RYERSON UNIVERSITY

FACULTY OF ENGINEERING, ARCHITECTURE AND SCIENCE

DEPARTMENT OF AEROSPACE ENGINEERING

The Implementation of Convolutional Neural Networks for Warp Detection in 3D Printed Components manufactured via Fused Filament Fabrication: A Bayesian-Based Automated Approach

Aditya Saluja

AER870 Aerospace Engineering Thesis

Faculty Advisor: Dr. Kazem Fayazbakhsh

Data: April 2020

ACKNOWLEDGMENT

I would like to thank my supervisor Dr. Kazem Fayazbakhsh of the Department of Aerospace Engineering at Ryerson University. His guidance and support helped me advance my knowledge and understanding in the field of 3D printing and applied machine learning for additive manufacturing.

Finally, I would like to thank my family for their support and encouragement throughout the project.

Abstract

Fused Filament Fabrication (FFF) is an additive manufacturing technique commonly used in industry to produce complicated structures sustainably. Although promising, the technology frequently suffers from defects, including warp deformation compromising the structural integrity of the component and, in extreme cases, the printer itself. To avoid the adverse effects of warp deformation, this thesis explores the implementation of deep neural networks to form a closed-loop in-process monitoring architecture using Convolutional Neural Networks (CNN) capable of pausing a printer once a warp is detected. Any neural network, including CNNs, depend on their hyperparameters. Hyperparameters can either be optimized using a manual or an automated approach. A manual approach, although easier to program, is often time-consuming, inaccurate and computationally inefficient, necessitating an automated approach. To evaluate this statement, classification models were optimized through both approaches and tested in a laboratory scaled manufacturing environment. The automated approach utilized a Bayesian-based optimizer yielding a mean accuracy of 100% significantly higher than 36% achieved by the other approach.

Table of Contents

List of Figures

List of Tables

Nomenclature

To avoid confusion, the term "parameter" or "model parameter" stands for any variable that is learned during the training process. In contrast, any variable defined before training is referred to as "hyperparameter" or "model hyperparameter." "Weight" is used interchangeably with the "model parameter" but denotes a learnable variable outside of the convolution layer.

1.0 Introduction

The past two decades have seen exponential growth in Additive Manufacturing (AM), an efficient technique that offers cost-effectiveness and sustainable manufacturing. Unlike conventional subtractive techniques, the ability of AM to handle complex geometries while conserving material has led to its widespread use in a variety of applications. A common AM technique used in industry is Fused Filament Fabrication (FFF). FFF (3D printing) adopts thermoplastic extrusion to fabricate a component from the base up, enabling streamline design with fewer overall components.

The aerospace industry is often dependent on such techniques to address supply chain constraints, limited warehouse space, and reduced wasted materials, typically seen in conventional manufacturing techniques. Airbus has been a strong advocate of AM to produce components for its A350 aircraft [1,2]. Components are printed with ULTEM 9085 resin wherein the resin is melted and extruded layer-by-layer to fabricate the part, resulting in an overall reduction of weight by about 20%. Similarly, GE manufactured components for Denali single-engine aircraft resulting in the elimination of 845 parts saving procurement, installation, and inspection efforts [1,2].

Although reliable, FFF printed components frequently exhibit defects that impair their functionality and performance leading to significant loss of material and time. Warpage represents one of the most typical defects in FFF [3,4,5,6]. Often, after material deposition, since the cooling rate is not uniform throughout the printed component, a temperature gradient forms within the print leading to residual thermal stresses. As the layers increase, the residual stress builds up, causing the component to distort, i.e., the bending force is higher than the adhesion between the component and the build platform, causing it to warp [9,10]. Warp deformation can happen at any time during the print and typically leads to print failure, compromised dimensional accuracy, and in extreme cases, damage to the 3D printer.

Multiple studies have been performed to analyze the factors that lead to warp deformation. Wang et al. [8] identified chamber temperature, material thermal expansion rate and the number of layers as critical parameters for warp deformation. Fitzharris et al. [12] studied the impact of material properties where altering polyphenylene sulphide to polyphenylene; they observed that materials with high thermal conductivity and low thermal expansion rates were less prone to warp deformation. Similarly, Peng et al. [10] indicated that decreasing the thermal expansion coefficient and glass transition temperature by treating a filament could reduce the chance of warpage. Other studies included analyzing geometric parameters on warp deformation. In a study conducted by Guerrero-de-Mier et al. [11] and Armillotta et al. [9], they concluded that decreasing the size of the horizontal plane of a coupon decreased the chance of deformation.

These studies provide an insight into critical factors that lead to warp deformation; however, due to design constraints, not all strategies can be implemented. Consistent monitoring of fabrication is thus required to detect defects limiting the loss of time and material.

In recent years, with the advancement of hardware accelerators, image processing and computer vision have been deployed for in-situ defect detection of 3D printed parts. Baumann et al. [17] utilized image processing to detect missing material and movement caused by the deformation of 3D printed components. The parts were recognized using a colouring threshold, followed by a binary difference image to identify defects. The model accuracy was severely affected by variations in lighting conditions and the colour of the filament, limiting the practicability of the monitoring system. A similar approach was used by Yi et al. [23], where a camera was utilized to record a print, layer by layer. The contours of each image were then extracted through image processing and analyzed using statistical process control to find defects. Although promising, these methods are often vulnerable to small variations in input data. Currently, machine learning methodologies provide an infrastructure capable of improving itself with data accumulation that adds flexibility by adapting to various industries and standards based on the manufacturing environment [26].

Delli and Chang [24] developed a system using a support vector machine (SVM) to classify printed components into either defective or non-defective by collecting images throughout the printing process to define an ideal print job. Similarly, Wu et al. [25] identified infill defects via Naïve Bayes Classifier and Decision Trees algorithm to achieve an accuracy of 85.26 and 95.51%, respectively. Recently, Z. Zhang et al. [18] and B. Zhang et al. [19] inspected defects in welded components Convolutional Neural Network (CNN) and explored the versatility and potential of machine learning in manufacturing process inspection.

This study focuses on developing a closed-loop in-process monitoring system capable of detecting warped corners in FFF printed components using CNN. In the following sections, a manual method is compared to an automated Bayesian-based black-box optimization algorithm for selecting the suitable CNN classification model. Next, a closed-loop architecture capable of pausing a print when a warp is detected is proposed. Finally, a discussion of the complete system is presented before the conclusion and recommendations for future work.

2.0 Theory

An Artificial Neural Network (ANN) is a mathematical model resembling the networked structure of neurons in the brain, with layers of connected nodes. Traditionally neural networks were shallow (containing two to three hidden layers), which were good at memorization, but lacked generalization limiting their practicability. Deep learning methodologies, however, utilize neural networks with several hidden layers to learn features at various levels of abstraction. A deeper network is better at generalizing as the network learns all the intermediate features between the input data and the high-level classification and thus achieves promising results for complex classification applications [27].

