functions:

$$Re_V = \frac{\rho \Delta^2 S}{\mu} \tag{3.19}$$

$$F_{onset1} = \frac{Re_v}{2.193 \cdot Re_{\theta c}} \tag{3.20}$$

$$F_{onset2} = min(max(F_{onset1}, F_{onset1}^4), 2.0) \tag{3.21}$$

$$R_T = \frac{\rho k}{\mu \omega} \tag{3.22}$$

$$F_{onset3} = max\left(1 - \left(\frac{R_T}{2.5}\right)^3, 0\right) \tag{3.23}$$

$$F_{onset} = max(F_{onset2} - F_{onset3}, 0) \tag{3.24}$$

where $Re_{\theta c}$ is the critical Reynolds number where the intermittency first starts to increase in the boundary layer. Based on T3B, T3A, T3A-, and the Schubauer and Klebanof test cases [23] a correlation for F_{length} based on $Re_{\theta t}$ from an empirical correlation is defined as:

$$F_{length} = \begin{cases} [398.189 \cdot 10^{-1} + (-119.270 \cdot 10^{-4})\tilde{Re}_{\theta t} + (-132.567 \cdot 10^{-6}\tilde{Re}_{\theta t}^2)], & \tilde{Re}_{\theta t} < 400 \\ [263.404 + (-123.939 \cdot 10^{-2})\tilde{Re}_{\theta t} + (-194.548 \cdot 10^{-5})\tilde{Re}_{\theta t}^2 + (-101.695 \cdot 10^{-8})\tilde{Re}_{\theta t}^3], & 400 \le \tilde{Re}_{\theta t} < 596 \\ [0.5 - (\tilde{Re}_{\theta t} - 596.0) \cdot 3.0 \cdot 10^{-4}], & 596 \le \tilde{Re}_{\theta t} < 1200 \\ [0.3188], & 1200 \le \tilde{Re}_{\theta t} \end{cases} \tag{3.25}$$

In some situations, particularly during transition at higher Reynolds numbers, the $\tilde{Re}_{\theta t}$ scalar tends to decrease significantly in the boundary layer shortly after the transition. Since F$_{lenght}$ is based on $\tilde{Re\theta t}$, this can cause a local increase in the source term for the intermittency equation, leading to a sharp rise in skin friction. Although the skin friction eventually returns to its fully turbulent value, this effect is not a physically realistic one. To eliminate this effect, F$_{length}$ can be forced to always remain equal to its maximum value in the viscous sublayer. The modification required to achieve this is presented below.

$$F_{sublayer} = e^{-(\frac{R_\omega}{0.4})^2} \tag{3.26}$$

$$R_\omega = \frac{\rho \Delta^2 \omega}{500\mu} \tag{3.27}$$

$$F_{length} = F_{length}(1 - F_{sublayer} + 40.0 \cdot F_{sublayer}) \tag{3.28}$$

The correlation between $Re_{\theta c}$ and $\tilde{R}e_{\theta t}$ is defined as follows:

$$Re_{\theta c} = \begin{cases} [\tilde{R}e_{\theta t} - (396.035 \cdot 10^{-2} + (-120.656 \cdot 10^{-4})\tilde{R}e_{\theta t} + (868.230 \cdot 10^{-6})\tilde{R}e_{\theta t}^2 & \\ +(-696.506 \cdot 10^{-9})\tilde{R}e_{\theta t}^3 + (174.105 \cdot 10^{-12})\tilde{R}e_{\theta t}^4)], & \tilde{R}e_{\theta t} \leq 1870 \\ [\tilde{R}e_{\theta t} - (593.11 + (\tilde{R}e_{\theta t} - 1870.0) \cdot 0.482)], & \tilde{R}e_{\theta t} > 1870 \end{cases} \tag{3.29}$$

The constants for the intermittency equation are:

$c_{e1} = 1.0$, $c_{\alpha 1} = 2.0$, $c_{e2} = 50$, $c_{\alpha 2} = 0.06$, $\sigma_f = 1.0$

The modification to the intermittency for predicting separation induced transition is:

$$\gamma_{sep} = min\left(s_1 max\left(0, \left(\frac{Re_v}{3.235 Re_{\theta c}}\right) - 1\right) F_{reattach}, 2\right) F_{\theta t} \tag{3.30}$$

$$F_{reattatch} = e^{-(\frac{R_T}{20})^4} \tag{3.31}$$

$$\gamma_{eff} = max(\gamma, \gamma_{sep}) \tag{3.32}$$

$$s_1 = 2 \tag{3.33}$$

The transport equation for the transition momentum-thickness Reynolds number $\tilde{R}e_{\theta t}$ reads

$$R^{\tilde{R}e_{\theta t}} = \frac{\partial(\rho u_j \tilde{R}e_{\theta t})}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\sigma_{\theta t}(\mu + \mu_t)\frac{\partial \tilde{R}e_{\theta t}}{\partial x_j}\right) - P_{\theta t} = 0 \tag{3.34}$$

where the source term is defined as follows:

$$P_{\theta t} = c_{\theta t}\frac{\rho}{t}(Re_{\theta t} - \tilde{R}e_{\theta t}(1.0 - F_{\theta t})) \tag{3.35}$$

$$t = \frac{500\mu}{\rho U^2} \tag{3.36}$$

where t is a time scale. The $F_{\theta t}$ blending function is defined as follows:

$$F_{\theta t} = min\left(max\left(F_{wake} \cdot e^{-(\frac{\Delta}{\delta})^4}, 1.0 - \left(\frac{\gamma - 1/c_{e2}}{1.0 - 1/c_{e2}}\right)^2\right), 1.0\right) \tag{3.37}$$

$$\theta_{BL} = \frac{\tilde{R}e_{\theta t}\mu}{\rho U}, \quad \delta_{BL} = \frac{15}{2}\theta_{BL}, \quad \delta = \frac{50\Omega\Delta}{U} \cdot \delta_{BL} \tag{3.38}$$

$$Re_\omega = \frac{\rho\omega\Delta^2}{\mu}, \quad F_{wake} = e^{-(\frac{Re_\omega}{1E+5})^2} \tag{3.39}$$

The model constants for the $\tilde{R}e_{\theta t}$ equation are:
$c_{\theta t} = 0.03$ $\sigma_{\theta t} = 2.0$

The transition model interacts with the k-ω SST model modifying the production and destruction terms as follows:

$$\frac{\partial}{\partial x_j}(\rho u_j k) = \tilde{P} - \tilde{D} + \frac{\partial}{\partial x_j}\left((\mu + \sigma\mu_t)\frac{\partial k}{\partial x_j}\right) \tag{3.40}$$

$$\tilde{P} = \gamma_{eff} P \tag{3.41}$$

$$\tilde{D} = min(max(\gamma_{eff}, 0.1), 1.0)D \tag{3.42}$$

$$R_\Delta = \frac{\rho\Delta\sqrt{k}}{\mu}, \quad F_3 = e^{-(\frac{R_\Delta}{120})^8} \quad F_1 = max(F_{1orig}, F_3) \tag{3.43}$$

where P and D are the original production and destruction terms for the SST model and F_{1orig} is the original SST blending function. The production term in the ω is not modified.

3.5 The GPU-enabled CFD Solver PUMA

In this Diploma Thesis, in order to predict the flow field around an airfoil and blade, the compressible GPU enabled flow solver PUMA, developed by the PCOpt/NTUA, is used [24]. PUMA is a software designed for solving Navier-Stokes equations and turbulence model equations using the finite volume method on unstructured grids such as tetrahedra, pyramids, prisms, hexahedra, and even structured grids. Additionally, it includes a suite of shape parameterization tools and computational grid morphers to support shape optimization studies.

The PUMA system leverages the CUDA programming environment and utilizes shared on-board memory for transferring data between GPUs within the same computational node, as well as the MPI protocol for communication between GPUs on different nodes. This system is highly efficient, thanks to its use of the Mixed Precision Arithmetic technique, as described in [24]. This technique involves computing all quantities in double precision, but storing the left-hand side of the equations in single precision in order to decrease memory requirements and improve efficiency, without sacrificing solution accuracy, which only depends on the accuracy of the residuals being computed.

The implementation of PUMA on GPUs provides a significant speed-up compared to CPU-implemented software, reducing the turnaround time of a CFD analysis. Specifically, the implementation on GPUs is approximately several times faster than the CPUs, comparing one card against one core. The GeForce RTX 2070 GPUs were used in this study.

Chapter 4

Optimization using Evolutionary Algorithms - EA

4.1 Introduction to Evolutionary Algorithms

An Evolutionary Algorithm (EA) is a popular optimization approach in nonlinear optimization that uses a population-based search heuristic inspired by Charles Darwin's theory of natural evolution. In this algorithm, the fittest individuals are selected from a population based on a defined objective function, similarly to how natural selection favors the fittest individuals in nature for reproduction. The population is initially initialized using either default or random values, and the algorithm evaluates these individuals to select the fittest members for reproduction using a user-defined reproduction function. This process is repeated until the desired number of iterations are completed, after which the best members of the population, as determined by the objective function, have been found. Although there are various types of EAs that differ in their procedures and purposes, they all have some important similarities in their basic approach.

There are many advantages of evolutionary algorithms over traditional optimization algorithms. Three most notable [25], are:

- **Capability to tackle complex problems, as they do not require any information about the fitness gradient**
  EAs can handle a range of optimization types, including problems with stationary or non-stationary objective functions, linear or nonlinear objectives, continuous or discontinuous objectives, and objectives with random noise. This versatility makes EAs a valuable tool in a wide range of optimization scenarios.

- **Parallelism**
  Multiple offspring in a population function as independent agents. This enables the population or any subgroup to explore the search space in multiple directions simultaneously, making it ideal for parallelization. As a result, various parameters and even different groups of encoded strings can be simultaneously processed. This characteristic of EAs enhances their capabilities as an optimization tool and makes them well-suited for many applications.

- **The ability to escape local minima**
  A situation where deterministic optimization methods might not be effective or even applicable. By searching the search space in multiple directions and utilizing a population-based approach, EAs are able to escape local minima and continue to search for a global optimal solution, making them a valuable tool in complex optimization problems. This characteristic makes EA a valuable option for many optimization scenarios where other methods might not work as effectively.

The flexibility of EAs, which allows a large number of parameters and functions to be selected, can also be seen as one of its disadvantages. When using EAs, it is necessary to carefully choose parameters such as the objective function formulation, the population size of parents and offspring, the reproduction function including the mutation and crossover rate, and the criteria for forming the new population. If these parameters are not chosen optimally, it can lead to difficulties in convergence or result in unmeaningful solutions. Hence, the selection of parameters in EAs requires careful consideration and expertise to ensure optimal performance.

4.2 Basic structure of EAs

The key steps involved in EA include [26]:

1. Select some basic parameters like the size of parents population ($S^{g,\mu}$) and the size of offspring population ($S^{g,\lambda}$) and initialize the $S^{0,\lambda}$ population with random candidate solutions.
2. Evaluate each candidate of the $S^{0,\lambda}$, through an objective function (fitness).
3. Repeat the evolution steps until stopping criterion satisfied:
   (a) Renew the set of elite population ($S^{g,e}$) with the best candidates of $S^{g,\lambda}$ based on the fitness.
   (b) Select candidates from the new elite $S^{g+1,e}$ to replace candidates at $S^{g,\lambda}$ based on the fitness.
   (c) Select the individuals/parents for reproduction ($S^{g+1,\mu}$).
   (d) Recombine pairs and mutate the resulting offspring, creating the new generation of offspring ($S^{g+1,\lambda}$).
   (e) Evaluate the fitness of the $S^{g+1,\lambda}$.

4. Report the best solution corresponding the fittest individual.

The flowchart of an EA is depicted in Figure 4.1.

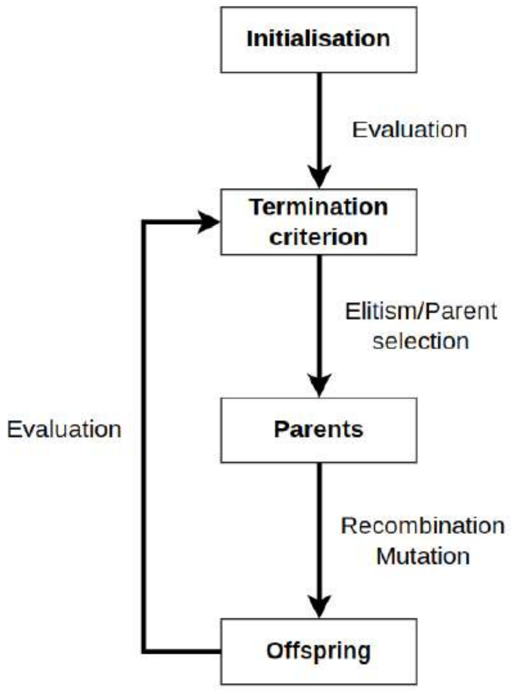


Figure 4.1: *Flowchart of an EA.*

4.3 Constituents of an EA

In this section we discuss evolutionary algorithms in detail. EAs have a number of components, procedures or operators that must be specified in order to define a particular EA. The most important components are [25]:

- Representation (coding of individuals)
- Objective function
- Population
- Parent selection mechanism
- Evolution operators, recombination and mutation
- Survivor selection mechanism (elitism)
- Termination Criterion

Each of these components must be specified in order to define a particular EA. Furthermore, to obtain a running algorithm the initialisation procedure and a termination condition must also be defined.

Representation (Coding of Individuals)

Representation in EAs refers to the encoding of candidate solutions for a problem as a set of variables or parameters (genotype). In the context of optimization problems, these variables are commonly referred to as "optimization" variables. The representation of these variables is a key aspect of EAs, as it determines the way in which candidate solutions are represented within the algorithm and how they can be manipulated and evolved over generations.

In EAs, the encoding of optimization variables is an important design decision that can greatly influence the performance and efficiency of the algorithm in finding optimal solutions. Different encoding methods can be used to represent optimization variables, such as binary[1] and real-valued[2].

Objective function

The objective function, also known as the fitness function, is a crucial component of evolutionary algorithms. The objective function plays a central role in guiding the search process by evaluating the quality of candidate solutions. In optimization problems, the objective function is typically defined as a mathematical function that maps a set of optimization variables (e.g the flow field quantities around an airfoil) to a scalar value (e.g total pressure losses). The objective of the optimization problem is to find the values of the optimization variables that maximize or minimize the objective function.

Population

The population in an evolutionary algorithm is a collection of genotypes that serve as potential solutions. The population, not the individuals within it, is the unit that undergoes evolution. Defining a population involves specifying the number of individuals in it, also known as setting the population size. In most cases, the population size remains constant throughout the evolutionary search process. The population acts as a store of possible solutions, and the choice of population size is an important factor in determining the success of the evolutionary algorithm.

Parent Selection Mechanism

Parent selection or mating selection is a crucial aspect of evolutionary algorithms that helps to determine the quality of individuals and pick the best ones to become parents for the next generation. The goal of this selection mechanism is to push quality improvements in the population by choosing individuals with high quality to undergo variations to create offspring. Parent selection is usually probabilistic, meaning that individuals with better quality have a higher chance of being chosen as parents, but at the same time, low-quality individuals are also given a small probability to be selected in order to avoid getting stuck in a local optimum. Along

[1]Binary encoding is one of the most straightforward encoding methods where solutions are represented as a binary string of 0s and 1s.

[2]In real-valued encoding, solutions are represented as a sequence of real numbers.

with survivor selection, parent selection plays a key role in guiding the direction of the evolutionary search.

Evolution Operators

The role of evolution operators is to create new individuals from old ones. From the generate-and-test search perspective, evolution operators perform the "generate" step. Evolution operators are divided into two types:

- **Mutation**
  The mutation process involves making small, random changes to a single genotype to generate a new offspring. The exact outcome of mutation is determined by the results of a series of random selections, making it a probabilistic and unpredictable process. The purpose of mutation is to avoid getting stuck in local optima, which are regions of the solution space where a candidate solution is good, but not the best. By randomly introducing small changes into the population of candidate solutions, mutation can help the algorithm escape from local optima. Mutation helps maintain diversity in the population of candidate solutions, which is important for the long-term success of the evolutionary algorithm. If the population becomes too homogeneous, the algorithm may converge to a suboptimal solution and stop making progress. By randomly introducing diversity through mutation, the algorithm can continue exploring the solution space and finding better candidate solutions.

- **Recombination**
  The recombination operator in EAs involves combining information from two parent genotypes to produce one or two offspring genotypes. This process is also a stochastic operation and the result depends on random choices made during the merging of information from the two parents. Though it is possible to use more than two parents in the recombination process, this approach is not typically seen in biological systems. The aim of recombination is to create an offspring that incorporates desirable traits from both parents. This allows for the combination of different, yet beneficial, features in a single individual, leading to the generation of a more well-rounded solution to the problem being addressed.

Survivor Selection Mechanism

Survivor selection is a mechanism in EAs that determines which candidate solutions from the current generation will survive and be included in the next generation. The goal of survivor selection is to balance the exploration[3] and exploitation[4] of the solution space, ensuring that the algorithm continues to search for better solutions while also maintaining a diverse population of candidate solutions.

[3]Exploration refers to the strategy of randomly searching the solution space for new candidate solutions that have not been discovered yet.

[4]Exploitation refers to the strategy of using the information that has already been obtained to focus on the best candidate solutions found so far and improve them further.

One common mechanism for survivor selection is elitism, which is a strategy that ensures that the best candidate solution from the current generation is always included in the next generation. This helps to ensure that the algorithm does not move away from the best solution found so far and allows the algorithm to make progress towards the optimal solution. In elitism, the top N best candidate solutions from the current generation are selected to be included in the next generation, where N is a parameter that determines the size of the elite group ($S^{g+1,e}$).

Termination Criterion

Termination criteria in EAs specify the conditions under which the algorithm should stop. There are several termination criteria in EAs. The maximum number of evaluations criterion is utilized in this Diploma Thesis, and it states that the algorithm stops upon completion of the specified number of evaluations. Alternatives commonly used options as termination conditions are the following:

- The maximum allowed CPU time elapses.
- For a given period of time, the cost improvement remains under a threshold value.
- The population diversity drops under a given threshold.

4.4 Metamodel-Assisted Evolutionary Algorithms (MAEA)

The implementation of a conventional EA can be quite resource-intensive, as it requires numerous calls to the problem-specific evaluation software for each generation. To mitigate this, a metamodel-assisted EA can be employed to reduce the CPU cost of the optimization. This approach involves creating low-cost surrogates, also known as a metamodels, which are trained on the fly on previously evaluated individuals to pre-evaluate the evolving populations. This significantly reduces the number of calls required to the evaluation software and enhances the overall performance of the optimization.

It is worth noting that metamodels can be applied to both single-objective optimization problems (SOO) [27], as well as, multi-objective evolutionary algorithms (MOEAs) [28]. The ability to use metamodels to pre-evaluate populations allows for more efficient and effective optimization, regardless of the type of problem being solved. Since, the metamodel training requires a minimum amount of previous evaluations, the starting population is evaluated on the problem-specific model.

4.5 The Evolutionary Algorithms SYstem - EASY software

EASY [29], created by the PCOpt/NTUA, is a versatile software tool that is used to find optimal solutions to problems that can have a single or multiple objectives, with or without constraints. It combines stochastic, deterministic, or a combination of optimization methods and has been widely applied in various engineering fields. EASY provides users with a high degree of control over the optimization process, and also offers presets for those who are new to the software. The tool supports the approximation of single and multi-objective functions using Artificial Neural Networks, which is especially useful when the problem at hand requires a lot of computational time. Additionally, EASY offers a wide range of options, including the use of genetic algorithms or evolution strategies, and the ability to incorporate metamodels, which significantly reduces cost when dealing with computationally intensive evaluation software.

In a conventional (μ, λ) EA, where μ refers to the number of parents and lambda refers to the number of offspring, the cost of evaluating each generation is as high as λ calls to the problem-specific software. However, MAEA begins just like a conventional EA by evaluating the initial randomly generated population using the problem-specific tool and storing the paired inputs and outputs in a database. In subsequent generations, the population members are pre-evaluated using either fitness inheritance or local metamodels. The first few generations rely on fitness inheritance, and once the database reaches a minimum size specified by the user, local metamodels are employed instead. These models are trained on-the-fly, separately for each new individual, based on a few entries properly selected from the database. The most promising offspring population members in each generation are, then, re-evaluated using the problem-specific tool. The evolution operators in the MAEA use both exact and approximate scores to create the next generation [28].

In this Diploma Thesis, EASY was employed for the MAEA optimization of DNN architecture and inputs and the MAEA aerodynamic optimizations.

Chapter 5

Replacing Turbulence Model by DNNs in the Optimization of an Isolated Airfoil

The following steps summarize the subsections of the chapter, which outline the process of replacing the SA turbulence model by DNNs in a MAEA-based (shape) optimization of an isolated airfoil with target to minimize CD, keeping CL close to its baseline. The idea behind this concept is to examine the potential of a DNN to replace the SA turbulence model (without transition), predicting the turbulence viscosity field (μ_t), in order to reduce the computational cost of the optimization.

- **The NACA4318 Case**. In this subsection the NACA4318 airfoil profile is plotted, and the flow conditions imposed to PUMA are presented.
- **Parameterization and training data creation**. The shape of the NACA4318 airfoil is altered to create a variety of blade shapes that will make up the training database for the DNN (DB_{DNN}). Afterwards, the DB_{DNN} undergoes additional processing to improve its performance when used to train the DNN.
- **Validation of the DNN configuration**. The hyper-parameters of the DNN were determined through a trial-and-error process. The performance of the DNN was evaluated by testing it on a group of 10 new airfoil shapes, with the aim of determining its ability to generalize and make predictions about unseen data. This step was crucial in ensuring that the DNN has the capacity to not only accurately model the training data, but also to make accurate predictions for new and unseen data points.

- **Shape optimization studies**. This subsection describes a shape optimization process for the NACA4318 shape using MAEA implemented using EASY. The optimization is evaluated using two different software, the ”standard” model in which the RANS equations are coupled with the SA turbulence model (without transition) within PUMA (PUMA-TM model), and a model where the RANS equations solver is coupled with the DNN (Figure 5.1) within PUMA (PUMA-DNN model). A comparison and discussion of the results is conducted.

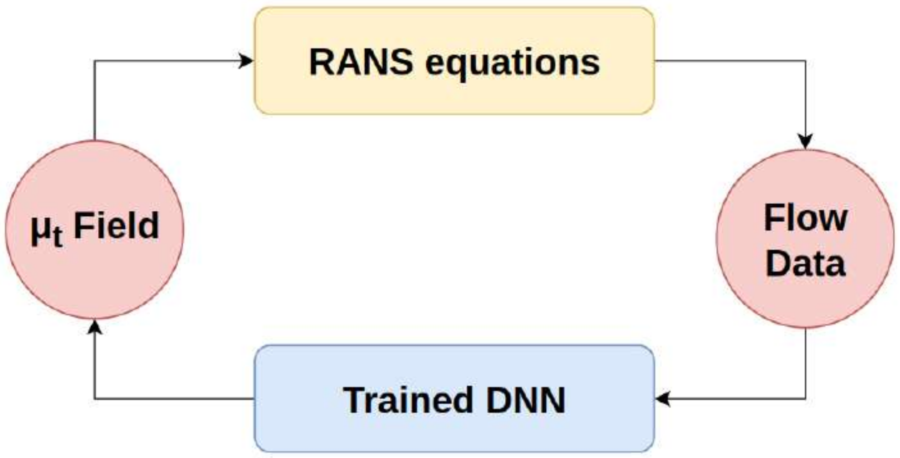


Figure 5.1: *An iterative solver of the RANS equations using the DNN to predict the μ_t field instead of SA turbulence model. Each iteration of the loop corresponds to a single pseudo-time iteration of the PUMA-DNN model.*

The overall procedure in order for the MAEA, using PUMA-DNN as the evaluation software, to optimize the NACA4318 airfoil shape is shown in Figure 5.2.

