Design Optimization of Switched Reluctance Motors Using Deep Learning

DESIGN OPTIMIZATION OF SWITCHED RELUCTANCE MOTORS USING DEEP LEARNING

BY

YOUSSEF ASHAM, B. Eng. Mgmt.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2024) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Design Optimization of Switched Reluctance Motors Using Deep Learning
AUTHOR:	Youssef Asham B. Eng. Mgmt., Electrical Engineering McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Mohamed Bakr
CO-SUPERVISOR:	Dr. Ali Emadi
NUMBER OF PAGES:	lxxvii, 87

To my Parents and Family

Abstract

Switched Reluctance Motors (SRMs) are known for their low manufacturing costs, simple structure, high torque-to-inertia ratio, and minimal maintenance. However, they have high torque ripples due to the salient nature of the stator and rotor poles. To address this problem, this thesis applies a Deep Learning optimization approach to minimize the torque ripple and maximize the average torque by changing the stator and rotor pole arc angles, β_s and β_r respectively, using Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) implemented in Python.

The training data consists of cross-section images of a 6/14 SRM and static and dynamic characteristics were captured using the Finite Element Method and MATLAB Simulink models. The CNN model takes the cross-section image and predicts the average torque and torque ripple. The results of the CNN model are compared to previous papers that applied a similar method to predict various parameters of different motors.

A variation of the GAN model called FastGAN was used to generate cross section motor images. The model takes a noise vector and generates a cross-section quarter model image.

The combined FastGAN-CNN model produced an optimal design that increases the average torque by 2% and decreases the torque ripple by 24% while being 17 times faster than the traditional FEM.

Acknowledgements

I would like to thank God for always being with me throughout this journey. Without his blessings and grace, I would not have been able to finish.

I would like to thank my supervisor Dr. Mohamed Bakr for his guidance and for his patience with me. I would like to also thank Drs. Mohamed Bakr, Ali Emadi, Mohamed Elamien, and Philip Kollmeyer for being on my defense committee and for their valuable feedback. I would like to thank Dr. Ali Emadi for his encouragement. Thank you both for giving me the opportunity to work on some of the projects at the McMaster Automotive Resource Center. It was an honor to be part of the team.

I would like to thank my parents, Nazih and Neveen, for their unwavering support throughout my masters degree. I would like to thank my brother Yousam Asham for his assistance with some of the programming challenges I faced. You are the only constant in my life.

Special mention to Mohamed Omar for providing me with the 6/14 SRM dataset and for his encouragement throughout this journey. I would like to also thank my friends at McMaster and outside of McMaster for their moral support.

Nomenclature

Abbreviations

SRM	Switched Reluctance Motor
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
GDA	Gradient Descent Algorithm
BPNN	Back-propagating Neural Network
GRNN	Generalized Regression Neural Network
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
MBGD	Mini-Batch Gradient Descent
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation
GAN	Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
cGAN	Conditional Generative Adversarial Network
cCGAN	Continuous Conditional Generative Adversarial Network
SLE	Skip Layer Excitation
GLU	Gated Linear Unit

MAE	Mean Absolute Error
MRE	Mean Relative Error
MSE	Mean Squared Error
SSIM	Structural Similarity Index Measurement

Symbols

v	Phase Voltage
R_m	Phase Motor Resistance
L_{ph}	Phase Inductance
i	Phase Current
λ	Flux Linkage
θ	Rotor Position
N	Number of Turns
μ_r	Relative Permeability
μ_0	Permeability of Free Space
A_c	Cross Section Area of the Air Gap
l_c	Air Gap Length
W_m	Mechanical Energy
W_f	Magnetic Energy
T	Torque
ω	Rotor Speed
W_c	Co-Energy
T_{avg}	Average Torque
ΔT_{RMS}	Torque Ripple
w_{jk}^{l}	Weight from Neuron k in Layer $l-1$ to Neuron j in Layer l
g	Activation Function
$\mathcal{L}$	Loss Function

β_s	Stator Pole Arc Angle
β_r	Rotor Pole Arc Angle
η	Learning Rate
α	Momentum
G	Generator Model
D	Discriminator Model
z	Noise Vector
c	Conditional Input
p_{data}	Probability Distribution of Dataset
p_g	Probability Distribution of Generated Images
p_z	Probability Distribution of Noise

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The world is slowly shifting towards using Electric Vehicles (EVs) for transportation. In December 2023, the Canadian environment minister announced that by 2035 only EVs will be sold [1]. As the production of EVs increases, so does the production of electric motors. More emphasis will be placed on improving their performance. There are several kinds of motors used in EVs. Some of these motors are Permanent Magnet Synchronous Motors (PMSMs), Induction Motors (IMs) and Switched Reluctance Motors (SRMs).

What provides SRMs a competitive edge over other motors is that they require no rare-earth materials to manufacture, making them cheaper to produce [2]. SRMs also have a simpler construction than PMSMs and IMs as the stator has simple salient poles, concentrated windings around each stator pole, and a rotor with no rotor bars or permanent

magnets (see Figure 1.1). As a result of this simple rotor design, SRMs have a high torque-to-inertia ratio [3] and low maintenance.

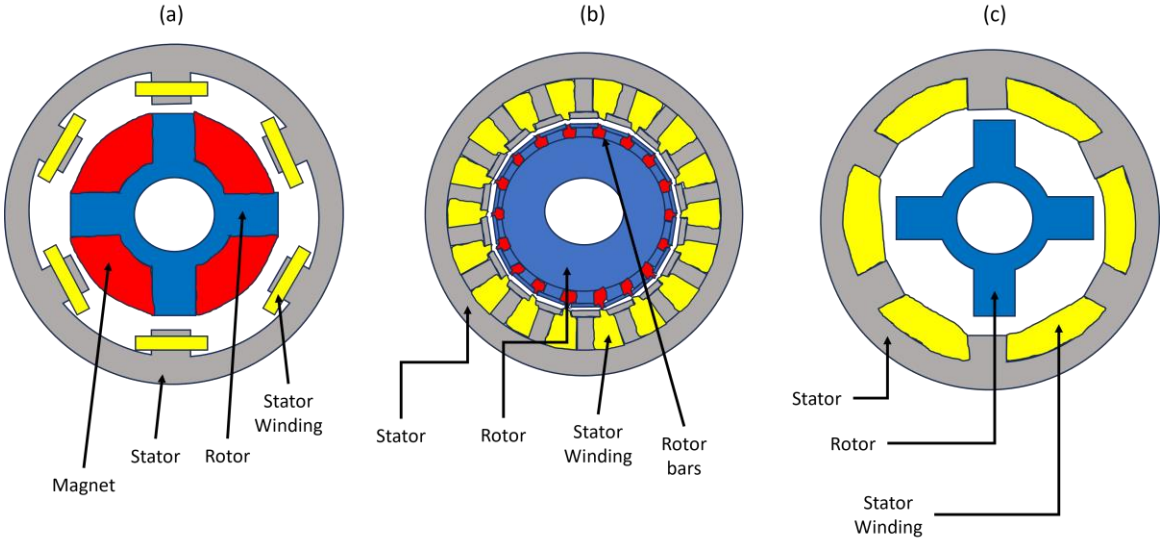

Figure 1.1: Comparison between (a) PMSM, (b) IM, and (c) SRM motor structures.

Despite these advantages, SRMs still face challenges. One of these challenges is high torque ripples due to the salient nature of the stator and rotor poles [2]. High acoustic noise is another disadvantage due to the high radial and axial forces in SRMs. Depending on the geometry dimensions of the motor, the torque ripples can be reduced [3]. To achieve this, different optimization techniques can be used. One of the techniques that has increased in popularity in the past five years is the implementation of Deep Learning (DL). To the author's knowledge, there has been no previously reported work on the application of DL to the design of SRMs

1.2 Contribution

This work contributed to the following developments, which are presented in this thesis and summarized as follows:

1. A literature review covering the modeling techniques and design optimization techniques of SRMs using Machine Learning (ML) and DL.
2. A data augmentation algorithm developed to train the convolutional neural network (CNN) model and the FastGAN model using computer vision techniques.
3. A CNN model that takes a quarter model cross section image of a motor and identifies the average torque and torque ripple with very high accuracy.
4. A FastGAN model that generates quarter model cross section motor images given a noise vector. To verify the correct images are generated, another CNN model is trained to predict the stator and rotor pole arc angles given a motor image.
5. An optimization algorithm that identifies the motor design that produces the lowest torque ripple and the corresponding average torque, stator pole arc angle, rotor pole arc angle, and cross-section motor image.

1.3 Outline

This thesis is outlined as follows:

Chapter 2 presents the necessary background on SRM operational characteristics and the different electromagnetic modeling techniques used to design SRMs.

Chapter 3 looks at the applications of ML algorithms to the design optimization of SRMs.

In chapter 4, background and literature review on how DL is used to optimize design parameters of SRMs is presented. This is achieved by looking at CNNs and generative adversarial networks (GANs) as they are the most common models used when processing images.

Chapter 5 presents the methodology used to develop the CNN and FastGAN by looking at the data preprocessing and cleaning procedure. The data cleaning algorithm, the two models' architecture, and optimization algorithm will be presented.

Chapter 6 looks at the results obtained from the two models by looking at their performance metrics. The optimal design candidate chosen by the optimization algorithm is also presented. Moreover, a discussion of the results obtained from the CNN-FastGAN and how they are compared to previous papers' models is shown.

Finally, chapter 7 concludes this thesis and provides any potential future work.

Chapter 2

SRM Operational Characteristics and Modeling Techniques

2.1 SRM Operational Characteristics

Before discussing the modeling techniques, it is important to have a general understanding of the operating characteristics of SRMs. The stator and rotor are made of laminated ferromagnetic materials with windings on the stator only. The windings contain current that is independently excited from a power converter circuit [4]. Looking at one phase of the motor, the SRM's operation follows the laws of electromagnetism. The single-phase diagram and equivalent electrical circuit are shown in Figure 2.1.

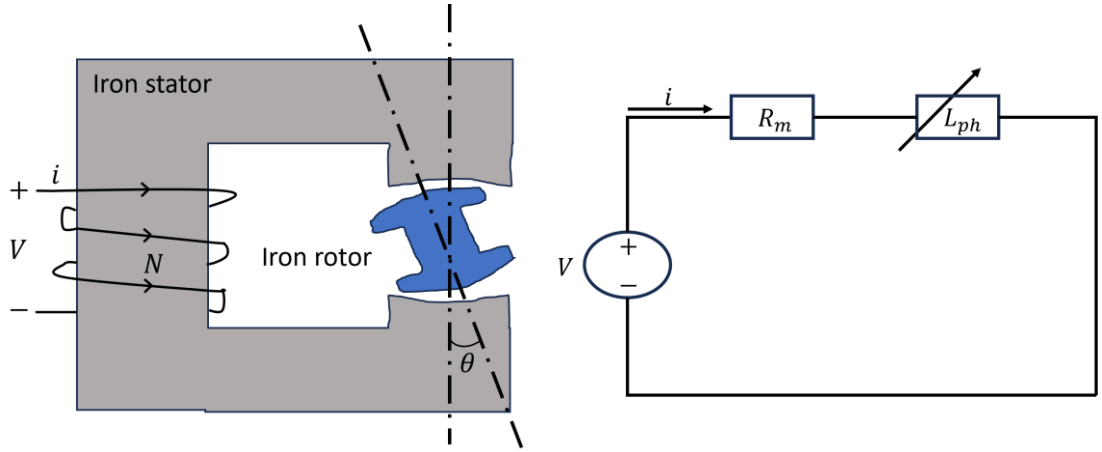

Figure 2.1: The (a) single phase SRM diagram and (b) equivalent electric circuit.

From first principles, the governing equation for the voltage across a single phase of the SRM is given by [4]:

$$v = R_m i + \frac{\partial \lambda}{\partial t}, \tag{2.1}$$

where v is the phase voltage, i is the phase current, R_m is the phase motor resistance, and λ is the flux linkage. In integral form equation (2.1) with respect to time becomes:

$$\lambda(t) = \int_0^t v(t) - R_m i(t) dt + \lambda(0), \tag{2.2}$$

where $\lambda(0)$ is the initial flux linkage at $t = 0$. The flux linkage varies with the rotor position θ and i [4]. This can be explained by expanding equation (2.1):

$$v = R_m i + \frac{\partial \lambda}{\partial i}\frac{di}{dt} + \frac{\partial \lambda}{\partial \theta}\frac{d\theta}{dt}, \tag{2.3}$$

where $\partial\lambda/\partial i$ is the instantaneous inductance and $\partial\lambda/\partial\theta$ is the instantaneous back EMF. It can be seen that the relationship between λ, θ, and i is non-linear in nature, making it challenging to model the electromagnetic characteristics of an SRM. An ideal flux linkage plot with respect to the rotor position for one electrical cycle is shown in Figure 2.2. The inductance-rotor position plot is almost the same as Figure 2.2. The flux linkage is at its maximum when the rotor pole is fully aligned with the stator pole and at its minimum when the rotor and stator poles are unaligned. Figure 2.3 shows the difference visually. This is because the flux linkage is inversely proportional to the air gap length between the stator and rotor poles as stated by the following equation:

$$\lambda = \frac{N^2 i \mu_r \mu_0 A_c}{l_c}, \tag{2.4}$$

where N is the number of turns, μ_r is relative permeability, μ_0 is permeability of free space, A_c is the cross section of the air gap, and l_c is the air gap length. Due to the salient structure of the stator and rotor poles in SRMs, the airgap is non-uniform. This affects l_c and causes a non-linear relationship between λ and θ [5].

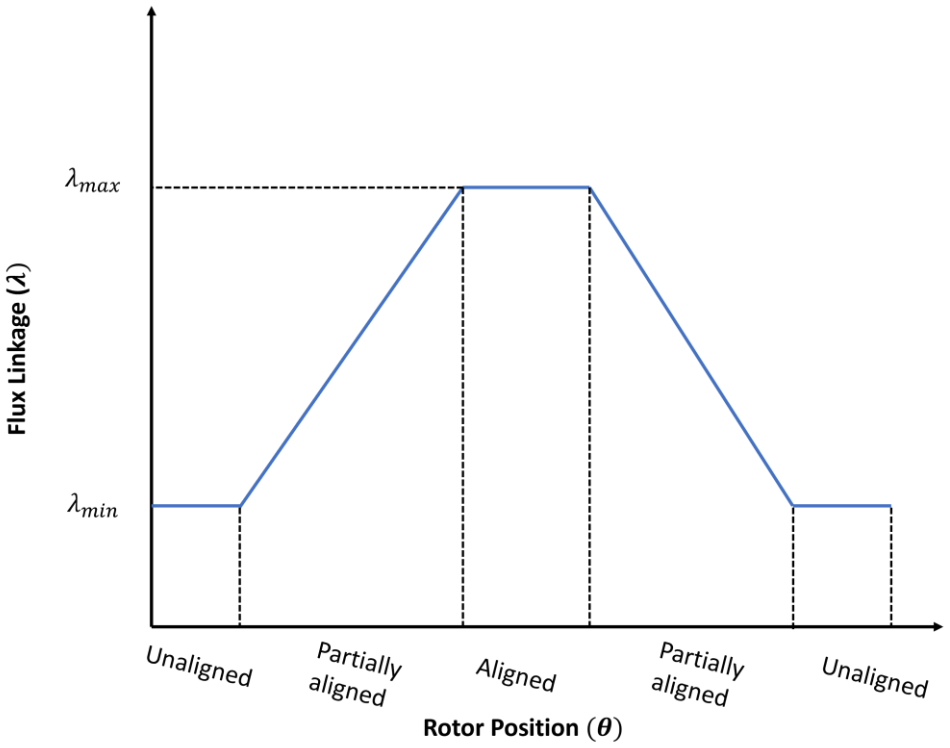

Figure 2.2: Ideal flux linkage-rotor position plot.

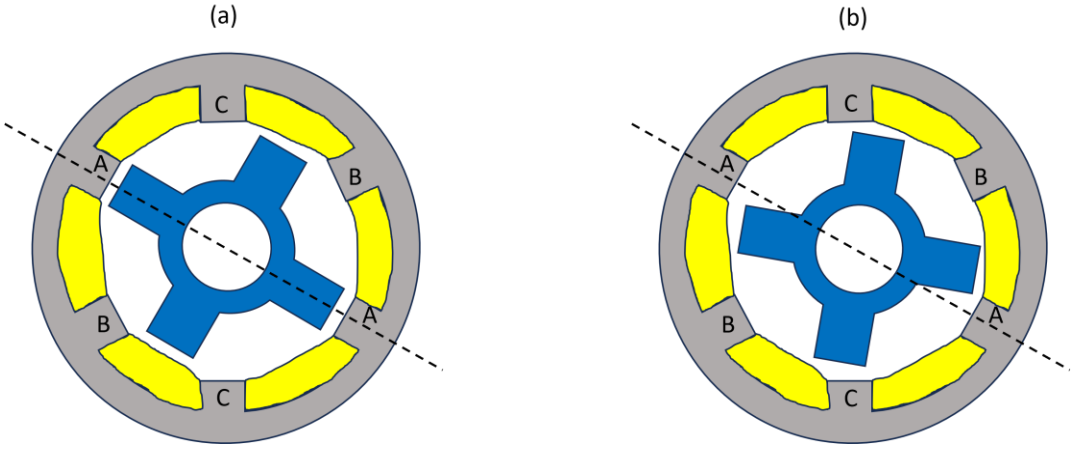

Figure 2.3: SRM rotor position relative to phase A in the (a) aligned and (b) unaligned positions.