One of the most popular types of a deep neural networks is the Convolutional Neural Network (CNN). CNN learns relevant features while training on a collection of images, making it suitable for computer vision tasks such as object classification. This section focuses on the basic concepts of CNN and its applicability in image classification [20,27].

2.1 Convolutional Neural Network: An Overview

CNN is a mathematical construct designed for processing spatial hierarchies of features within an image adaptively. Typically, CNN is composed of three types of layers: convolution, pooling, and fully connected layers. The first two layers deal with feature extraction whereas, the fully connected layer, associates the extracted features to the final output [20].

A convolution layer comprises a series of learnable filters known as kernels. A kernel is a matrix that convolves over a subset of input pixels to produce a feature map. The creation of multiple feature maps in each convolutional layer helps extract different features from the input image. As the classification function modeled through the neural network is unlikely to have a linear relation with the input, nonlinearities are added to generalize the system, enhancing the performance of the neural network. These nonlinearities are added by employing activation functions analogous to synapses in a brain. To improve the performance of the model, it is essential to make features invariant to small translations. This is typically achieved by downsampling the feature maps through means of pooling. The feature maps are then flattened and passed through fully connected layers, and finally, the output layer completing the architecture of a typical CNN. The process of optimizing parameters to minimize the difference between outputs and truth labels is called training and is achieved by minimizing the error function through means of backpropagation.

2.2 Building blocks of CNN architecture

This section provides an overview of a typical CNN architecture and the training process.

2.2.1 Convolution

Convolution is a mathematical operation to extract features by applying kernels across an input image. Element-wise product between each element of the kernel and the image is then calculated and summed, as illustrated in Figure 1. The values of each element in a kernel are learned throughout the training process to ensure essential features are captured, a critical factor for getting the right prediction [20,27].

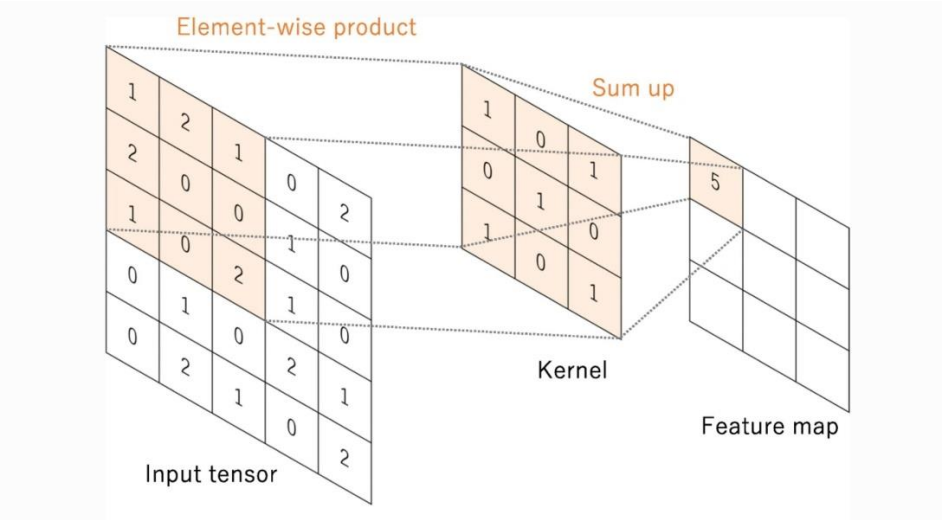


Figure 1: An example of convolution operation with a kernel size of 3 × 3 applied across an input tensor to obtain an output value for a feature map. [20]

Figure 2 presents an example of different features extracted from an image.

Figure 2: An example of how kernels obtain feature maps in a convolution layer. [30]

A 32×32-pixel image was passed through a convolution layer with six kernels to produce six feature maps. Each feature map is unique and represents an essential feature of the input. Brighter spots in the map represent the extracted feature; for example, the first map (left-most) extracts the horizontal aspects of the number six while the fifth map extracts vertical ones. With additional layers and an optimal number of kernels, a network ensures a wholesome understanding of images within the dataset; however, these hyperparameters are often hard to select and will be discussed later in the investigation.

2.2.2 Activation Function

An activation function is a mathematical gate that maps an input (in the current neuron) to the output in the subsequent layer deciding which neurons are relevant (for a feature) influencing the accuracy and efficiency of any ANN.

Activation functions can typically be divided into two groups:

1. Linear Activation Function
2. Non-Linear Activation Function

Linear Activation Functions

In a linear activation function, the activation is proportional to the input. The derivative of such a function is constant. This implies that the error in prediction is constant and does not depend on the input, making it impossible to understand what weight in the input neurons can provide a better prediction. Besides, a linear combination of a linear function is still linear; thus, all layers of the ANN can collapse into one. This creates a linear regression model that fails to learn the complex functional mapping from input data [22].

Non-Linear Activation Functions

Non-linear activation functions address these problems by:

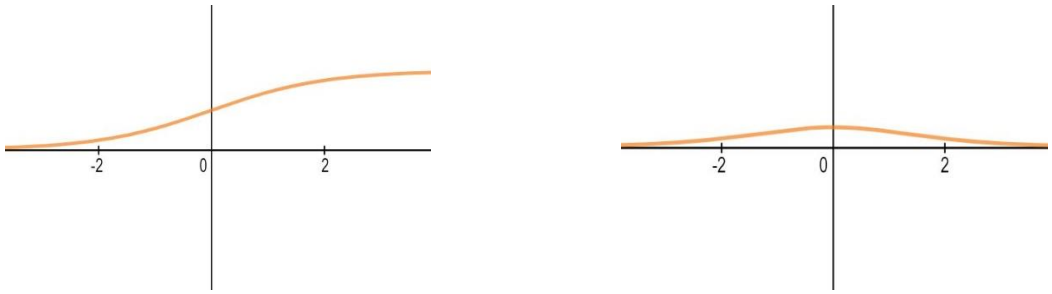
1. Enabling backpropagation as the derivative function depends on the input.
2. Allowing the stacking of neurons to develop deep neural networks. Deep networks have various hidden layers of neurons that allow complex features to be extracted with high accuracy.

Common non-linear activation function includes sigmoid, *hyperbolic tangent*[1], Rectified Linear Unit (ReLU) and Leaky ReLU.

[1] Similar to the sigmoid function, however, the output is zero centered, making it easy to identify neurons that have a strong positive, negative, or no impact on the network.

Sigmoid

Sigmoid offers two significant advantages, bounding the output values to 0 and 1, thus, normalizing the output of each neuron and creating a smooth gradient to enable backpropagation. As shown in Figure 3, the derivative of the function resembles a bell curve. This leads to the vanishing gradient problem whereby very large and very low values of **x** have no impact on the prediction, preventing the network from learning further [22].