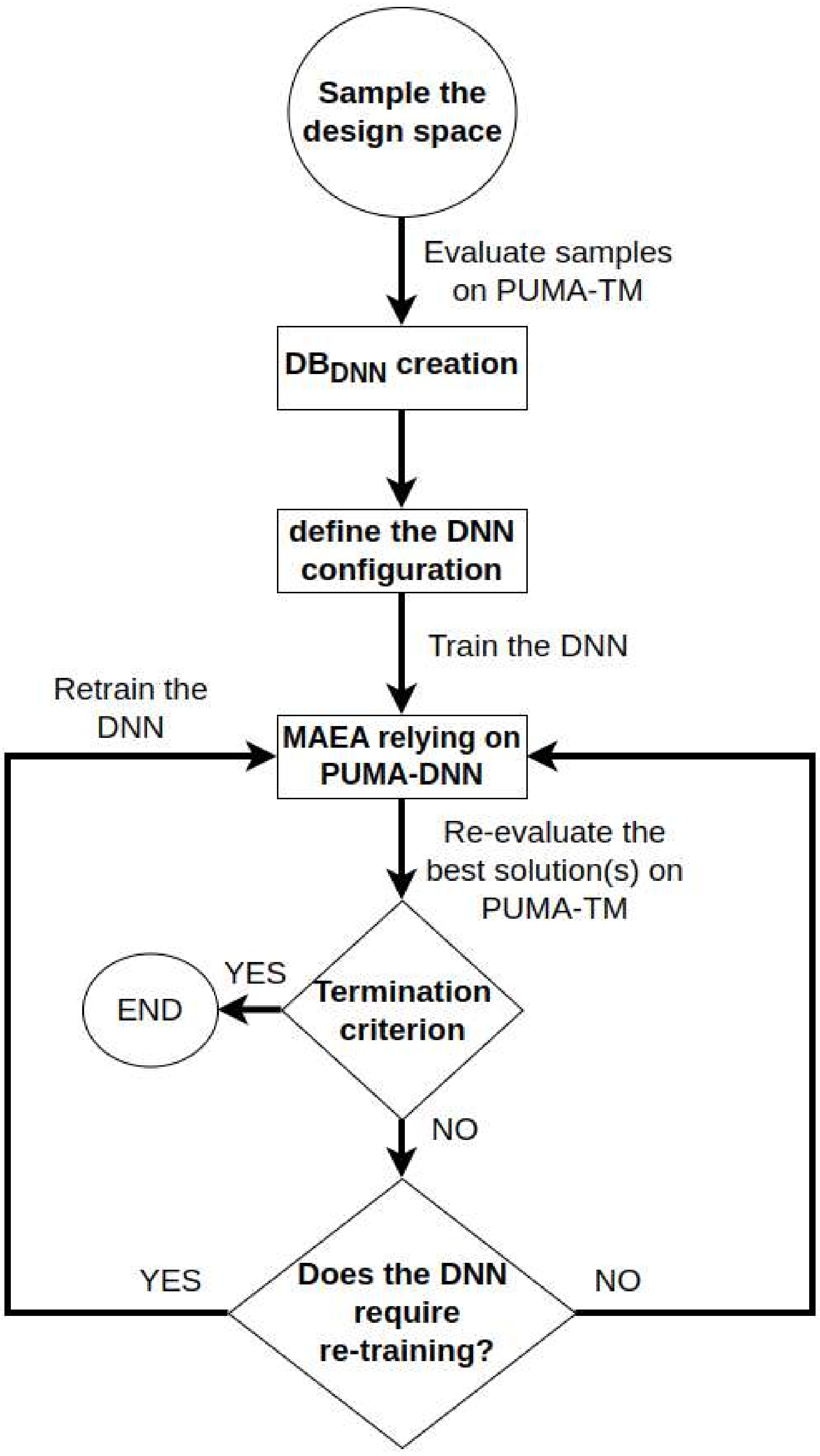

Sample the design space
Evaluate samples on PUMA-TM
DB$_{DNN}$ creation
define the DNN configuration
Train the DNN
Retrain the DNN
MAEA relying on PUMA-DNN
Re-evaluate the best solution(s) on PUMA-TM
YES
END
Termination criterion
NO
YES
Does the DNN require re-training?
NO

Figure 5.2: *The algorithm of the procedure.*

5.1 The NACA4318 Airfoil Case

In this Diploma Thesis, the NACA4318 airfoil is studied. The airfoil profile is plotted in Figure 5.3.

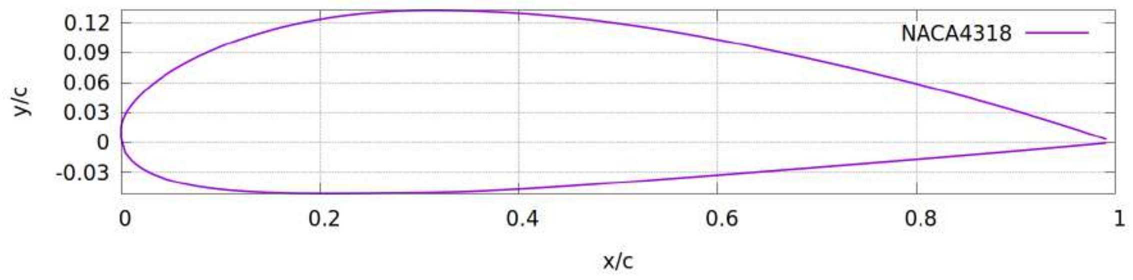


Figure 5.3: *NACA4318 shape*

The flow conditions, imposed in PUMA, are listed in Table 5.1.

Flow conditions	
M_∞	0.19
Angle α_∞ [degrees]	2.8
Reynolds number	$5.46 \cdot 10^6$

Table 5.1

5.2 Parameterization and Training Data Creation

The training database of the DNN (DB_{DNN}) is constructed by altering the geometry of the NACA4318 airfoil and computing the flow fields using the PUMA-TM model. This dataset will serve as the basis for training the DNN to predict turbulence viscosity.

The airfoil shape is controlled by the 5x3 NURBS lattice of Figure 5.4. The control points in blue are allowed to move only in the normal-to-the-chord direction within $\pm 10\%$ of their reference values. The 2 control points in red at the leading and trailing edge are fixed. Thus, there are 13 design variables in total. The same parameterization is utilized for the MAEA shape optimization of section 5.4.

A Latin Hypercube Sampling (LHS) is used to sample the design space of 13 variables by generating 60 different geometries. Then, the flow fields are computed by PUMA-TM, thus forming the DB_{DNN} to be used for training the DNN. The geometry of 60 different airfoil shapes, is shown in Figure 5.5.

A single CFD run takes a bit less than 4 mins on a single NVIDIA GeForce RTX

2070. Using a database of 60 different geometries to train the DNN, the total CFD run takes a bit less than 4 hours on the same GPU.

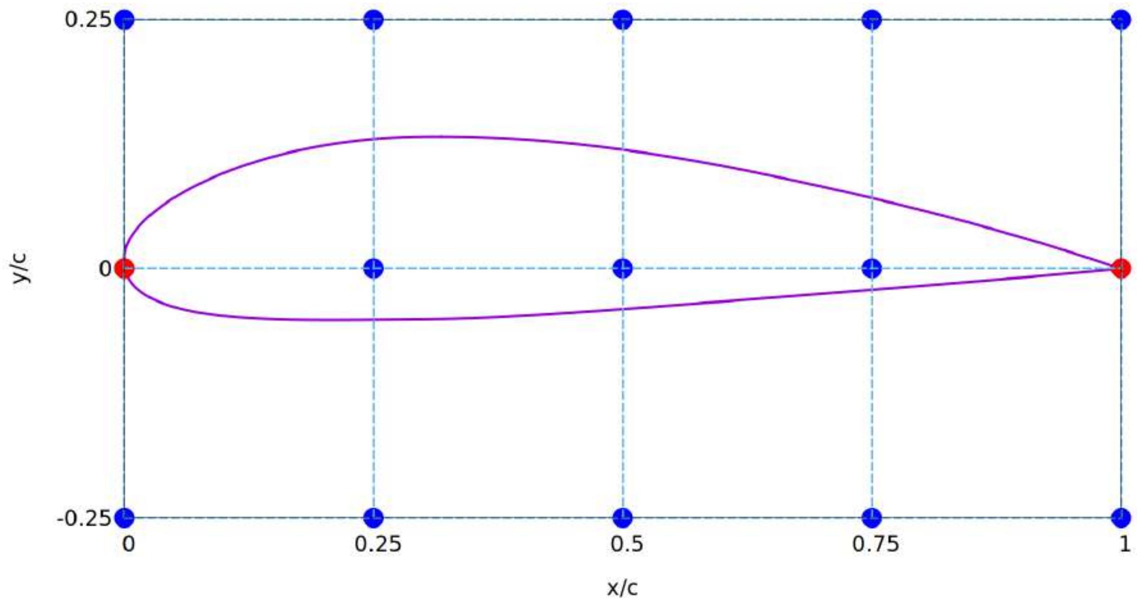


Figure 5.4: *Parameterization of the airfoil shape. Blue control points are allowed to move in the normal-to-chord direction only, while red points are kept fixed.*

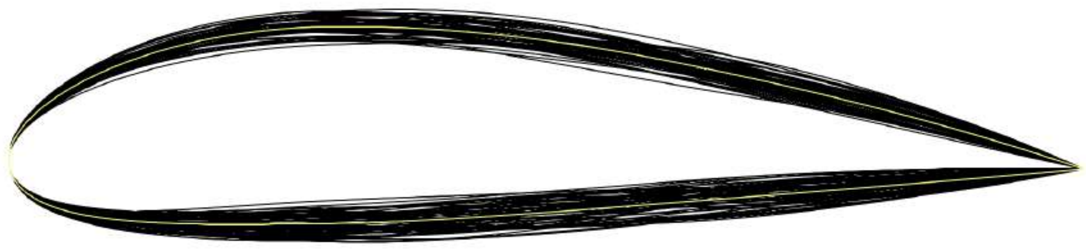


Figure 5.5: *The geometries of the 60 airfoils in DB_{DNN} colored in black and the baseline geometry colored in yellow. It shows the area in which the new airfoil shapes can be created.*

The geometrical and flow data computed for each and every grid node and stored in the DB$_{DNN}$, are listed in Table 5.2.

• Coordinates x_k =(x,y)	Input
• Density ρ	Input
• Pressure P	Input
• Velocity vector $u_k = (u, v)$	Input
• Turbulent viscosity μ_t	Output

Table 5.2: *The data that are inputs and outputs of the DNN. The DNN predicts only the turbulence viscosity.*

DNNs are trained to learn the relationships between input variables and output variable (the μ_t). However, because the input variables have different scales, this can pose a challenge for the network. This is because the parameters associated with each input will exist on different scales, leading to vanishing error gradients[1] or exploding error gradients[2] in the back-propagation algorithm, which can hinder the performance of the network.

To address this issue, normalizing the data is recommended. One way of normalization is a rescaling of the data so that all values be within 0 and 1. This have the following benefits:

- It can help to improve the performance of activation functions, such as the sigmoid or tanh, by preventing saturation and improving the convergence of the network [30].
- It can simplify the learning process for the network by removing the need to adjust the parameters of the activation function to handle different ranges of inputs [31].

The data are rescaled by the following equation:

$$x_j = \frac{X_j - min(Q_i)}{max(Q_i) - min(Q_i)} \tag{5.1}$$

where Q_i represents an array with all the values of i-th geometrical or flow quantity in the DB$_{DNN}$, $max(Q_i)$ and $min(Q_i)$ are the maximum and minimum values of

[1]The vanishing gradients problem occurs when the gradients of the loss function with respect to the parameters of the network become too small during the training process. This makes it difficult for the parameters to update effectively, leading to slow convergence and a suboptimal solution.

[2]Exploding gradients problem occurs when the gradients of the loss function with respect to the parameters of the network become too large during the training process. This can cause the parameters to update in an uncontrolled manner, leading to numeric instability and, ultimately, to the failure of the training process.

i-th quantity in the whole DB$_{DNN}$, X_j represents each value of Q_i and x_j is the normalized value of X_j that belongs in the interval [0,1].

Because the Rectified Linear Unit (ReLU) activation function is used in the DNNs training, rescaling the data to the interval [0,1] instead of [-1,1] is considered the best option for training a DNN. This is because the derivative of ReLU with respect to its input is 1 for positive values and 0 for negative values. Therefore, if the input to a ReLU neuron is negative, its gradient with respect to the weights and biases will be zero, and the neuron will stop learning. This can result in a significant portion of the data being disregarded and not utilized to train the DNN. By normalizing the data to the interval [0,1], the DNN is able to better utilize all the available data, improving the training process and ultimately leading to more accurate predictions.

5.3 Validation of the DNN configuration

The development and training of DNNs is carried out in the TensorFlow framework using Python. Table 5.3 summarizes the DNN configuration utilized for this case. The loss during DNN training is shown in Figure 5.6, where it can be seen that, after the 600 epochs, it is unlikely for the losses to decrease any further thus, the training procedure stopped. It is apparent that the validation loss and training loss are nearly identical.

Table 5.3

Optimal DNN configuration obtained from a trial-and-error approach.

Layers	Neurons/Layer	Input data	Act.Functions
5	64, 128, 256, 2048, 512	$(x_k, \rho, p, u_k,)$	ReLu/tanh

As the goal of PUMA-DNN in a shape-optimization procedure is to minimize CD, it is more important and practical to assess the generalization performance and errors based on the computed CD, rather than just the predicted turbulence viscosity. Thus, all the errors are computed with respect to the CD values. The generalization capability of the PUMA-DNN was assessed by generating 10 additional airfoil shapes using the same process used to create the DB$_{DNN}$. The CD values computed by both PUMA-TM and PUMA-DNN for these new shapes are shown in Figure 5.7. The predictions made by the PUMA-DNN show a consistent tendency to overestimate CD.

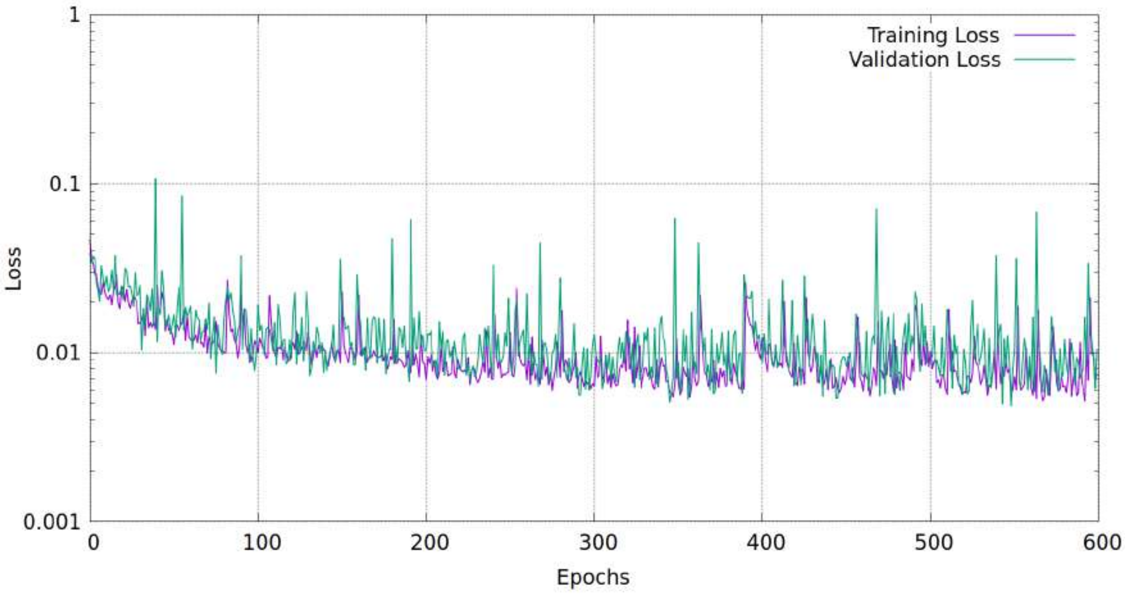


Figure 5.6: *Training and Validation loss during the training of the DNN. There is no indication of overfitting, since the training and validation loss are converging to similar values. The training loss cannot decrease any further and training procedure stops at 600 epochs.*

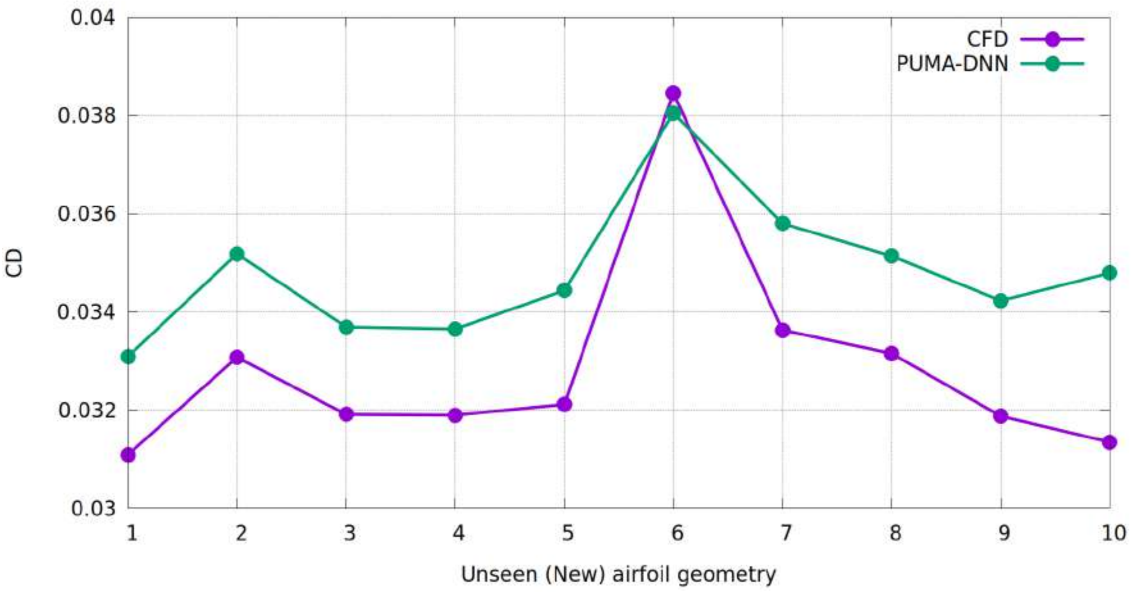


Figure 5.7: *The computed CD values of the 10 new airfoils by PUMA-TM and PUMA-DNN. The PUMA-DNN predictions exhibit a nearly consistent overestimation of CDs, with the exception of the 6th geometry, which though has the worst CD value. Despite differences in the predicted CD values, the relative gain or loss between any two of these geometries is correctly predicted by PUMA-DNN; a small exception regarding the 10th solution is practically of no importance.*

The percentile errors of computed CDs by PUMA-DNN are calculated as:

$$error\ [\%] = \frac{CD^j_{PUMA-TM} - CD^j_{PUMA-DNN}}{CD^j_{PUMA-TM}} \cdot 100 \tag{5.2}$$

where j represents the index of the 10 new airfoil shapes and takes on values of [1,2,..,9,10]. The errors of PUMA-DNN are shown in Figure 5.8, where it can be seen that the average error is 6.3%, with a minimal variation in error values.

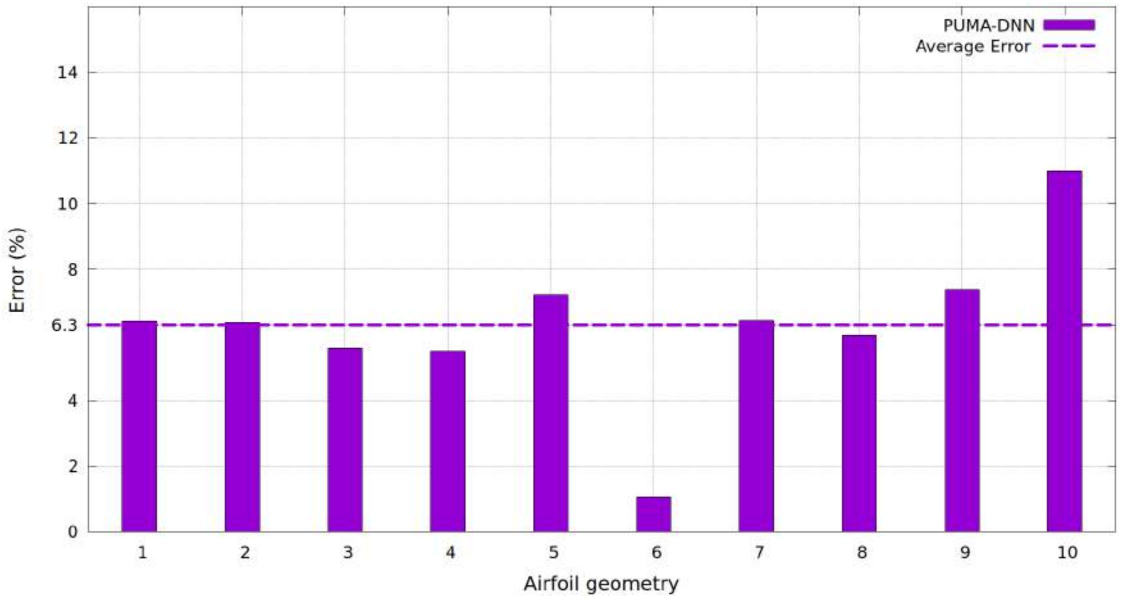


Figure 5.8: *The errors of PUMA-DNN; the average error is 6.3 % with a minimal variation in error values.*

5.4 Shape Optimization Studies

In the shape optimization process using MAEA-based optimization in EASY, two different evaluation software namely, the standard PUMA-TM and PUMA-DNN were used. The MAEA optimization was configured to have $\mu = 10$ parents and $\lambda = 30$ offspring. The target is to minimize the CD while ensuring that the Lift Coefficient (CL) remains close to its baseline value. The CD and CL of the baseline (evaluated by PUMA-TM) are: CD$_{baseline}$ = 0.03196 and CL$_{baseline}$ = 1.51.

PUMA-TM as evaluation software

To assess the independence from the randomly generated number seed (RNG seed) of MAEA, the optimization process was repeated three times for three different

RNG seeds (RNG_1, RNG_2, and RNG_3). The convergence history of the MAEA-based optimization using PUMA-TM is shown in Figure 5.9, where it can be seen that the choice of RNG seed has an impact on the optimization outcome, given that it was decided to terminate the optimization run quite early in order to save computational cost. Therefore, to compare with the PUMA-DNN model, the results of these three optimizations are averaged. One cost unit (or time unit) is defined as the cost (time) of one evaluation using the PUMA-TM (i.e a pure CFD evaluation). The termination criterion using PUMA-TM model, was set to 350 evaluations.

PUMA-DNN as evaluation software

The second evaluation software used in the MAEA (shape) optimization was the PUMA-DNN, which has lower cost per evaluation, with one evaluation costing 0.8 cost units as opposed to one cost unit for the PUMA-TM. The objective function and constraints remained unchanged. The DNN didn't require retraining, and the optimization process was terminated after 440 evaluations, which roughly correspond to 350 cost units. The results of the optimization procedure are summarizes in Table 5.4, where the results of PUMA-TM are averaged.

Table 5.4

Results of the Optimization procedure

Evaluation software	Best solution (CD/CL)	Re-evaluated (CD/CL)	Percentage % of CD reduction
PUMA-TM (averaged)	0.0309/1.506	-	-3.46%
PUMA-DNN	0.03288/1.506	0.0307/1.508	-3.91%

The convergence histories of the MAEA relying on PUMA-DNN and PUMA-TM using three different RNG seed are shown in Figure 5.10, where it can be noticed that the PUMA-DNN evaluation software results in faster convergence compared to optimizations using PUMA-TM with RNG_2 and RNG_3, however, convergence is similar when using RNG_3.