Multiplying i on both sides of equation (2.1) yields the equation:

$$vi = R_m i^2 + i\frac{\partial \lambda}{\partial t}. \tag{2.5}$$

The left-hand side of equation (2.5) is the total instantaneous power delivered to the SRM. The first term on the right-hand side is the windage losses. For the power to be conserved, this means the second term on the right-hand side is the sum of the mechanical and magnetic power output of the SRM [4]. In other words, $i\,\partial\lambda/\partial t$ can be represented as:

$$i\frac{\partial \lambda}{\partial t} = \frac{\partial W_m}{\partial t} + \frac{\partial W_f}{\partial t}, \tag{2.6}$$

where $\partial W_m/\partial t$ is the instantaneous mechanical power and $\partial W_f/\partial t$ is the instantaneous magnetic power. By definition, mechanical power is the product of torque and speed,

$$\frac{dW_m}{dt} = T\omega = T\frac{d\theta}{dt}, \tag{2.7}$$

where T is the torque and ω is the angular velocity of the rotor. Substituting equation (2.7) into (2.6) and solving for T results in:

$$i\frac{\partial \lambda}{\partial t} = T\frac{\partial \theta}{\partial t} + \frac{\partial W_f}{\partial t} \tag{2.8}$$

$$T(\theta, \lambda) = i(\theta, \lambda)\frac{\partial \lambda}{\partial \theta} - \frac{\partial W_f(\theta, \lambda)}{\partial \theta}. \tag{2.9}$$

Assuming that the flux linkage does not change with rotor position, $\partial\lambda/\partial\theta = 0$, the torque is thus defined as:

$$T = -\frac{\partial W_f}{\partial \theta}. \tag{2.10}$$

Normally torque is represented in terms of co-energy W_c instead of magnetic power [4]. To understand the concept of co-energy, recall from equation (2.4) that there is a relationship between flux linkage and phase current. This is represented in a plot called the magnetization curve. Integrating equation (2.8) with respect to λ and assuming a constant rotor position, $\partial\theta/\partial t = 0$, this results in the following integral,

$$W_f = \int_0^{\lambda} i(\theta, \lambda) d\lambda. \tag{2.11}$$

Unlike W_f, which is the integral of current as a function of flux, W_c is the integral of the magnetization curve:

$$W_c = \int_0^{i} \lambda(\theta, i) di \tag{2.12}$$

Figure 2.4 shows an example of a magnetization curve along with the visual representation of magnetic and co-energy.

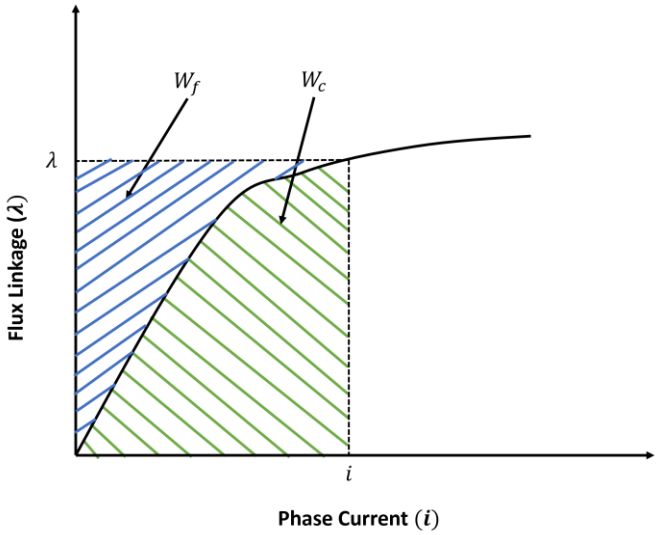

Figure 2.4: Graphical representation of magnetic energy and co-energy.

Torque is represented as the change in co-energy with respect to the rotor position under constant current [4][6],

$$T = \frac{dW_c}{d\theta}\Big|_{i=constant}. \tag{2.13}$$

Two other parameters related to torque that are studied are average torque, T_{avg}, and torque ripple, ΔT_{RMS} [3].

$$T_{avg} = \frac{1}{\theta_{cycle}} \int_0^{\theta_{cycle}} T(\theta) d\theta, \tag{2.14}$$

$$\Delta T_{RMS} = \sqrt{\frac{1}{\theta_{cycle}} \int_0^{\theta_{cycle}} \left[T(\theta) - T_{avg}\right]^2 d\theta}, \tag{2.15}$$

where θ_{cycle} is one complete electrical cycle. It can be seen in Figure 2.4 that to operate at a high torque, the phase current need to be high. However, the rate of change of the flux linkage decreases as i increases. This is known as the saturation region and is where the SRM operates. This further shows that there is a non-linear relationship between the flux linkage, rotor position, and current. To improve the performance of SRMs, different modeling techniques have been used to address this non-linear relationship and improve parameters such as average torque, torque ripple, radial force, and loss density on the stator and rotor [5]. In the next section, these modeling techniques will be discussed.

2.2 Modeling Techniques of SRMs

SRM modeling techniques can be divided into two main categories: numerical and analytical techniques [5][6]. Figure 2.5 summarizes the two main techniques.

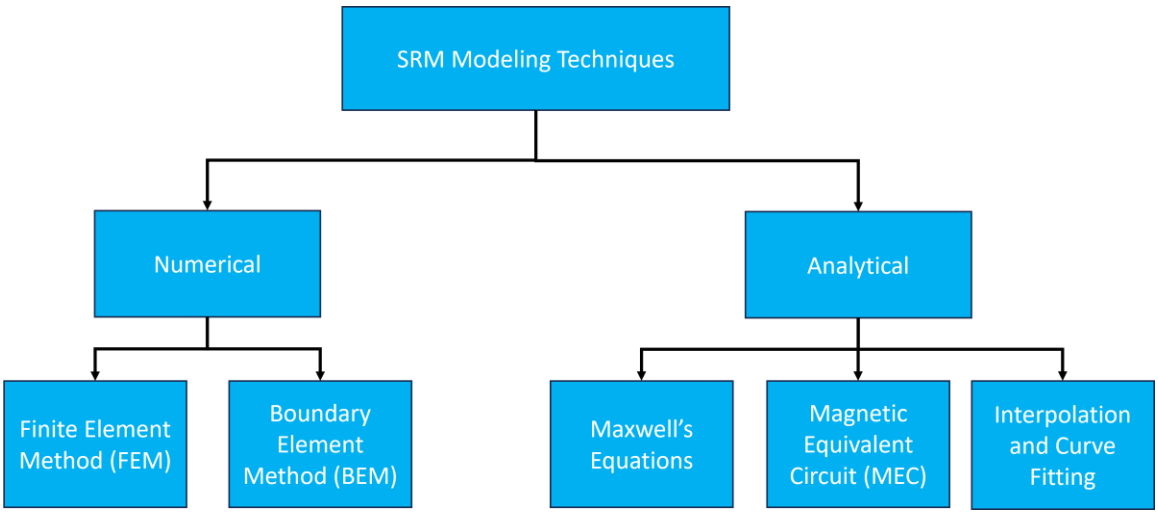

Figure 2.5: Summary of modeling techniques of SRMs.

2.2.1 Numerical Techniques

The two most common numerical techniques are the finite element method (FEM) and boundary element method (BEM). FEM is used in the design and optimization of SRMs [7]. The method involves discretizing the machine geometry into finite elements to create meshes and a system of equations for each mesh is obtained. The series of equations for all the meshes is then collected and solved and post-processed to obtain quantities such as torque or flux linkage [7]. An example procedure of FE analysis is shown in Figure 2.6. FEM is very accurate and it is highly dependent on the number of finite elements formed. Since SRM operates in the saturation region, dense finite elements are needed in such a

region. Moreover, an accurate mesh is needed near the airgaps because the rotor position is always varying. This is why the main drawback of FEM is high computational cost.

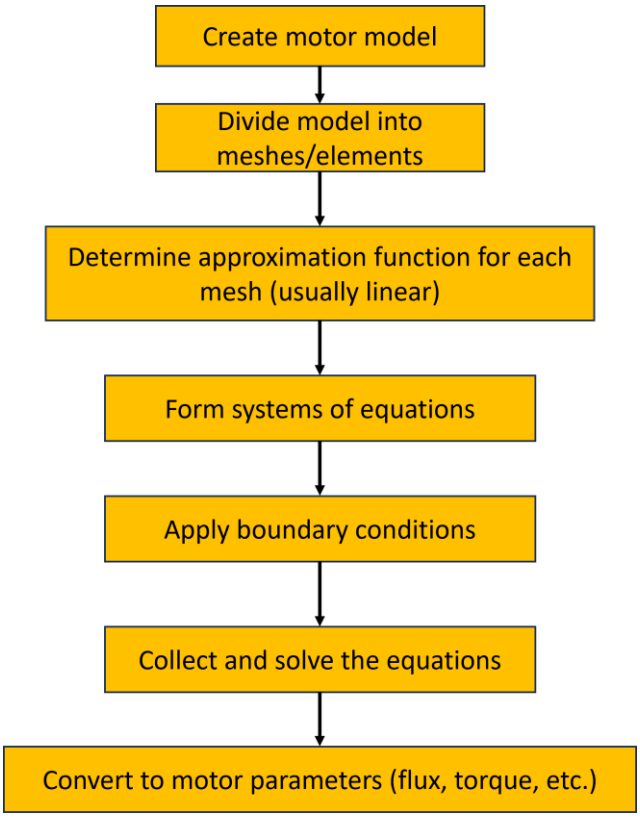

Figure 2.6: Example of FEM procedure.

BEM is the second numerical technique used to optimize SRMs. It utilizes integral equations to solve electromagnetic fields on the boundary domain [6][7]. The advantage of BEM is that it uses surface meshes whereas FEM uses volume meshes, thus reducing the computation time [5]. The drawback however is that BEM requires more memory resources compared to FEM due to highly dense coefficient matrices, especially in the saturation region of SRM [6]. This is why it is combined to solve electromagnetic fields with another method such as FEM or an analytical method called magnetic equivalent

circuit (MEC). For instance, BEM is used to solve electromagnetic fields in the linear region and FEM is used in the saturation region.

2.2.2 Analytical Techniques

Analytical techniques consist of Maxwell's equations, MEC, and interpolation and curve fitting. Interpolation and curve fitting approaches are comprised of four sub-techniques: lookup tables, interpolation, Fourier series, and ML/DL algorithms [6][7]. Figure 2.7 displays the analytical techniques used for SRMs.

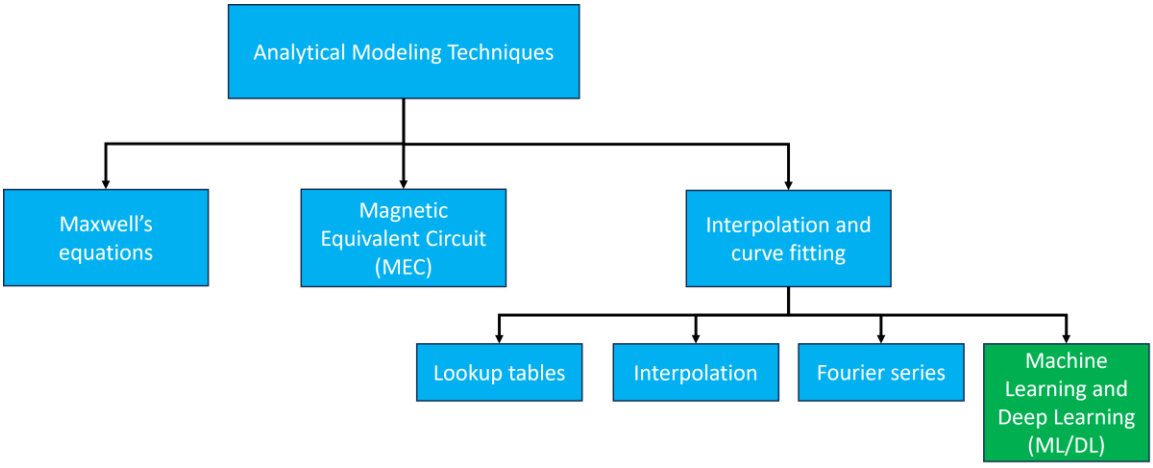

Figure 2.7: Types of analytical methods for SRM modeling.

Maxwell's equations is a method used to investigate the magnetic vector potential or magnetic scalar potential [7]. A set of equations are solved under boundary conditions and transformations take place to identify the electromagnetic field in the airgap. Similar to BEM, this method is accurate in the linear region, but it is challenging to apply in local saturation regions[6].

MEC is a popular technique to analyze electromagnetic fields and design various electric machines. The motor geometry is represented as a magnetic circuit and the flux distribution is calculated of that circuit. From there the flux density, magnetic field intensity, phase flux linkage, and electromagnetic torque can be computed [7]. MEC can achieve a close accuracy to FEA and can consider saturation regions [6]. As the rotor position changes, so does the flux distribution. This can be solved by stating assumptions for flux paths gained from experience in FEA simulations. However, the drawback of MEC is that the flux paths need to be defined in advance which can be complicated as the motor geometry becomes more complex [7].

Lookup tables are a simple technique that can be implemented with good accuracy. FEA is used to retrieve the flux linkage, induced voltage, and torque characteristics for various rotor positions and phase input currents and are stored in 2D lookup table. Interpolation is then applied on these data points to obtain an expression for the torque or phase inductance as a function of the rotor position and/or phase current [6] [7]. Tools that can be used to implement this method are MATLAB [8], OCTAVE [9], and Compose by Altair [7].

Interpolation modeling technique uses piece-wise functions to model the non-linear relationship between flux linkage and inductance to the rotor position and phase current. Some of the interpolation functions are 2D bivariate polynomials, 2D bicubic spline polynomials, Gaussian functions, and exponential functions [6] [7].

Fourier series modeling technique uses Fourier decomposition to model the phase flux linkage and phase inductance as a function of the phase current and rotor position. For instance, the phase inductance can be estimated using Fourier series as follows,

$$L_{ph}(i,\theta) = \sum_{n=0}^{k_t} L_{ph,n}(i)\cos\big(n(N_r\,\theta + \delta_n)\big), \tag{2.16}$$

where N_r is the number of rotor poles. $L_{ph,n}(i)$ and δ_n are the n^{th} Fourier coefficients. The more coefficients that are required to be calculated, the more accurate is the model [7]. Coefficients can be determined using FEA, experimental and analytical techniques. Typically, the number of terms are limited from two to four [7].

ML/DL algorithms involve training an algorithm or a model that can predict flux linkage and torque characteristics based on motor geometry parameters. The learning algorithm uses a limited training dataset from numerical techniques or experiments and can make reasonably accurate predictions for input on data it has not seen before. The advantage of ML/DL algorithms is that it uses low computational power compared to numerical methods [6]. ML/DL algorithms have gained popularity in the last five years and is an active research subject. In chapters 3 and 4, the ML/DL algorithms used to optimize performance in electric motors will be discussed.

Chapter 3

Application of ML in the Design Optimization of SRMs

3.1 Taxonomy of ML Algorithms

ML algorithms can be split into four main categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [10]. Figure 3.1 shows the various kinds of ML algorithms. Supervised learning is the most widely used ML algorithm type [6]. It is a problem where given a dataset, the ML algorithm tries to find a mapping between the input and output. For example, consider a dataset of cats and dogs images. The expected outputs (or labels) of the ML algorithm are either 0 to indicate a cat image and 1 to indicate a dog image. The input of the model is an image. The supervised learning problem is to find a mapping between the input image and the output values. There are two main types of supervised learning problems: classification and regression problems. Classification supervised learning algorithms deal with predicting discrete outputs. The previous example is a classification problem because the image can be classified as either a cat or a dog. Another example is recognizing handwritten digits from an input image. Regression problems on the other hand deal with continuous outputs. Predicting the house

price from the size of the house is an example of a regression problem. Another example is predicting the height of a person from an image.

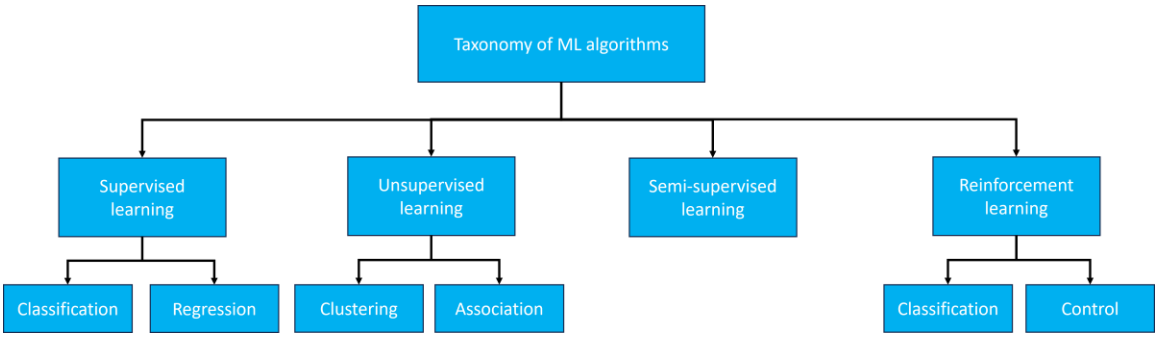

Figure 3.1: Taxonomy of ML algorithms.

Unsupervised learning is a problem type where the dataset given has no labels and the ML algorithm tries to find features or patterns without supervision and any prior knowledge [6]. There are two kinds of unsupervised learning problems: clustering and association. Clustering problems deal with categorizing objects into groups or clusters [10]. For example, given a dataset of fruit images, the ML algorithm clusters these images into apples, oranges, bananas, and so on. Association finds the relationship between the datasets. For instance, someone who buys a PlayStation console tends to buy PlayStation games [10].

Semi-supervised learning involves datasets that are partially labelled. Similar to supervised and unsupervised learning, the ML algorithm can be used to solve either problems of classification or clustering. An example of an application of semi-supervised learning is text document classifier [10].

Reinforcement learning is a problem where the ML algorithm (called the agent) learns to behave in an environment and depending on the agent's actions, it can be rewarded or penalized. The agent uses a trial and run method to come up with an outcome in the environment [10]. It is analogous to training a pet dog where it is rewarded with a treat when it performs the necessary action by the dog owner and it penalized (not getting a treat) when it does not. Reinforcement learning can be used as classification algorithms or for control applications such as self-driving cars or robotics [10].

3.2 ML Algorithms used for Design Optimization of SRMs

Reference [6] reviewed the applications of ML algorithms for modelling and design optimization of SRMs. There were two ML algorithms used in the literature for geometry optimization of SRMs: back-propagating neural network (BPNN) and generalized regression neural network (GRNN).