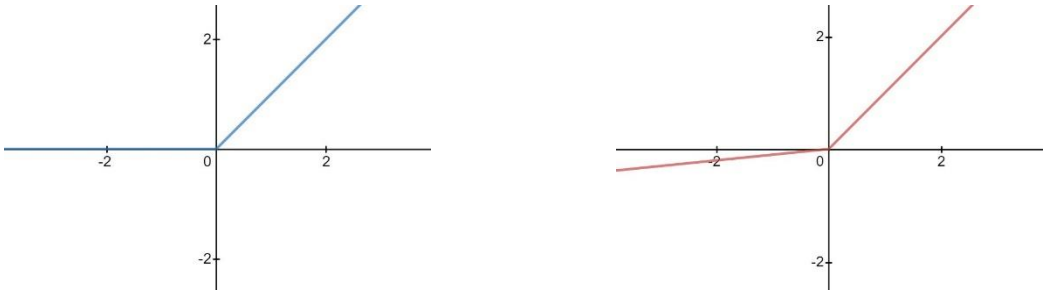


(a) **(b)**

Figure 3: A graphical representation of sigmoid function: (a) Sigmoid Function; (b) Derivative of Sigmoid Function

ReLU and Leaky ReLU

In a deep neural network, sigmoid or hyperbolic tangent will cause all neurons to fire, i.e., all activations will be processed to obtain an output making them computationally expensive. Visible in Figure 4, ReLU prevents this by setting all negative values (neurons deemed not useful for a specific feature) to zero and proportional to the input for all values above or equal to zero. However, inputs that approach zero or are negative cannot perform backpropagation leading to the dying ReLU problem. Leaky ReLU attempts to solve the issue by having a small positive slope in the negative region to enable backpropagation [22].



(a) **(b)**

Figure 4: A graphical representation of two commonly used activation functions: (a) ReLU Function; (b) Leaky ReLU

2.2.3 Pooling

Typically feature maps produced through convolution are sensitive to the location of features in the input image, limiting the generalizability of the model. An approach to address this sensitivity is to down-sample the feature maps. Pooling is a downsampling operation that reduces the in-plane dimensionality of feature maps, making them invariant to small translations and distortion, simultaneously reducing the number of learnable parameters in the subsequent layers [20].There are no learnable model parameters in any of the pooling layers; however, similar to convolution, the hyperparameters can be optimized to enhance model performance. The two conventional pooling methodologies are:

Max Pooling & Global Average Pooling (GAP)

As the name suggests, max-pooling identifies the most activated presence of a feature in a feature map. It is a simple mathematical operation wherein patches of data are extracted from an input feature map, then storing the maximum value within that patch to produce a downsampled map. Another common approach is to summarize the presence of a feature within a feature map. GAP works by extracting several patches of data and storing the mean value of each patch to reduce the dimension of the feature map [20].

2.2.4 Fully Connected Layer

Feature maps from the last convolution/pooling layer are flattened, i.e. transformed into a vector and connected to each neuron in the subsequent layer by a learnable weight. This is done to map extracted features to the final output of the network. The final fully connected layer of a CNN has the same number of output nodes as the number of classes. An activation function is then applied to get the predicted class [20].

2.2.5 Last Layer Activation

The activation function applied to the last fully connected layer of the network is different then the non-linear activation function covered in Section 2.2.2. For a multiclass classification problem, a softmax function is often used which normalizes the outputs for each class between 0 and 1, divided by their sum to yield the probability of a specific class.

Mathematically a softmax function is represented as:

$$P(y = j \mid z^i) = \frac{e^{z^i}}{\sum_{j=0}^{K} e^{z_j}} \tag{1}$$

where

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w^T x} \tag{2}$$

Here $\boldsymbol{w}$ represents the weight vector and $\boldsymbol{x}$ the feature vector of one training sample. The softmax function computes the probability, $P(y = j \mid z^i)$, of the training sample belonging to some class given an input z^i to predict a class label j [28].

2.2.6 Training the Network

Training involves finding the right set of kernels and weights in convolution and fully connected layers to minimize error in the predicted output. A model is analyzed through a loss function during forward propagation, and learnable model parameters are updated with respect to the error through backpropagation and gradient descent (Figure 5) [20]. A comprehensive explanation of backpropagation, loss function and gradient descent is beyond the scope of this thesis.

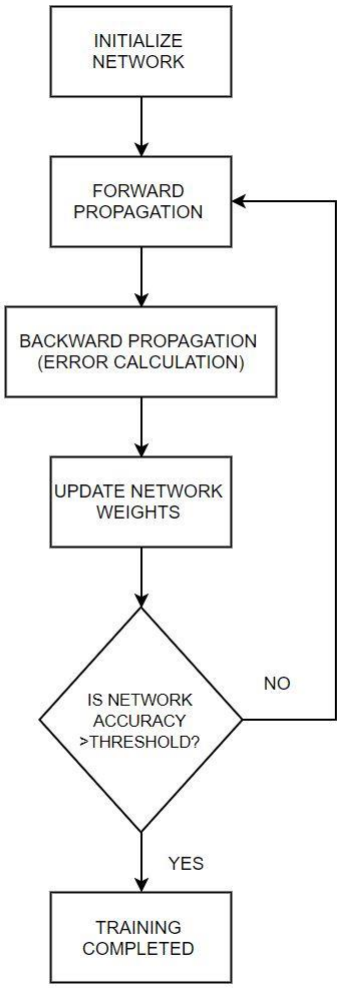


Figure 5: A simplified representation of the training process

2.3 Building the Network

Depending on the nature of the problem and computational resources, a CNN architecture can be constructed using a manual or an automated approach.

2.3.1 Manual Approach

As the name suggests, this approach requires a user to make design decisions manually. Design decisions include selecting hyperparameters, *training and regularization procedures*[2] to ensure models perform as intended. The performance of machine learning models, in general, is susceptible to a wide range of hyperparameter choices. Each time a different set of hyperparameters is selected, the machine learning model has to be retrained. Some machine learning models, such as CNNs, take several days to be trained, making the process irrational to do by hand [21].

2.3.2 Automated Approach

With an increase in hardware accelerators, the past decade has seen exponential growth in machine learning models and their applications in various domains.

The idea of extending optimization from model parameters to design decision is covered in the field of automated machine learning (AutoML). AutoML involves automating various stages of machine learning application pipeline: data preparation, feature extraction, hyperparameter optimization, and model selection. Since the performance of machine learning models changes considerably with hyperparameters, this paper focuses on developing a general-purpose pipeline to optimize hyperparameters and subsequently select models for the problem at hand. Although an automated pipeline reduces human efforts, thereby improving efficiency and performance, the underlying optimization algorithms faces two significant challenges:

1. The hyperparameter configuration space is a complex, multidimensional problem often composed of discrete, continuous, conditional, and categorical data types. The combination of datatypes makes it challenging to define a range in which hyperparameters need to be optimized [21, 29].
2. A machine learning model is optimized by minimizing a cost function. However, for a hyperparameter optimization problem, the shape of the function is unknown, nonlinear, non-convex, and multidimensional. As little is known about the cost function, black-box optimization methodologies are often employed to optimize the hyperparameter configuration space [21].

At a higher level hyperparameter optimization can be represented as:

[2] This paper encapsulates training procedure and regularization into hyperparameters for simplicity.