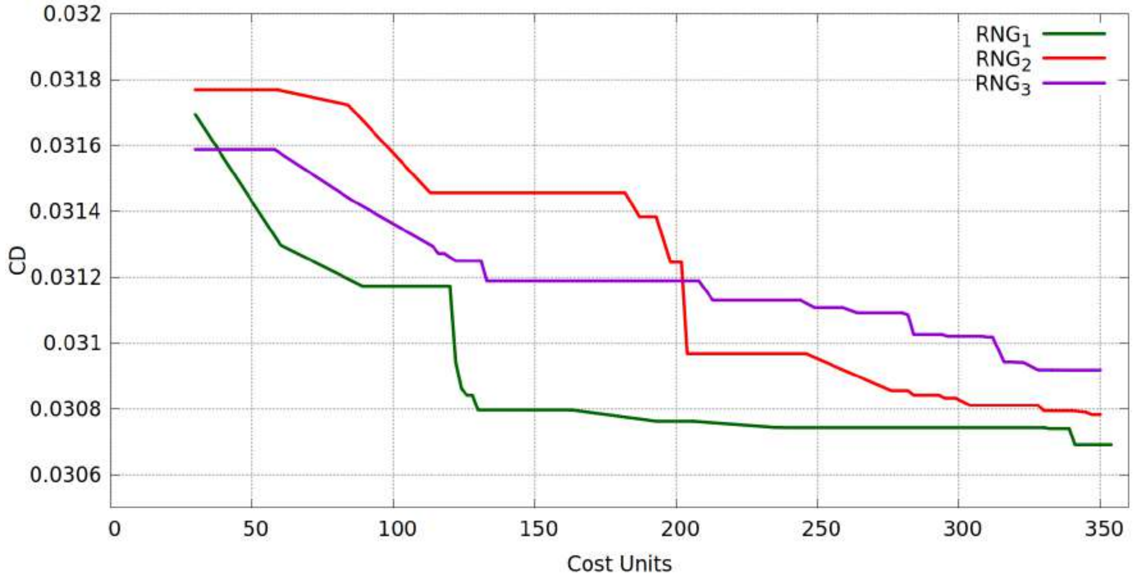


Figure 5.9: *The convergence history of the MAEA-based optimization relying on the PUMA-TM.*

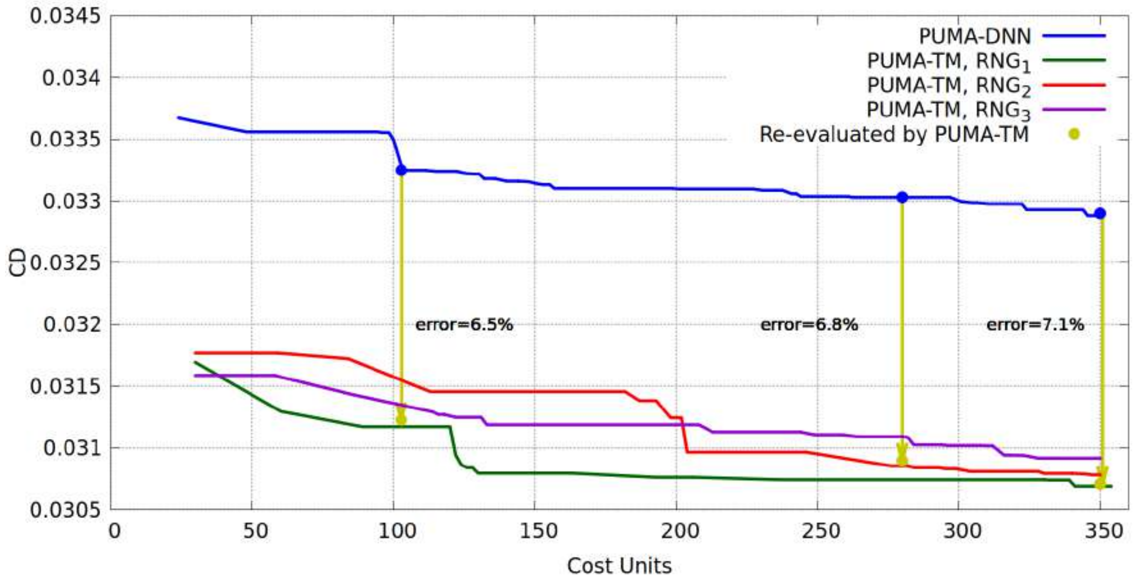


Figure 5.10: *The convergence histories of the MAEA-based optimization relying on the PUMA-DNN and PUMA-TM, using three different RNG seed. The PUMA-DNN evaluation software results in faster convergence compared to optimizations using PUMA-TM with RNG$_2$ and RNG$_3$, however, convergence is similar when using RNG$_3$.*

The shapes of the best solutions obtained from the MAEA optimization relying on both the PUMA-TM and PUMA-DNN are shown in Figure 5.11.

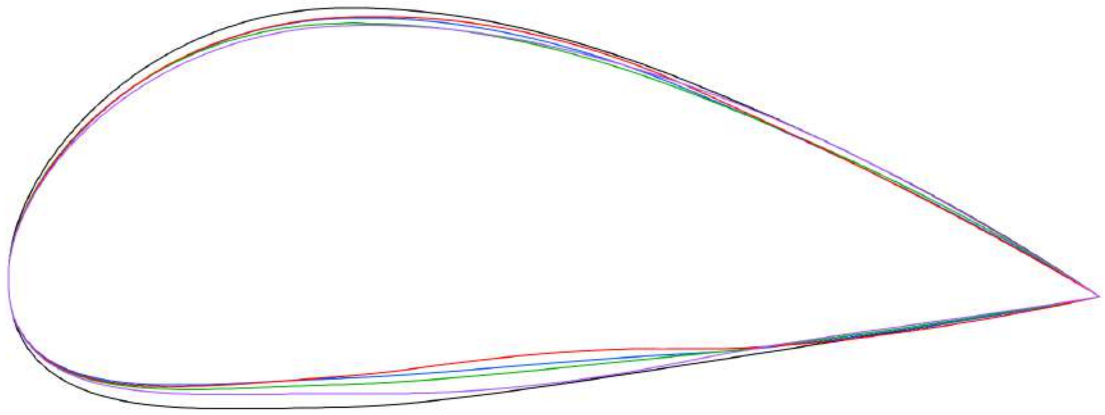


Figure 5.11: *The best results of PUMA-TM using RNG$_1$ (green), RNG$_2$ (red), and RNG$_3$ (violet), the best result of PUMA-DNN (blue) and the baseline (black) are displayed. It is evident that the shapes of PUMA-TM (RNG$_1$) and PUMA-DNN are the most alike. The y-axis is scaled by 2 to enhance the visibility of the variations in shape.*

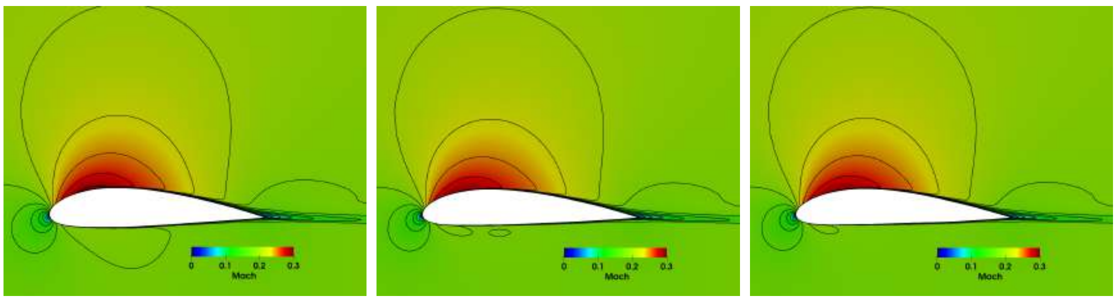


Figure 5.12: *Baseline's Mach number iso-areas (left), best's solution obtained by PUMA-TM (RNG$_1$) (center) and best's solution obtained by PUMA-DNN (right) models, evaluated by PUMA-TM.*

5.5 Conclusions

The presented studies show that the DNN can help in the shape optimization process using an EA. The results from using the PUMA-DNN model in shape optimization studies are encouraging, even though the cost of the PUMA-TM run is quite small (it is a 2D case with SA turbulence model). Although there is only a slight improvement in the overall time of the optimization considering the averaging performance of PUMA-TM using the different RNGs, this serves as an initial strong indication that the gain can be even greater in cases where evaluations are more costly, such as in unsteady or 3D cases.

Chapter 6

Replacing Turbulence and Transition Models by DNNs in a Turbine Blade Shape Optimization

The following steps summarize the subsections of the chapter, which outline the process of replacing the turbulence and transition model by DNNs in a shape optimization with target to minimize the total pressure losses ($P_{t,losses}$) while ensuring that the exit angle remains close to its baseline value.

- **The LS89 case**. The LS89 case is described, defining the convective heat transfer coefficient and the flow conditions that are imposed in PUMA.
- **CFD analysis of LS89 blade - Validation**. The configurations of PUMA are optimized with the aim to meet the experimental data of the heat transfer coefficient, thereby highlighting the crucial role played by the $\gamma - \tilde{Re}_{\theta t}$ transitional model.
- **Parametrization and training data creation**. The LS89 turbine blade airfoil shape is parameterized to generate a diverse set of blade shapes that will constitute DB_{DNN}. The DB_{DNN} is then subjected to post-processing to enhance its effectiveness during the DNN training process.
- **Optimization of DNN architecture and Inputs**. This subsection outlines the optimization of the DNN's hyper-parameters and inputs using a Single Objective MAEA based optimization in EASY. However, since the optimization procedure does not assess the generalization ability of the DNNs, a separate evaluation is performed to identify the best DNN configuration from the set

provided by the optimization that has both low losses and good generalization capabilities.

- **Regularization of the optimal DNN configuration**. In this subsection, the use of regularization techniques such as dropout and early stopping is applied to further reduce the losses of the selected DNN, while simultaneously improving its generalization performance and numerical stability when coupled with the RANS solver within PUMA.

- **Shape optimization studies**. This subsection describes a shape optimization process for the LS89 blade airfoil using the MAEA based optimization in EASY. The optimization is assessed using two different software, a standard model where the RANS equations' solver is coupled with the turbulence and transition model withing PUMA (PUMA-TM), and a PUMA-DNN model where the RANS solver is coupled with the DNN within PUMA. Also a comparison and discussion of the results is conducted.

6.1 The LS89 turbine case

The LS89 turbine nozzle guide vane is studied [32]. The blade profile is plotted in Figure 6.1.

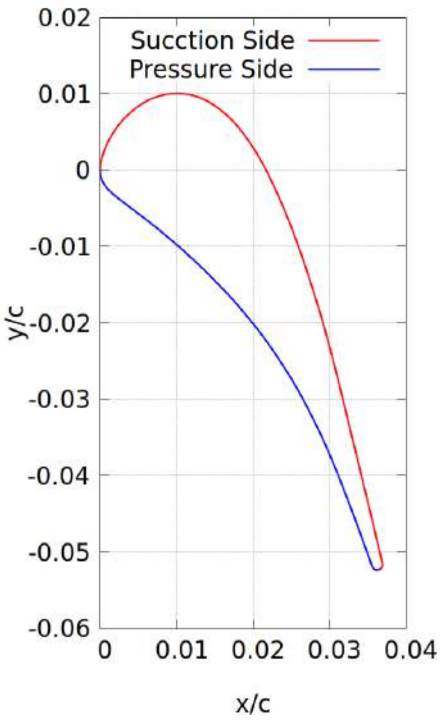


Figure 6.1: *LS89 turbine airfoil*

One of the measured quantities of the test number MUR132 at VKI's investigation [32], that is compared, in section 6.2, with the numerical results of LS89 flow field computed by PUMA, for validation, is the distribution of the convective heat transfer coefficient H, which is defined as the ratio of the measured wall heat flux and the difference between the total free stream and the local wall temperatures:

$$H = \frac{\dot{q}_w}{T_{t,\infty} - T_w} \tag{6.1}$$

The units of H are $(W/m^2/K)$. The flow conditions for test number MUR132, imposed in PUMA, are listed in Table 6.1.

Free stream conditions for MUR132		
	Inlet Conditions	Outlet Conditions
Total Temperature [K]	408.50	
Total pressure [bar]	1.757	
Pressure [bar]		1.289
Flow angle [degrees]	0	

Table 6.1

6.2 CFD Analysis of LS89 blading - Validation

In order to optimize the shape of the LS89 blade airfoil, it's good to find the optimal configuration of PUMA that meets the experimental data of the heat transfer coefficient with a constant wall temperature of 299.75 K, and inlet and outlet conditions as listed in Table 6.1.

The heat transfer coefficient is greatly affected by the location of transition over the suction side. Thus, the LS89 case is studied with the $k - \omega\ SST$ turbulence model assisted by the $\gamma - \tilde{Re}_{\theta t}$ transition model. The importance of the latter can be seen in Figure 6.2 that compares the heat transfer coefficient over the pressure and suction sides of the blade airfoil, with and without the transition model, computed by PUMA.

To achieve the minimum error of the computed heat transfer coefficient by PUMA, a variety of spatial derivative computation techniques and discretization schemes have been tested. The comparison of the heat transfer coefficient computed using two different spatial derivative computation techniques (P1 elements[1] [33], Weighted

[1]P1 elements, also known as linear finite elements, are a type of discretization technique commonly used in CFD simulations. They are defined by linear interpolation functions and are used

Least Squares[2] [34]), is shown in Figure 6.3, where it can be noticed that P1 elements meet slightly better the experimental data.

The comparison of the computed heat transfer coefficient using three different discretization schemes (Roe [35], JST scalar, and JST matrix [36]), is shown in Figure 6.4, where it can be noticed that the Roe scheme gives the best prediction. Thus, the use of P1 elements and Roe discretization scheme with the $k-\omega$ SST turbulence model assisted by the $\gamma - \tilde{Re}_{\theta t}$ transition model, is the optimal PUMA configuration that reproduces the experimental distribution of the heat transfer coefficient.

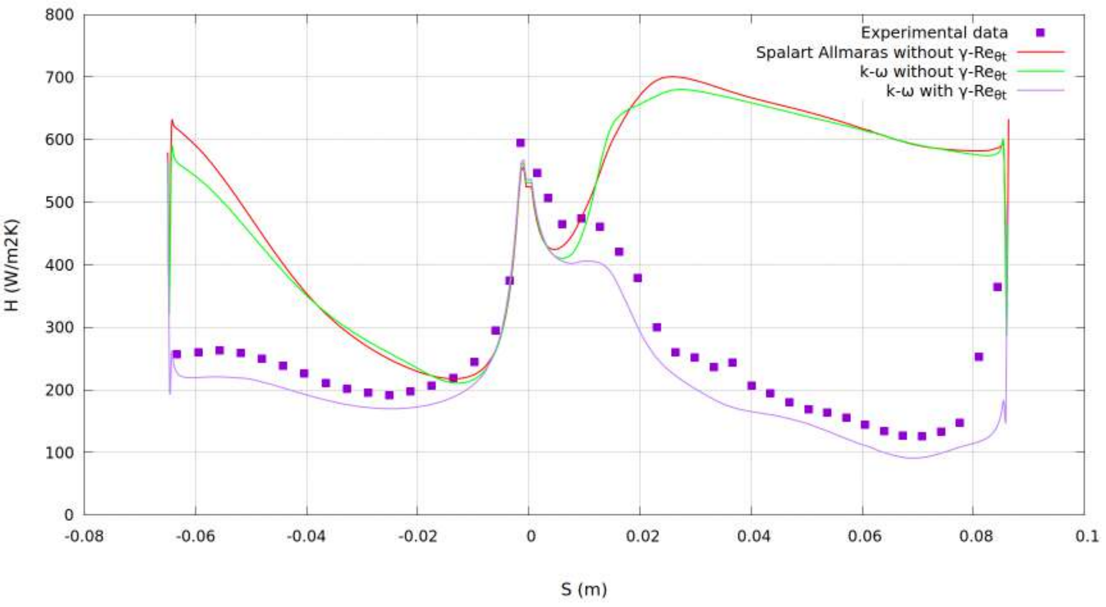


Figure 6.2: *Comparison of the heat transfer coefficient computed using PUMA with and without the $\gamma - \tilde{Re}_{\theta t}$ transition model with experimental data. Important differences can be seen when the transition model is omitted.*

to represent the solution field (e.g., velocity, pressure, temperature) in a CFD simulation. P1 elements are used to discretize the governing equations (such as the Navier-Stokes equations) into a system of algebraic equations that can be solved numerically.

[2]Weighted Least Squares (WLS) is a statistical estimation method that is used to minimize the sum of the squared residuals between the observed data and the predicted values by the model. In the context of CFD, WLS can be used as a spatial derivative computation technique to estimate derivatives of flow fields.

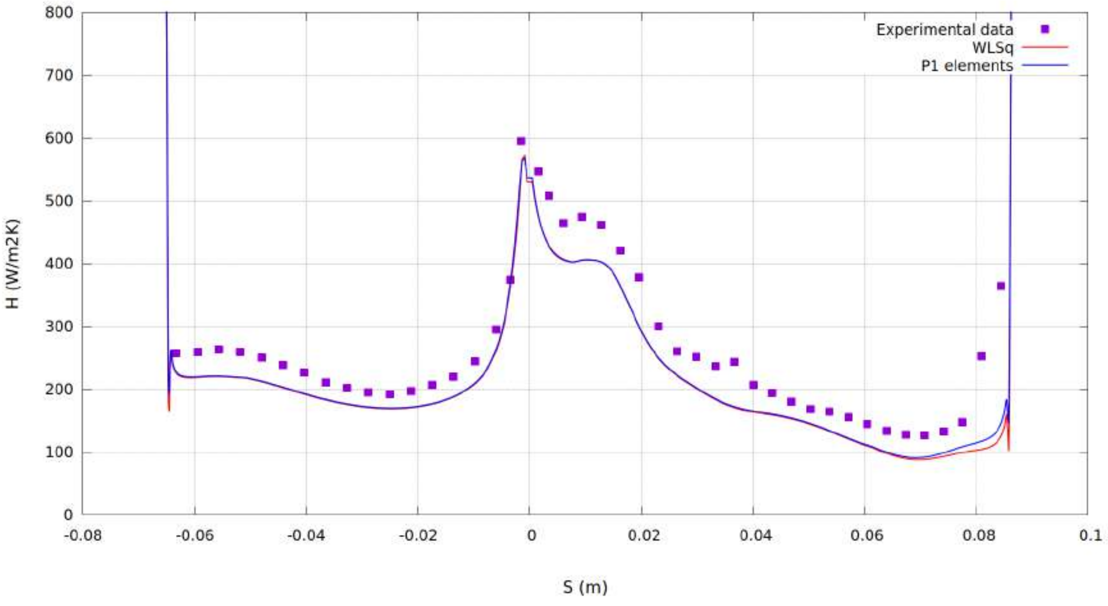


Figure 6.3: *Comparison of the heat transfer coefficient computed using PUMA with two different spatial derivatives' computation techniques: WLSq and P1 elements.*

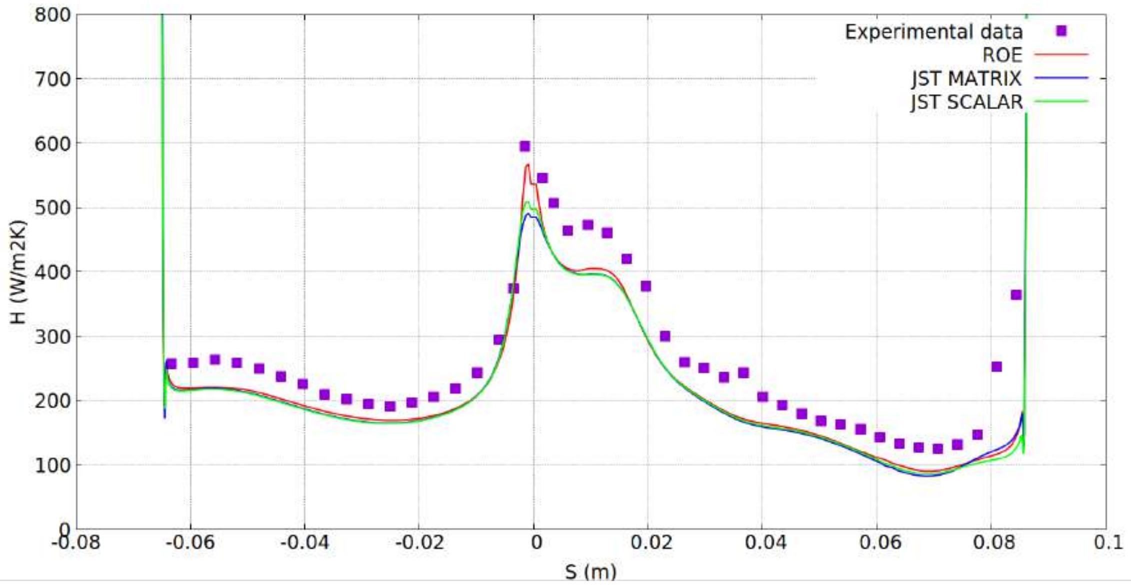


Figure 6.4: *Comparison of the heat transfer coefficient computed using PUMA with three different discretization schemes: Roe, JST matrix and JST scalar.*

6.3 Parameterization and Training Data Creation

The goal is to replace the turbulence and transition model with a DNN that predicts the turbulence viscosity. This goal can be achieved by first creating a training dataset. This dataset can be generated by varying the geometry of the LS89 blade and computing the flow fields with the RANS, which are coupled with the turbulence and transition model. Having this dataset would serve as the foundation for training the DNN to accurately predict the turbulence viscosity.

The turbine blade airfoil shape of LS89 is controlled by the 8x5 NURBS lattice of Figure 6.5. The control points on blue circles are allowed to move in the chordwise and the normal-to-the-chord directions within ±10% of their reference values. The 10 control points in red circles along the left and right are fixed. Thus, there are 60 design variables in total. The same parameterization was selected for the MAEA optimization of section 6.6.

The Latin Hypercube Sampling (LHS), which generates 150 near-random samples, is used to sample the design space of 60 variables in order to generate a set of different geometries. Then, the flow fields are computed by PUMA using the $k - \omega\ SST$ turbulence model assisted by the $\gamma - \tilde{Re}_{\theta t}$ (to be referred as the PUMA-TM model), creating the database used for training the DNN (DB_{DNN}). The geometries of the 150 blades is shown in Figure 6.6.

A single CFD run takes $\sim$ 10 mins, on a single NVIDIA GeForce RTX 2070. Using a database of 150 different geometries to train the DNN, the total CFD run to collect 150 evaluated samples takes less than 26 hours.

The geometrical and flow data were computed for each and every grid node and stored in the DB_{DNN}, are listed in Table 6.2.

• Coordinates (x,y)	Input
• Density ρ	Input
• Pressure P	Input
• Velocity vector $\vec{U} = (u, v)$	Input
• Vorticity	Input
• Strain Rate	Input
• Wall Distance	Input
• Turbulent viscosity μ_t	Output

Table 6.2: *Inputs and outputs in the training process of the DNN. The DNN predicts only the turbulence viscosity, thus it is the only output.*

The obtained data are rescaled by Equation 5.1, from the original range so that all values are within the range of 0 and 1, similar to the NACA4318 case (see section 5.2).