3.2.1 Back-propagating Neural Networks

BPNN is an ML algorithm made up of neurons connected to each other. A neuron is a computational node with a number associated with it. Each two neurons are connected and that connection has an associated weight to it. A group of neurons can be grouped vertically to layers. A BPNN usually consists of three layers: an input layer, a hidden layer, and an

output layer. The input layer has the number of neurons the same as the number of inputs to the network. The hidden layer number of neurons is determined heuristically. The number of neurons in the output layer is based on how many outputs the model is supposed to show. Figure 3.2 shows an example of a BPNN.

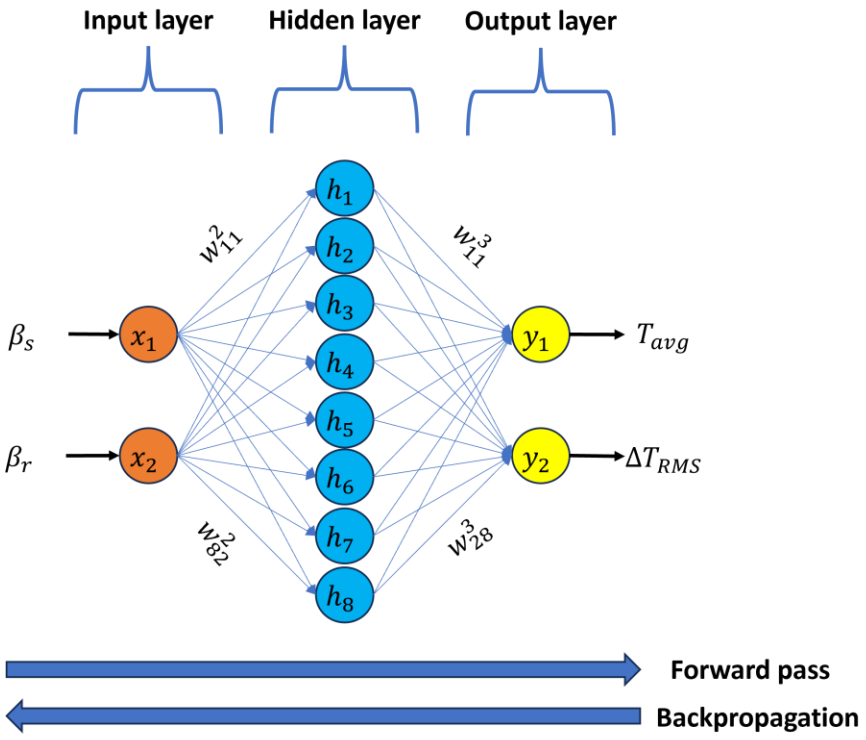

Figure 3.2: Structure of a BPNN.

The input layer can be called layer 1, hidden layer is layer 2, and output layer is layer 3. w_{jk}^{l} represents the weight going into layer l coming from neuron k in layer $l-1$ to neuron j in layer l. For example, w_{82}^{2} is the weight going into layer 2 coming from neuron 2 of layer 1 into neuron 8 of layer 2. The training for this network involves two processes: forward pass and backpropagation. In forward pass, the input vector $X = [x_1\ x_2]^T$ which

corresponds to the features from the data, gets fed into the input layer and forwarded to the output layer to produce an output $Y = [y_1\ y_2]^T$. To get this output Y, a weighted sum of the features and a bias for each layer is computed. Then an element-wise activation function $g(a)$ is used for each layer [6]. Mathematically, this can be represented as follows.

$$X \to \underbrace{g_2(W_2^T X + b_2)}_{a_2} \to \underbrace{g_3(W_3^T a_2 + b_3)}_{Y}, \tag{3.1}$$

Where, $\to$ is a vector function that takes in a vector and outputs another vector, $W_l^T \in \mathbb{R}^{n \times m}$ is the weight matrix of layer l with n neurons in layer l and m neurons in layer $l-1$, $b_l \in \mathbb{R}^n$ is the bias term in layer l, and g_l is the non-linear activation function for layer l. $a_2 \in \mathbb{R}^8$ is the resultant vector/activation after the second layer's computation. There are many activation functions that can be used. Some of the popular ones are RELU (Rectified Linear Unit), Sigmoid, Tanh, and Tansig [6]. In the process of backpropagation, a loss function is being minimized by differentiating it relative to the weights and biases of the network. The weights and biases are then updated based on the resulting derivative [6]. An optimizer is used to find the optimal weights and biases that would minimize the loss function. Normally gradient descent algorithm (GDA) is the optimizer used [6]. This process is repeated across the whole training time of the model.

Reference [3] investigated the static and dynamic characteristics of a 6/14 SRM by developing a BPNN. The model's architecture is the same as in Figure 3.2, which takes the stator and rotor pole arc angles, β_s and β_r respectively and predicts the average torque and torque ripple. Increasing β_s can reduce the torque ripple since the flux linkage increases however an excessive increase can reduce the output torque. Moreover, increasing β_r can

increase the average torque as it increases the flux linkage however an excessive increase in β_r can reduce the output torque due to a reduction in motor saliency [3]. FEA and MATLAB were used to capture the dynamic characteristics and the dataset of the motor. The optimizer used to train the BPNN was Levenberg-Marquardt as it has a faster convergence time than GDA to find the local minimum of the loss function [3]. The loss function used was mean squared error (MSE). A sample of 10,000 design candidates were generated and the optimal motor design was the one that had the lowest torque ripple. Compared to the baseline model, the average torque increased by 2% and torque ripple decreased by 24%.

3.2.2 Generalized Regression Neural Network

GRNN is another type of ML algorithm that has a much simpler training procedure than BPNN because it relies on only one parameter, the spread factor σ [11] . Unlike BPNN, it is not an iterative algorithm, which means it is a forward pass algorithm only. This makes the training time shorter than BPNN [6]. Figure 3.3 shows an example of a GRNN. It consists of four layers: an input layer with a number of neurons equal to the number of features in the training data. The second layer is called a pattern layer (or radial basis layer) and the number of neurons is the same as the number of training examples. A Euclidean distance function is applied between the input and the training data input [11]. The pattern layer applies a gaussian kernal for each row in the training dataset according to the equation:

$$f = e^{\frac{-D_i{}^2}{2\sigma^2}}, \tag{3.2}$$

where D_i is the Euclidean distance between the input to the model and input in row i of the training dataset. The third layer is a summation layer that has two units, a numerator and denominator. The numerator unit, N^s, contains the same number of neurons as the number of GRNN outputs and the denominator unit, D^s, has one neuron [11]. The equations governing the summation layer are:

$$N^s = \sum_{i=1}^{n} Y_i e^{\frac{-D_i{}^2}{2\sigma^2}} \tag{3.3}$$

$$D^s = \sum_{i=1}^{n} e^{\frac{-D_i{}^2}{2\sigma^2}}. \tag{3.4}$$

Y_i in equation (3.3) is the response in row i of the training dataset. The output layer is the division of the numerator unit by the denominator unit [11].

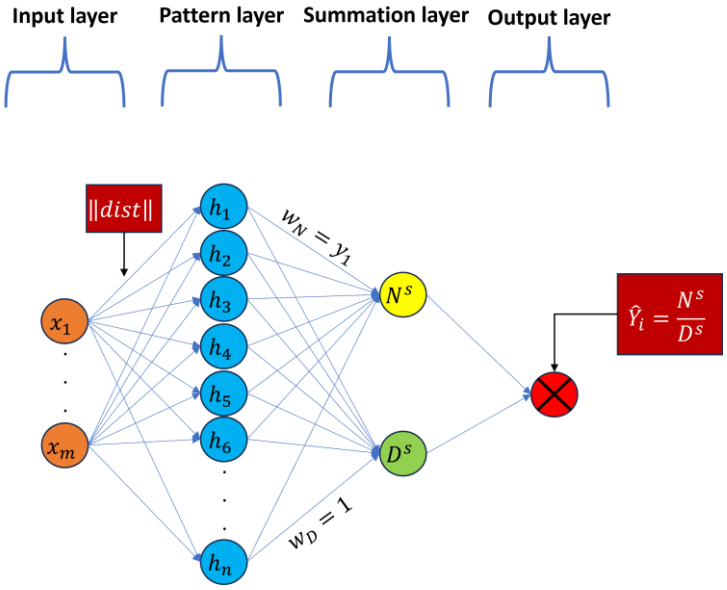

Figure 3.3: Structure of a GRNN. ||dist|| is a distance function and $\hat{Y}_i$ is the predicted output from the model.

Reference [12] looked at a 12/8 SRM and implemented GRNN. The model considered the stator and rotor angle parameters and predicted the average torque and torque ripples at low-speed operation. The torque ripples were reduced by 12% compared to the baseline design given in [12]. GRNN was also used in [13] for a 12/8 SRM. A Fruit fly optimization algorithm (FOA) was used to improve the spread parameter. The model takes the stator pole arc angle, rotor pole arc angle, and the rotor yoke height and predicts the torque ripple and efficiency. The mean relative error (MRE) of the prediction of average torque was found to be 0.74% and 0.21% for the efficiency. Both papers used FEA to gather the dynamic characteristics of the motors. While BPNN and GRNN use lower dimensional inputs, there exist other model architectures that can process higher-dimensional input features, such as images. This will be discussed in chapter 4.

Chapter 4

Deep Learning: Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs)

4.1 Convolutional Neural Networks

CNNs are a subclass of DL algorithms as seen in Figure 4.1. They are one of the popular DL algorithms for image and audio processing applications [14]. The focus of this thesis will be training CNNs on images. Unlike the networks discussed in the previous section which are made of dense layers, CNNs consist of one or more convolutional layers, which have the ability to interpret the spatial structure of the images [15]. In this section, a general description of CNNs will be provided.

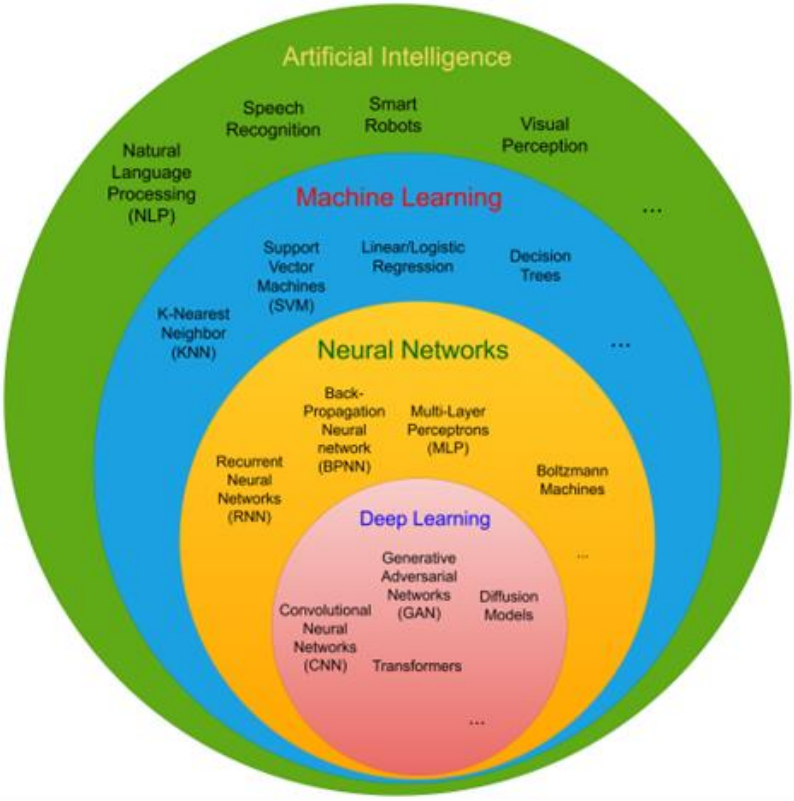

Figure 4.1: Classification of AI.

4.1.1 CNN Layers

CNN models consist of usually five types of layers: convolutional layer, activation layer, pooling layer, dense layer, and loss layer [16]. Figure 4.2 shows an example of a CNN model. The convolutional layer is the most important layer in any CNN model. It uses filters or also known as kernels to be convolved with the input image to produce feature maps. Input images consist of three dimensions: width, height, and number of channels, or $m \times m \times c$. Common practice in training CNN models is to have the width and height of the images to be the same [16]. c is three if the image is colored (or in RGB format) and

one if it is grayscale. Images are made of pixel values which range from 0 to 255. The kernel can be thought of as a matrix with dimensions $n \times n \times q$, where $n \ll m$ and $q \leq c$. In a convolutional layer, there can be more than one kernel and thus more than one feature can be extracted from an image [16]. The values of the kernel act as the weights of the model and are initially assigned to random numbers.

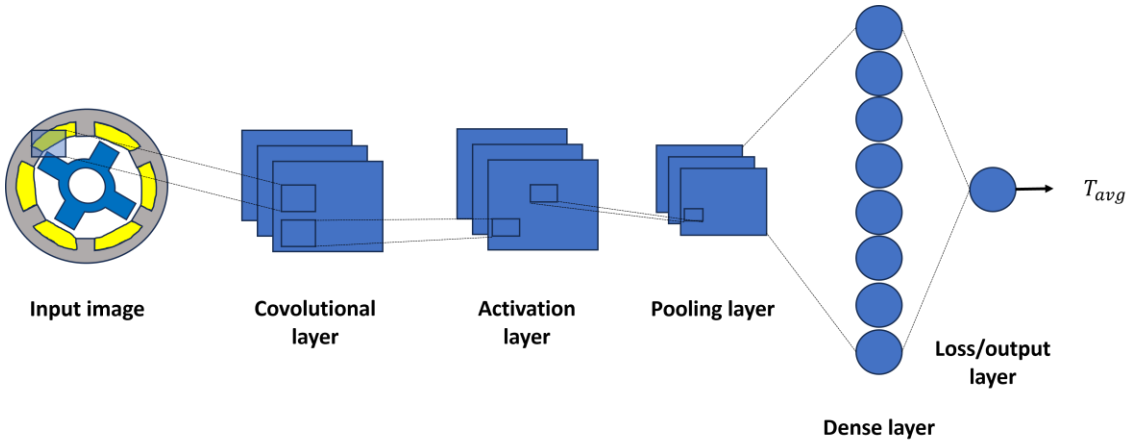

Figure 4.2: Example of a CNN architecture for regression.

There are techniques to initially assign values to those weights, such as Principal Component Analysis or random features, and are then adjusted by the model through backpropagation [17]. The convolution process involves performing a dot product operation between the values of the input image and the weights of the input kernel. Figure 4.3 shows an example of how the convolution process works. The convolution process is performed until the end of the image. The user can change the number of kernels for each layer, the size of each kernel, amount of padding for the input image, and the stride. Padding increases the width and height of the input image [17]. In Figure 4.3 no padding was applied however there are different kinds of padding such as zero padding or mirror

padding [17]. Stride is the step size for the kernel and in Figure 4.3, a stride of one is used. The higher the stride, the lower the dimensions of the feature map [16]. Convolutional layers are beneficial due to their sparse connectivity, which reduces the number of trainable weights compared to dense layers and reduces memory [16]. Moreover, the weights of the kernels are shared, causing the training model time to decrease [16][17].

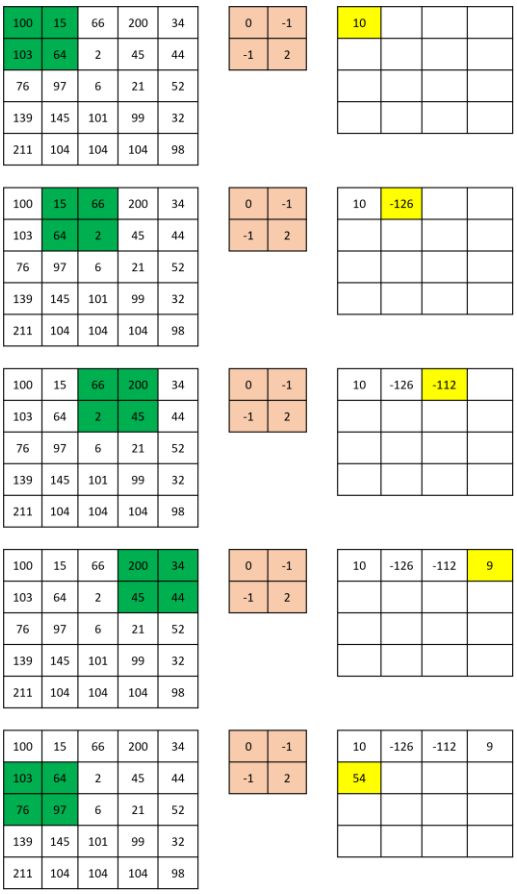

Figure 4.3: Calculation performed for each step of the convolutional layer. Left table is the input image, middle table is the kernel matrix, and right table is the resulting feature map.

The activation layer detects if the values of the feature map pass a threshold. There are many kinds of activation functions. One of them is the sigmoid function that takes a real number and outputs a value between zero and one according to this equation:

$$f_{sigmoid}(x) = \frac{1}{1 - e^{x}}. \tag{4.1}$$

Usually, sigmoid is used for classification tasks and applied in the last layer where 0 can indicate a cat image and 1 is a dog image for example. Another common activation function is tanh. This is similar to the sigmoid as it takes a real number, but it outputs a value between -1 and 1. Tanh is used in generative adversarial networks (GANs), which will be discussed in the next section. The equation for the tanh activation function is given as:

$$f_{tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4.2}$$

The most commonly used activation function in CNN models is the rectified linear unit or ReLU [16]. This is because it has a low computational load. It is a function that takes a real number and if the input is positive then it is left as is and if it is negative it is converted to 0. The equation for the ReLU activation function is given as:

$$f_{ReLU}(x) = \max(0, x). \tag{4.3}$$

The challenge with using ReLU however is that it can suffer from dying ReLU, where the gradient is 0 during backpropagation in the initial training phase [15]. To solve this problem, one alternative is to use LeakyReLU, which does not ignore negative inputs [16]. The mathematical representation of LeakyReLU is given by:

$$f_{LeakyReLU}(x) = \begin{cases} x, & x > 0 \\ mx, & x \leq 0' \end{cases} \tag{4.4}$$

where m is the leak factor. It is commonly set to a small value like 0.001 [16].