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} f(\lambda) \quad (3)$$

Here, $f(\lambda)$ refers to the objective function to be minimized – typically evaluated on the validation set; λ^* is a vector that minimizes the objective function and, λ a vector in the hyperparameter in configuration space that can take any value in the domain Λ.

Although global optimization algorithms are usually preferred, the non-convex nature of the problem often leads to premature convergence. This local convergence is still useful as it allows to make progress within a few function evaluations. Common algorithms include model-free blackbox optimization and Bayesian optimization [21].

Model-Free Blackbox Optimization Methods

Model-free blackbox optimization methodologies include grid search and random search. Grid search is a simple optimization algorithm wherein a user defines a finite set or grid of values for each hyperparameter. The algorithm then performs the Cartesian product of this grid; however, since the number of evaluations increases with the dimensionality of the hyperparameter configuration space, this method suffers from the curse of dimensionality. Furthermore, to improve the resolution of discretization, additional function evaluations are required, which further increases the computational cost. These costs can be limited by setting a computational budget. Random search is an alternative wherein the algorithm randomly searches for samples within a hyperparameter configuration space until a certain computational threshold is reached. Random search is often used to initiate an optimization process. This is because the algorithm searches throughout the hyperparameter configuration space achieving performance close to the optimum. Although better than manual tuning, both methodologies ignore past evaluations and often spend time and computational resources on unimportant hyperparameters [21].

Bayesian Optimization Methodologies

In contrast to the previous approaches, a Bayesian optimizer is an iterative algorithm that develops a surrogate model and an acquisition function to select hyperparameters in an informed manner. The algorithm approximates the objective function, $f(\lambda)$ by drawing samples from $f(\lambda)$ to obtain posterior that better describe $f(\lambda)$. The probabilistic model developed to approximate the objective function is called a surrogate model [21, 29].

The acquisition function uses the probabilistic model to select next samples from the hyperparameter configuration space. A common acquisition function is the expected improvement (EI) which provides a balance between exploitation and exploration. Mathematically, the function is represented as follows:

$$\mathbb{E}[\mathbb{I}(\lambda)] = \mathbb{E}[\max(f(\lambda) - f(\lambda^+),0)] \quad (4)$$

where $f(\lambda^+)$ is the value of the best sample so far. The expected improvement is typically calculated using a Gaussian Process model. Analytically this process is evaluated as:

$$\mathbb{E}[\mathbb{I}(\lambda)] = \begin{cases} (\mu(\lambda) - f(\lambda^+) - \xi)\Phi(Z) + \sigma(\lambda)\phi(Z) & \sigma(\lambda) > 0 \\ 0 & \sigma(\lambda) = 0 \end{cases} \quad (5)$$

where

$$Z = \begin{cases} \dfrac{(\mu(\lambda) - f(\lambda^+) - \xi)}{\sigma(\lambda)} & \sigma(\lambda) > 0 \\ 0 & \sigma(\lambda) = 0 \end{cases} \quad (6)$$

$\mu(\lambda)$ and $\sigma(\lambda)$ are the mean and standard deviation of the Gaussian Process, respectively; Φ and ϕ the Cumulative Density Function (CDF) and Probability Density Function of the normal distribution. The expression $\mu(\lambda) - f(\lambda^+) - \xi$ is the exploitation term while $\sigma(\lambda)$ represents the exploration. Exploration refers to searching over the entire sample space, while exploitation consists of exploring a limited but promising search space. The term ξ determines the amount of exploration. Increasing ξ will lead to more exploration as the improvement predicted by the mean decreases with relative to $\sigma(\lambda)$ describing high prediction uncertainty [21, 29].

The terms $\mu(\lambda)$ and $\sigma(\lambda)$ are obtained by:

$$\mu(\boldsymbol{\lambda}) = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y}, \sigma^2(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_* \quad (7)$$

where $k(\boldsymbol{\lambda}, \boldsymbol{\lambda})$ denotes the covariance matrix, $\boldsymbol{k}_*$ the vector of covariance between $\boldsymbol{\lambda}$ and all previous observations, $\boldsymbol{K}$ the covariance matrix of previous evaluated configurations and y the values of observed functions [21].

This mathematical construct allows evaluating the degree of impact of specific hyperparameters from previous iterations to select the next set of hyperparameters in an informed manner. Bayesian optimizers spend more time selecting; however, the time spent in selecting is often inconsequential to the frequent call made to the objective function as in model-free blackbox optimization methodologies.

3.0 Methodology

This section provides details regarding experimental setup, dataset preparation and an in-depth comparison of designing a CNN classification algorithm through two approaches: manual and automated.

3.1 Experimental Setup

Figure 6 outlines the setup used for this study. All tests were carried out on Prusa i3 MK2S, a commercial desktop 3D printer. As depicted, the camera was in line with the build platform to have a clear view of the printed component.

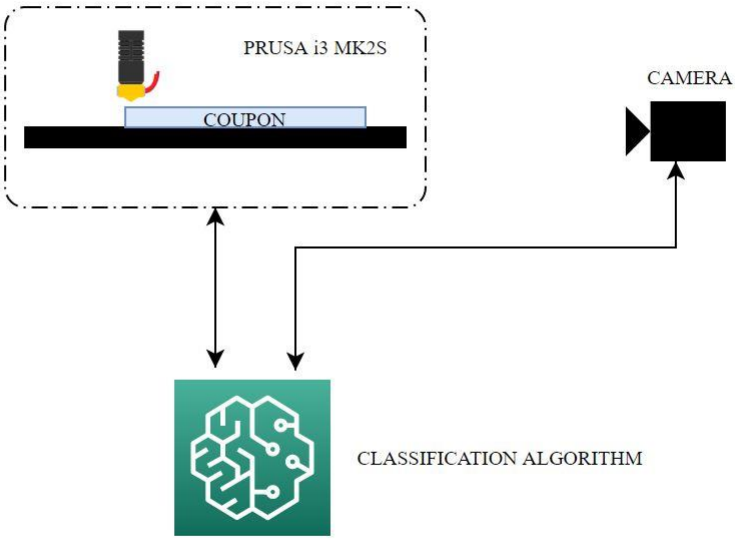


Figure 6: Experimental setup for the proposed closed-loop architecture

The camera and 3D printer were managed via subroutines running on a local computer to form a closed loop architecture (Figure 7).