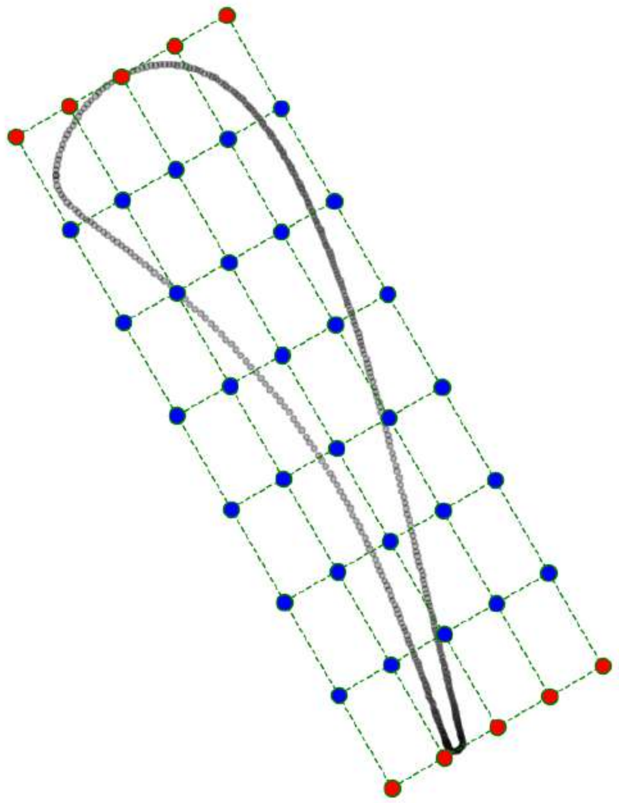


Figure 6.5: *Parameterization of the LS89 turbine blade airfoil. The control points marked with blue circles are allowed to move in the chordwise and the normal-to-the-chord directions within $\pm 10\%$ of their reference values. The 10 control points in red circles along the left and right are fixed.*

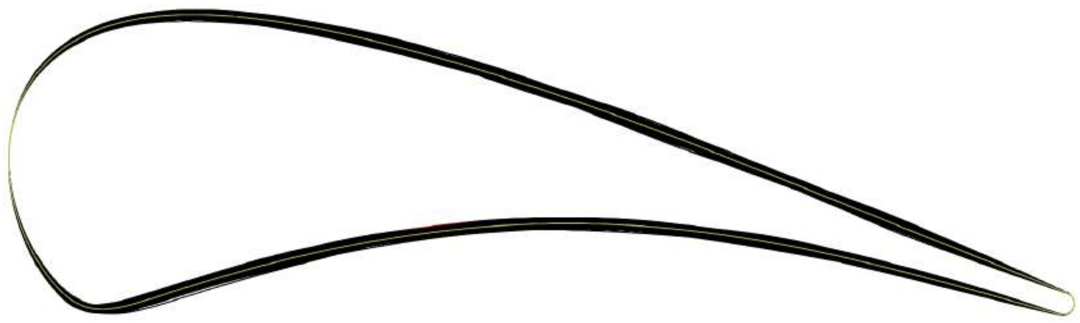


Figure 6.6: *The geometries of the 150 blades in DB_{DNN} colored with black and the baseline geometry colored with yellow. It broadly shows the area in which the new blade shapes can move.*

The target of the MAEA optimization of the turbine blade airfoil at section 6.6 is to minimize the total pressure losses ($P_{t,losses}$). In the DB_{DNN}, shapes with lower $P_{t,losses}$ than the baseline, already exist. Thus, it is essential to know the minimum $P_{t,losses}$ in the DB_{DNN} in order to compare it with the minimum $P_{t,losses}$ obtained by the MAEA optimization relying on the PUMA-DNN model. The total pressure losses of the different geometries in DB$_{DNN}$ are shown in Figure 6.7, where it can be seen that the minimum value recorded is 1378 Pa while the value of the baseline is 1469 Pa.

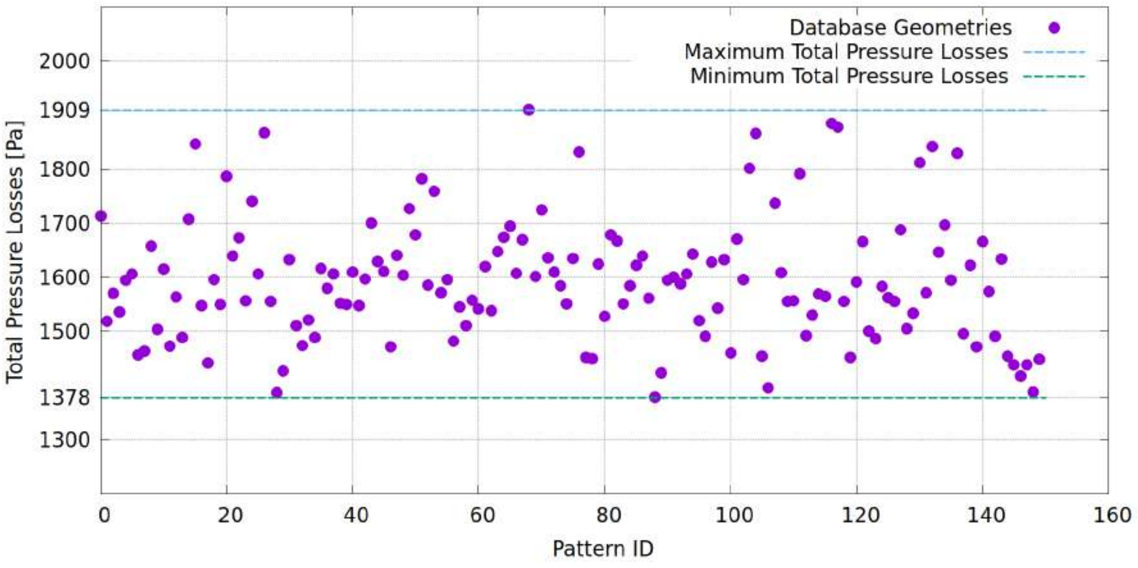


Figure 6.7: *Total pressure losses of each patterns of the DB$_{DNN}$.*

6.4 Optimization of DNN Architecture and Inputs

The optimization of DNN hyper-parameters and inputs is still an active area of research in the field of ML. Currently, the most common method for optimizing a DNN is through experience and trial-and-error. Another approach is to use genetic algorithms, which are automatic and require less human intervention in finding the optimal solution. In this case, the optimization of DNN requires a targeted optimization approach due to the complexity of the PDEs being replaced. To minimize the prediction error of the DNN, the optimal DNN configuration is achieved through a MAEA by using EASY. The target is to minimize the prediction error of the DNN by optimizing the following design variables:

- The pool of possible quantities (Table 6.2).
- The number of hidden layers.
- The number of neuron units at each hidden layer.
- Activation functions for hidden and output layers.

For the number of hidden layers, the lower and upper bounds are set to 3 and 10 layers respectively. For each layer (hidden or output), the number of neurons is defined in powers of 2 with the lower and upper bounds to be set at 2^5 and 2^{12} units respectively. The ReLu, tanh, and sigmoid activation functions can be selected for each layer. During the optimization procedure, the DNN used the 20% of the DB_{DNN} data for validation, and the dataset was randomly shuffled to avoid any overfitting in training by passing samples in different orders making the model more robust. Also, the ADAM optimizer was used.

The overfitting problem

A way to asses the generalization ability of a DNN during the training is by using a validation set. The validation set is a portion of the training data that is held out and not used during the training process. Instead, the model's performance on the validation set is used to monitor the model's progress during training and to avoid overfitting. The validation loss is the loss computed on the validation set. It represents the difference between the predicted outputs and the true outputs for the validation set. During training, the validation loss is used to monitor the model's performance on the validation set and to determine if the model is overfitting or underfitting. If the validation loss is increasing, the model is overfitting.

Upon evaluating 44 different DNNs, the MAEA was stopped due to the observation that the best performing DNNs were starting to overfit the training data when tested on new unseen geometrie. This occurred despite no indications of overfitting being detected through monitoring the model's performance (validation loss) on the validation set during training. The reliability of validation loss as an indicator of overfitting in DNNs that are coupled with PDEs can depend on various factors and the specifics of the problem at hand. Some studies have reported that validation loss can provide useful information on overfitting in these types of problems, while others have found that it may not accurately reflect the generalization ability of the model. For example, a study by Raissi et al. [37], used a physics-informed neural network to model PDEs and found that the validation loss did not necessarily provide reliable information on overfitting. In conclusion, while validation loss can be a useful indicator of overfitting in some cases where a DNN prediction is coupled with PDEs, the reliability of this indicator may depend on various factors and the specifics of the problem. In this case, the validation loss fails to accurately reflect overfitting, therefore, it becomes difficult to gauge the generalization performance during the optimization process, and the only way to assess it is by testing the PUMA-DNN model on new geometries not seen during training.

It was crucial to test the optimal DNN configurations on new geometries to determine which DNN had the lowest losses and generalized well simultaneously. This step was necessary to ensure that the DNN selected would not only perform well on the training data but also on unseen data, making it a more robust and reliable model for predicting turbulence viscosity and therefore for computing the total pressure losses in order to minimize them in the shape-optimization procedure of section 6.6.

As the goal of using a DNN coupled with RANS in a shape-optimization procedure is to minimize total pressure losses, it is more important and practical to assess the generalization performance and errors based on the computed total pressure losses, rather than just the predicted turbulence viscosity. Thus, all the errors are calculated with respect to the computed total pressure losses. Testing the DNNs on new geometries, it was possible to make an informed decision on which DNN was best suited for the task, and could generalize well to new and unseen data.

Selection of DNNs to be tested

Because the best-so-far DNN configuration, which has 10 layers, was overfitting, testing a smaller one was a good practice. Overfitting occurs when a DNN model is too complex and performs well on the training data but poorly on the test data. By reducing the size of the DNN, for example, by decreasing the number of layers or reducing the number of neurons in each layer, it may be possible to reduce overfitting. A study by Liu et al. [38], found that reducing the number of layers in a DNN can lead to improved generalization performance and prevent overfitting. The authors showed that smaller DNN models with only a few hidden layers could achieve similar or even better performance compared to larger DNN models.

As a result, two DNN configurations were selected from the optimization process based on two criteria. The first and most important criterion was to have fewer hidden layers than the best-performing DNN so far. The second criterion was to have the best performance compared to other DNNs with the same number of hidden layers. These two criteria led to the chosen DNN configurations that are summarized in Table 6.3. These are marked with the letters A, B, and the best-so-far DNN with the letter C (referred to as DNN$_A$, DNN$_B$, and DNN$_C$).

Table 6.3

Optimal DNN configurations to be tested.

Optimal DNN	Layers	Neurons/Layer	Input data	Act.Functions
DNN$_A$	3	256, 2048, 2048	$(x, \rho, p, d, u_k, \Omega, S)$	ReLu/tanh
DNN$_B$	3	128, 64, 2048	(y, ρ, p, d, u_k, S)	ReLu/tanh
DNN$_C$	10	256,2048,4096,1024,32,64,4096,256,256,1024	$(x_k, \rho, p, d, u_k, \Omega, S)$	ReLu/tanh

The best performing DNN of each generation in the MAEA is shown in Figure 6.8, where it can be seen that DNN$_A$, DNN$_B$, and DNN$_C$ are the best performing DNNs for the first, second, and fourth generation respectively.

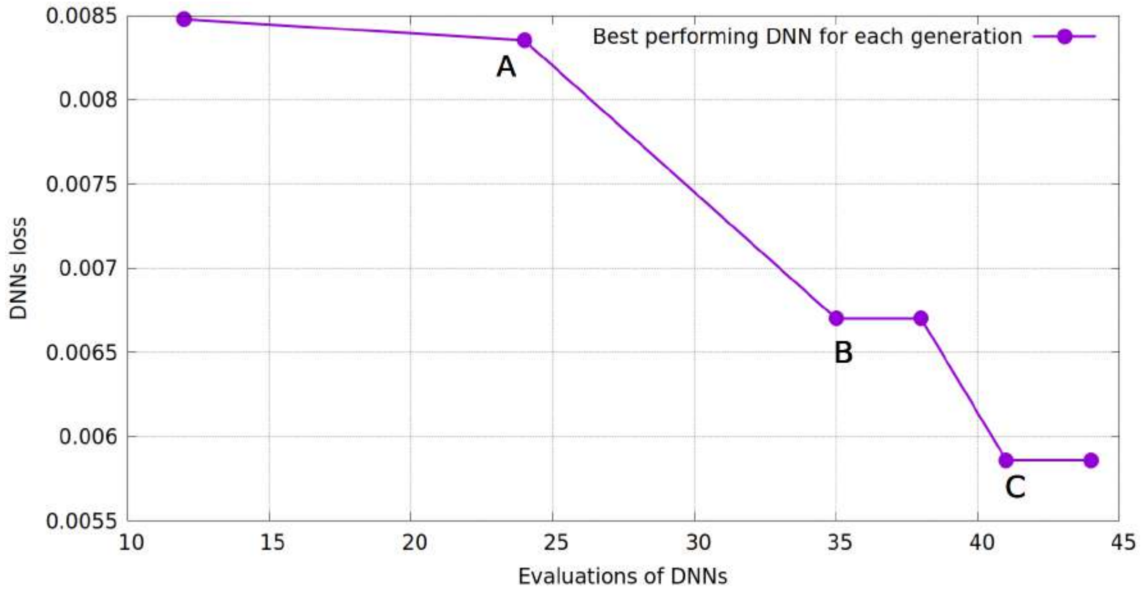


Figure 6.8: *The best DNN of each generation.*

In the following testing results, the DNN$_C$ results are included in order to demonstrate its overfitting and allow for comparison with the DNN$_A$ and DNN$_B$ configurations. The DNNs have been trained for a certain number of epochs, and it has been observed that the loss function has stopped improving significantly after reaching a certain level. This is known as a plateau in the loss function, where the rate of improvement in the loss function slows down and eventually stops.

As a result, it has been decided to stop the training process at 600 epochs, as further iterations are unlikely to result in any significant improvement in the performance of the DNNs. This approach is commonly used to prevent overfitting, where the model becomes too specialized to the training data and fails to generalize well to new data.

Testing DNN$_C$

The loss during the training of DNN$_C$ model is shown in Figure 6.9. The plot shows that the validation loss is consistently higher than the training loss, with no obvious indications of overfitting. However, when the DNN$_C$ model was tested on 10 new blade geometries within PUMA (to be referred as PUMA-DNN$_C$), it was discovered that the model was overfitting the data and did not generalize well. Despite this, the DNN$_C$ model was found to provide similar numerical stability as the PUMA-TM. These results indicate that while the DNN$_C$ model performed well on the training data, it did not generalize well to new and unseen data, making it a less robust model for predicting turbulence viscosity.

Testing DNN$_B$

As previously observed, the validation loss consistently remained larger than the training loss with no signs of overfitting (Figure 6.10). However, when the PUMA-DNN$_B$ model was tested on new blade geometries, overfitting was revealed. Additionally, the numerical solution was found to converge with difficulty, as 6 out of the 10 new geometries flows diverged. This indicates that while the DNN$_B$ model performed well on the training data, it did not generalize well to new and unseen data, resulting in overfitting and a lack of robustness in the numerical solution. These results highlight the importance of evaluating the performance of the DNNs model on new geometries to ensure it generalizes well and produces accurate predictions.

Testing DNN$_A$

The testing of the PUMA-DNN$_A$ showed a significant increase in prediction accuracy compared to previous DNN models. The implementation of dropout effectively addressed the divergent behavior of the numerical solution in two out of ten test geometries (see section 6.5). This not only improved the numerical stability, but also led to enhanced generalization capabilities for the model. As a result, DNN$_A$ is considered the best configuration for the DNN. The loss during the training of DNN$_A$ model is shown in Figure 6.11.

The need of testing the best solutions generated by the MAEA based optimization, is due to the limitation of validation loss as an indicator of overfitting. This does not mean that the optimization process was unnecessary or inefficient, as it provides a collection of high-performing DNN configurations, obtained faster compared to a trial and error method. These DNNs can be selected for testing and evaluation in order to find the optimal DNN configuration that can generalize well simultaneously.

The results of the comparison of the absolute percentage errors of the computed total pressure losses by the PUMA-DNN$_A$, PUMA-DNN$_B$, and PUMA-DNN$_C$, are presented in figure Figure 6.12. Some of the error bars for PUMA-DNN$_B$ and PUMA-DNN$_A$ are missing due to the numerical instability in the solution. However, after evaluating all three configurations, it is concluded that the DNN$_A$ configuration provides the best results, with the smallest average error and generalizing better than the other two DNNs. Additionally, PUMA-DNN$_A$ is twice as fast as PUMA-TM. The error is calculated as:

$$error_{ij}(\%) = \left| \frac{P_{tlosses}^{PUMA-TM} - P_{tlosses}^{PUMA-DNN_j}}{P_{tlosses}^{PUMA-TM}} \right|_i * 100 \tag{6.2}$$

where $P_{tlosses}^{PUMA-TM}$ are the computed total pressure losses by PUMA-TM at i_{th} testing geometry, $P_{tlosses}^{PUMA-DNN_j}$ are the computed total pressure losses at i_{th} testing geometry by PUMA-DNN$_j$ with j=A,B,C.

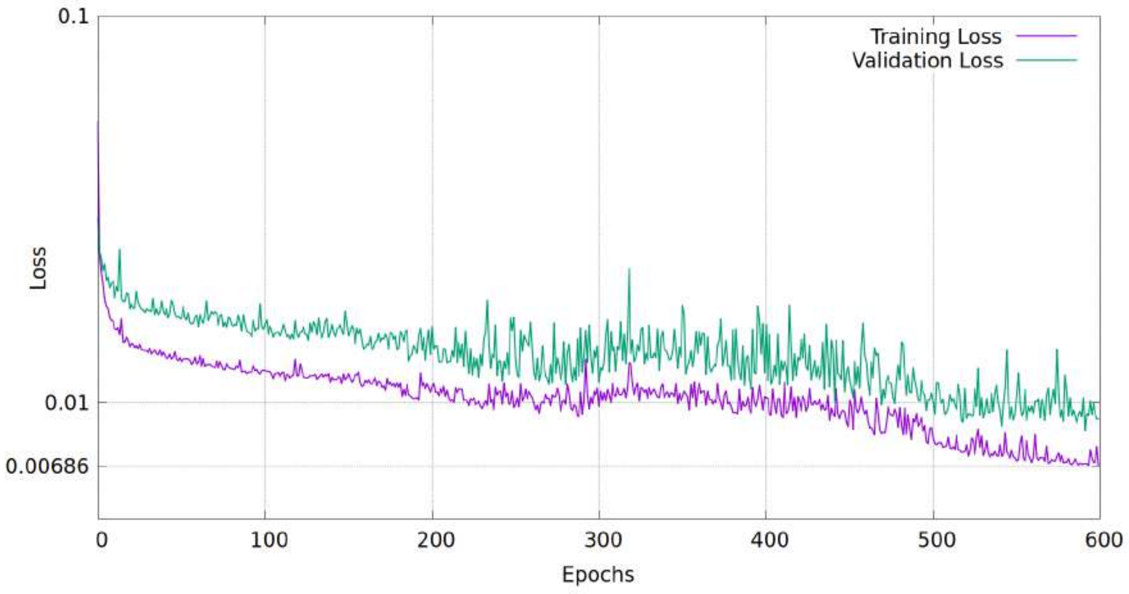


Figure 6.9: *The training and validation loss during the training of DNN_C. It observed that the DNN_C trained for 600 epochs, overfits the training data, when tested on new unseen geometries. As a result, even though the training loss could potentially be reduced further, stopping the training at 600 epochs was recommended not to damage generalization.*

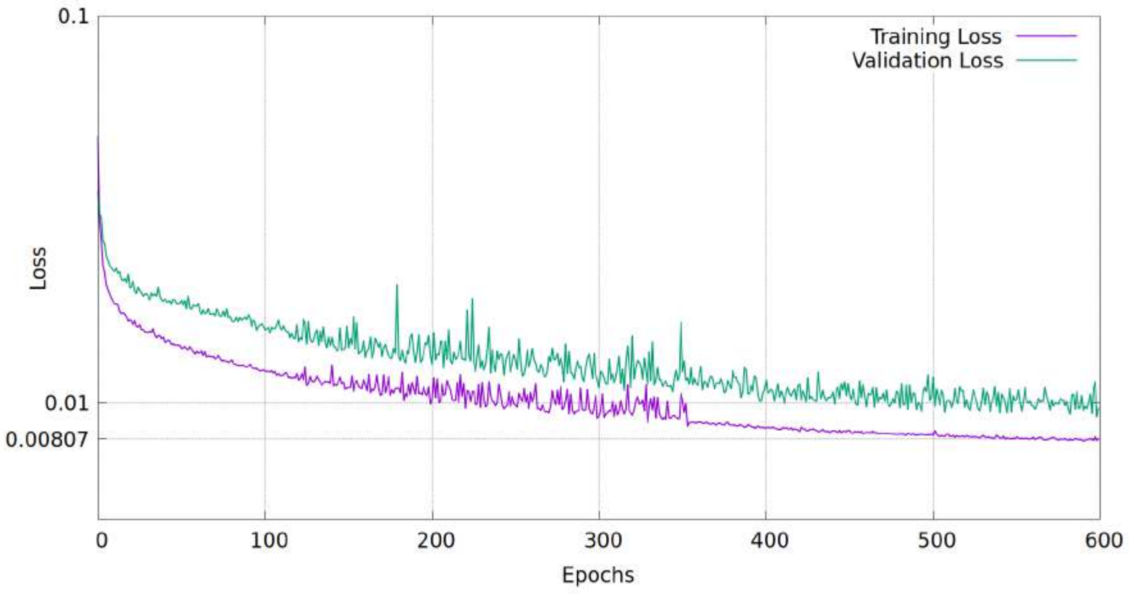


Figure 6.10: *The training and validation loss during the training of DNN_B. The validation loss is consistently higher than the training loss, with no obvious indications of overfitting.*

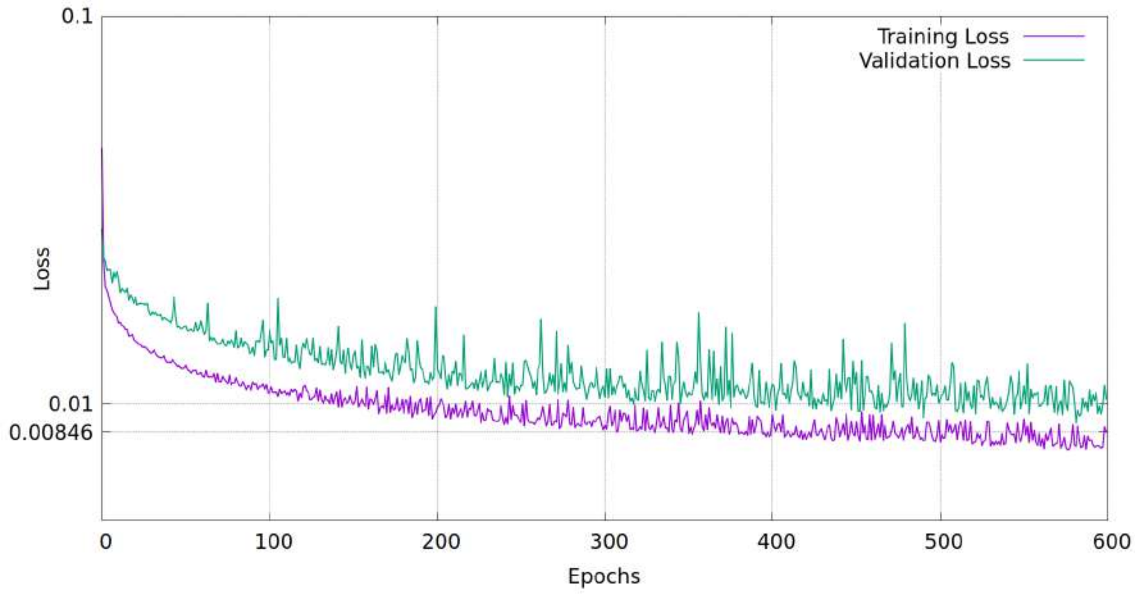


Figure 6.11: *The training and validation loss during the training of DNN$_A$. The validation loss is consistently higher than the training loss, with no obvious indications of overfitting.*

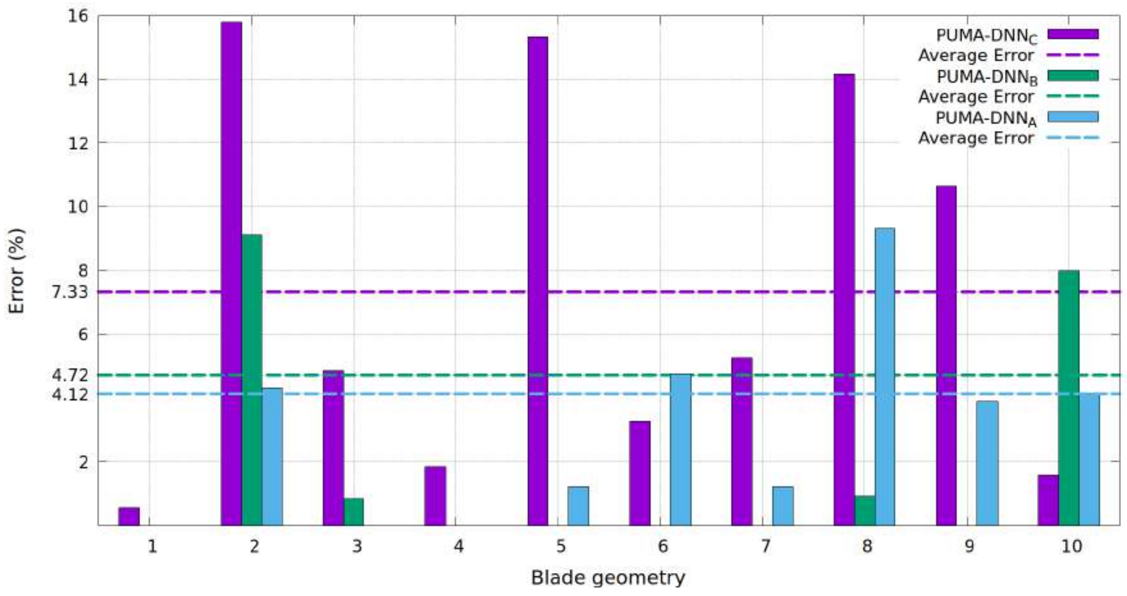


Figure 6.12: *The absolute percentage errors of PUMA-DNN$_A$, PUMA-DNN$_B$ and PUMA-DNN$_C$ on new blade geometries. It can be seen that the PUMA-DNN$_A$ have the smallest average error and generalizes better than the other two DNNs. The problem of PUMA-DNN$_A$ with the numerical instability in the solution, since 2 out of 10 geometries flow diverged, was addressed effectively through the implementation of dropout. The optimal DNN configuration is the DNN$_A$.*

The Mach number iso-areas computed for the baseline geometry by the PUMA-TM and PUMA-DNN$_A$ models and the relative error between the two fields are shown in ??.