The pooling layer decreases the size of the feature maps. This reduces computation time for the next layer [17]. It also makes the features invariant to local translations. For example, applying a pooling layer for a handwritten digit classifier allows the detection of centered digits in the image as well as left aligned digits in the image. There are many kinds of pooling layers, but the most frequently used ones are max-pooling, min-pooling, and global average pooling (GAP) [16]. Similar to the convolutional layer, the user can adjust the width of the kernel, stride, and padding. Figure 4.4 shows an example of those three pooling operations.

The dense layer (or also known as the fully connected layer) is a restructuring of the previous layer in the network [15]. For instance, if the previous layer has an output shape of $2 \times 2 \times 8$, then the dense layer will have the product of the dimensions, 32 neurons. Sometimes another dense layer can be added to add more depth to the network.

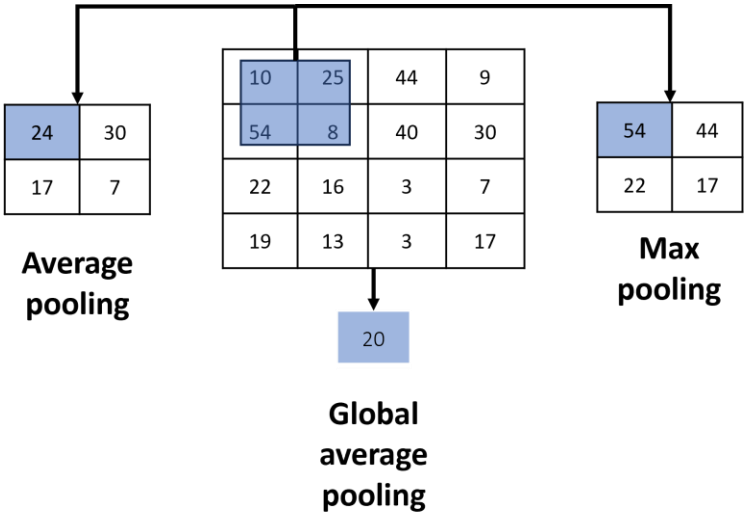

Figure 4.4: Examples of pooling operations.

Loss layer is the last year in a CNN model. It is a dense layer that uses a loss function depending on whether the model solves a classification or regression problem. For classification problems, the cross-entropy loss function is usually used with a SoftMax activation function, and it is represented by the following equation:

$$H(p, y) = -\sum_{i=1}^{N} y_i \log p_i \tag{4.5}$$

$$p_i = \frac{e^{a_i}}{\sum_{j=1}^{N} e^{a_j}}, \tag{4.6}$$

where N is the number of neurons in the output layer, y_i is the desired output, and p_i is called the log loss function, which is a probability ranging from 0 to 1 [16]. e^{a_i} is the non-normalized output in layer a from neuron i. Mean Squared Error (MSE) is used for regression tasks, and it is the square difference between the predicted output value, $\hat{y}_i$, and the desired output.

$$MSE = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2 \tag{4.7}$$

The loss function is optimized throughout the training time.

4.1.2 Regularization Techniques

It is desirable to have the CNN model be well-fitted to the training data and test data. If the model does very well at predicting outputs for the training data but poorly on unseen data (called test data), it is called over-fitting. If the model performs poorly on the training data, it is called under-fitting. Figure 4.5 highlights the differences between under-fitting, over-fitting, and a good fit. There are several techniques that can be employed to reduce over-fitting and improve the performance of CNN models. The most common ones are dropout, data augmentation, batch normalization, and early stopping [15] [16] [18]. Dropout is a technique that chooses a set of neurons from the preceding layer and sets them equal to 0. This improves the generality of the model and prevents it from depending on all its weights. This is achieved by adding a dropout layer and specifying a value from 0 to 1 indicating the percentage of neurons in the previous layer to be dropped.

One of the reasons the model could be over-fitting is due to a complex model and few data is present. To solve this problem, data augmentation can be applied to increase the diversity of the dataset. Transformations can be applied to the dataset, such as rotating the image, scaling, changing brightness of the image. These transformations can then be added to the original dataset.

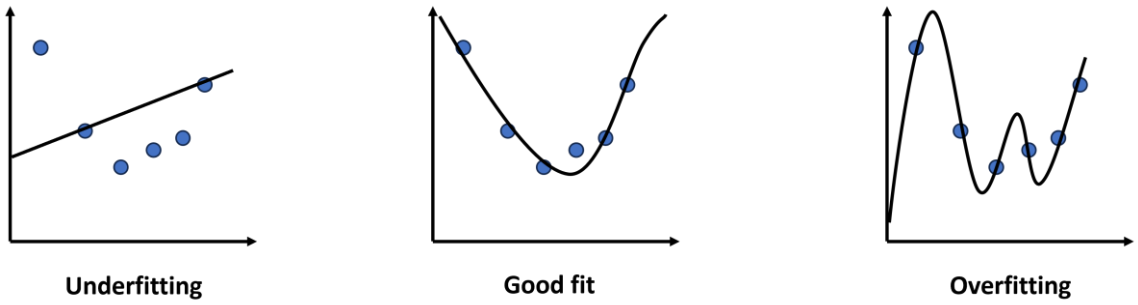

Figure 4.5: Difference between over-fitting and under-fitting.

When training a CNN model or a DL model, it is important that the weights of the model are within a reasonable range. If they go beyond that range, then the loss function becomes very large. This is known as the exploding gradient problem [15]. This problem could happen anytime during training. Usually when training a model, the input data is normalized between 0 and 1 or -1 to 1 instead of 0 to 255 to avoid the gradient from becoming very large. While this can help maintain the weights at a reasonable range, as the training time continues sometimes the weights can be very far off from their random initial values. This can cause the loss function to also increase in value, known as covariate shift [15] [16]. To solve this problem, batch normalization can be applied to make sure these weights do not go beyond that range. The mean of the preceding layer is subtracted from the outputs of the previous layer and divided by the standard deviation of that previous layer.

Early stopping is another technique used to avoid over-fitting. This is applied during training to monitor the loss function. An example of its usage is to stop training the model when the validation error starts to increase or if it goes below a threshold.

4.1.3 Types of Optimizers

Backpropagation is the most popular method used train CNN models. The weights are updated from the output layer to the first layer of the model using GDA [16] [17] using the update formula:

$$w_{il}^{k+1} = w_{il}^{k} - \eta^{k} \frac{\partial \mathcal{L}}{\partial w_i}, \tag{4.8}$$

where w_{il}^{k} are the weights between layers l and i at epoch k. An Epoch is an iteration throughout the whole training dataset. η is the learning rate and $\partial \mathcal{L}/\partial w_i$ is the partial derivative of the error/loss function relative to the weights of the model. η can be understood as the step taken to update the weights of the model and it is a choice by the user. It can be constant throughout all epochs, or it can vary [18]. If η is too large, then the model will take a large step and thus can miss finding the optimal weights and if it is too low then the model will take too long to find a solution. The problem with using GDA or batch gradient descent in equation (4.8) is that is computationally expensive because the whole training dataset is used. Nowadays, backpropagation is used with two main kinds of optimizers: stochastic optimizers and adaptive optimizers [18].

Stochastic optimizers update the weights for each training sample [16]. Stochastic gradient descent (SGD) updates the weights at every training sample. This makes SGD more memory efficient especially for large training datasets. The disadvantage is that because the weights are updated frequently based on every training sample, this adds noise and causes unstable convergence [16]. The equation for SGD is similar to equation (4.8):

$$w^{k+1} = w^k - \eta^k \nabla \mathcal{L}_i(w^k), \tag{4.9}$$

where $\nabla \mathcal{L}_i(w^k)$ is the gradient of the loss function with respect of the weights at epoch k evaluated at training sample i. Mini-batch gradient descent (MBGD) [19] is an improvement to SGD by splitting the data into batches with no overlap and the weights are updated for every batch. This makes MBGD more computationally efficient and have a stable convergence.

Adaptive optimizers keep the history of previous sampled points. The two most common adaptive optimizers are root mean square propagation (RMSProp) [20] and adaptive momentum estimation (Adam) [21]. RMSProp and Adam use the concept of momentum [18]. Momentum is a parameter set by the user ranging from 0 to 1 and it controls how much of the history of previous gradients to keep. The higher the momentum parameter α, the higher the convergence rate but the higher the chance the model could miss the global minimum solution [16]. Adam is the most popular optimizer used to train neural networks and it uses the following equation [18]:

$$w^{k+1} = w^k - \frac{\eta^k}{\sqrt{\epsilon + \widehat{v^{k+1}}}} \widehat{\alpha^{k+1}}, \tag{4.10}$$

where ϵ is a scalar used for stability and $\widehat{v^{k+1}}$ is a scalar variable related to the gradient of the loss function similar to equation (4.9).

4.1.4 Examples of CNN Architectures

There are many CNN architectures that have been developed. Some of the well-known architectures and that have been used in the literature to optimize SRM performance are visual geometry group (VGG) and residual network (ResNet). VGG was proposed by Simonyan et al. [22]. It is a very deep neural network that can handle non-linear patterns in an image [17]. VGG uses small convolutional filters with kernel size 3×3 to avoid overfitting. All layers use the RELU activation function with the exception of the layer using softmax for classification. VGG's main disadvantage is its computation time because there are many parameters to be updated [16]. There are two main types of VGG architectures: VGG-16 and VGG-19. VGG-16 uses 13 convolutional layers and 3 dense layers whereas VGG-19 has 16 convolutional layers and 3 dense layers.

ResNet is another architecture first proposed in [23]. It solves the problem of degradation, which is when the network has so many convolutional layers that the accuracy saturates or gets worse [17]. This is done by bypassing the original dense layers and adding a residual mapping instead. This allows better optimization due to the residual mapping. There are many variations to ResNet such as ResNet-18, Resnet-34, ResNet-50, and ResNet-101 depending on the processor available and application [17]. The most common one is ResNet-50 [16]. Figure 4.6 and Figure 4.7 show example architectures of ResNet and VGG respectively.

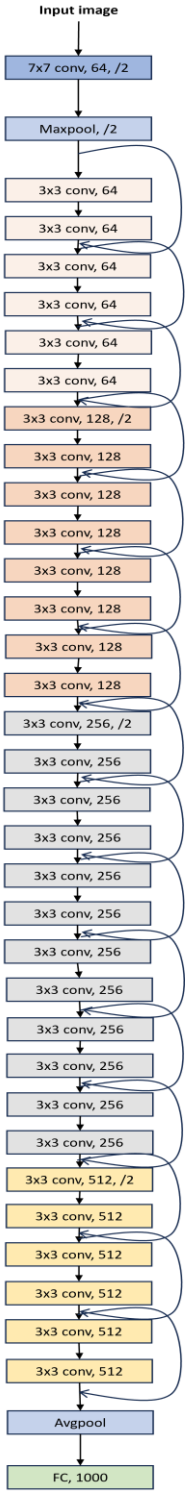

Figure 4.6: The architecture of the ResNet-34.

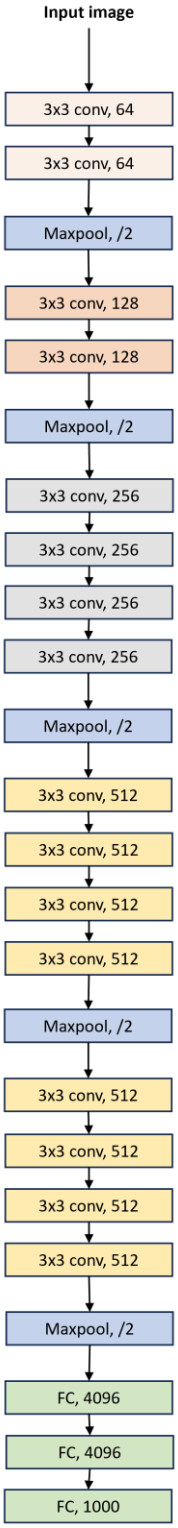

Figure 4.7: The architecture of the VGG-19.

4.1.5 CNN Applications for Design Optimization of Electric Motors

Reference [24] used VGG16 CNN model to predict the average torque and torque ripple of two kinds of Inner Permanent Magnet (IPM) motors where the input images were material configuration and magnetic flux density. The CNN model was then used along with GA to find the maximum torque and minimum torque ripple. Mean absolute error (MAE) was used as the metric. VGG16 was also used in [25] and trained on two types of 2D IPM motors images to predict average torque and torque ripple. Correlation coefifcient was the metric used to measure the performance of the CNN model. In [26], VGG16 model was used to predict d-axis and q-axis inductance and d-axis permanent magnet flux of an IPM motor. Correlation coefficient was the metric used to measure the performance.

Shimizu et al [27] used ResNet-18 to predict torque-speed characteristics of a PMSM with three rotor shapes. CNN model was used to predict d- and q-axis inductance and flux linkage. Mean squared error (MSE) was used to measure the performance of the CNN.

Reference [14] used two CNN models to optimize the rotor design of a IPM motor that provides the maximum torque and back-EMF. Barmada et al. [28] used CNN to predict average torque of a synchronous reluctance motor. Mean absolute percentage error was used as the metric.

4.2 Generative Adversarial Networks (GANs)

In ML, algorithms can be split into two broad categories: discriminative and generative algorithms [15]. Discriminative algorithms aim to predict labels from a given observation. Example is a classification algorithm that takes an image and predicts whether an image is a car (labelled as 0) or a horse (labelled as 1). The model would learn the spatial structure of the image and determine a mapping from the image to the label. Generative algorithms aim to produce new data that is similar to a given dataset [15]. Suppose that the dataset of observations is represented by X, which have been generated according to some distribution p_{data}. A generative algorithm is trained such that the distribution of the model with respect to its parameters $p_{model}(\gamma)$ is almost the same as p_{data}.

There are two main approaches to estimate such distribution: implicit and explicit density modelling (see Figure 4.8). Explicit density modeling aims to model $p_{model}(\gamma)$ by using X to train the parameters γ [29]. The disadvantage of using such technique is that p_{model} needs to be constrained in order to be calculated and may not be able to fully map the complex distribution of p_{data} [15][29]. Implicit density modeling does not solve p_{model} but rather stochastically generates data and uses that generated data to train the model [29]. Generative adversarial network (GAN) is an example of an implicit density model, which will be discussed in this section.

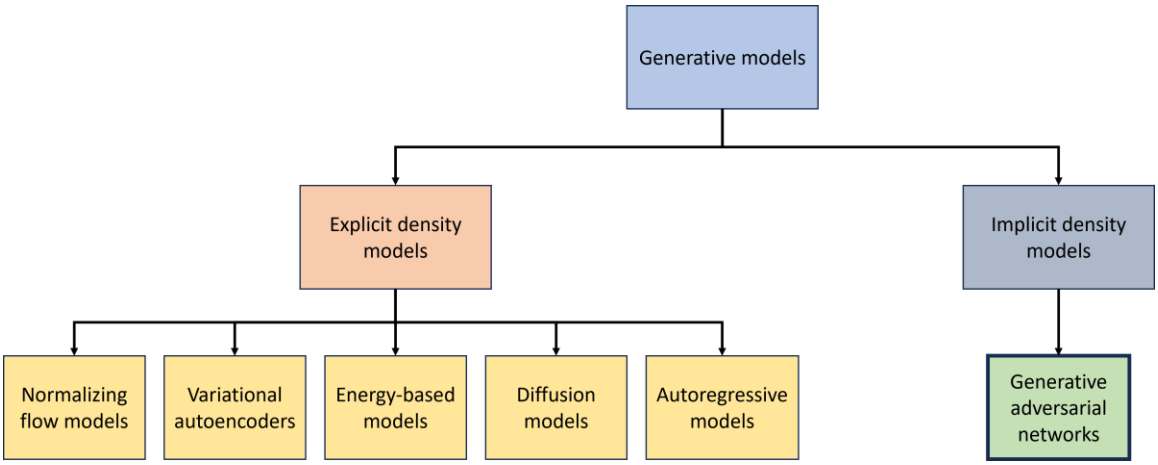

Figure 4.8: Types of generative models.

4.2.1 Objective Function

GANs consist of two DL models: a generator and a discriminator. A generator synthesizes realistic cross-section images based on input noise and/or conditional input parameters [30]. The discriminator is nothing but a classifier, which evaluates whether the images generated by the generator are real or fake compared to the original dataset. Figure 4.9 shows the GAN architecture. The generator and discriminator models play an adversarial game, where the generator aims to generate data that is identical to the real dataset and fool the discriminator whereas the discriminator tries to differentiate between the real and fake data.

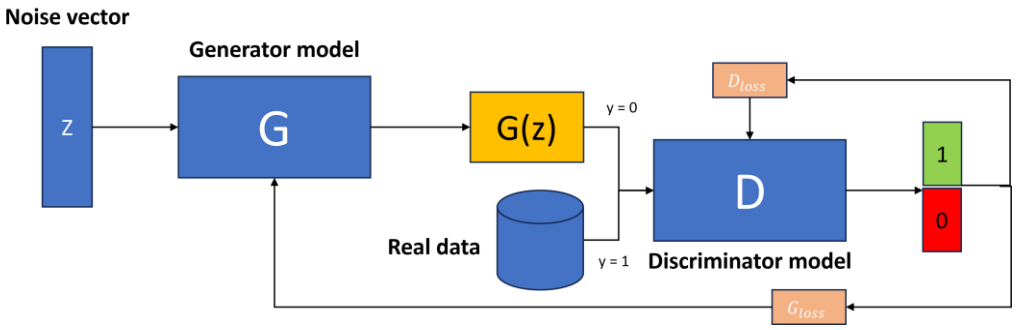

Figure 4.9: The GAN Architecture.

In mathematical notation, the generator G performs the following,

$$G(z) = x, \tag{4.11}$$

where $z \in \mathbb{R}^d$ is a d-dimensional noise vector or latent vector and $x \in \mathbb{R}^{m \times m \times c}$ is the generated image for the purpose of this thesis and $d \leq m$. The discriminator D is a binary classifier than performs the following task.

$$D(x) = \begin{cases} 1, & \text{if } x \text{ is real} \\ 0, & \text{if } x \text{ is generated/fake} \end{cases}. \tag{4.12}$$

The adversarial game can be represented mathematically through the objective function as follows [29]:

$$V(D,G) = \min_G \max_D E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p(z)}\big[\log 1 - D\big(G(z)\big)\big], \tag{4.13}$$

where $E_{x \sim p_{data}(x)}$ in the first term is the expectation over the real dataset. To understand equation (4.13), the discriminator wants to maximize the first term because it assigns a value of 1 to the real data, which is in equation (4.12). In the second term, $E_{z \sim p(z)}$ is the expectation over the noise. The discriminator wants to assign a low value (minimize) to the generated/fake data, $D\big(G(z)\big)$, which is equivalent to maximizing $1 - D\big(G(z)\big)$. On

the other hand, the first term has no generator term, thus it can be omitted. Since the generator wants to fool the discriminator that the fake data is real, it is then trying to minimize $1 - D\big(G(z)\big)$. The log function is the cross entropy and another name for equation (4.13) is the minmax game [29].