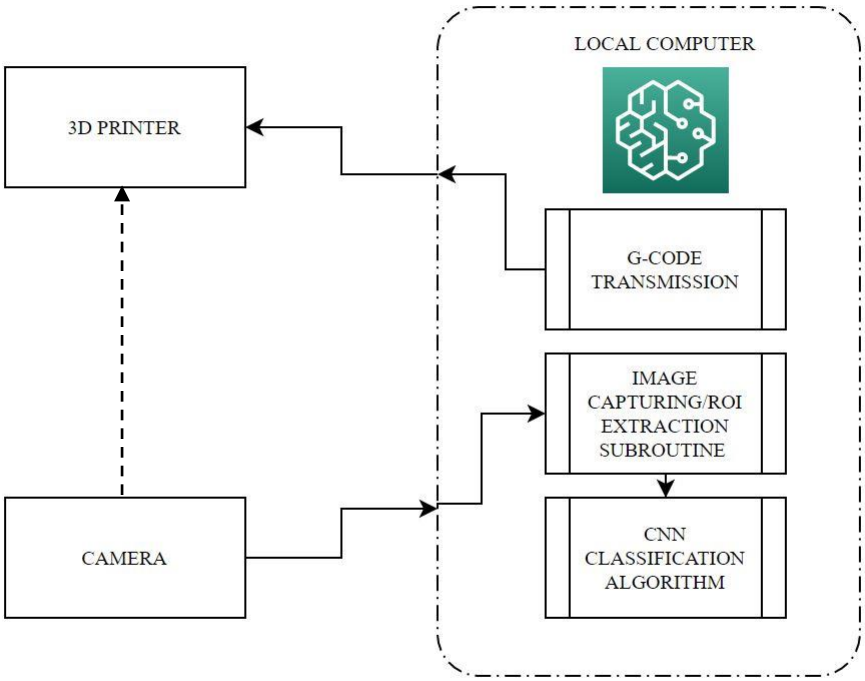


Figure 7: A simplified representation of the dataflow for the proposed closed-loop architecture

The computer was responsible for running three *subroutines*[3] in the background: A G-Code transmission script to interface with the 3D printer via its serial port, an image capturing and processing script and the CNN classification algorithm. The G-Code was altered to pause the printer at the *end of each layer* [4]and resume automatically after two seconds, allowing the extruder to move out of the frame to have an uninterrupted view of the component being printed. An image was then captured and processed to extract, reshape and grayscale a region of interest (ROI) to reduce computational complexity. Next, the ROI

[3] All scripts were written in a Python 3.6 environment and processed on an Intel® Core™ i5-6300U processor @ 2.4 GHz and 8GB memory running on Windows 10 64-bits.

[4] The system was initiated at the end of the second layer to ensure enough material was deposited to distinguish the part from the build-platform.

was analyzed using the classification algorithm to get the probability of an event, $X = unwarped$, where X represents the corner of the part being printed. A $P(X)$ value above or equal to 0.5 indicates the component is unwarped, allowing the print to continue. A $P(X)$ value below 0.5 indicates a warping occurrence, and a signal is sent to pause the printer, thereby creating a closed-loop system.

Figure 8 outlines the image acquisition and preprocessing and classification pipeline.

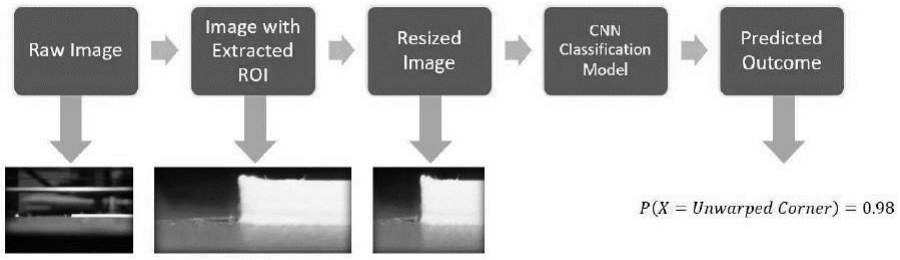


Figure 8: Image acquisition and analysis pipeline for the classification algorithm

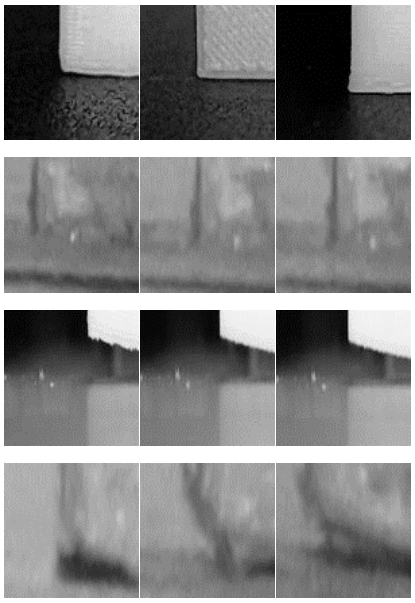
3.2 Dataset Preparation

Two datasets[5] were proposed for this study; the first set contained images taken in an ideal environment, i.e. lighting conditions were closely monitored to warrant clear images. For the second set however, ambient conditions (including lighting conditions and camera position) were altered to obtain grainy images. These images were acquired using the setup described in Section 3.1. Each set consisted of 550 coloureds, 6000 × 4000 pixeled images divided (equally) into two classes. Several 30 mm ×15mm × 5mm cuboids were printed (individually) and recorded layer-by-layer to collect images for unwarped corners. The corners of these cuboids were then peeled from the build platform to imitate warp deformation.

Training a classification algorithm with this dataset would require optimizing 3.96×10^{10} nodes ($550 \times 6000 \times 4000 \times 3$), a computationally expensive process. To improve computational efficiency a region-of-interest (Figure 9) was extracted, greyscale and resized to 100 × 100 pixels by performing nearest-neighbor interpolation. The processed dataset was then one-hot encoded, shuffled and split into training and validation sets in a $^{80}/_{10}$ split, the remaining 10% was used for testing.

[5] The rationale for varying image quality is discussed in Section 4.0.

Figure 9 shows three training samples for each class from both datasets.



(a)

(b)

(c)

(d)

Figure 9: Samples from the training set: (a) unwarped samples from Test Set 1; (b) unwarped samples from Test Set 1; (c) warped samples from Test Set 1; (d) warped samples from Test Set 2

3.3 Building the Network

This section provides details regarding the underlying CNN classification model, including approaches taken for hyperparameter optimization and integration with the overall closed-loop architecture. Depending on computational resources, any ANN can be constructed manually or autonomously. Although simple to program, the manual approach necessitates multiple trials and error to optimize a model and is often time-consuming, and inaccurate. In contrast, an automated approach is suitable for an agile environment but requires a good understanding of black-box optimization. The following sections provide a comparative study of the two approaches and a rationale for choosing one over the other.

3.3.1 Manual Approach

Table 1 presents the final architecture of CM-M1, the CNN classification model constructed via a manual approach.