6.5 Regularization of the Optimal DNN configuration

Many techniques have been developed to combat overfitting in DNNs, including early stopping, weight penalties (L1 and L2 regularization), soft weight sharing, and dropout.

In this Diploma Thesis, it was found that using dropout in the optimal DNN configuration (DNN$_A$), improves the PUMA-DNN predictions, reduces overfitting and boosts the numerical stability of PDEs (see Figure 6.14). Different implementations on the hidden layers tested with three possible probabilities for each unit to retrain (0.1, 0.2, 0.4) and found that dropping out the units of the last two hidden layers with probability of 0.2, is the optimal. The errors of PUMA-DNN predictions on new blade geometries, with and without dropout, are shown in Figure 6.15; dropout reduces the average error by 1.13% while resolving the divergence issue when coupling RANS with the DNN.

Further overfitting reduction can be achieved by stopping the training process at the point where the validation loss stops improving (early stopping). This was found to occur at about 400 epochs, as can be seen in Figure 6.16. The PUMA-DNN errors on new blade geometries, with the DNN trained for 400 and 600 epochs are shown in Figure 6.17, where it can be seen that there is a reduction in the average error by 0.59% while the DNN trained for 400 instead of 600 epochs.

While dropout improves PUMA-DNN stability, some geometries may still show oscillatory behavior on flow-field quantities during the numerical solution of PUMA-DNN. In this Diploma Thesis, it was found that calculating the error with respect to the average value of the computed total pressure losses over the final X iterations (1000 iterations found to be optimal) of the PUMA-DNN, instead of just the final value, reduced the error by an additional 0.3% and improved the generalization performance of the PUMA-DNN as the errors were closer to the average value. The effect of averaging is shown in Figure 6.18.

Finally, the DNN$_A$ errors without using any of the aforementioned techniques (Initial DNN$_A$) and the DNN$_A$ errors using them (Final DNN$_A$), are shown in Figure 6.19, where it can be seen that it was achieved a reduction in total error by 2.02%, addressing at the same time the divergence problem.

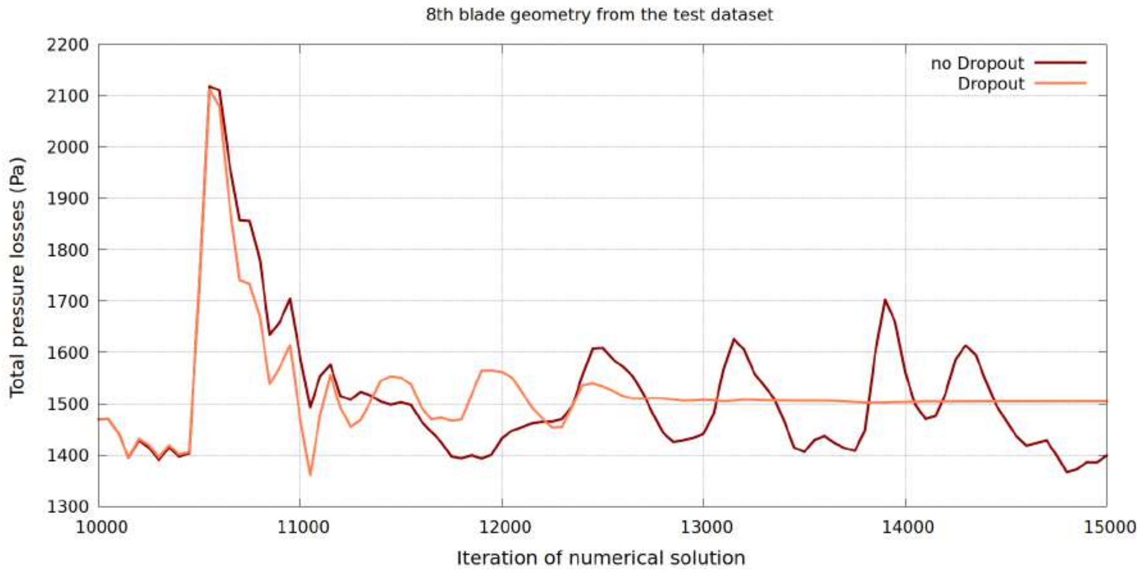


Figure 6.14: *Total pressure loss per Iteration* ($5 \cdot 10^3$ *iterations in total) computed by PUMA-DNN with and without dropout. It's obvious that dropout boosts numerical stability.*

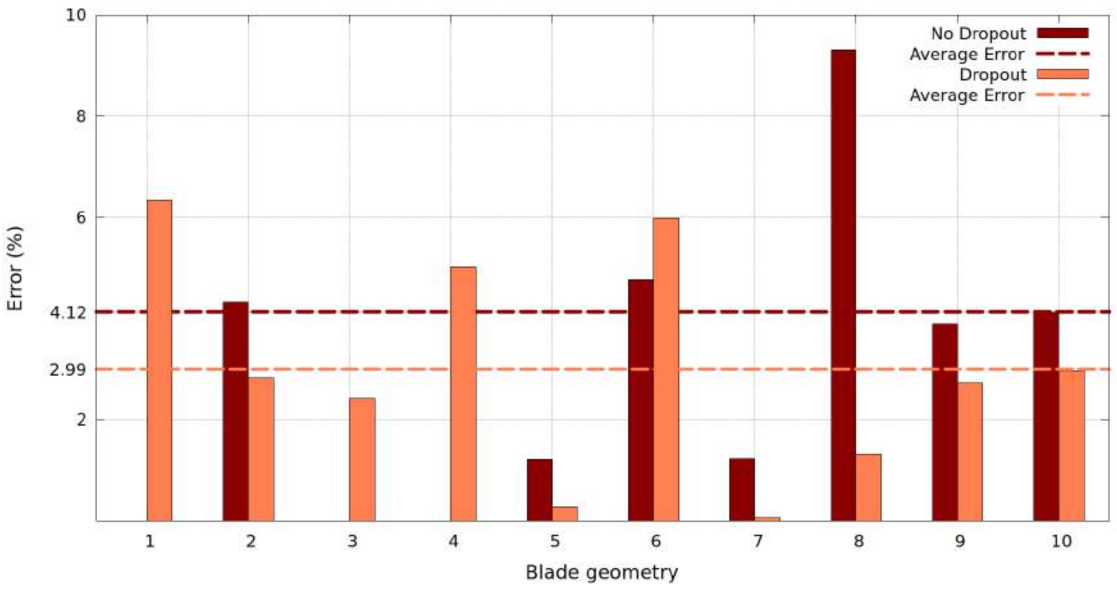


Figure 6.15: *The errors of PUMA-DNN predictions on new blade geometries, with and without dropout. It can be seen that dropout reduces the average error by* 1.13% *while resolving the divergence issue when coupling RANS with the DNN at geometry IDs: 1, 3 and 4.*

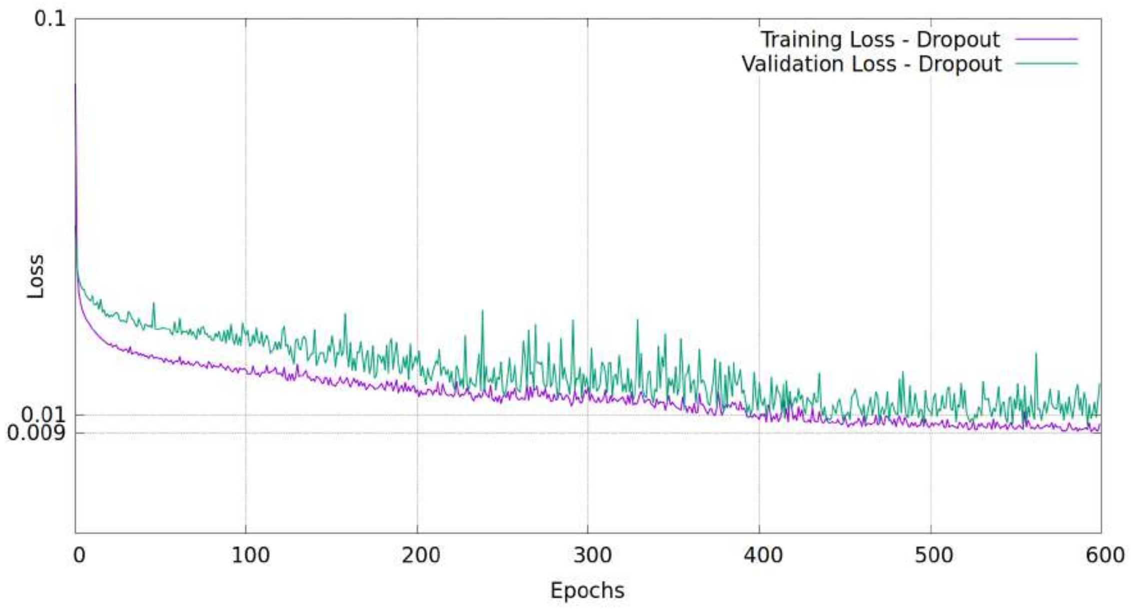


Figure 6.16: *The training and validation loss during the training process of optimal DNN configuration using Dropout.*

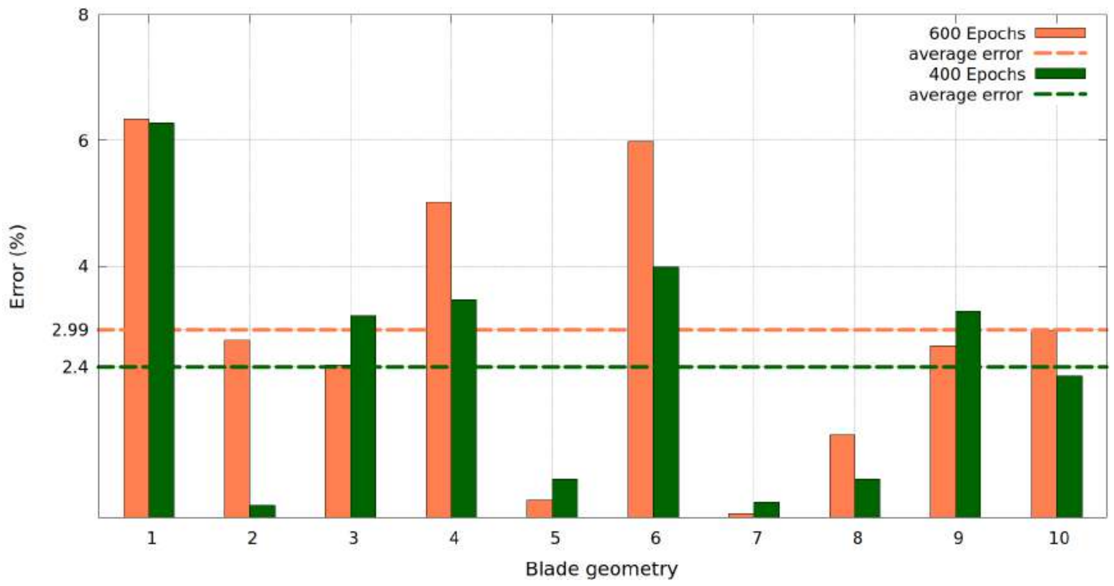


Figure 6.17: *The errors of PUMA-DNN predictions on the new blade geometries, having trained the DNN for 400 and 600 epochs. It can be seen that there is a reduction in the average error by 0.59% while the DNN trained for 400 instead of 600 epochs.*

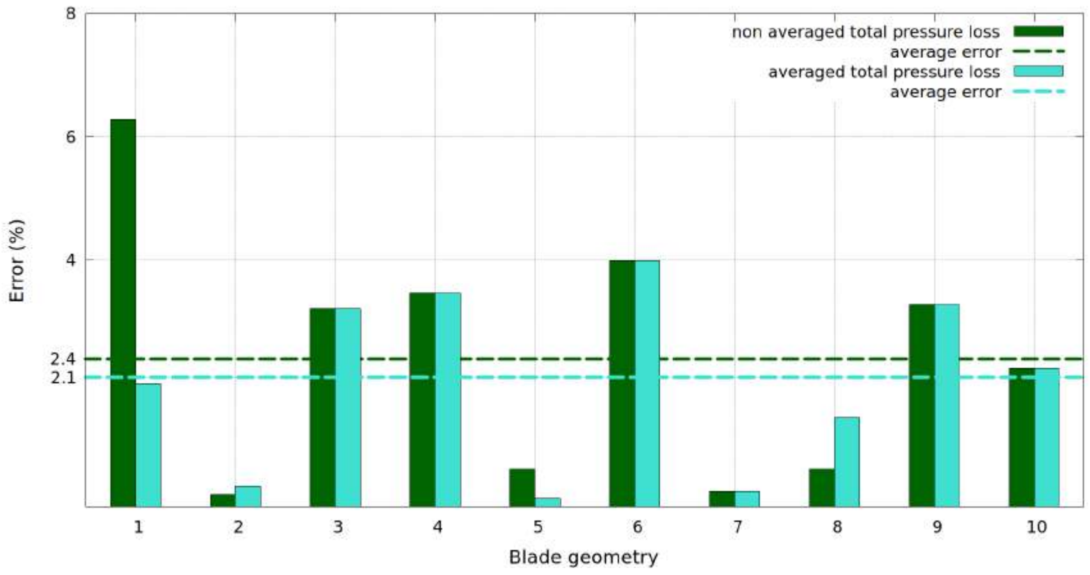


Figure 6.18: *The errors of PUMA-DNN using dropout with and without averaging the final* 10^3 *total pressure loss values. It can be seen that averaging the computed total pressure losses over the final 1000 iterations of PUMA-DNN, decreases the error by* 0.3%.

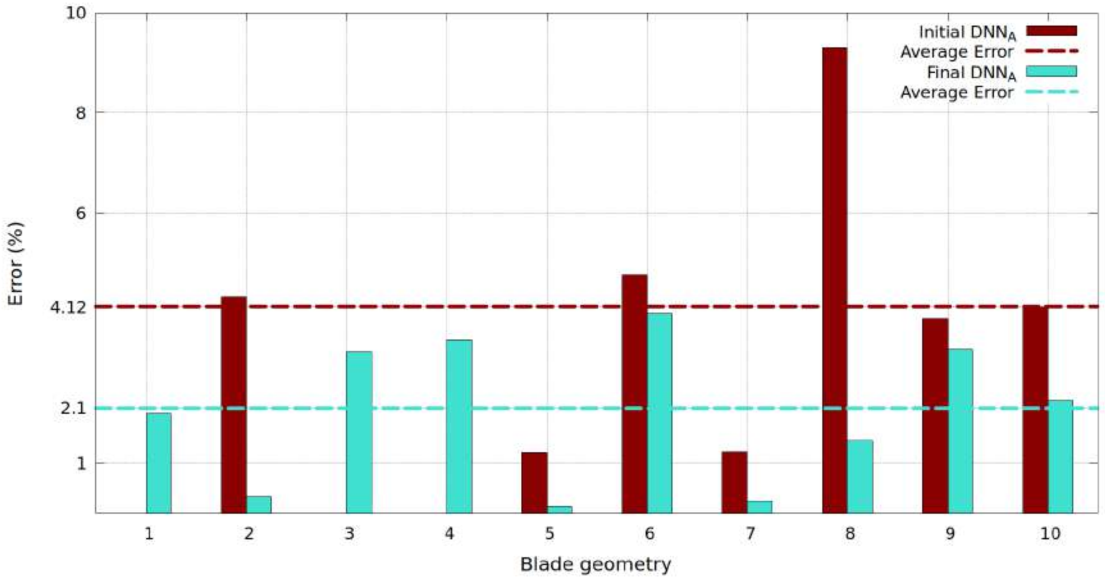


Figure 6.19: *The initial* DNN_A *errors and the final* DNN_A *errors using dropout, early stopping and averaging the total pressure losses over the final 1000 iterations of PUMA-DNN. It can be seen that it was achieved an error decrement by 2.02%, addressing at the same time the divergence problem.*

6.6 Shape Optimization Studies

In the shape optimization process using MAEA-based optimization in EASY, two different evaluation software were compared, the standard PUMA-TM and PUMA-DNN (using the final DNN_A model from section 6.5). The MAEA optimization configured with $\mu = 10$ parents and $\lambda = 18$ offspring.

PUMA-TM as evaluation software

When the PUMA-TM was used as the evaluation software, the RANS equations were solved with turbulence and transition models (PUMA-TM). However, during the computation of flow fields in some of the newly generated geometries, an oscillatory behavior in the total pressure losses was observed (see Figure 6.20) without reaching convergence. To overcome this and avoid the possibility of choosing some of them as the best ones in the optimization process because the numerical solution ended in a valley, the best practice is to average the total pressure losses over the last iterations of the numerical solution and use their mean as the objective function in the optimization process. The objective was to minimize the total pressure losses while ensuring that the exit angle remains close to its baseline value. After implementing this technique, the geometries that exhibited oscillatory behavior were deemed suboptimal, and this phenomenon ceased after a certain number of evaluations.

To guarantee independence from the randomly generated number seed (RNG seed) of MAEA, the optimization process was repeated three times for different RNG seeds. The convergence history of the MAEA-based optimization using PUMA-TM is shown in Figure 6.21, where it can be seen that both optimizations resulted in very similar outcomes, practically converging after 150 evaluations. One cost unit is defined as the cost of one evaluation using the PUMA-TM. A single evaluation takes ~ 10 mins, on a single NVIDIA GeForce RTX 2070.

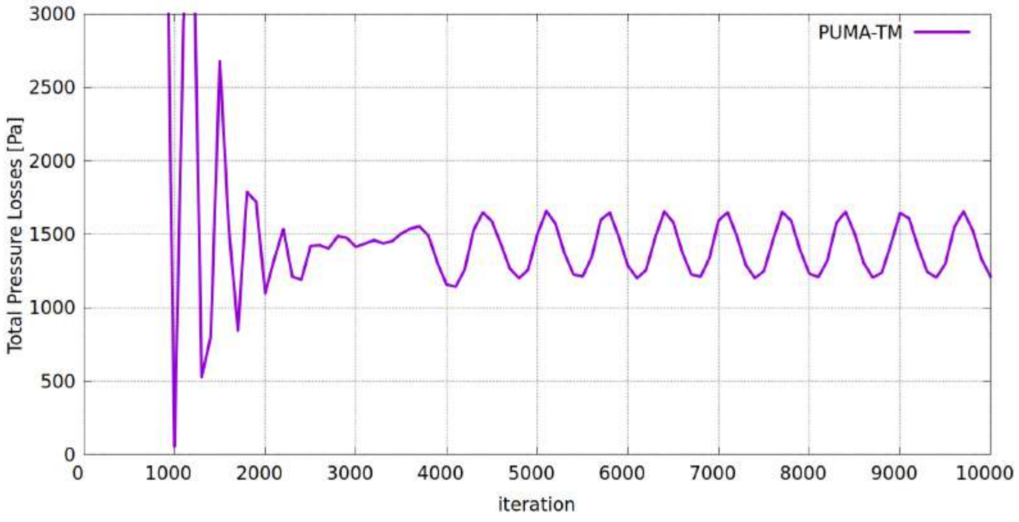


Figure 6.20: *The total pressure losses do not converge but they present a oscillatory behavior during the numerical solution of PUMA-TM. Because steady flows are studied and this unsteadiness is not accepted, the best practice is to average the losses over the last* $2 \cdot 10^3$ *iterations and consider this value as the convergent one.*

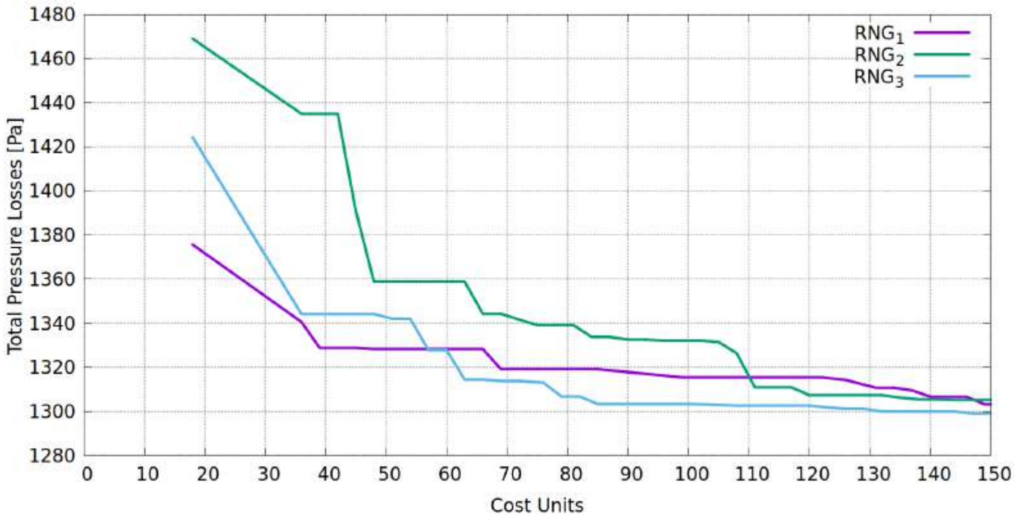


Figure 6.21: *The convergence history of the MAEA-based optimization relying on the PUMA-TM. It can be noticed that the optimizations led to very similar results.*

PUMA-DNN as evaluation software

The second evaluation software used in the shape optimization process was the PUMA-DNN, which is the DNN$_A$ predicting turbulence viscosity coupled with RANS. It was selected due to its faster speed, with one evaluation costing 0.5 cost units as opposed to one cost unit for the PUMA-TM. The objective function and constraints remained unchanged.

The re-evaluation of the two best-so-far solutions after 50 evaluations of the optimization procedure showed that the error in the computed total pressure losses was nearly double the average error. The two best solutions were then replaced the "worst" ones in the DNN database (DB$_{DNN}$) and the DNN was retrained, which was found to be more effective than simply enriching the DB$_{DNN}$. The cost of retraining was deemed negligible in comparison to the initial training cost.

The optimization process then continued for 200 more evaluations using the retrained DNN. At the end, the optimization algorithm could not further improve the best solutions and re-evaluation showed that the errors were equal to or less than 2.1%, which was equal to the average error of PUMA-DNN (see section 6.5). As a result, the DNN was deemed acceptable and there was no need for further retraining.

Although further improvement of the average error could have been achieved by replacing the worst solutions in the DB$_{DNN}$ with the better new ones from the optimization, this would have resulted in the DNN losing its generalization capability. Thus, although the DNN would have better accuracy in solutions near to the best ones, however, the solutions far from the best would be evaluated with significant errors, overestimating or underestimating the total pressure losses, causing the optimization procedure to consider the significantly underestimated solutions as the best ones. Therefore, the DNN was not retrained and the optimization process was stopped as it could not improve the best solution any further. The results of the optimization procedure are summarized in Table 6.4.