4.2.2 Training GANs and its Challenges

In terms of model architectures, the discriminator consists of convolutional layers with activation function LeakyReLU. Dropout and batch normalization layers are usually added to avoid overfitting [15]. The last layer is a convolutional layer with one filter and the sigmoid is the activation function as a value between 0 and 1 needs to be outputted. For the generator, the input is a dense layer with d neurons, symbolizing the dimension of the noise vector. The noise vector can be based on the gaussian distribution or normal distribution [15] [31]. This vector then gets reshaped from one dimensional to three dimensional because the output of the generator is supposed to be an image. Transposed convolutional layers or up sampling layers are used to perform the task of increasing the dimensions [15]. Dropout layers and batch normalization are used to regularize training of the generator. In the last layer, the tanh activation function is usually used because normalizing images between -1 and 1 improves the performance of the generator compared to normalizing images between 0 and 1 [32]. Both the generator and discriminator use the binary cross-entropy as the loss function.

One of the disadvantages of GANs is that it is challenging to train as two models are being trained instead of one [29]. The key is to alternate the training of the two models. The discriminator is trained by creating a dataset with half of it being real data and half being fake data. The discriminator is treated as a supervised classification problem, where the label for the real data is assigned a 1 and the fake data is assigned a 0 [15]. During the training of the generator, the weights of the discriminator are kept constant, and the generator produces the images and are sent to the discriminator and the labels of those images are 1. This is because the generator's aim is to fool the discriminator into believing the images it produced are real. The loss of the generator is the output of the discriminator and the vector of ones [15]. Figure 4.10 summarizes the training process of the generator and discriminator.

The two most common problems encountered when training GANs are vanishing gradients and mode collapse [15]. Vanishing gradients occur if the discriminator is much stronger than the generator, causing the discriminator loss to be very close to 0. Thus, the discriminator will need to be weaker, and this can be done by increasing the dropout percentage of the discriminator, reducing the learning rate, decreasing the number of convolutional layers, adding noise to the labels, or randomly alternating the real and fake labels to some of the images [15]. Mode collapse is the opposite of vanishing gradients, where the generator overpowers the discriminator. The generator can find a point mapped to the latent space where the discriminator is fooled, causing the generator loss to be close to 0. To solve this problem, the discriminator will need to be strengthen using the opposite techniques of the vanishing gradient [15].

One might think that if the generator loss is decreasing, this means that the quality of the generated images would be better. However in practice, as the generator loss increases, the quality of the generated images improves [15]. This is known as uninformative loss, making monitoring GAN losses challenging. Many researchers mitigated these challenges by developing variations of the GAN, which will be discussed in the next sub-section.

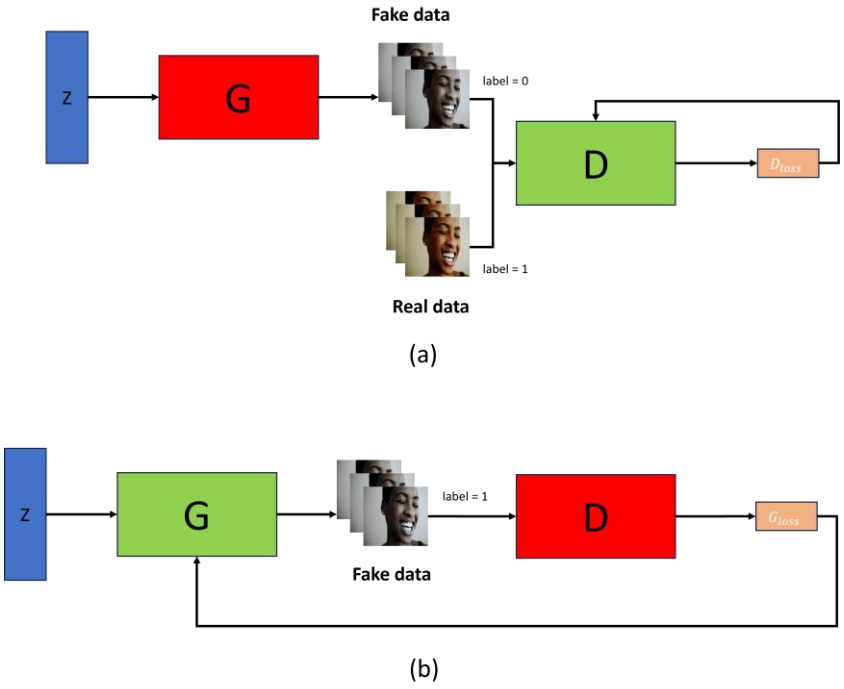

Figure 4.10: The training process of (a) discriminator and (b) generator models. Red boxes indicate the weights are frozen for the model.

4.2.3 GAN Variants

In this sub-section, five GAN variants will be discussed. It should be noted that there are more variations. The first GAN was developed in 2014 by Ian Goodfellow [33]. Dense layers were used for both the generator and discriminator. The optimal D^* for a fixed G is as follows [29] [33].

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}. \tag{4.14}$$

Since the generator is aiming to generate data that has the same distribution as the real data, $p_g(x) = p_{data}(x)$, this means the optimal discriminator should give 0.5. This paper acts as the backbone for GAN variants.

In the same year, Mirza et al [34] developed a GAN model that takes a conditional input. This model was called conditional GAN (cGAN). The architecture of the model is in Figure 4.11. The objective function for the cGAN is similar to equation (4.13) but there is an extra term c for the conditional input.

$$V(D,G) = \min_G \max_D E_{x\sim p_{data}(x)}[\log D(x|c)] + E_{z\sim p(z)}\big[\log 1 - D\big(G(z|c)\big)\big]. \tag{4.15}$$

The authors trained the cGAN model on the MNIST dataset. The cGAN model has the ability to generate a specific class from the MNIST dataset by embedding the class label as a one-hot encoded vector to the generator and discriminator models.

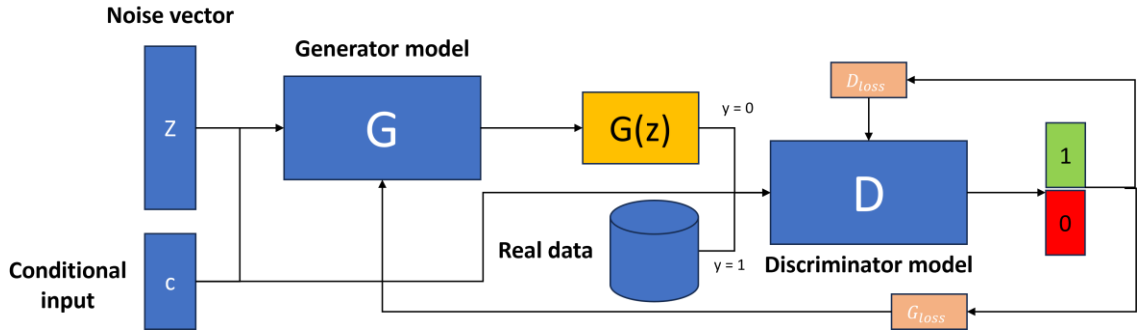

Figure 4.11: The architecture of the cGAN.

In 2015, a convolutional network version of the GAN was developed in [35] called Deep Convolutional GAN (DCGAN). DCGAN is superior to GAN because it reduces the problem of mode collapse by making the networks deeper [14]. The concept of using LeakyReLU and batch normalization layers was also introduced in this paper. This is the reason the GAN variations after 2015 are made of CNN layers.

In 2017, a variation of the cGAN was developed where c is an image known as image-to-image translation (or pix2pix) by Isola et al [36]. The objective function is slightly different than equation (4.13) as it does not use a noise vector.

$$L_{cGANs}(D,G) = E_{x,y}[\log D(x,c)] + E_y\left[\log\left(1 - D\left(c, G(c)\right)\right)\right], \qquad (4.16)$$

where x is the input image to the generator, c is the expected generated image, and $G(c)$ is the actual generated image from the generator. Figure 4.12 shows the architecture of the pix2pix model. An example of a use case for the pix2pix model is to convert an aerial image to a map image or a grayscale image to a colored image. The authors also added an L1 loss term (also known as MAE) to the objective function so that generator outputs an

image that is as close to the ground truth image as possible [36]. The final objective function is [29]:

$$L_{cGANs}(D,G) + \rho\mathcal{L}_{L_1}(G), \tag{4.17}$$

where ρ is a tunable parameter and it is set to 100 in the original paper. The generator model architecture used is the U-Net architecture and the discriminator uses the PatchGAN network.

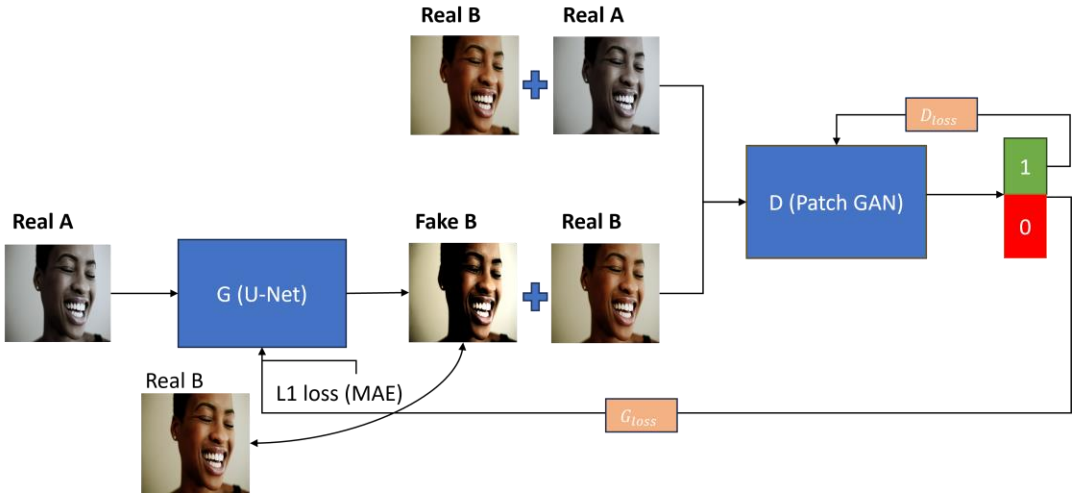

Figure 4.12: The architecture of the pix2pix model.

The challenge with using cGAN is that the conditional input is discrete. Examples are generating digit 3 from the MNIST dataset, a cat image, or an image of a truck. For cases when a continuous conditional input (such as an age of a person or angle of a chair in degrees) is available, a different network is required. Reference [37] tackled this issue by developing a continuous cGAN (cCGAN). The cCGAN architecture is the same as Figure 4.11 however the conditional input first goes through a dense layer network and is

converted to an embedding vector that is fed into the generator and discriminator models. The authors compared this method to the method in [34] and concluded that their method had better results.

A common problem with training those GAN variants is that a lot of training data is required and lots of computational power is used. For these reasons, Liu et al [38] developed a light-weight GAN (or FastGAN) model that produces high quality images with a low number of training images and low computational costs. The generator uses a technique called skip-layer excitation (SLE), which provides a more robust gradient flow and provides faster training. The discriminator on the other hand, uses the feature maps technique, which focuses on certain parts of an image and thus provides more signals to train the generator. The discriminator is also treated as an encoder with small decoders [38]. The generator and discriminator losses use hinge losses according to the given equations:

$$\mathcal{L}_D = -E_{x\sim p_{data}(x)}\left[\min\left(0,-1+D(x)\right)\right] - E_{\hat{x}\sim G(z)}\left[\min\left(0,-1-D(\hat{x})\right)\right] + \mathcal{L}_{recons} \quad (4.18)$$

$$\mathcal{L}_G = -E_{z\sim N}\left[D\left(G(z)\right)\right], \quad (4.19)$$

where $\mathcal{L}_{recons}$ is called the reconstruction loss of the discriminator and the noise vector is sampled form the normal distribution.

4.2.4 GAN Applications for Design Optimization of Electric Motors

Few papers used GAN and its variations in electric motor design optimization. The pix2pix model was used in [30] to generate magnetic field plot images from a geometry image of a V-shaped interior permanent magnet machine. Further optimization using the genetic algorithm to find the optimal design that gave the highest torque and lowest weight. The authors found 23% of the time was saved using pix2pix and FEM compared to only FEM.

Shimizu et al [27] used GAN as a data augmentation method. Genetic algorithm was used to find the design that provides minimum volume of the PMSM and highest torque. GAN was used in [39] to generate cross-section images for a PMSM with a double V rotor shape. Optimization was conducted to find the optimal design that generated the maximum axle torque and driving cycle efficiency. Reference [14] used a DCGAN to optimize the rotor design of a IPM motor that provides the maximum torque and back-EMF. As it can be seen from the CNN applications subsection and this subsection, there is no record of applying CNN and GAN models to optimize the performance of SRMs. In the next chapters, the implementation of the CNN and GAN models will be discussed on a 3-phase 6/14 SRM.

Chapter 5

Proposed Approach

5.1 Introduction

The motor model that is being investigated is the 3-phase 6/14 SRM at different stator and rotor pole angles. This motor can be found at MARC and was used in [40] for furnace blower applications rated at 1 HP and 1100 RPM speed. The parameters of the baseline SRM model are summarized in Table 5.1. The full cross-section model of the motor is shown in Figure 5.1. The FEM analysis was done using the JMAG software where the cross-section images of the SRM at different stator and rotor pole angles, β_s and β_r respectively, and the corresponding average torque and torque ripple were obtained through simulation. Figure 5.2 shows the dynamic baseline's phase current plot. The baseline's stator and rotor pole arc angles are constrained such that a fully unaligned

position exists and to guarantee self-starting capability [3]. The range of the β_s and β_r values for this motor is $[8.57°: 12.85°]$.

Table 5.1: Parameters of the 6/14 SRM under investigation.

Parameter	**Symbol**	**Value**
DC-link Voltage [V]	V_{DC}	163
Stack length [mm]	L	74
Stator outer diameter [mm]	D_s	139.21
Shaft diameter [mm]	D_{sh}	12.7
Air gap length [mm]	l_c	0.4
Stator back iron thickness [mm]	y_s	10
Rotor back iron thickness [mm]	y_r	35.78
Stator pole height [mm]	h_s	10
Rotor pole height [mm]	h_r	7.08
Stator pole arc angle [°]	β_s	9.5
Rotor pole arc angle [°]	β_r	9.3
Stator taper angle [°]	τ_s	4
Rotor taper angle [°]	τ_r	4

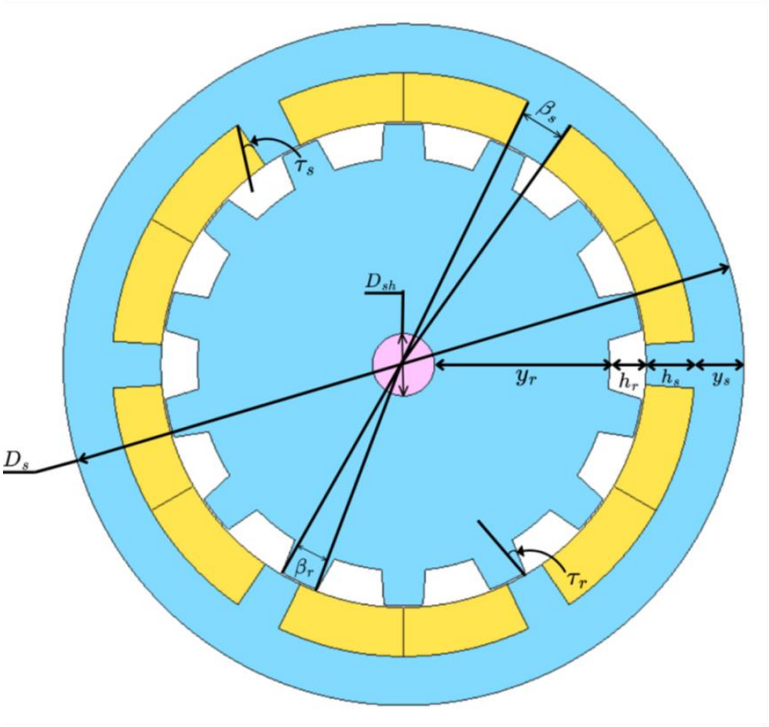

Figure 5.1: The full model of the 6/14 SRM under investigation.

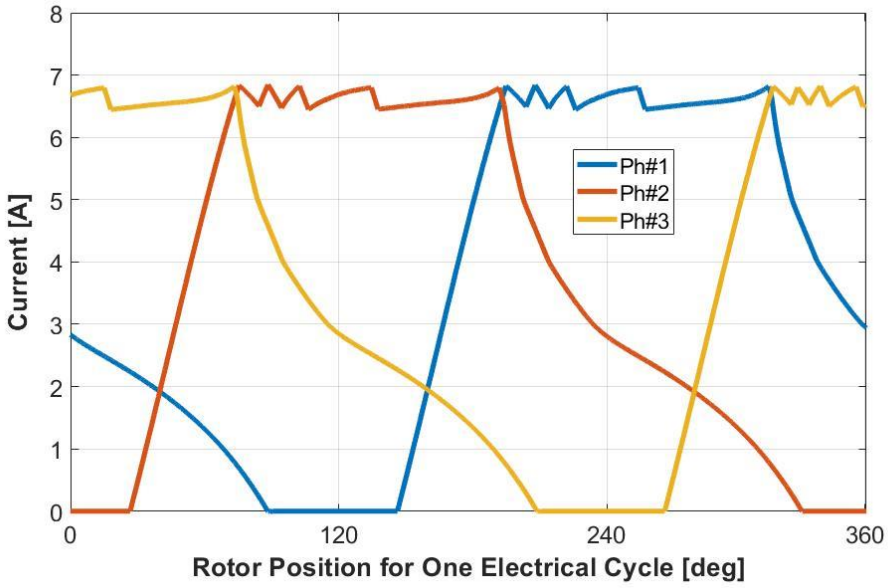

Figure 5.2: The dynamic phase current plot at various rotor positions.