Table 1: Summarised architecture of CNN classification model CM-M1

Layer	Operator	Kernel Size	Stride	Number of Filters/Nodes/%	Method
LY1 – C1	Convolution	2 × 2	2	8	-
LY2 – P1	Pooling	2 × 2	2	-	Max Pooling
LY3 – C2	Convolution	2 × 2	2	8	-
LY4 – P2	Pooling	2 × 2	2	-	Max Pooling
LY5 – C3	Convolution	2 × 2	2	8	-
LY6 – P3	Pooling	2 × 2	2	-	Max Pooling
LY7 – C4	Convolution	2 × 2	2	16	-
LY8 – P4	Pooling	2 × 2	2	-	Max Pooling
LY9 – C5	Convolution	2 × 2	2	16	-
LY10 – P5	Pooling	2 × 2	2	-	Max Pooling
LY11 – C6	Convolution	2 × 2	2	16	-
LY12 – P6	Pooling	2 × 2	2	-	Max Pooling
LY13 – C7	Convolution	2 × 2	2	24	-
LY14 – P7	Pooling	2 × 2	2	-	Max Pooling
LY15 – C8	Convolution	2 × 2	2	24	-
LY16 – FC1	Fully Connected		Flattened to a vector		
LY17 – DP1	Dropout		60% of Nodes Retained		
LY18 – FC2	Fully Connected	-	-	2	SoftMax (Activation)

CNN typically contains an input, convolution, pooling, fully connected and output layer. The dataset was passed through the convolution layer LY1 – C1, where a kernel of size 2×2 was applied to the input image with a stride of 2. The first layer contained eight learnable kernels producing eight feature maps. These feature maps were then passed through subsequent convolutional layers with additional filters, as outlined in Table 1 – furthermore, even layers from 2 to 14 performed max-pooling of stride 2. The number of layers, filters and stride were selected at random and altered until the model yielded acceptable results. The feature maps were then flattened into a vector and fully connected to the subsequent layer. As the underlying model performs binary classification, the output layer had two nodes, one for each class; completing the basic framework of the classification model.

Next, an optimization algorithm was chosen to minimize the loss function. Figure 10 compares the evolution of loss function with the number of epochs for three optimization algorithms: Root Mean Square (RMS), Stochastic Gradient Descent (SGD) and Adaptive moment estimation (Adam) optimizer. The details of these optimizers are beyond the scope of this paper.

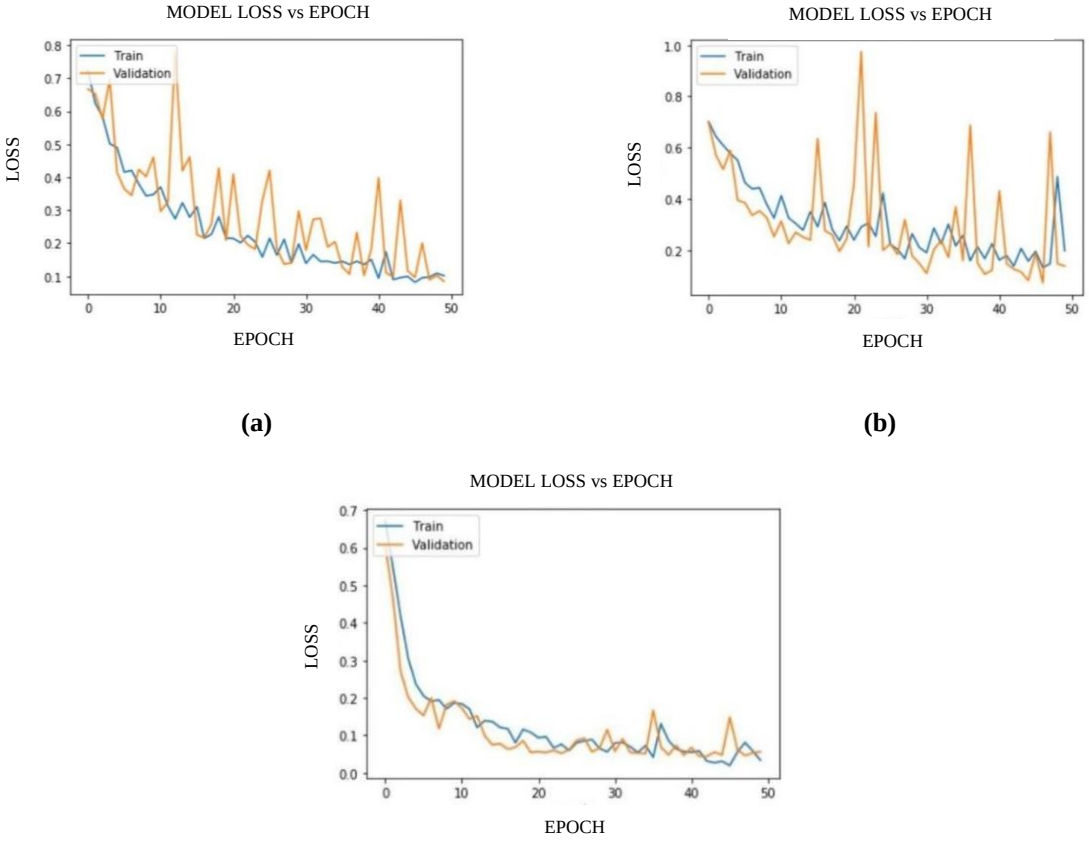
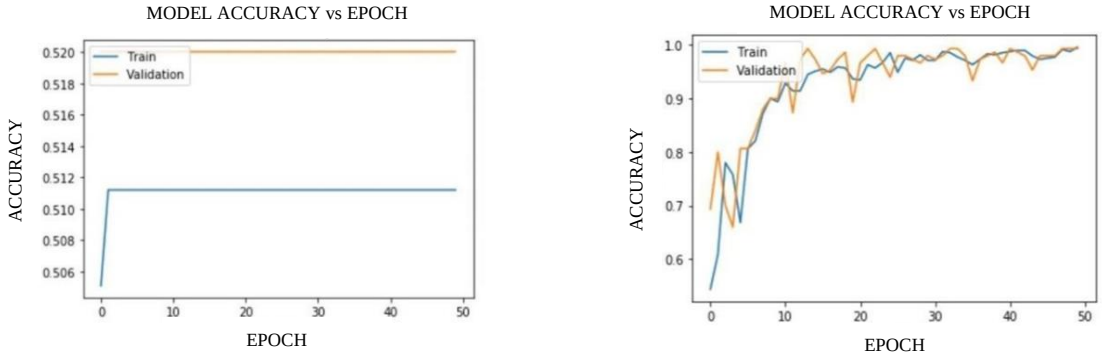


Figure 10: A comparison of loss function decay with the number of epochs minimised by common optimization algorithms: (a) RMS; (b) SGD; (c) Adam

All optimizers performed reasonably well[6]; however, RMS and SGD were prone to substantial fluctuations. In contrast, Adam optimizer minimized the loss function to less than 10% with little fluctuation and was therefore selected.

[6] Although Adam performs well for this dataset, recent publications have shown that the optimizer often fails to converge under specific conditions. In hindsight, all optimizers would have yielded similar results given the optimal number of epochs, learning rate and mini-batch size. Therefore, any optimizer could have been employed for this investigation.

For reasons explained in Section 2.2.2, each layer requires an activation function. Based on existing literature, two activation functions were considered ReLU and Leaky ReLU. Their performance was evaluated by training a model for 50 epochs. Epochs are essential to the generalizability of the model and must be selected carefully. Callback addresses this issue by evaluating the internal state of any machine learning algorithm to prevent a model from training further once the accuracy plateaus. The training and validation accuracy for both activation functions were then graphed and compared.