Table 6.4
Results of the Optimization procedure

Evaluation software	Best solution	Re-evaluated	percentage %
PUMA-TM (averaged)	1302	-	-11.3%
PUMA-DNN	1296	1318	-10.3%

The convergence history of the MAEA-based optimization relying on the PUMA-DNN model, is shown in Figure 6.22 and the convergence histories of the MAEA-based optimization relying on the PUMA-DNN and PUMA-TM using three different RNGs seeds, is shown in Figure 6.23.

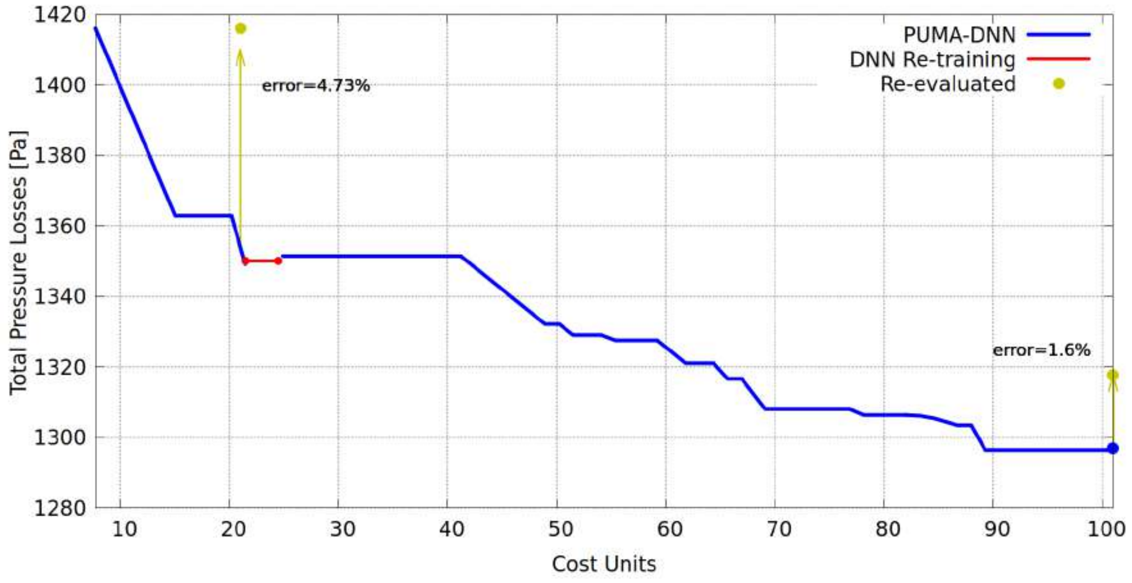


Figure 6.22: *The convergence history of the MAEA-based optimization relying on the PUMA-DNN model.*

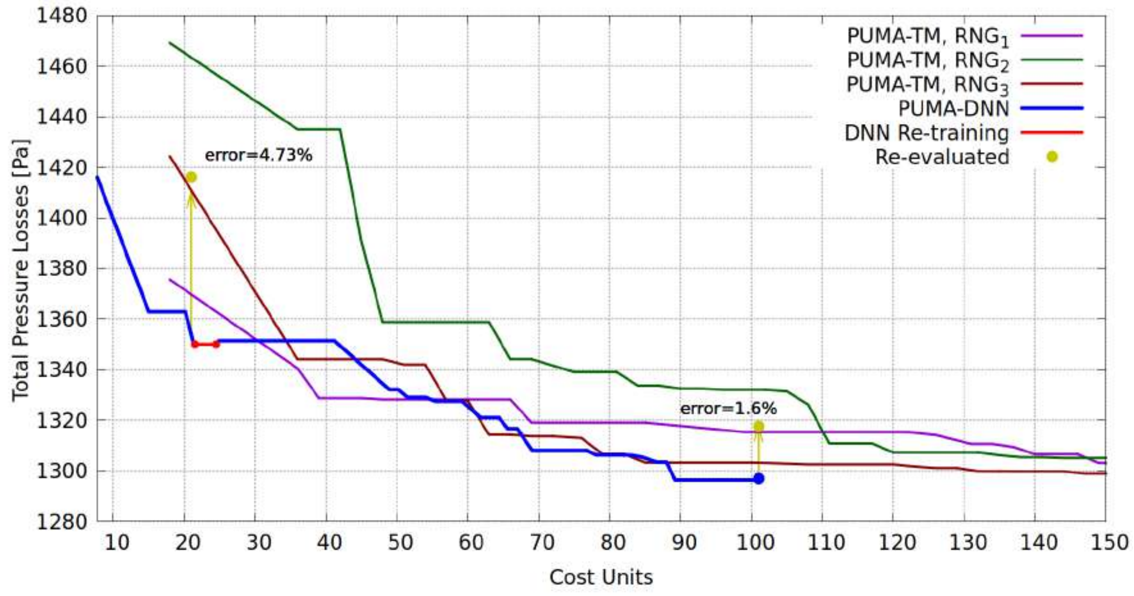


Figure 6.23: *The convergence histories of the MAEA-based optimization relying on the PUMA-DNN (without re-evaluation) and PUMA-TM using three different RNG seeds. The re-evaluated geometries by PUMA-TM are colored in yellow. The PUMA-DNN evaluation software leads to a faster converges but due to its predicted error, the best solution is worser than the final best solution using PUMA-TM as evaluation software.*

The baseline shape and the shapes of the best solutions of the MAEA-based optimization relying on the PUMA-TM and PUMA-DNN, are shown in Figure 6.24, where the x-axis is scaled at twice the size of the y-axis to enhance the visibility of the variations in the shapes.

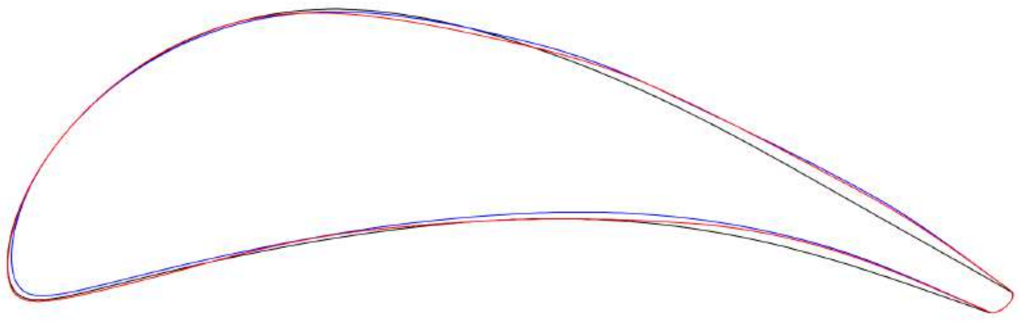


Figure 6.24: *The blade airfoil shape of the baseline (black), the shape of the best solution using PUMA-TM (red) and the shape of the best solution using PUMA-DNN (blue). The x-axis is scaled at twice the size of the y-axis to enhance the visibility of the variations in the shapes.*

The heat transfer coefficient H of the best solution of the MAEA-based optimization relying on the PUMA-DNN, computed by PUMA-DNN and re evaluated by PUMA-TM is shown in Figure 6.25, where it can be seen that the results are almost identical.

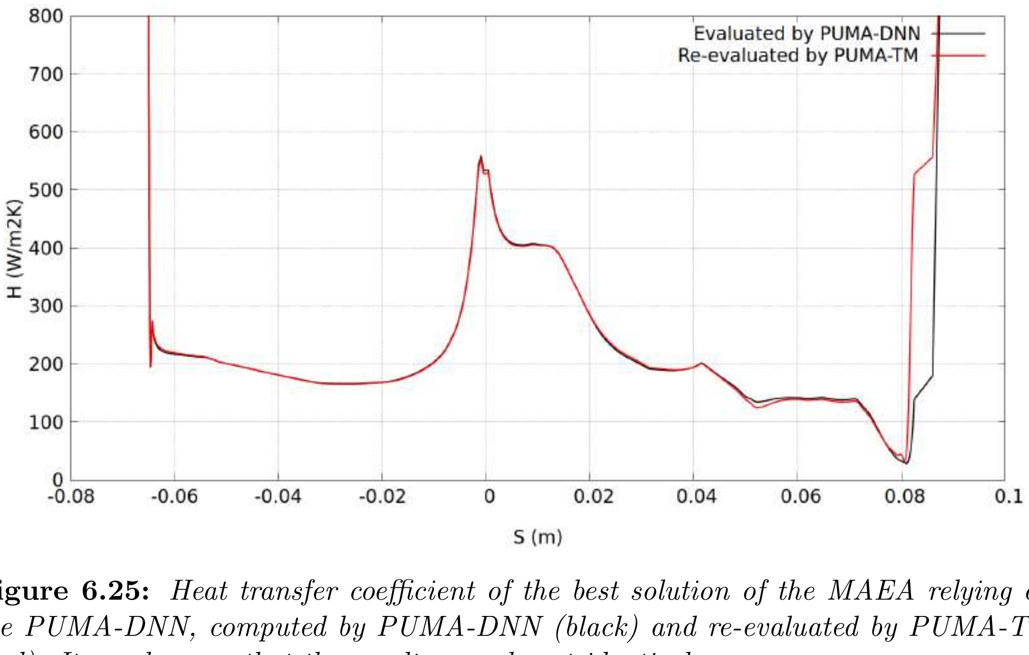


Figure 6.25: *Heat transfer coefficient of the best solution of the MAEA relying on the PUMA-DNN, computed by PUMA-DNN (black) and re-evaluated by PUMA-TM (red). It can be seen that the results are almost identical.*

The results of the heat transfer coefficient H computed by PUMA-TM shown in Figure 6.26 for both the baseline and the optimal solutions obtained using PUMA-TM and PUMA-DNN as evaluation software in the shape optimization procedure. The variations in the results stem from the dissimilarities in blade shapes.

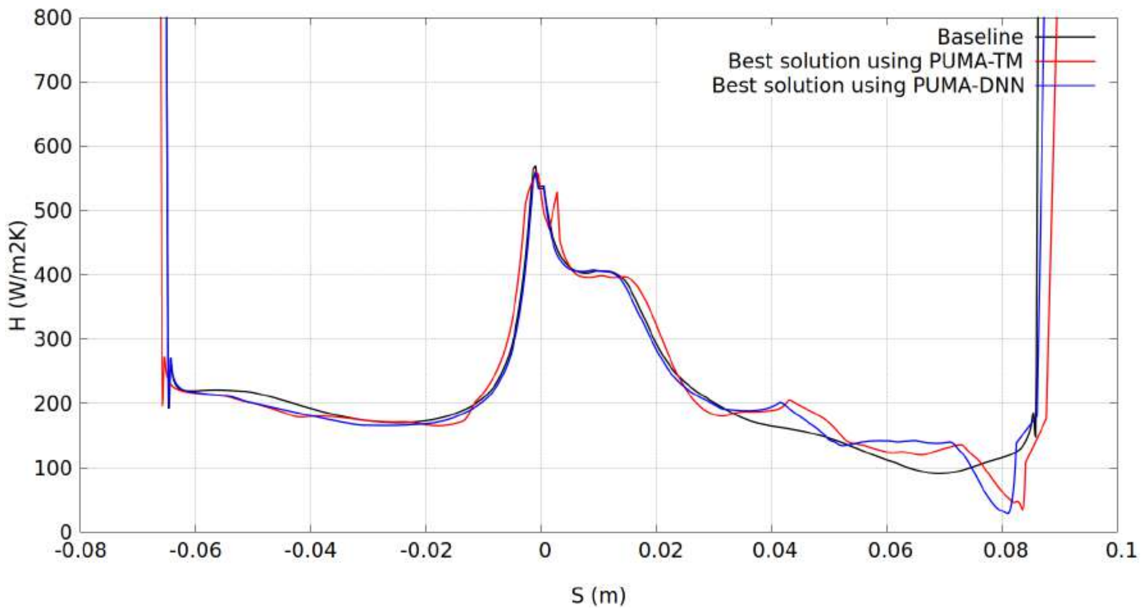


Figure 6.26: *Distributions of the heat transfer coefficient H computed by PUMA-TM for both the baseline and the optimal solutions obtained using PUMA-TM and PUMA-DNN as evaluation software. The variations in the results stem from the dissimilarities in blade shape.*

Also the Mach number iso-areas computed by PUMA-TM for the baseline and the best solutions of the shape optimization using PUMA-TM and PUMA-DNN are shown in Figure 6.27.

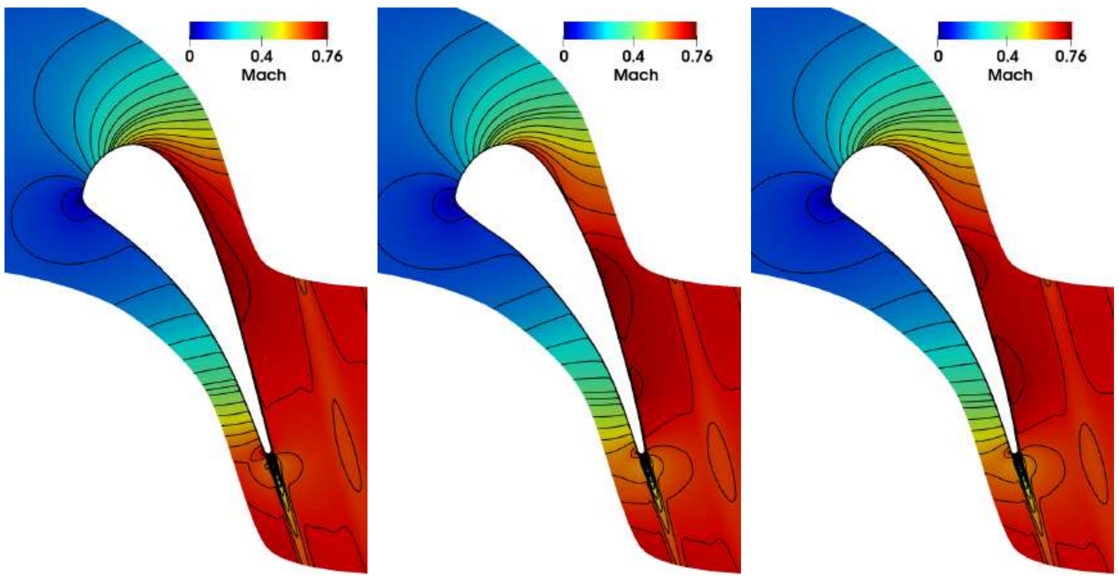


Figure 6.27: *Mach number iso-areas computed by PUMA-TM for the baseline (left), for the best blade geometry presented by the shape optimization procedure using PUMA-TM (center) and using PUMA-DNN (right).*

6.7 Conclusions

The optimization process utilizes the PUMA-DNN evaluation software, which allows for optimization convergence to occur nearly twice as fast as the PUMA-TM. However, the best solution achieved using PUMA-DNN resulted in a 10.3% decrease in total pressure losses compared to an 11.3% decrease using the PUMA-TM. The model could not be retrained to improve accuracy, as its predictive errors for the best-so-far solutions were equal to or smaller than its average error. This suggests that the model's predictions are still likely to be underestimated (like the best-so-far solutions that the PUMA-DNN found) or overestimated (the true best-so-far solutions when re-evaluated by PUMA-TM) with an average error of 2.1%.

In conclusion, it is proven that the DNN can effectively replace the k-ω turbulence and $\gamma - \tilde{R}e_{\theta t}$ transitional models (a total of 4 PDEs) and help the shape optimization process using EA. PUMA-DNN has a cost per evaluation that is about half that of PUMA-TM cost per evaluation, resulting in a faster convergence. The best-so-far solution of PUMA-DNN was trapped into a region due to its average error which was nearly 5 times less than the reduction in the objective function. This suggests that using PUMA-DNN can be even more beneficial, when the ratio of the maximum objective function decrement and the average error of the DNN is higher, and also when the evaluations are even more costly.

Chapter 7

Conclusion

7.1 Overview

In this Diploma Thesis, two cases were studied, specifically the NACA4318 airfoil and LS89 turbine blade cases, and employed DNNs to predict the turbulence viscosity field. To generate the training dataset, the baseline geometries were parameterized using NURBS lattices, and a set of 60 and 150 geometries was created for the first and second case, respectively, using Latin Hypercube Sampling (LHS). For the first case, the flows were simulated, using the RANS equation solver coupled with the SA turbulence model and for the second, using the RANS equations solver coupled with the k-ω turbulence model assisted by $\gamma - \tilde{Re}_{\theta t}$ transition model.

For the LS89 case, the DNN configurations and inputs were optimized in a MAEA optimization in EASY. Subsequently, the best DNN configuration was tested on 10 new geometries, which revealed overfitting to the training data. As a result, two alternative DNNs from earlier generations that have smaller number of hidden layers, were selected and tested, eventually leading to the identification of the optimal DNN configuration that exhibited good generalization and low loss simultaneously. For the training of the optimal DNN configuration, the early stopping and dropout generalization techniques were utilized. The findings demonstrat two key points: firstly, that the DNN's ability to generalize cannot be discerned from the training process, even when using a validation set; and secondly, that dropout in particular can enhance its generalization ability.

The MAEA (shape) optimization of the two cases was presented. The first, was the optimization of the NACA4318 case, with target to minimize the CD, keeping CL close to its baseline value. The second, was the optimization of the LS89 case, with target to minimize the total pressure losses while ensuring that the exit angle remains close to its baseline value. For each case, two types of MAEA optimizations were performed. For NACA4318 airfoil case, the first type used the RANS equations coupled with SA turbulence model as evaluation software (the standard model), when the second used the RANS equations coupled with the DNN instead (PUMA-DNN). The comparison of the two types, showed that the use of the DNN as turbulence model surrogate in the MAEA (shape) optimization, managed to achieve the same quality results as the standard model with a slight improvement in computational cost. For LS89 turbine blade case, the first type used the RANS equations coupled with the k-ω turbulence and $\gamma - \tilde{Re}_{\theta t}$ transition model as evaluation software (the standard model), when the second used the RANS equations coupled with the DNN instead (PUMA-DNN). The results revealed that the PUMA-DNN led to a faster convergence than the standard model with similar best solution quality. Thus, in both cases, the DNN can successfully replace the turbulence/transition model(s) and assist the MAEA shape optimization.

7.2 Conclusions

By completing the studies in this diploma thesis, the following conclusions are drawn:

- The DNNs, used to predict the turbulence viscosity field (by replacing the turbulence/transition model(s) in CFD runs), are capable of making useful predictions.

- Generalization techniques like dropout and early dropping can increase effectively the generalization ability of the DNN improving the accuracy of the predictions while testing on new unseen geometries. Also, dropout can indirectly improve the numerical stability of the RANS equations coupled with the DNN, making the predictions less sensitive to small perturbations in the input data.

- For both cases, the MAEA optimization relying on PUMA-DNN, managed to reach a solution of similar quality with the standard model. At the NACA4318 case, using the PUMA-DNN there was a slight improvement in the overall time of the optimization, as opposed to the LS89 case where the computational cost of the optimization reduced nearly by half. Thus, the benefit of using the PUMA-DNN model in optimization is higher when the application and the turbulent/transition model(s) being replaced are more computationally expensive.

7.3 Future Work for Proposals

Based on the implementation of the DNNs in CFD, the following future works are proposed:

- DNNs can be employed as turbulence and transition models surrogates in computationally expensive cases, such as 3D flows, resulting in greater time savings during the optimization process.
- By utilizing Fourier feature mapping, it is possible to improve the ability of DNNs to learn the high frequency functions present in turbulence flows. As a result, DNNs can be used to simulate more computationally expensive cases or reduce the required size of the training database.
- DNNs can be trained to learn from the data generated by Large-Eddy Simulation (LES) and reproduce their results. By utilizing the high-fidelity data from LES simulations, the DNNs can learn to predict the behavior of the fluid flow with greater accuracy and efficiency compared to using low-fidelity models.

Bibliography

[1] Gershgorn, D.: *The Quartz guide to artificial intelligence: What is it, why is it important, and should we be afraid?* Quartz, September 2017. https://qz.com/

[2] Kahneman D., Sibony O., and Sustein C.R.: *Noise: A flaw in human judgment.* Little, Brown Spark, New Work Boston London, May 2021, ISBN 978-0-316-45138-3

[3] Michalski R.S., Carbonell J.G., Mitchell T.M.: *Machine learning: An Artificial Intelligence Approach.* Springer-Verlag, Berling Heidelberg, 1984, ISBN 978-3-662-12407-9

[4] Russell S.J., Norvig P.: *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Clidds, New Jersey, page:546, 1995, ISBN 0-13-103805-2

[5] Ryan J.: *Machine Learning: In plain English*, DZone, July 2018, https://dzone.com/

[6] Rouhiainen L.: *Artificial intelligence: 101 things you must know today about our future*, CreateSpace Independent Publishing Platform, January 2018, ISBN 978-1982048808

[7] Kontou M., Asouti V, Giannakoglou K.: *DNN surrogates for turbulence closure in CFD-based shape optimization.* Applied Soft Computing, 134:1-11, January 2023, `https://doi.org/10.1016/j.asoc.2023.110013`

[8] Tracey, D.B., Duraisamy, K., Alonso, A. J.: *Machine learning strategy to assist turbulence model development.*, 53rd AIAA Aerospace Sciences Meeting, January 2015. doi:10.2514/6.2015-1287, 10.2514/6.2015-1287

[9] Maulik R., San O., Jacob D.J, Crick C.: *Sub-grid scale model classification and blending through deep learning*, Journal of Fluid Mechanics, 870:784-812, May 2019. doi:10.1017/jfm.2019.254, 10.1017/jfm.2019.254

[10] Haykin S.: *Neural Networks and Learning Machines*, Prentice Hall/Pearson, New York, 2009, ISBN 978-0131471399.

[11] Crick F.: *The Astonishing Hypothesis: The Scientific Search for the Soul*, Charles Scribner's Sons, New York, 1995, ISBN 0-684-19431-7

[12] Rumelhart D.E., Hinton G.E, Williams R.J.: *Learning representations by back-propagating errors*, Nature, 323(6088):533-536, October 1986. doi:10.1038/323533a0, 10.1038/323533a0.

[13] Elbrachter, D., Perekrestenko, D., Grohs P., and Bolcskei H.: *Deep Neural Network Approximation Theory.* IEEE Transactions on Information Theory, 67(5):2581-2623, May 2021. doi:10.1109/tit.2021.3062161, 10.1109/tit.2021.3062161.

[14] Qi, J., Du, J., Siniscalchi, S.M., Ma, X., Lee, C.: *On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression*, IEEE Signal Processing Letters, 1-1, 2020. doi:10.1109/lsp.2020.3016837, 10.1109/lsp.2020.3016837.

[15] Kingma, D. P. and Ba, J.: *Adam: A method for stochastic optimization.*, In Bengio, Yoshua and LeCun, Yann (editors): 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. `http://arxiv.org/abs/1412.6980`

[16] Vapnik, S.M. *Deep Neural Networks for Modeling Nonlinear Systems: A Review of Cross-Validation Strategies*, 2017

[17] Brownlee, J. *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*, Machine Learning Mastery, December 2018.

[18] Mohamed, D. *What is Overffitting and Underfitting?* Medium, November 11, 2020. https://medium.com/

[19] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15:1929-1958, 2014.

[20] Spalart, P., Allmaras, S.: *A One-Equation Turbulence Model for Aerodynamic Flows*, 30th Aerospace Sciences Meeting and Exhibit, 1992. https://doi.org/10.2514/6.1992-439, 10.2514/6.1992-439.

[21] Menter, F.R.: *Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications*, AIAA Journal, 32(8):1598-1605, 1994. doi:10.2514/3.12149, 10.2514/3.12149.

[22] Langtry, R.B., and Menter, F.R.: *Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes*, AIAA Journal, 47(12), 2894-2906. doi:10.2514/1.42362, 10.2514/1.42362.

[23] Schubauer, G. B., and Klebanoff, P. S., *Contribution on the Mechanics of Boundary Layer Transition,* January 1956. id: 19930092285.