A total of 60 data points were obtained. To get a better understanding of the dataset, plots of the average torque and torque ripple at various β_s and β_r values are presented in Figure 5.3.

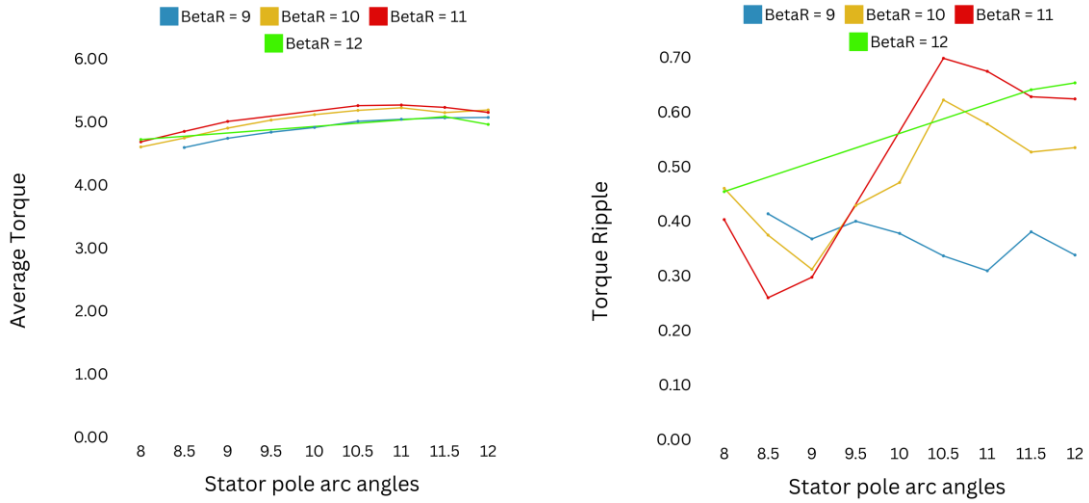

Figure 5.3: The plot of (a) average torque and (b) torque ripple at various stator pole arc angles with constant rotor pole arc angles.

It can be observed that the average torque increases as the stator pole arc angle increases. However, the torque ripple exhibits a nonlinear behavior as the stator and rotor pole arc angles increase. This shows how sensitive these geometric parameters can be in relation to the average torque and torque ripple. The lowest torque ripples occur when β_r is between 10° and 11° and when β_s is around 9°. This can further be seen by observing the average torque ripple plot as a function of the rotor pole arc angles at constant β_s values in Figure 5.4.

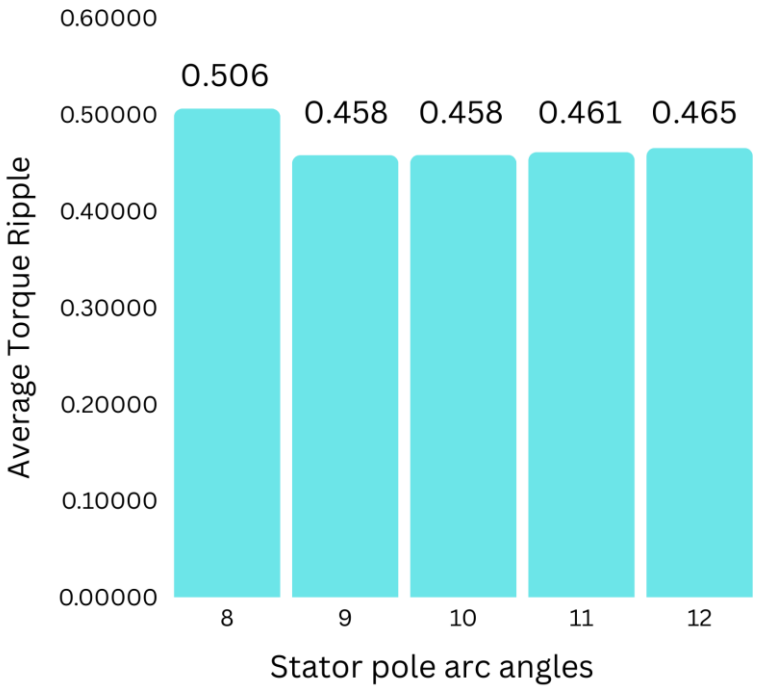

Figure 5.4: Average torque ripple for various rotor angles at constant stator angles.

In this chapter, a detailed explanation of the data preprocessing technique used on the training dataset, the CNN and FastGAN models' architecture, and the optimization algorithm developed are discussed.

5.2 Data Extraction

To train a DL model, more data is needed and thus data augmentation was applied. Each motor image generated by JMAG had the stator and rotor color to be blue. Since the focus is on the stator and rotor angles, color detection was applied for each image using Python's OpenCV library. In OpenCV, the red and green color channels are in the opposite order

when the image is read. In other words, the images are in BGR format instead of RGB. The image was converted from RGB format to HSV, which stands for Hue, Saturation, and Volume. The concept behind the color detection algorithm is to create a mask that is then applied on the original image to extract the color desired. In this case, a blue mask was applied on each image to extract the blue color of the stator and rotor. To apply the mask, a lower and upper bound of the HSV blue color is needed. After several experimentations, the lower HSV code for blue was found to be $[90, 50, 50]$ and the upper bound was $[130, 255, 255]$. Then a pixelwise AND operation was applied between the image and the mask. Any pixels in the original image is within the mask is kept and anything outside the mask is set to pixel 0 (i.e. the code for black).

Due to the symmetry of the motor image and to simplify model training, the extracted image was split into four quarters. To increase the dataset more, the colors of the stator and rotor were changed randomly for each quarter image 30 times. Since each motor image is split into four images and each of the four images is duplicated 30 times with different colors, this resulted in a new dataset consisting of 7440 images. A summary of the feature extraction process is displayed in Figure 5.5.

Figure 5.5: The data cleaning algorithm.

The programming language used to model the architecture of the CNN model and FastGAN is Python 3. The Python libraries that were used for the CNN and FastGAN data preprocessing were pandas, PIL, matplotlib, and numpy. Pandas was used to arrange the data into a table. For the CNN model, the table consisted of three columns: image path, average torque value, and torque ripple value (see Table 5.2). PIL library was used to read

the images and rescale them. The images were rescaled to $128 \times 128 \times 3$ for CNN model training. Matplotlib library was used to plot distribution of the dataset, plot the models' performance, and ensure the correct images have been imported. Numpy library was used to represent the images in a matrix format and to normalize the images for more accurate training. TensorFlow was the DL framework used for training the CNN and FastGAN models. The images were normalized from zero to one for the CNN model and the dataset was split into 5208 images for training, 1116 images for validation, and 1116 images for testing.

For the FastGAN model, the same data cleaning algorithm was also applied but with an original dataset of 114 images. This resulted in a training dataset of 14136 images. The images were rescaled to $256 \times 256 \times 3$ to achieve high quality images. The images were normalized to the range (-1, 1) since the generator model uses tanh as the activation function. All 14136 images were used for training the FastGAN.

Table 5.2: Example of CNN training data.

Image Path	Average Torque [Nm]	Torque Ripple [Nm]
…/14_0.png	4.685286	0.443294
…/14_colored_8.png	4.685286	0.443294
…/1_0.png	4.478206	0.547822
…/28_colored_108.png	5.263512	0.674058
…/60_1.png	5.197041	0.618147

5.3 CNN Model Training

The VGG19 was used as the CNN model for this study. In the original paper by Simonyan et al. [22], VGG19 is a classification model that consists of 16 convolutional layers and three fully connected layers. Since the current problem is a regression problem, the model was slightly altered. This was done by removing the three fully connected layers and replacing it with two layers: a dense layer comprising of 256 neurons and a layer consisting of two output neurons, one for average torque and the other is for torque ripple. The activation function for the dense layer was ReLU and for the last layer was the linear activation function. The initial weights for training were set to the ImageNet weights. The model architecture can be seen in Figure 5.6.

Choosing the right losses and metrics are key for an effective CNN model training for a regression task. The loss function used was the mean relative error (MRE) defined by the following equation:

$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - p_i}{y_i}\right|, \tag{5.1}$$

where N is the number of training images, y_i is the ground truth average torque and/or torque ripple from image i, and p_i is the predicted average torque and/or torque ripple of image i from the CNN model. The metric used for training the CNN model is the mean squared error (MSE). The Adam optimizer was used with an initial learning rate of 0.00007. To avoid overfitting, exponential learning decay method was applied every three epochs without going below $1e-10$ using the following equation:

$$\eta_{new} = \eta_{old} e^{-0.2}, \tag{5.2}$$

where η_{old} is the previous learning rate and η_{new} is the new learning rate three epochs later. The second method used to avoid overfitting is early stopping by monitoring the validation set's MRE with a patience of five epochs and restoring the best weights. Batch size was set to 25 and number of epochs were set to 150.

A second CNN model was developed to validate the FastGAN's generated images. The CNN takes a $128 \times 128 \times 3$ image and outputs the corresponding β_s and β_r. The same training images for the first CNN model was used to train this model with the target variables changed to the stator and rotor pole arc angles as seen in Table 5.3. Likewise, the same model architecture and hyperparameters were used.

Table 5.3: Example of the second CNN training data.

Image Path	**Stator Pole Arc Angle [°]**	**Rotor Pole Arc Angle [°]**
…/14_0.png	10	8
…/14_colored_8.png	10	8
…/1_0.png	8	9.1428
…/28_colored_108.png	11	11
…/60_1.png	10.75	12

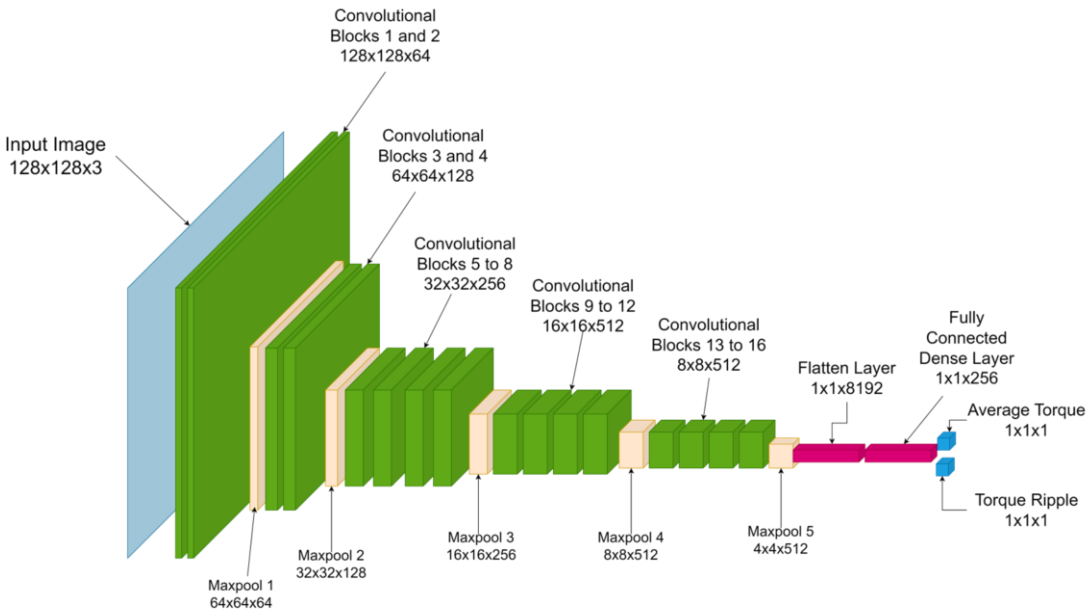

Figure 5.6: The model architecture of the CNN models. The second CNN model has stator and rotor pole arc angles instead of average torque and torque ripple.

5.4 FastGAN Model Training

As mentioned in chapter 4, the generator uses the SLE technique, and the discriminator is self-supervised and uses feature maps technique. The authors in [38] provided the source code for the FastGAN model in [41]. The generator consists of an input block that takes a noise vector with a length of 256 and outputs a matrix of dimensions $4 \times 4 \times 1024$. The input block consists of convolutional transposed layers, which does the opposite of convolutional layers, and uses batch normalization to reduce overfitting and gated linear unit (GLU) as the activation function [42]. Up-sampling is then performed with SLE

blocks to reach the desired image dimension, which is $256 \times 256 \times 3$. The up-sampling blocks are made of up-sampling 2D and convolutional layers with batch normalization to reduce overfitting and the GLU as the activation function. The SLE blocks consist of average pooling and convolutional layers with activation functions of LeakyReLU and sigmoid. The output block consists of convolutional layers and tanh as the activation function to output an image.

For the discriminator, the image is taken as the input and is then down sampled through a series of convolutional and pooling layers to reduce the dimension of the image to 8×8. The input block is made of the same layers as the input block in the generator and down-sampling blocks are made from convolutional layers with batch normalization and LeakyReLU as the activation function. The decoder blocks are made from convolutional layers, batch normalization layers, and uses GLU as the activation function. Partial and full reconstructions of the image are output so that both the generator and discriminator are able to extract the important features in the image using the tanh activation function. Classification block is made from two convolutional layers with batch normalization and LeakyReLU as the activation function. Figure 5.7 and Figure 5.8 show a summary of the generator and discriminator model architectures respectively.

The training of both the CNN and FastGAN models was conducted on a Dell G15 laptop with NVIDIA GeForce RTX 3060 as the GPU. The adversarial hinge loss was used for the generator and discriminator along with an additional reconstruction loss for the discriminator [38]. The metrics used were Fréchet Inception Distance (FID) and learned perceptual similarity (LPIPS). The optimizer used for training both the generator and

discriminator was Adam with the learning rate set to 0.0002. Due to GPU limitations and to avoid running out of memory, batch size was set to 6 and the number of epochs were 1000.

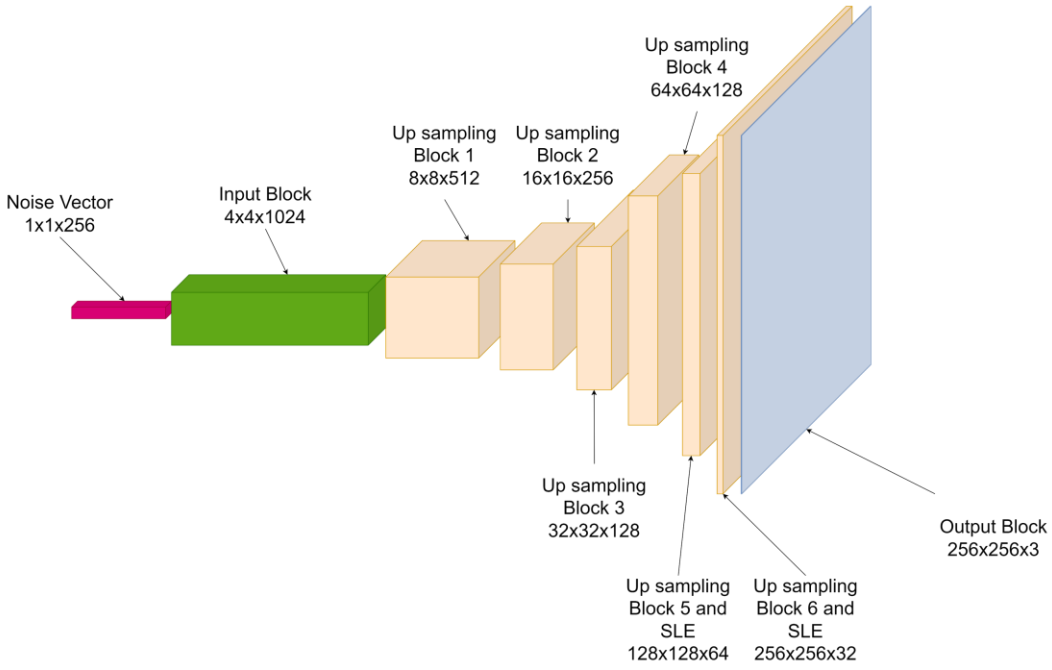

Figure 5.7: The FastGAN generator architecture.

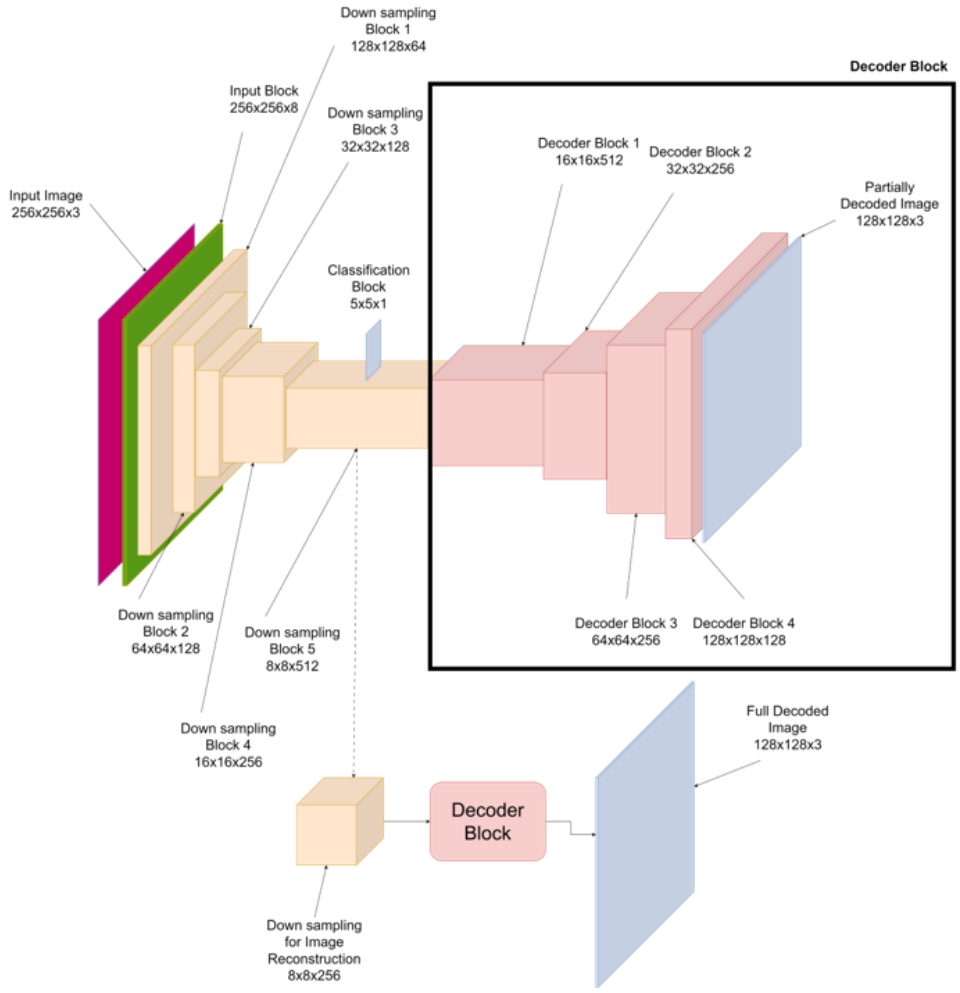

Figure 5.8: The FastGAN discriminator.