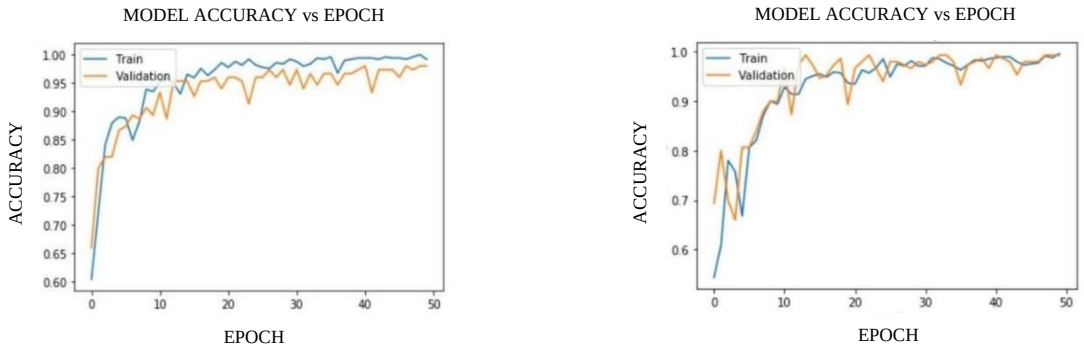


(a) **(b)**

Figure 11: Training and validation accuracy of the model with different activation functions: (a) ReLU; (b) Leaky ReLU

ReLU provides a training and validation accuracy of 0.520 and 0.511, respectively. The poor performance could be attributed to the "dying ReLU problem" described in Section 2.2.2; consequently, Leaky ReLU was chosen.

Figure 12 indicates a discrepancy between training and validation accuracy, suggesting overfitting. Overfitting is commonly observed in CNN models and is corrected by adding a dropout layer. A dropout layer randomly discards nodes to allow a model to learn from all the inputs instead of relying on a few inputs regularizing the neural network.



(a) **(b)**

Figure 12: Training and validation accuracy with the number of epochs: (a) without dropout; (b) with dropout

The approach may seem promising at first glance, yielding an accuracy of 0.98; however, the number of manual iterations exposes the inherent vulnerabilities with this method. The subsequent section proposes an alternative approach.

3.3.2 Automated Approach

Figure 13 outlines the high-level algorithm utilized for an automated approach.

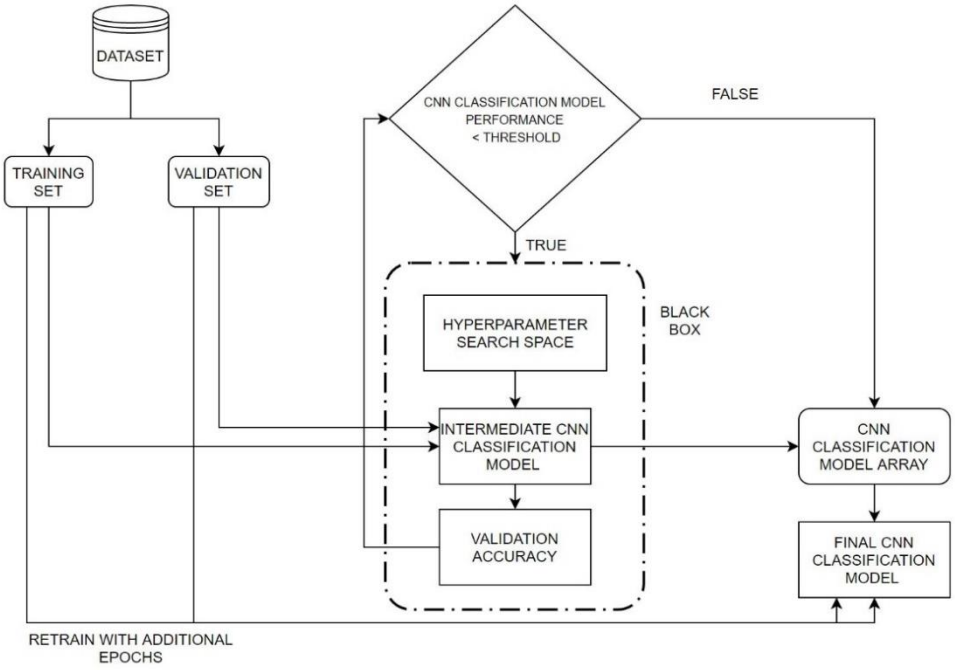
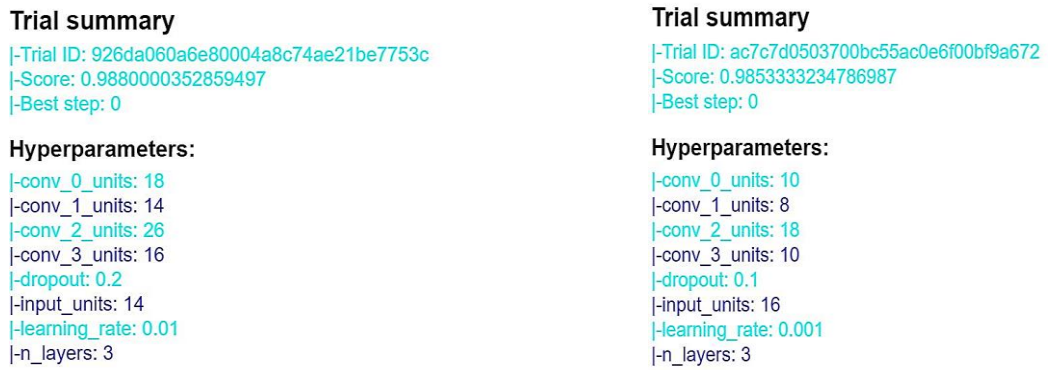


Figure 13: A simplified representation of the underlying algorithm utilized for the automated approach

The algorithm involved repeating an optimization loop until a certain threshold was met. In each iteration, hyperparameters were selected from a hyperparameter configuration space defined by the user. The configuration space for this study contained the total number of convolution layers (including stride length and kernel size), the number of units (neurons) and feature maps in each layer; the optimal dropout and learning rate; and the activation function for each layer. An intermediate classification model was then created and trained using the selected hyperparameters. The validation accuracy was recorded and utilized for estimating hyperparameters for subsequent iterations. The models were stored in an array, and the one with the highest score (validation accuracy) was *serialized*[7] for later use.

[7] Here, serialization refers to the (OOP language) process of converting an Object (stored model) into a transportable form for later use.

Figure 14 shows the best classification model for each dataset proposed by the optimization algorithm.



(a) **(b)**

Figure 14: Best hyperparameter values selected by the optimization algorithm for each dataset: (a) CM-A1; (b) CM-A2

CM-A1 was trained on the first dataset, whereas CM-A2 was trained on both datasets. Like, the previous approach, both models had a validation accuracy of 0.98. Once serialized, the models were retrained on additional epochs, yielding an accuracy of 0.991 and 0.996, respectively. The number of epochs was limited by implementing early stopping, as defined in the previous section. The serialized architecture for CM-A1 and CM-A2 is summarised in Tables 2 and 3 respectively.