[24] Asouti V., Trompoukis X., Kampolis I., Giannakoglou K. (2011) *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. . `https://doi.org/10.1002/fld.2352`, 10.1002/fld.2352.

[25] Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computing*, Springer Berlin, Heidelberg, 2015, ISBN: 978-3-662-44873-1. https://doi.org/10.1007/978-3-662-44874-8

[26] Giannakoglou, K: *Optimization Methods in Aerodynamics*, NTUA, Athens, 2006. http://velos0.ltt.mech.ntua.gr

[27] Giannakoglou, K.: *Designing turbomachinery blades using evolutionary methods*, ASME Paper 99-GT-181 1999.

[28] Karakasis, K.M., Giannakoglou, K.: *Metamodel-Assisted Mutli-Objective Evolutionary Optimization*, 2005.

[29] Agoston E. Eiben, J.E Smith (2008), *EASY*. http://velos0.ltt.mech.ntua.gr/EASY/

[30] Ching, J., Gulcehre, C., Cho, K., and Bengio, Y.: *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, arXiv, December 2014, https://doi.org/10.48550/arXiv.1412.3555

[31] Goodfellow, I., Bengio, Y., and Courville, A.: *Deep Learning*, MIT press, 2016. http://www.deeplearningbook.org

[32] Arts, T., Rouvroit De Lambert, M., Rutherford, W.A.: *AERO-Thermal investigation of a highly loaded transonic linear turbine guide vane cascade: A test case for inviscid and viscous flow computations*, Von Karman Institute for fluid dynamics, Technical Note 174, September 1990.

[33] Pironneau, O.: *Finite Elements for Fluid*, Willey, 1st edition, January 1990, ISBN 978-0471922551.

[34] Bochev P.B, Gunzburger, M.D.: *Analysis of least squares finite element methods for the Stokes equations*, Mathematics of Computation, 63(208):479-479. doi:10.1090/s0025-5718-1994-1257573-4, 10.1090/s0025-5718-1994-1257573-4.

[35] Roe, L.P.: *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*, Journal of Computational Physics, 135(2):250-258. doi:10.1006/jcph.1997.5705, 10.1006/jcph.1997.5705.

[36] Jameson, A.: *Origins and Further Development of the Jameson–Schmidt–Turkel Scheme*, AIAA Journal, 55(5):1487–1510, 2017. doi:10.2514/1.j055493, 10.2514/1.j055493.

[37] Raissi, M., Perdikaris, P., and Karniadakis, G.E.: *Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations*, Journal of Computational Physics, 378:686-707, 2019. https://doi.org/10.1016/j.jcp.2018.10.045

[38] Zhang, C., Patras, P., and Haddadi, H.: *Deep learning in mobile and wireless networking: A survey*, IEEE Communications Surveys and Tutorials, 43(3):2224-2287, 2019. doi:10.1109/COMST.2019.2904897.

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

Τα Βαθιά Νευρωνικά Δίκτυα ως Υποκατάστατα Μοντέλων Τύρβης στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση

Διπλωματική εργασία - Εκτενής Περίληψη στην Ελληνική

Ιωάννης Κορδός

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2023

Εισαγωγή

Στόχος της Διπλωματικής αυτής Εργασίας, είναι η χρήση Βαθιών Νευρωνικών Δικτύων (ΒΝΔ), ως υποκατάστατα μοντέλων τύρβης στην αεροδυναμική ανάλυση και βελτιστοποίηση, με στόχο τη μείωση του συνολικού υπολογιστικού κόστους της διαδικασίας αυτής. Τα ΒΝΔ εκπαιδεύονται σε ένα σύνολο δεδομένων με στόχο να αναπαράγουν όσο το δυνατόν ακριβέστερα το πεδίο της τυρβώδους συνεκτικότητας.

Οι περιπτώσεις βελτιστοποίησης που χρησιμοποιούνται είναι η βελτιστοποίηση του σχήματος της μεμονωμένης αεροτομής $NACA4318$ με στόχο την ελαχιστοποίηση του συντελεστή αντίστασης και σταθερό συντελεστή άνωσης, και του πτερυγίου συμπιεστή $LS89$ με στόχο την ελαχιστοποίηση των απωλειών ολικής πίεσης και σταθερή γωνία εξόδου της ροής. Το ακριβές μοντέλο για την πρώτη περίπτωση, είναι ο επιλύτης των $RANS$ με το μοντέλο τύρβης $Spalart-Allmaras$, ενώ για τη δεύτερη είναι ο επιλύτης των $RANS$ με το μοντέλο τύρβης k-ω και μετάβασης $\gamma-\widetilde{Re}_{\theta t}$. Τα εναλλακτικά μοντέλα είναι ο επιλύτης των $RANS$ εξισώσεων με το αντίστοιχο ΒΝΔ της κάθε περίπτωσης.

Τεχνητή Νοημοσύνη και Βαθιά Νευρωνικά Δίκτυα

Η τεχνητή νοημοσύνη είναι ένας ραγδαία αναπτυσσόμενος κλάδος της επιστήμης των υπολογιστών και έχει επηρεάσει και αναδιαμορφώσει πολλές πτυχές της βιομηχανίας και της κοινωνίας. Η ΤΝ περιλαμβάνει την ανάπτυξη αλγορίθμων ικανών να ανιχνεύουν μοτίβα σε ένα σύνολο δεδομένων και να λαμβάνουν, εν συνεχεία, αποφάσεις με βάση νέα δεδομένα.

Η πρώτη υποκατηγορία του πεδίου της ΤΝ είναι η Μηχανική Μάθηση (ΜΜ). Τα μοντέλα ΜΜ εξετάζουν δεδομένα, μαθαίνουν από αυτά βασιζόμενα σε στατιστικά μοντέλα και, εν συνεχεία, κάνουν προβλέψεις/ταξινομήσεις σχετικά με νέα δεδομένα. Το σημαντικό τους χαρακτηριστικό είναι πως η διαδικασία μάθησης είναι αυτόματη χωρίς να χρειάζεται εξατομικευμένος προγραμματισμός στο εκάστοτε πρόβλημα.

Τα Βαθιά Νευρωνικά Δίκτυα (ΒΝΔ ή DNN) αποτελούν υποκατηγορία της ΜΜ και είναι εμπνευσμένα από την λειτουργία των εγκεφαλικών νευρικών κυττάρων. Έτσι, τα ΝΔ αποτελούνται από διασυνδεδεμένα δίκτυα τεχνητών νευρώνων τα οποία έχουν είσοδο και έξοδο. Η πληροφορία ρέει από την είσοδο στην έξοδο και επηρεάζεται σύμφωνα με την αρχιτεκτονική και τα βάρη των συνάψεων. Η αρχιτεκτονική ενός ΒΝΔ ορίζεται από το πλήθος των επιπέδων και το πλήθος των νευρώνων σε κάθε επίπεδο. Η εκπαίδευσή τους σε ένα σύνολο εισόδων έγκειται στον προσδιορισμό των κατάλληλων συναπτικών βαρών (σύμφωνα με έναν αλγόριθμο μάθησης) που ελαχιστοποιούν το σφάλματα μεταξύ της εξόδου τους και της γνωστής απόκρισης. Το τελικό σύνολο τιμών των βαρών αποτελεί τη μνήμη του ΒΝΔ η οποία του επιτρέπει να κάνει προβλέψεις σε νέα δεδομένα.

Τα ΒΝΔ έχουν χρησιμοποιηθεί ευρέως και στον τομέα της Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ) καθώς έχουν την ικανότητα να μαθαίνουν πολύπλοκες μη-γραμμικές σχέσεις μεταξύ των εισόδων και των εξόδων. Ενδεικτικά, λόγω του μικρού υπολογιστικού κόστους έχουν χρησιμοποιηθεί και ως υποκατάστατα λογισμικών ΥΡΔ σε διαδικασίες βελτιστοποίησης [1].

Αεροδυναμική Ανάλυση σε Τυρβώδες Ροές

Οι μόνιμες και συμπιεστές ροές μοντελοποιούνται με τις $RANS$ εξισώσεις οι οποίες γράφονται:

$$R_n^{MF} = \frac{\partial f_{nj}^{inv}}{\partial x_j} - \frac{\partial f_{nj}^{vis}}{\partial x_j} = 0 \tag{7.1}$$

όπου $n = 1,..,5$ και $j = 1,..3$. $f_j^{inv} = [\rho u_j \quad \rho u_j u_1 + p\delta_{1j} \quad \rho u_j u_2 + p\delta_{2j} \quad \rho u_j u_3 + p\delta_{3j} \quad \rho u_j h_t]^T$ είναι οι ατριβείς ροές, $f_j^{vis} = [0 \quad \tau_{1j} \quad \tau_{2j} \quad \tau_{3j} \quad u_k\tau_{kj} + q_j]^T$ είναι οι συνεκτικές-τυρβώδες ροές. ρ, p, u_j,h_t και δ_{jm} είναι η πυκνότητα του ρευστού, η πίεση, οι συνιστώσες της ταχύτητας, η ολική ενθαλπία και το σύμβολο του $Kronecker$, αντίστοιχα. Βασιζόμενοι στην υπόθεση του $Boussinesq$, οι συνιστώσες του τανυστή τάσης είναι:

$$\tau_{jm} = (\mu + \mu_t)\left(\frac{\partial u_j}{\partial x_m} + \frac{\partial u_m}{\partial x_j} - \frac{2}{3}\delta_{jm}\frac{\partial u_k}{\partial x_k}\right) - \frac{2}{3}\delta_{jm}\rho k \tag{7.2}$$

όπου μ είναι η συνεκτικότητα του ρευστού και μ_t η τυρβώδης συνεκτικότητα (ο τελευταίος όρος της εξίσωσης 7.2 χρησιμοποιείται μόνο στο k-ω SST μοντέλο) και q_j είναι η ροή θερμότητας. k είναι η τυρβώδης κινητική ενέργεια (TKE), $k = \frac{1}{2}\overline{u_j' u_m'}$, η οποία μπορεί να υπολογιστεί με το k-ω SST μοντέλο τύρβης. Οι συνιστώσες της ταχύτητας και πίεσης, υπολογίζονται απο τις εξισώσεις μέσης ροής ενώ το πεδίο τυρβώδους συνεκτικότητας υπολογίζεται απο το μοντέλο τύρβης που επιλέγεται.

Το $Spalart - Allmaras$ (SA) Μοντέλο Τύρβης

Το μοντέλο τύρβης SA λύνει μία διαφορική εξίσωση για τον υπολογισμό του πεδίου $\tilde{\nu}$. Η εξίσωση αυτή για μόνιμη και συμπιεστή ροή είναι:

$$\begin{aligned} R^{SA} = &\frac{\partial(\rho\tilde{\nu}u_k^R)}{\partial x_j} - \frac{\rho}{Re_0\sigma}\left\{\frac{\partial}{\partial x_j}\left[(\nu+\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_k}\right] + c_{b2}\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k}\right\} \\ &- \rho c_{b1}\tilde{S}\tilde{\nu} + \frac{\rho}{Re_0}c_{w1}f_w\left(\frac{\tilde{\nu}}{\Delta}\right)^2 \end{aligned} \tag{7.3}$$

όπου το Δ είναι η απόσταση κάθε κόμβου του πλέγματος απο τον πλησιέστερο σύνορο. Η επιθυμητή ποσότητα μ_t μπορεί να υπολογιστεί απο την εξίσωση:

$$\mu_t = \rho\tilde{\nu}f_{v1} \tag{7.4}$$

Οι συμπληρωματικές εξισώσεις της 7.3, μπορούν να βρεθούν στο [2].

Το k-ω Μοντέλο Τύρβης

Το μοντέλο τύρβης k-ω λύνει δύο διαφορικές εξισώσεις για τα μεγέθη k και ω, οι οποίες είναι οι εξής:

$$\frac{\partial(\rho u_j k)}{\partial x_j} = P - D + \frac{\partial}{\partial x_j}\left((\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_j}\right) \tag{7.5}$$

$$\frac{\partial(\rho u_j \omega)}{\partial x_j} = \frac{\gamma}{\nu_t}P - \beta\rho\omega^2 + \frac{\partial}{\partial x_j}\left((\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right) + 2(1-F_1)\frac{\rho\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{7.6}$$

Η επιθυμητή ποσότητα μ_t μπορεί να υπολογιστεί από τη σχέση:

$$\mu_t = \frac{\rho \alpha_1 k}{max(\alpha_1 \omega, \Omega F_2)} \tag{7.7}$$

Οι συμπλρωματικές εξισώσεις και σταθερές των εξισώσεων 7.5 και 7.6, μπορούν να βρεθούν στο [3].

Το $\gamma - \tilde{R}e_{\theta t}$ Μοντέλο Μετάβασης

Το μοντέλο αυτό βασίζεται σε δύο διαφορικές εξισώσεις για τα μεγέθη γ και $\tilde{R}e_{\theta t}$, οι οποίες είναι:

$$R^{\gamma} = \frac{\partial(\rho u_j \gamma)}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\left(\mu + \frac{\mu_t}{\sigma_f}\right)\frac{\partial \gamma}{\partial x_j}\right) - P_\gamma + E_\gamma = 0 \tag{7.8}$$

$$R^{\tilde{R}e_{\theta t}} = \frac{\partial(\rho u_j \tilde{R}e_{\theta t})}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\sigma_{\theta t}(\mu + \mu_t)\frac{\partial \tilde{R}e_{\theta t}}{\partial x_j}\right) - P_{\theta t} = 0 \tag{7.9}$$

Όλες οι συμπληρωματικές εξισώσεις και σταθερές μπορούν να βρεθούν στο [4]. Εν τέλει, σκοπός είναι να υπολογιστεί η ποσότητα γ_{eff} με την οποία το μοντέλο μετάβασης αλληλεπιδρά με το μοντέλο τύρβης k-ω ως εξής:

$$\frac{\partial}{\partial x_j}(\rho u_j k) = \tilde{P} - \tilde{D} + \frac{\partial}{\partial x_j}\left((\mu + \sigma\mu_t)\frac{\partial k}{\partial x_j}\right) \tag{7.10}$$

$$\tilde{P} = \gamma_{eff} P \tag{7.11}$$

$$\tilde{D} = min(max(\gamma_{eff}, 0.1), 1.0)D \tag{7.12}$$

όπου τα P και D είναι αρχικοί όροι παραγωγής και καταστροφής του k-ω SST μοντέλου.

Εξελικτικοί Αλγόριθμοι

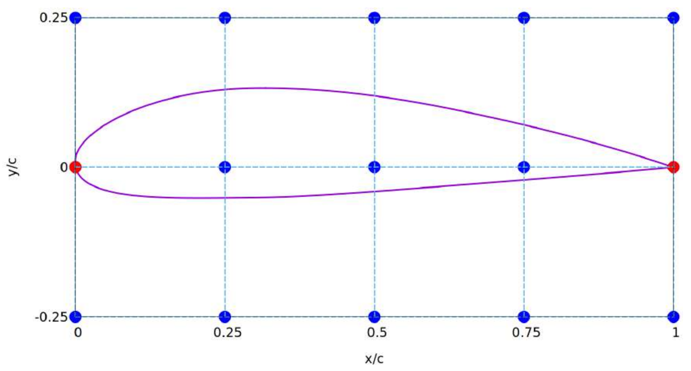
Οι εξελικτικοί αλγόριθμοι (ΕΑ) είναι βασισμένοι στη θεωρία του Δαρβίνου περί της φυσικής εξέλιξης. Ένας ΕΑ διαχειρίζεται πληθυσμούς λύσεων (γονείς) οι οποίες συνδυάζονται/αναπαράγονται δίνοντας νέες λύσεις (απογόνους) από γενιά σε γενιά σύμφωνα με μηχανισμούς εμπνευσμένους από τη φύση. Ο σκοπός των μηχανισμών αυτών είναι τα καλά χαρακτηριστικά των γονέων να περάσουν και να ενισχυθούν στους απογόνους τους. Τα "καλά" χαρακτηριστικά ενός ατόμου, προσδιορίζονται μέσω μιας συνάρτησηστόχου, η οποία ανάλογα το πρόβλημα, αφορά στην ελαχιστοποίησή ή μεγιστοποίησή της. Τα άτομα που ελαχιστοποιούν/μεγιστοποιούν την συνάρτηση-στόχος είναι αυτά που θεωρούνται και τα "καλύτερα" από τα υπόλοιπα άτομα.

Τα βασικά πλεονεκτήματα ενός ΕΑ, είναι το μη-μαθηματικό υπόβαθρό τους, η ικανότητα να προσαρμόζονται σε διαφορετικά προβλήματα αρκεί να υπάρχει το κατάλληλο λογισμικό αξιολόγησης, η δυνατότητα να αξιολογούνται τα άτομα παράλληλα και η δυνατότητά τους να μην εγκλωβίζονται σε τοπικά ακρότατα λόγω του στοχαστικού τους χαρακτήρα και των μηχανισμών αναπαραγωγής, οι οποίοι ενισχύουν την τυχαιότητα.

Χρήση ΒΝΔ ως Υποκατάστατου του Μοντέλου Τύρβης SA στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση της Μεμονωμένης Αεροτομής $NACA4318$

Το ΒΝΔ εκπαιδεύεται με σκοπό να αναπαράγει με όσο το δυνατόν μεγαλύτερη ακρίβεια το πεδίο μ_t. Αφού επιτευχθεί αυτό, το ΒΝΔ υποκαθιστά το μοντέλο τύρβης (SA) στη βελτιστοποίηση σχήματος της αεροτομής με ΜΑΕΑ με στόχο την ελαχιστοποίηση του συντελεστή οπισθέλκουσας CD και διατήρηση του συντελεστή άνωσης CL στην αρχική του τιμή.

Για την εκπαίδευση του ΒΝΔ η γεωμετρία της αεροτομής παραμετρικοποιείται με την χρήση ενός κουτιού $NURBs$ αποτελούμενο από 5Χ3 σημεία ελέγχου όπως φαίνεται στο Σχήμα.7.1. Τα σημεία ελέγχου με μπλε χρώμα, επιτρέπεται να μετακινηθούν μόνο κατά την κάθετη κατεύθυνση, ενώ τα σημεία ελέγχου με κόκκινο κρατούνται σταθερά. Επομένως, υπάρχουν 13 μεταβλητές σχεδιασμού.



Σχήμα 7.1: *Παραμετρικοποίηση με κουτί 5Χ3 $NURBS$.*

Για τη δημιουργία των δεδομένων εκπαίδευσης (ΔΕ) του ΒΝΔ, δημιουργούνται 60 δείγματα με την τεχνική *Latin Hypercube Sampling* (*LHS*), η οποία κάνει σχεδόν τυχαία δειγματοληψία του χώρου σχεδιασμού των 13 μεταβλητών. Εν συνεχεία, η ροή γύρω από κάθε γεωμετρία μοντελοποιείται με τον επιλύτη των $RANS$ εξισώσεων και του μοντέλου τύρβης SA (ακριβές μοντέλο ή $PUMA-TM$). Η μοντελοποίηση της ροής των 60 γεωμετριών παίρνει περίπου 4 ώρες στην κάρτα γραφικών $NVIDIA$ $GeForce$ RTX 2070.

Οι ποσότητες στο σύνολο των ΔΕ, κανονικοποιούνται στο διάστημα [0,1] από τη σχέση:

$$x_i = \frac{X_j - min(Q_i)}{max(Q_i) - min(Q_i)} \tag{7.13}$$

όπου το Q_i είναι ένας πίνακας που περιέχει τις τιμές της μεταβλητής i σε όλες τις γεωμετρίες των ΔΕ. $max(Q_i)$ και $min(Q_i)$ είναι η μέγιστη και ελάχιστη τιμή της μεταβλητής i όλων των ΔΕ. Το X_j είναι η j τιμή του πίνακα Q_i και το x_j είναι η κανονικοποιημένη τιμή του X_j, η οποία ανήκει στο διάστημα [0,1].

Η ανάπτυξη και εκπαίδευση του ΒΝΔ έγινε στο $TensorFlow$ με χρήση της γλώσσας προγραμματισμού *Python*. Η επιλεγμένη διαμόρφωση του ΒΝΔ φαίνεται στον πίνακα 7.1

Πίνακας 7.1

Επίπεδα	Νευρώνες/Επίπεδο	ΔΕ εισόδου	Συνάρτηση ενεργοποίησης
5	64, 128, 256, 2048, 512	$(x_k, \rho, p, u_k,)$	$ReLu/tanh$

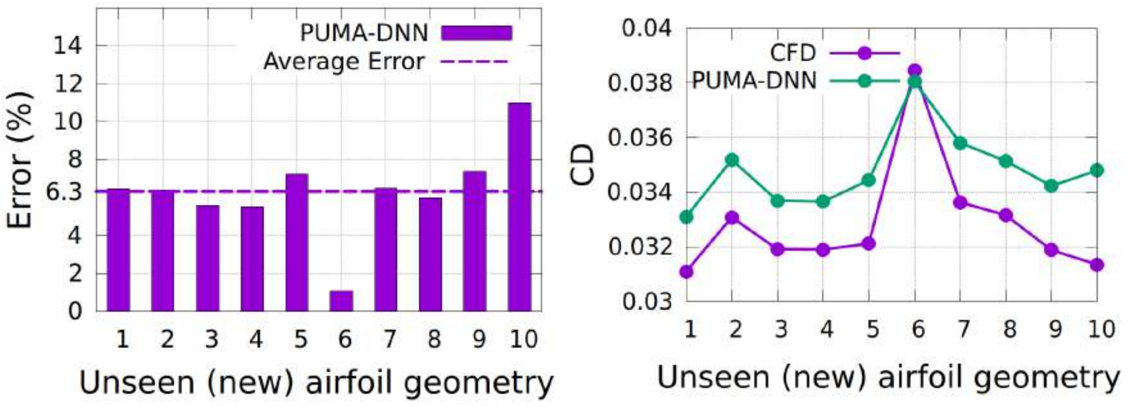
Προκειμένου να εκτιμηθεί το σφάλμα και η ικανότητα γενίκευσης, το ΒΝΔ και οι $RANS$ εξισώσεις (εναλλακτικό μοντέλο ή $PUMA-DNN$) επιλύονται για την μοντελοποίηση της ροής σε 10 νέες γεωμετρίες και το σφάλμα των προβλέψεων εκτιμάται ως προς τις υπολογιζόμενες τιμές του CD και όχι του πεδίου μ_t. Αυτό γίνεται για πρακτικτούς λόγους, αφού η βελτιστοποίηση σχήματος αφορά στην ελαχιστοποίηση του CD. Η σχέση για τον υπολογισμό του σφάλματος είναι:

$$error\ [\%] = \frac{CD^j_{PUMA-TM} - CD^j_{PUMA-DNN}}{CD^j_{PUMA-TM}} \cdot 100 \tag{7.14}$$

όπου j είναι ο δείκτης των 10 νέων γεωμετριών και παίρνει τιμές [1,2,..,9,10].

Στο Σχ. 7.2, φαίνονται τα αποτελέσματα των προβλέψεων του ΒΝΔ στις νέες γεωμετρίες.

Αφού το ΒΝΔ έχει εκπαιδευτεί, πραγματοποιείται βελτιστοποίηση σχήματος της αερο-



Σχήμα 7.2: *Ποσοστιαία σφάλματα σε κάθε νέα γεωμετρία (αριστερά) και οι τιμές των* CD *(δεξιά) υπολογισμένες απο το ακριβές μοντέλο* $(PUMA-TM)$ *και το εναλλακτικό μοντέλο* $(PUMA-DNN)$*. Το μέσο σφάλμα είναι 6.3%*

τομής με $MAEA$, με χρήση δύο λογισμικών αξιολόγησης. Το πρώτο είναι ο επιλύτης των $RANS$ εξισώσεων με το μοντέλο τύρβης SA (ακριβές μοντέλο) και το δεύτερο είναι ο επιλύτης των $RANS$ εξισώσεων με το μοντέλο του ΒΝΔ (εναλλακτικό μοντέλο). Στόχος της βελτιστοποίησης είναι η ελαχιστοποίηση του CD, διατηρώντας το CL κοντά στην αρχική του τιμή.