5.5 Optimization Algorithm

After constructing the FastGAN and CNN models, a method is needed to find the optimal design candidate with the lowest torque ripple and an average torque greater than 4.9 Nm. An algorithm was developed to find the optimal design candidate for a series of 500 values of β_s and β_r ranging from 9° to 12°. For each pair of pole arc angles, an image was

generated from the FastGAN. This image was then passed to the β_s-β_r-CNN model to retrieve the predicted angles. If the predicted angles lie within 1% of the required pole arc angles, then the generated image's quality was compared to the quality of images of the FastGAN training dataset. The metric used was structural similarity index (SSIM) according to the following equation:

$$\frac{1}{N_s}\sum_{i=1}^{N_s} SSIM(X, Y_i), \tag{5.3}$$

where N_s is a sample of s images from the training dataset, X is the generated image, and Y_i is a random image form the training dataset. The SSIM function used is found in [43] and s was set to 100. From experimentation, a SSIM score of 0.14 or less was found to be the threshold for the generated image to be of high quality. Once the generated image passes these two conditions, the average torque and torque ripple were then predicted using the T_{avg}-ΔT_{RMS}-CNN model. Figure 5.9 shows the chart of the DL models used.

From previous experience in the 6/14 SRM, a β_s greater than 10° results in high torque ripples [3]. This was also observed in Figure 5.3 and Figure 5.4. The designs from the algorithm were then sorted in ascending order of ΔT_{RMS}.

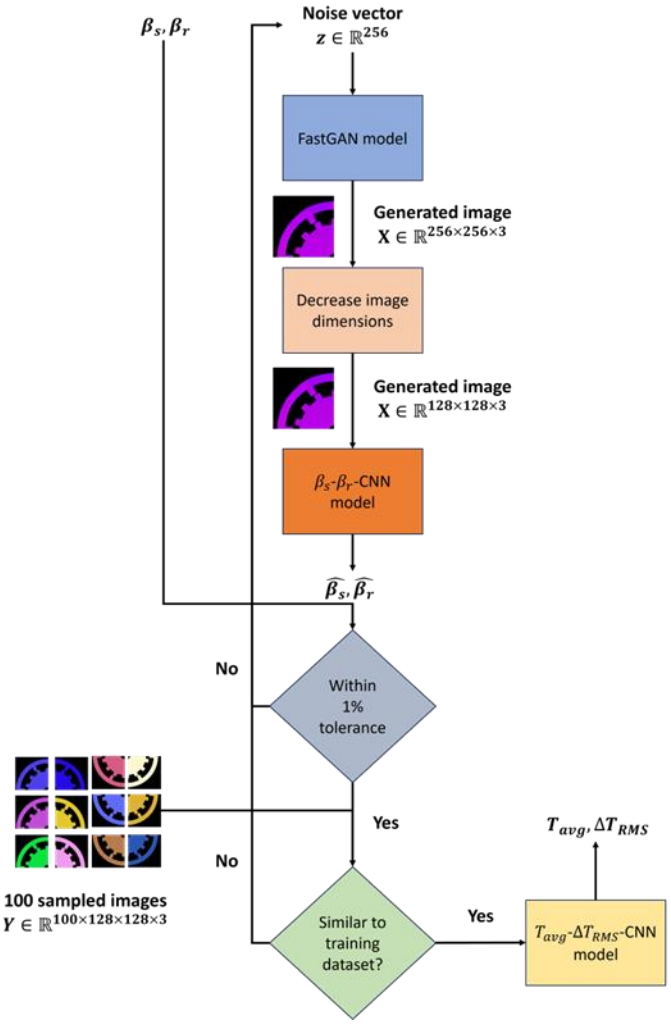

Figure 5.9: Flowchart of the overall algorithm developed to find the optimal design candidate.

Chapter 6

CNN and FastGAN Results and Discussion

6.1 CNN Models' Results

The T_{avg}-ΔT_{RMS}-CNN model was successfully trained after 66 epochs. Figure 6.1 and Figure 6.2 show the MRE and MSE plots for the average torque and torque ripple of the training and validation data as the number of epochs increased. The early stopping method detected epoch 61 had the best validation loss where the MRE for the average torque and torque ripple were 6.5155e-04 and 0.0024, respectively. The validation MSE for average torque and torque ripple were found to be 3.7594e-05 and 7.5812e-06, respectively. At epoch 61, the training MRE for average torque and torque ripple were 5.6061e-04 and 0.0019, respectively, and the training MSE for average torque and torque ripple were 2.6624e-05 and 1.3667e-06, respectively. The average torque and torque ripple MRE for the test dataset were calculated to be 6.775e-04 and 0.002399, respectively. The MSE values were 4.6429e-05 for T_{avg} and 6.7663e-06 for ΔT_{RMS}. Figure 6.3 and Figure 6.4 illustrate that the CNN predictions closely match the FEM results from JMAG, as indicated

by the alignment of blue points with the red reference line. Table 6.1 summarizes the training and testing results.

(a) (b)

Figure 6.1: The MRE plot of (a) average torque and (b) torque ripple of the training and validation datasets.

(a) (b)

Figure 6.2: The MSE plot of (a) average torque and (b) torque ripple of the training and validation datasets.

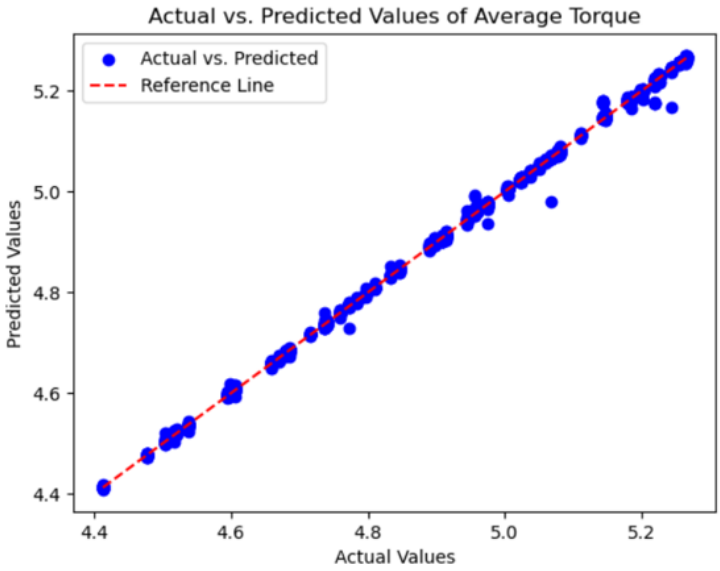

Figure 6.3: The CNN predictions of the test dataset for average torque compared to the FEM results.

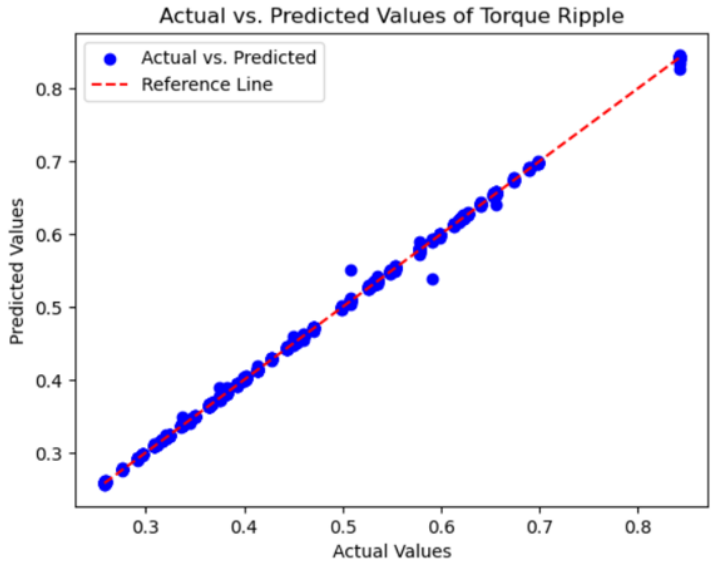

Figure 6.4: The CNN predictions of the test dataset for torque ripple compared to the FEM results.

Table 6.1: Summary of training and testing results of the T_{avg}-ΔT_{RMS}-CNN model.

Metrics	**Training**	**Testing**
MRE	T_{avg}: 5.6061e-04 ΔT_{RMS}: 0.0019	T_{avg}: 6.775e-04 ΔT_{RMS}: 0.002399
MSE	T_{avg}: 2.6624e-05 ΔT_{RMS}: 1.3677e-06	T_{avg}: 4.6429e-05 ΔT_{RMS}: 6.7663e-06

The β_s-β_r-CNN model was successfully trained after 80 epochs. The early stopping method detected epoch 75 had the best validation loss where the MRE for β_s and β_r were 4.7435e-04 and 4.0467e-04, respectively. The validation MSE for β_s and β_r were found to be 5.7833e-05 and 3.9346e-05, respectively. At epoch 75, the training MRE for β_s and β_r were 2.9248e-04 and 2.6494e-04, respectively, and the training MSE for β_s and β_r were 1.5056e-05 and 1.1507e-05, respectively. The β_s and β_r MRE for the test dataset were calculated to be 4.7951e-04 and 4.2142e-04, respectively. The MSE values were 5.1922e-05 for β_s and 4.3140e-05 for β_r. Table 6.2 summarizes the training and testing results. Figure 6.5 and Figure 6.6 show the β_s-β_r-CNN model predictions of the test dataset for the stator and rotor pole arc angles respectively.

Table 6.2: Summary of training and testing results of the β_s-β_r-CNN model.

Metrics	Training	Testing
MRE	β_s: 2.9248e-04 β_r: 2.6494e-04	β_s: 4.7951e-04 β_r: 4.2142e-04
MSE	β_s: 1.5056e-05 β_r: 1.1507e-05	β_s: 5.1922e-05 β_r: 4.3140e-05

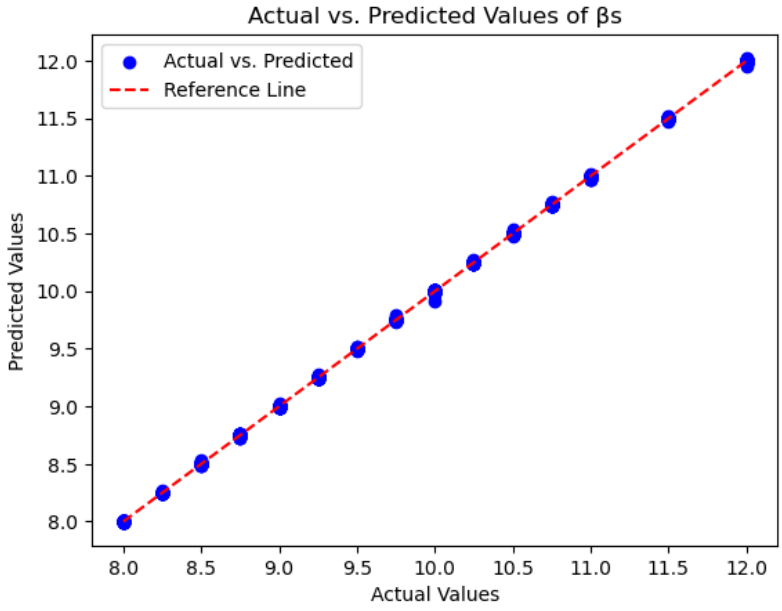

Figure 6.5: The CNN predictions of the test dataset for stator pole arc angle.

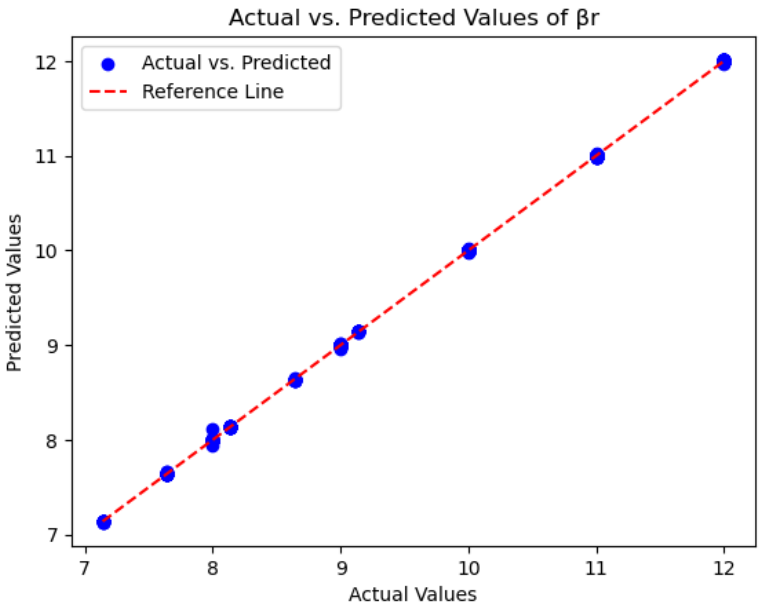

Figure 6.6: The CNN predictions of the test dataset for rotor pole arc angle.

6.2 FastGAN Model's Results

Figure 6.7 shows the progress of the FastGAN training from the quality of images generated for every 20 epochs. It was observed that the images generated at epoch 90 were of high quality. Thus, training was terminated at that epoch. To verify that FastGAN generated the correct quarter model motor images, the β_s-β_r-CNN model took the image generated by the FastGAN and predicted the stator and rotor pole arc angles with 1% tolerance. Figure 6.8 shows an example of the generated image with the baseline β_s and β_r to be 9.5 and 9.3, respectively.

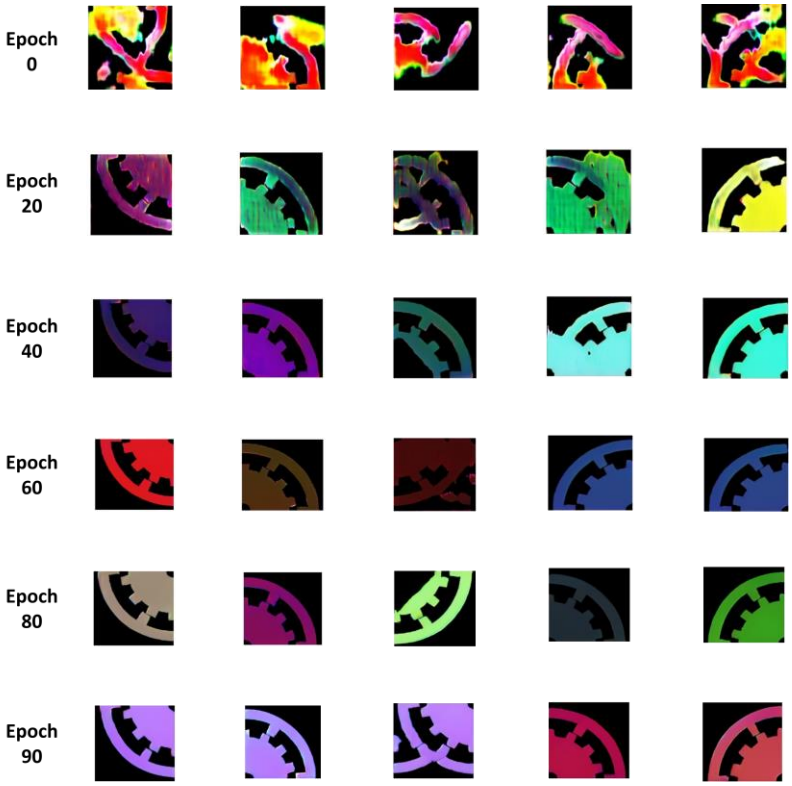

Figure 6.7: Generated images quality from FastGAN for every 20 epochs.

Figure 6.8: The generated image for the baseline. The predicted β_s and β_r were 9.541617 and 9.368641 respectively.

6.3 Optimization Algorithm's Results

After retrieving the average torque and torque ripple of the generated images using the optimization algorithm, the designs that had a stator pole arc angle less than 10° and a T_{avg} greater than 4.9 Nm were sorted in ascending order of torque ripple. The first point resulted in an average torque of 4.95597 Nm and the lowest torque ripple of 0.30418 Nm. The corresponding β_s was 9.03876° and β_r was 10.72958°. Figure 6.9 shows the baseline quarter motor model image and the generated optimal design's quarter motor model image. Figure 6.10 displays a 2-D scatter plot of the design candidates with $\beta_s < 10°$ and $T_{avg} > 4.9\ Nm$.

Figure 6.9: The baseline quarter model motor image compared to the optimal design's generated image.

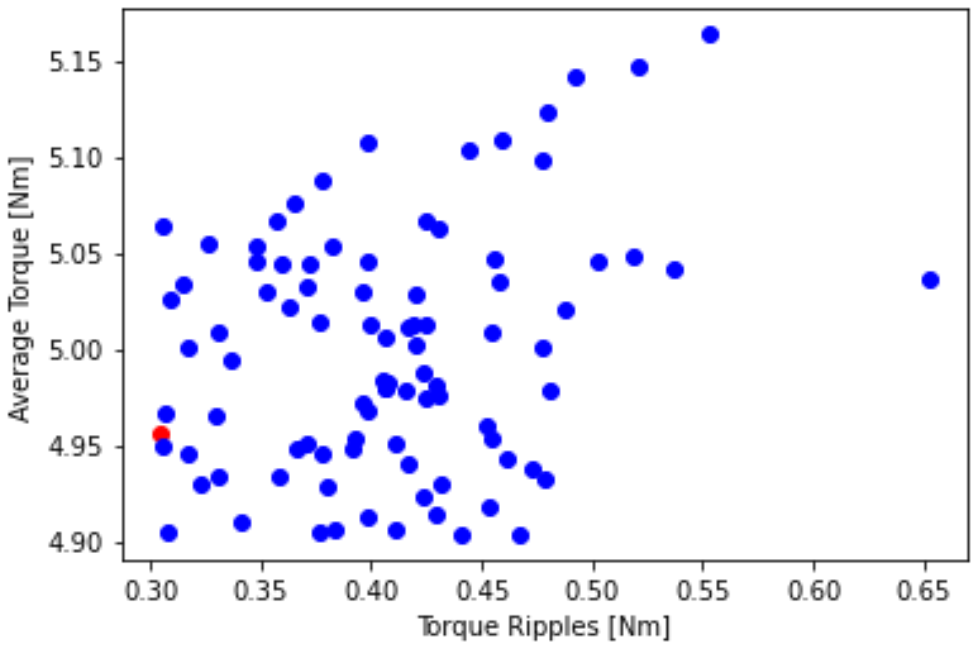

Figure 6.10: A scattered plot of the design candidates that had $\beta_s < 10°$ and $T_{avg} > 4.9\ Nm$. The optimal point is marked in red.

6.4 Discussion of Findings

Table 6.3 provides a comparison of the proposed CNN model to the previous CNN models developed. It is evident through the metrics that the proposed CNN model predicts T_{avg} and ΔT_{RMS} with high precision.

Table 6.3: Comparison of the proposed CNN model with previous CNN models.

Paper Reference	Results	Our Results
[24]	MAE for T_{avg}: 0.0473 MAE for ΔT_{RMS}: 0.0331	**MAE for T_{avg}: 0.00334** **MAE for ΔT_{RMS}: 0.00112**
[27]	MSE for flux linkage (Φ): 0.0129 MSE for d-axis inductance L_d: 0.0278 MSE for q-axis inductance L_q: 0.0577	**MSE for T_{avg}: 4.6429e-05** **MSE for ΔT_{RMS}: 6.7664e-06**
[25]	Correlation coefficient for d-axis flux (Φ_d): 0.992 Correlation coefficient for L_d: 0.881 Correlation coefficient for L_q: 0.988	**Correlation coefficient for T_{avg}: 0.9996** **Correlation coefficient for ΔT_{RMS}: 0.9998**
[26]	Correlation coefficient for T_{avg}: 0.996 Correlation coefficient for ΔT_{RMS}: 0.959	
[28]	Mean Absolute Percentage Error (MAPE) for T_{avg}: 7.54%	**MAPE for T_{avg}: 0.0675%**

The optimal design had a reduction of 4.86% in β_s and an increase of 15.39% in β_r compared to the baseline. Moreover, the optimal design had an increase in average torque by roughly 2% and a drastic decrease in torque ripple by 24% compared to the baseline. The estimated average torque and torque ripple of the optimal design were also verified by FEM. The MRE for average torque was 0.62%, and for the torque ripple was 0.29%. This

shows how accurate the FastGAN-CNN architecture is. Table 6.4 summarizes the comparison of the baseline, and the optimal design and Table 6.5 compares the estimated and actual FEM values of the optimal design's average torque and torque ripple. Figure 6.11 shows the dynamic torque plot of the baseline and optimal design. It can be observed that the total torque plot of the optimized design is more centralized on 5 Nm than the baseline plot, causing a reduction in the torque ripple.

Table 6.4: Comparison between the baseline and optimal design variables.

Parameters	**Baseline**	**FastGAN-CNN Proposed Model**
β_s [°]	9.5	9.03876
β_r [°]	9.3	10.72958
T_{avg} $[Nm]$	4.88	4.95597
ΔT_{RMS} $[Nm]$	0.398	0.30418

Table 6.5: Comparison of results between the FastGAN-CNN and FEM.

Parameters	**FastGAN-CNN Proposed Model**	**Actual FEM Results**	**MRE**
T_{avg} $[Nm]$	4.95597	4.9865	0.62%
ΔT_{RMS} $[Nm]$	0.30418	0.30329	0.29%

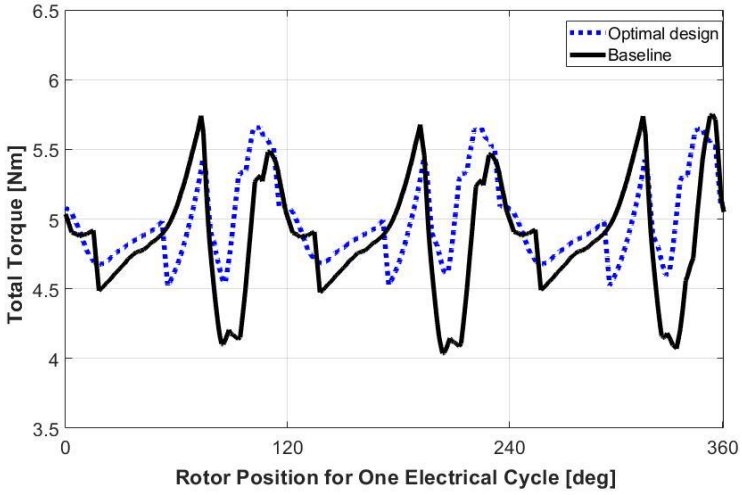

Figure 6.11: The total torque plot of one electrical cycle for the baseline and optimal design.

The proposed algorithm reduces the computational overhead compared to FEM. To get a single point using FEM and MATLAB, it would take approximately 40 minutes. The algorithm took a total of 20 hours to retrieve 500 points, which is roughly equivalent to 2.4 minutes. This means that the algorithm is 17 times faster than the traditional FEM method.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

This work proposed a design optimization approach using DL to minimize the torque ripple and maximize the average torque by changing the stator and rotor pole arc angles of a 6/14 SRM. This is achieved by using the FastGAN to generate cross-section images of the motor, followed by a CNN model that accurately predicts the stator and rotor pole arc angles with 1% tolerance. Then, the generated image gets passed into another CNN model that predicts the average torque and torque ripple. Using this architecture, a series of 500 design points were generated, and the optimal design was chosen. The optimal design was validated using FEM and it improved the average torque by 2% and decreased the torque ripple by 24% compared to the baseline. The proposed algorithm was also 17 times faster than the traditional FEM for one point.

7.2 Future Work

7.2.1 Physics Informed Neural Networks

The FastGAN-CNN architecture has demonstrated superior performance in optimizing the design. However, this architecture is heavily influenced by the training data provided to the models. In some situations, data acquisition can be very costly and conclusions will be drawn based on little information. Physics informed neural networks (PINNs) [44] solve supervised learning problems while following the physics laws represented as partial differential equations and added to the loss function of the model. In 2021, Zhao et al. [45] proposed a physics-informed convolutional neural network for a heat source layout application. PINNs are used in electromagnetic analysis [46][47] and can be explored in electric motor design.

7.2.2 Diffusion Models

One of the main disadvantages in GANs is that it is hard to train as there are many parameters that can be altered and can easily lead to mode collapse. In [48], the authors found that diffusion models outperform state-of-the-art GAN models in high quality image generation. To the author's knowledge, there is no previous work that applied diffusion models in electric motor design. Diffusion models rely on the concept of applying noise to an image and then denoising the image to get a diverse set of images. Using diffusion models can also affect the optimization algorithm as there will be no need to check the

similarity of the generated image to the training dataset. Thus, a genetic algorithm can be used in place of it.

7.2.3 Exploration of Additional Geometric Parameters and Topology Optimization using Deep Learning

This work focused on maximizing the average torque and minimizing torque ripple based on cross section motor images with various stator and rotor pole arc angles. Using only these two geometric parameters is not enough to design an SRM. Changing geometric parameters such as taper angle, stator and rotor pole heights, and stator and rotor back iron thickness affect the average torque and torque ripple [49]. These additions can provide a more robust design for the SRM. Altering these geometric parameters is a very sensitive task as they have an opposite effect on the average torque and torque ripple. Thus, understanding this trade-off is crucial to optimize the motor's performance. Including other outputs, such as motor efficiency, can also provide more insights on the performance of the SRM. GANs can be also useful for topology optimization of SRMs. For instance, generating a design for an SRM that has an optimal distribution of ferromagnetic material and air [2].

References

[1] "The federal government wants Canadians to switch to electric vehicles. Are they interested?", ici.radio-canada.ca. https://ici.radio-canada.ca/rci/en/news/2037482/the-federal-government-wants-canadians-to-switch-to-electric-vehicles-are-they-interested#:~:text=Environment%20Minister%20Steven%20Guilbeault%20last,emission%20vehicle%20sales%20by%202035, 2014, (accessed Mar. 18, 2024).

[2] M. Abdalmagid, E. Sayed, M. H. Bakr and A. Emadi, "Geometry and Topology Optimization of Switched Reluctance Machines: A Review," in *IEEE Access*, 2022.

[3] M. Omar, M. H. Bakr and A. Emadi, "Advanced Design Optimization of Switched Reluctance Motors for Torque Improvement Using Supervised Learning Algorithm," in *IEEE Access*, 2023.

[4] W. A. Aljaism, "Switched Reluctance Motor: Design, Simulation and Control," Ph.D. dissertation, University of Western Sydney, 2007. [Online]. Available: https://researchdirect.westernsydney.edu.au/islandora/object/uws:3650.

[5] N. Abdulah, F. Shukor, R. Othman, S. Ahmed, and N. Nasir, " Modelling methods and structure topology of the switched reluctance synchronous motor type machine: a review," in *International Journal of Power Electronics and Drive Systems (IJPEDS)*, 2023.

[6] M. Omar, E. Sayed, M. Abdalmagid, B. Bilgin, M. H. Bakr and A. Emadi, "Review of Machine Learning Applications to the Modeling and Design Optimization of Switched Reluctance Motors," in *IEEE Access*, 2022.

[7] G. Watthewaduge, E. Sayed, A. Emadi and B. Bilgin, "Electromagnetic Modeling Techniques for Switched Reluctance Machines: State-of-the-Art Review," in *IEEE Open Journal of the Industrial Electronics Society*, 2020.

[8] The MathWorks, Inc. (2022). *MATLAB version: 9.13.0 (R2022b).* Accessed: July 23, 2024. Available: https://www.mathworks.com.

[9] GNU Octave. (2024). *Octave version: 9.2.0 (2024).* Accessed: July 23, 2024. Available: https://octave.org.

[10] R. Shyam, and S. Singh, "A Taxonomy of Machine Learning Techniques," in *Journal of Advancements in Robotics*, 2021.

[11] S. Heddam, "Generalized Regression Neural Network Based Approach as a New Tool for Predicting Total Dissolved Gas (TDG) Downstream of Spillways of Dams: a Case Study of Columbia River Basin Dams," in *Environmental Process*, 2017.

[12] H. Sahraoui, H. Zeroug, and H. A. Toliyat, "Switched reluctance motor design using neural-network method with static finite-element simulation," in *IEEE Transactions on Magnetics*, 2007.

[13] Z. Zhang, S. Rao, and X. Zhang, "Performance prediction of switched reluctance motor using improved generalized regression neural networks for design optimization," in *CES Transactions on Electrical Machines and Systems*, 2018.

[14] C. Lee and W. Ha, "Optimal Design of IPM Rotor Shape Using Generative Adversarial Networks," in *2021 24th International Conference on Electrical Machines and Systems (ICEMS)*, 2021.

[15] D. Foster, "Generative Deep Learning Teaching Machines to Paint, Write, Compose, and Play," in *O'Reilly Media*, 2019.

[16] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep

learning: concepts, CNN architectures, challenges, applications, future directions," in *Journal of Big Data*, 2021.

[17] Benyamin Ghojogh, "Deep Learning, F23(5): Convolution, Pooling, Batch Normalization, LeNet, AlexNet, VGG, U-Net, ResNet," *YouTube*, Oct. 17, 2023 [Video file]. Available: https://www.youtube.com/watch?v=c8HdZOac0e0.

[18] Benyamin Ghojogh, "Deep Learning, F23(4): Deep Learning, F23(4): Backpropagation, SGD, AdaGrad, RMSProp, Adam, PyTorch code of network, CNN," *YouTube*, Oct. 3, 2023 [Video file]. Available: https://www.youtube.com/watch?v=95VQJX1mKds&t=3041s.

[19] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining,* New York, NY, USA, Aug. 2014, pp. 661–670.

[20] G. Hinton, N. Srivastava, and K. Swersky, ''Neural networks for machine learning—Lecture 6a: Overview of mini-batch gradient descent,'' Dept. Comput. Sci., Toronto Univ., Toronto, ON, Canada, 2012. Accessed: July 23, 2024. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980.*

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *arXiv*, 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition*, 2016.

[24] H. Sasaki, Y. Hidaka and H. Igarashi, "Prediction of IPM Machine Torque Characteristics Using Deep Learning Based on Magnetic Field Distribution," in *IEEE Access*, 2022.

[25] J. Asanuma, S. Doi and H. Igarashi, "Transfer Learning Through Deep Learning: Application to Topology Optimization of Electric Motor," in *IEEE Transactions on Magnetics*, 2020.

[26] H. Sato, and H. Igarashi, "Deep learning-based surrogate model for fast multi-material topology optimization of IPM motor", in *The international journal for computation and mathematics in electrical and electronic engineering (COMPEL)*, 2022.

[27] Y. Shimizu, S. Morimoto, M. Sanada and Y. Inoue, "Automatic Design System With Generative Adversarial Network and Convolutional Neural Network for Optimization Design of Interior Permanent Magnet Synchronous Motor," in *IEEE Transactions on Energy Conversion*, 2023.

[28] S. Barmada, N. Fontana, L. Sani, D. Thomopulos and M. Tucci, "Deep Learning and Reduced Models for Fast Optimization in Electromagnetics," in *IEEE Transactions on Magnetics*, 2020.

[29] J. Gui, Z. Sun, Y. Wen, D. Tao and J. Ye, "A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications," in *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[30] S. Yang, Y. Meng and X. Meng, "Image Transfer Applied in Electric Machine Optimization," in *2020 IEEE 61th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), Riga, Latvia*, 2020.

[31] Benyamin Ghojogh, " Deep Learning, F23(10): Deep Q-Network, Generative Adversarial Network, Knowledge Distillation," *YouTube*, Nov. 28, 2023 [Video file]. Available: https://www.youtube.com/watch?v=Wffuii2Yr2s&t=4547s.

[32] S. Chintala, E. Denton, M. Arjovsky, and M. Mathieu, "How to train a GAN? Tips and tricks to make GANs work." *GitHub*, 2016.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in *Neural Information Processing Systems 27*, Montreal, Quebec, Canada, 2014.

[34] M. Mirza and S. Osindero, "Conditional generative adversarial nets", in *arXiv preprint*, arXiv: 1411.1784, 2014.

[35] Radford, A., Metz, L., and Chintala S., “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” in *arXiv preprint*, arXiv:1511.06434, 2015.

[36] P. Isola, J. -Y. Zhu, T. Zhou and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017.

[37] X. Ding, Y. Wang, Z. Xu, W. J. Welch and Z. J. Wang, "Continuous Conditional Generative Adversarial Networks: Novel Empirical Losses and Label Input Mechanisms," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[38] B. Liu, Y. Zhu, K. Song, and A. Elgammal, ”Towards Faster and Stabilized GAN Training for High-fidelity Few-shot Image Synthesis.”, In *International Conference on Learning Representations*, 2021.

[39] M. Heroth, H. C. Schmid, R. Herrler and W. Hofmann, "Image-Based Optimization of Electrical Machines Using Generative Adversarial Networks," in *2023 IEEE International Electric Machines & Drives Conference (IEMDC)*, San Francisco, CA, USA, 2023.

[40] Kasprzak, M., 6/14 switched reluctance machine design for household HVAC system applications, MASc thesis, Department of Mechanical Engineering, McMaster University, Hamilton, Canada, 2016.

[41] Gábor Vecsei, " Towards Faster and Stabilized GAN Training for High-fidelity Few-shot Image Synthesis.", github.com, https://github.com/gaborvecsei/SLE-GAN, 2020, (accessed Jun. 19, 2024).

[42] Y. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language Modeling with Gated Convolutional Networks.", In *arXiv preprint*, arXiv:1612.08083, 2017.

[43] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures," in *IEEE Signal Processing Magazine,* vol. 26, no. 1, pp. 98-117, Jan. 2009.

[44] M. Raissi, P. Perdikaris, and G. E. Karniadakis, ‘‘Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations,’’ *CoRR,* vol. abs/1711.10561, 2017.

[45] X. Zhao, Z. Gong, Y. Zhang, W. Yao, and X. Chen, "Physics-informed Convolutional Neural Networks for Temperature Field Prediction of Heat Source Layout Without Labeled Data," *Engineering Applications of Artificial Intelligence,* vol. 117, 2023.

[46] A. Khan and D. A. Lowther, "Physics Informed Neural Networks for Electromagnetic Analysis," in *IEEE Transactions on Magnetics,* vol. 58, no. 9, pp. 1-4, Sept. 2022, Art no. 7500404.

[47] O. Noakoasteen, S. Wang, Z. Peng and C. Christodoulou, "Physics-Informed Deep Neural Networks for Transient Electromagnetic Analysis," in *IEEE Open Journal of Antennas and Propagation,* vol. 1, pp. 404-412, 2020.

[48] P. Dhariwal and A. Nichol, "Diffusion Models Beat GANs on Image Synthesis," in *arXiv Preprint,* arXiv:2105.05233v4, 2021.

[49] B. Bilgin, J. W. Jiang, and A. Emadi, *Switched Reluctance Motor Drives: Fundamentals to Applications.* Boca Raton, FL, USA: CRC Press, 2019.