Η βελτιστοποίηση σχήματος με ΜΑΕΑ με το ακριβές λογισμικό αξιολόγησης πραγματοποιείται για τρείς διαφορετικές αρχικοποιήσεις της γεννήτριας τυχαίων αριθμών (RNG $seed$). Το κριτήριο τερματισμού κάθε βελτιστοποίησης τίθεται στις 350 αξιολογήσεις. Μιά μονάδα κόστους ισοδυναμεί με μία αξιολόγηση του ακριβούς μοντέλου.

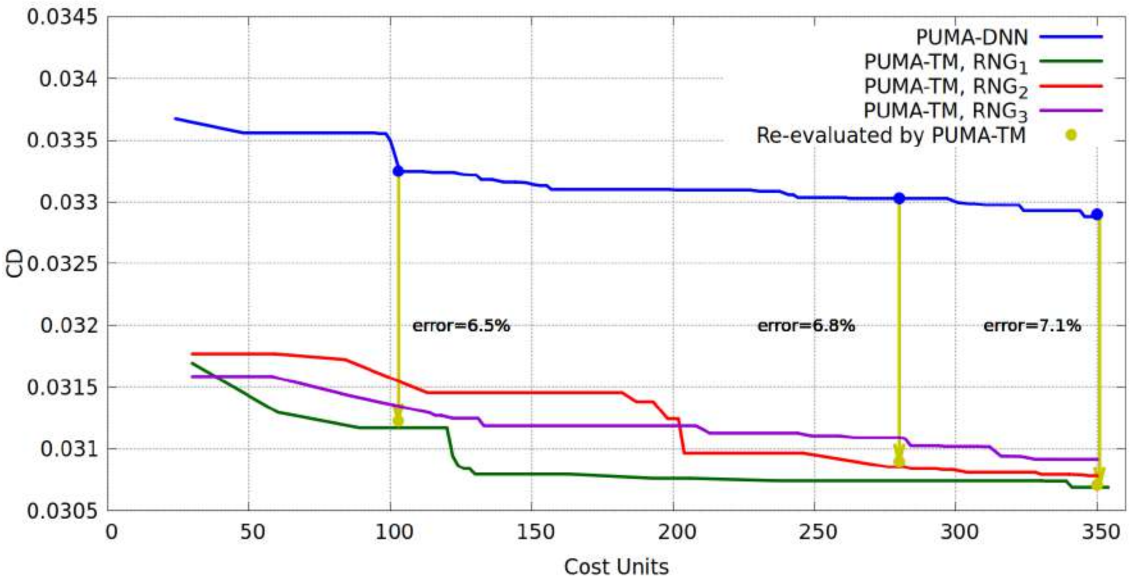
Το εναλλακτικό μοντέλο είναι πιο γρήγορο απο το ακριβές με μία αξιολόγηση να ισοδυναμεί με 0.8 φορές τη μονάδα κόστους. Έτσι, πραγματοποιούνται 440 αξιολογήσεις, οι οποίες ισοδυναμούν περίπου με 350 μονάδες κόστους (ή 350 αξιολογήσεις με το ακριβές μοντέλο). Το ΒΝΔ δεν χρειάστηκε επανεκπαίδευση καθόλη τη βελτιστοποίηση.

Τα αποτελέσματα όλων των βελτιστοποιήσεων αναγράφονται στον Πίνακα. 7.2, όπου για το ακριβές μοντέλο, αναγράφεται ο μέσος όρος των αποτελεσμάτων απο τη χρήση των τριών διαφορετικών RNG.

Πίνακας 7.2
Τα αποτελέσματα της βελτιστοποίησης

Λογισμικό αξιολόγησης	Βέλτιστη λύση (CD/CL)	Επαναξιολόγηση (CD/CL)	Ποσοστιαία μείωση του CD
Ακριβές μοντέλο (μέσος όρος)	0.0309/1.506	-	-3.46%
Εναλλακτικό μοντέλο	0.03288/1.506	0.0307/1.508	-3.91%

Στο Σχ. 7.3 φαίνονται οι πορείες σύγκλισης των βελτιστοποιήσεων.



Σχήμα 7.3: *Η πορεία σύγκλισης της βελτιστοποίησης με ΜΑΕΑ βασιζόμενη στα $PUMA-DNN$ και $PUMA-TM$. Το $PUMA-DNN$ λογισμικό αξιολόγησης επιταχύνει την βελτιστοποίηση συγκριτικά με το $PUMA-TM$ με επιλεγμένες τις RNG_2 και RNG_3, ενώ παρουσιάζει σχεδόν όμοια πορεία σύγκλισης όταν επιλέγεται η RNG_3.*

Συμπεράσματα

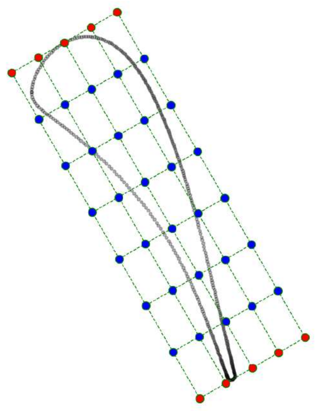
Τα αποτελέσματα από τη χρήση του μοντέλου $PUMA-DNN$ σε μελέτες βελτιστοποίησης σχήματος είναι ενθαρρυντικά, παρόλο που το κόστος της εκτέλεσης του $PUMA-TM$ είναι ελάχιστο σε αυτήν την περίπτωση. Παρόλο που υπάρχει μόνο μια μικρή βελτίωση στο συνολικό χρόνο της βελτιστοποίησης λαμβάνοντας υπόψη τη μέση απόδοση του $PUMA-TM$ με τη χρήση των διαφορετικών RNG, αυτό χρησιμεύει ως μια πρώτη ένδειξη ότι το κέρδος μπορεί να είναι ακόμη μεγαλύτερο σε περιπτώσεις όπου οι αξιολογήσεις είναι πιο δαπανηρές.

Χρήση ΒΝΔ ως Υποκατάστατο του Μοντέλου Τύρβης k-ω και Μετάβασης $\gamma-\tilde{R}e_{\theta t}$ στην Αεροδυναμική Ανάλυση και Βελτιστοποίηση του Σχήματος Πτερύγωσης Αξονικού Στροβίλου $LS89$

Η γεωμετρία της αεροτομής του στροβίλου παραμετρικοποιείται με τη χρήση ενός κουτιού μορφοποίησης $NURBS$ αποτελούμενο από 8X5 σημεία ελέγχου όπως φαίνεται στο Σχήμα.7.4. Τα σημεία ελέγχου με μπλε χρώμα, επιτρέπεται να κινούνται κατά τη χορδή και κάθετα αυτής εντός 10% των τιμών αναφοράς τους. Τα σημεία ελέγχου με κόκκινο χρώμα, κρατούνται σταθερά. Συνεπώς, υπάρχουν 60 μεταβλητές σχεδιασμού.

Τα 150 δείγματα των ΔΕ παράγονται με την τεχνική LHS και η ροή μοντελοποιείται με τον επιλύτη των $RANS$ εξισώσεων με το μοντέλο τύβης k-ω και μετάβασης

$\gamma - \tilde{R}e_{\theta t}$ (ακριβές μοντέλο ή $PUMA - TM$). Η μοντελοποίηση της ροής των 150 γεωμετριών χρειάζεται περίπου 26 ώρες στην $NVIDIA\ GeForce\ RTX$ 2070. Τα ΔΕ κανονικοποιούνται σύμφωνα με την εξίσωση 7.13.



Σχήμα 7.4: *Τα σημεία ελέγχου της γεωμετρίας του $LS89$.*

Επειδή το ΒΝΔ καλείται να υποκαταστήσει ένα σύνολο τεσσάρων διαφορικών εξισώσεων (μοντέλα τύρβης και μετάβασης) κάνοντας την εύρεση του κατάλληλου δικτύου δύσκολη, μία καλή πρακτική είναι η βελτιστοποίηση των υπερ-παραμέτρων και των εισόδων του με $MAEA$ που υλοποιείται στο λογισμικό $EASY$. Ο στόχος της βελτιστοποίησης είναι η μείωση του σφάλματος του ΒΝΔ εκπαιδευόμενου στο σύνολο των ΔΕ. Ωστόσο, το προκύπτον βέλτιστο δίκτυο παρουσιάζει υπερπροσαρμογή στα ΔΕ ($overfitting$). Η μείωση του σφάλματος στα ΔΕ δεν συνεπάγεται και ταυτόχρονη βελτίωση της ικανότητας γενίκευσης των ΒΝΔ και, επειδή δεν είναι εφικτό να αξιολογηθεί η ικανότητα γενίκευσης κατά τη διαδικασία της βελτιστοποίησης, κρίνεται απαραίτητη η δοκιμή των ΒΝΔ σε νέες γεωμετρίες.

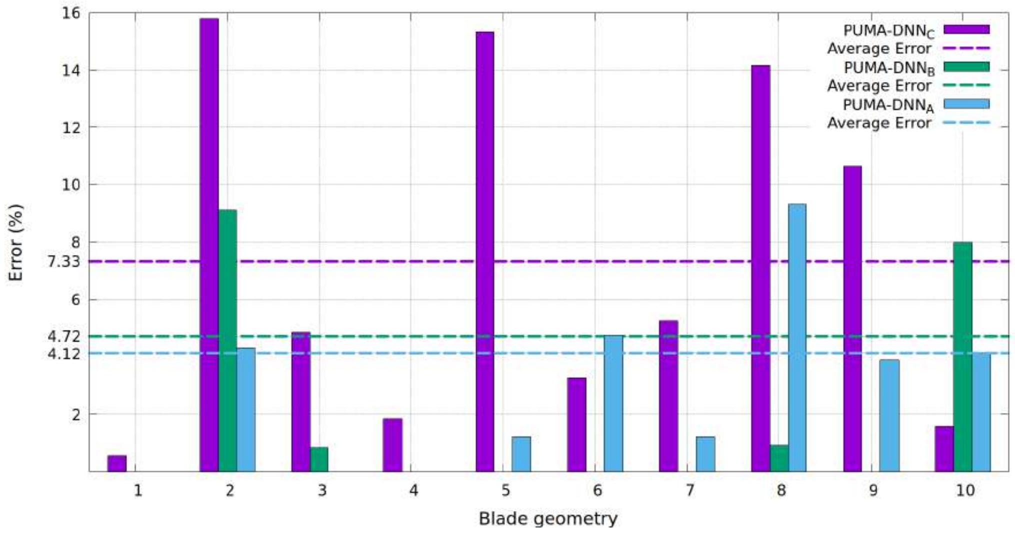
Τα επιλεγμένα ΒΝΔ τα οποία δοκιμάζονται σε νέες γεωμετρίες είναι το συνολικά βέλτιστο (DNN_C) και δύο ακόμα από το σύνολο των καλύτερων λύσεων της βελτιστοποίησης, τα οποία συμβολίζονται ως DNN_B και DNN_A. Ένας τρόπος να αντιμετωπιστεί η υπερ-προσαρμογή είναι μειώνοντας την πολυπλοκότητα της αρχιτεκτονικής του δικτύου. Επομένως, τα δύο τελευταία δίκτυα, επιλέγονται καθώς έχουν τα λιγότερα κρυφά επίπεδα και παράλληλα μικρό σφάλμα. Στον Πίνακα.7.3 αναγράφονται οι διαμορφώσεις των επιλεγμένων ΒΝΔ.

Πίνακας 7.3

Βέλτιστα DNN	Επίπεδα	Νευρώνες/Επίπεδο	Δεδομένα-Εισόδου	Συνάρτηση Ενεργοποίησης
DNN_A	3	256, 2048, 2048	$(x, \rho, p, d, u_k, \Omega, S)$	$ReLu/tanh$
DNN_B	3	128, 64, 2048	(y, ρ, p, d, u_k, S)	$ReLu/tanh$
DNN_C	10	256,2048,4096,1024,32,64,4096,256,256,1024	$(x_k, \rho, p, d, u_k, \Omega, S)$	$ReLu/tanh$

Τα σφάλματα των ΒΝΔ εκτιμώνται ως προς τις υπολογιζόμενες απώλειες ολικής πίεσης εν σύγκριση με αυτές του ακριβούς μοντέλου σε 10 νέες γεωμετρίες. Τα ποσοστιαία σφάλματα των τριών ΒΝΔ σε 10 νέες γεωμετρίες φαίνονται στο Σχ.7.5, όπου είναι φανερό πως το μοντέλο $PUMA-DNN_A$ έχει το μικρότερο μέσο όρο σφάλματος σε νέες γεωμετρίες και, άρα, τη μεγαλύτερη ικανότητα γενίκευσης. Η επίλυση της ροής στην 1η και 4η γεωμετρία με χρήση του DNN_A απέκλινε, ωστόσο, το πρόβλημα αυτό μπορεί να διευθετηθεί με περαιτέρω ρύθμιση του ΒΝΔ όπως θα φανεί στη συνέχεια. Επομένως το DNN_A επιλέγεται ως το βέλτιστο ΒΝΔ.

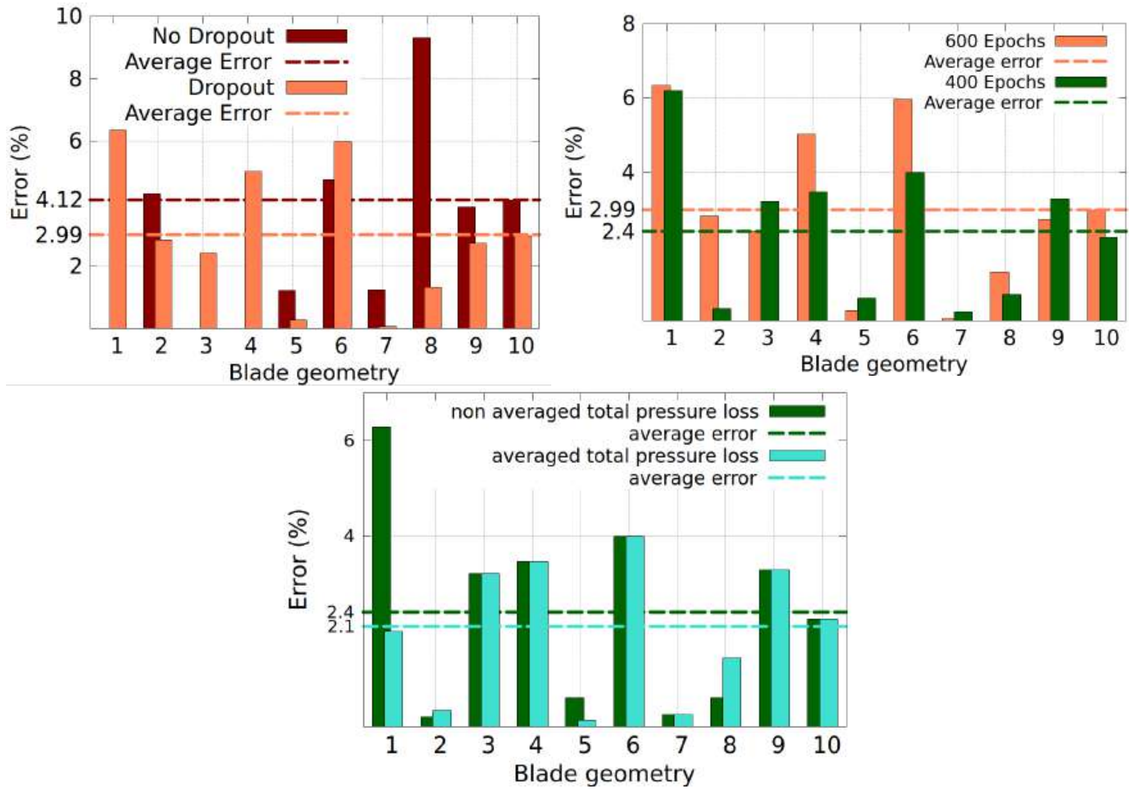
Για την περαιτέρω αύξηση της ικανότητας γενίκευσης του ΒΝΔ καθώς και την επίλυση του προβλήματος απόκλισης, εφαρμόζονται δύο τεχνικές, οι οποίες ονομάζονται *dropout* και *early-stopping*. Επιπλέον, η επαναληπτική αριθμητική επίλυση των $RANS$ εξισώσεων με το ΒΝΔ παρουσιάζει, σε μερικές περιπτώσεις, μία ταλαντωτική συμπεριφορά. Για την αντιμετώπιση αυτού του φαινομένου, μια πρακτική είναι να λαμβάνεται υπόψη ο μέσος όρος των απωλειών ολικής πίεσης λ.χ των τελευταίων 1000 επαναλήψεων αντί της τελικής τιμής. Η επίδραση των τεχνικών αυτών στα ποσοστιαία σφάλματα του ΒΝΔ στις 10 νέες γεωμετρίες, φαίνονται στο Σχ.7.6, όπου το μέσο σφάλμα του νευρωνικού μειώθηκε συνολικά κατά 2.11%.



Σχήμα 7.5: *Τα ποσοστιαία σφάλματα των $PUMA-DNN_A$, $PUMA-DNN_B$ και $PUMA-DNN_C$ σε νέες γεωμετρίες.*

Αφού έχει επιλεχθεί και εκπαιδευτεί το βέλτιστο ΒΝΔ, πραγματοποιείται βελτιστοποίηση σχήματος της γεωμετρίας του πτερυγίου στροβίλου με $MAEA$, με χρήση δύο λογισμικών αξιολόγησης. Το πρώτο είναι ο επιλύτης των $RANS$ εξισώσεων με το μοντέλο τύρβης k-ω και μετάβασης $\gamma-\tilde{R}e_{\theta t}$ (ακριβές μοντέλο - $PUMA-TM$) και το δεύτερο είναι ο επιλύτης των $RANS$ εξισώσεων με το μοντέλο του ΒΝΔ (εναλλακτικό μοντέλο - $PUMA-DNN$). Στόχος της βελτιστοποίησης είναι η ελαχιστοποίηση των απωλειών ολικής πίεσης, διατηρώντας σταθερή τη γωνία εξόδου της ροής.

Η βελτιστοποίηση σχήματος με ΜΑΕΑ βασιζόμενη στο ακριβές μοντέλο, πραγματο-



Σχήμα 7.6: *Τα ποσοστιαία σφάλματα σε κάθε νέα γεωμετρία με ή χωρίς τη χρήση του dropout (αριστερά), τα ποσοσταία σφάλματα με ή χωρίς early stopping και dropout (δεξιά) και τα ποσοστιαία σφάλματα με ή χωρίς την χρήση του μέσου όρου (κάτω).*

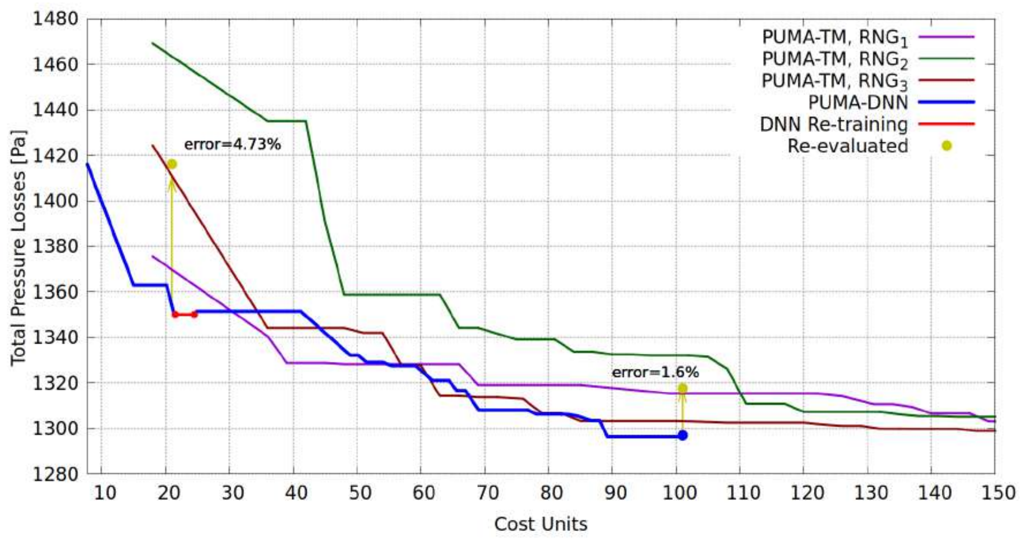
ποιείται για τρείς *RNG seeds* με κριτήριο τερματισμού τις 150 αξιολογήσεις. Μία μονάδα κόστους ισοδυναμεί με μία αξιολόγηση του ακριβούς μοντέλου.

Το εναλλακτικό μοντέλο σε αυτήν την περίπτωση είναι περίπου δύο φορές πιο ταχύ απο το ακριβές και, συνεπώς, μια αξιολόγηση με το $PUMA-DNN$ ισοδυναμεί περίπου με μισή μονάδα κόστους. Κατά τη διαδικασία της βελτιστοποίησης, το ΒΝΔ επανεκπαιδεύεται μετά από 50 αξιολογήσεις και συνεχίζει για ακόμα 200 αξιολογήσεις όπου και η βελτιστοποίηση τερματίζεται, καθώς δεν μπορεί να βελτιώσει περαιτέρω τη βέλτιστη λύση. Τα αποτελέσματα φαίνονται στον Πίνακα.7.4

Πίνακας 7.4

Λογισμικό Αξιολόγησης	Βέλτιστη λύση ($P_{t,losses}$)	Επαναξιολόγηση ($P_{t,losses}$)	Ποσοστό %
$PUMA-TM$ (μέσος όρος)	1302 $[Pa]$	-	-11.3%
$PUMA-DNN$	1296 $[Pa]$	1318 $[Pa]$	-10.3%

Στο Σχ.7.7 φαίνονται οι πορείες των βελτιστοποιήσεων με ΜΑΕΑ.



Σχήμα 7.7: *Η πορεία της βελτιστοποίησης με ΜΑΕΑ βασιζόμενη στο* $PUMA-DNN$ *και στο* $PUMA-TM$. *Οι επαναξιολογημένες λύσεις του* $PUMA-DNN$ *απο το ακριβές μοντέλο είναι χρωματισμένες με κίτρινο. Το μοντέλο* $PUMA-DNN$ *οδηγεί σε γρηγορότερη σύγκλιση, ωστόσο, λαμβάνοντας υπόψη το σφάλμα, η τελική λύση είναι χειρότερη αυτής του ακριβούς μοντέλου.*

Συμπεράσματα

Απο τις μελέτες, συμπεραίνεται πως το ΒΝΔ μπορεί να υποκαταστήσει αποτελεσματικά το μοντέλο τύρβης k-ω και μετάβασης $\gamma - \widetilde{Re}_{\theta t}$ και να βοηθήσει τη βελτιστοποίηση σχήματος με ΜΑΕΑ. Το μοντέλο $PUMA-DNN$ έχει ένα κόστος αξιολόγησης το μισό απο αυτό του ακριβούς μοντέλου, με αποτέλεσμα την ταχύτερη σύγκλιση. Όσο το υπολογιστικό κόστος του ακριβούς μοντέλου είναι μεγαλύτερο, τόσο υψηλότερο όφελος αναμένεται από τη χρήση ΒΝΔ. Βέβαια απαιτείται ακόμη αρκετή διερεύνηση ώστε και να αυτοματοποιηθελι περισσότερο η διαδικασία και να αυξηθεί το όφελος από τη χρήση ΒΝΔ.

Bibliography

[1] Kontou M., Asouti V, Giannakoglou K.: *DNN surrogates for turbulence closure in CFD-based shape optimization.* Applied Soft Computing, 134:1-11, January 2023, `https://doi.org/10.1016/j.asoc.2023.110013`

[2] Spalart, P., Allmaras, S.: *A One-Equation Turbulence Model for Aerodynamic Flows*, 30th Aerospace Sciences Meeting and Exhibit, 1992. https://doi.org/10.2514/6.1992-439, 10.2514/6.1992-439.

[3] Menter, F.R.: *Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications*, AIAA Journal, 32(8):1598-1605, 1994. doi:10.2514/3.12149, 10.2514/3.12149.

[4] Langtry, R.B., and Menter, F.R.: *Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes*, AIAA Journal, 47(12), 2894-2906. doi:10.2514/1.42362, 10.2514/1.42362.