Table 2: Serialized architecture of CNN classification model CM-A1

Layer	Operator	Kernel Size	Stride	Number of Filters/Nodes/%	Method/Activation
LY1 – C1	Convolution	2 × 2	2	18	ReLU
LY2 – P1	Pooling	2 × 2	2	-	Max Pooling
LY3 – C2	Convolution	2 × 2	2	14	ReLU
LY4 – C3	Convolution	2 × 2	2	26	ReLU
LY5 – C4	Convolution	2 × 2	2	16	ReLU
LY6 – F1	Fully Connected		Flattened to a Vector		
LY7 – DP1	Dropout		80% of Nodes Retained		
LY8 – FC2	Fully Connected	-	-	2	SoftMax (Activation)

Table 3: Serialized architecture of CNN classification model CM-A2

Layer	Operator	Kernel Size	Stride	Number of Filters/Nodes/%	Method/Activation
LY1 – C1	Convolution	2 × 2	2	10	ReLU
LY2 – P1	Pooling	2 × 2	2	-	Max Pooling
LY3 – C2	Convolution	2 × 2	2	8	ReLU
LY4 – C3	Convolution	2 × 2	2	18	ReLU
LY5 – C4	Convolution	2 × 2	2	10	ReLU
LY6 – F1	Fully Connected		Flattened to a Vector		
LY7 – DP1	Dropout		90% of Nodes Retained		
LY8 – FC2	Fully Connected	-	-	2	SoftMax (Activation)

In comparison to CM-M1, both CM-A1 and CM-A2 had fewer hidden layers; however, they had additional kernels per convolutional layer. This arrangement hypothetically allows the convolution layer to extract additional low-level features from the input image.

4.0 Results and Discussion

Figure 15 outlines the procedure undertaken to evaluate the performance of two machine learning architectures.

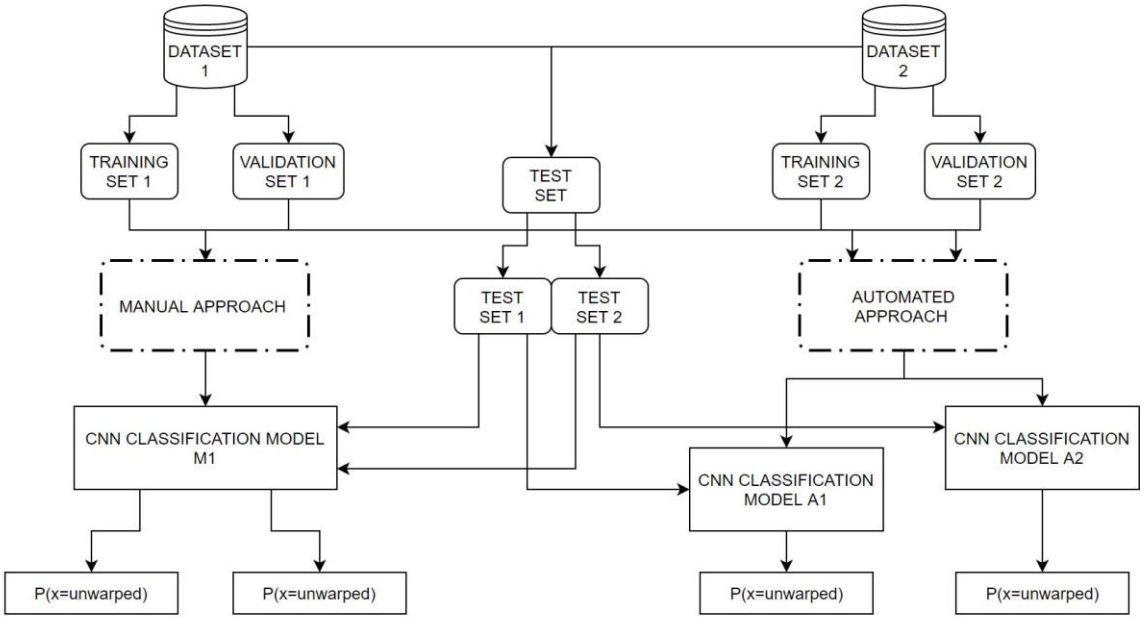


Figure 15: A representation of the test-plan for evaluating the performance of the proposed approaches

CNN models are sensitive to the quality of input images; for example, features are less prominent in darker images and reduce the performance of the system. This can be addressed by retraining the underlying model with additional images. This was evaluated by employing two datasets; Test Set 1 represented an ideal condition wherein ambient light condition was closely monitored, while Test Set 2, contained a combination of clear and grainy images to simulate varying lighting conditions. CM-M1 was tested on both test sets, while CM-A1 and CM-A2 were tested on Test Set 1 and 2, respectively. This setup allowed to form a comparative study of the approaches taken to build the underlying CNN classification model.

Figure 16 depicts a specimen printed following the lighting conditions specified for the first dataset.

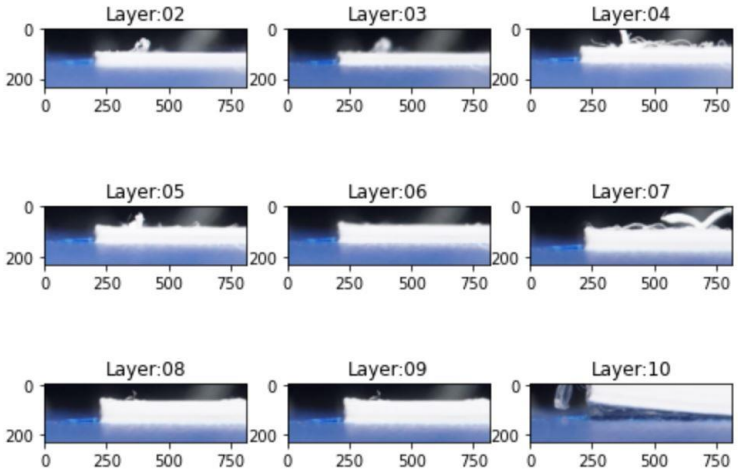


Figure 16: A subplot of different layers of a sample print showing warping in Layer 10

Table 4 present the predicted output for a specimen depicted in Figure 16.

Table 4: The predicted output by CM-M1 and CM-A1 for a specimen printing

CM-M1		**CM-A1**	
Layer #	$P(X = Unwarped\ Corner)$	**Layer #**	$P(X = Unwarped\ Corner)$
2	0.5786	**2**	0.5641
3	0.8349	**3**	0.8115
4	0.7567	**4**	0.7912
5	0.8849	**5**	0.9147
6	0.9543	**6**	0.9684
7	0.9384	**7**	0.9588
8	0.9848	**8**	0.9975
9	0.9932	**9**	0.9998
10	0.0004	**10**	0.0008

Both models were able to classify the printed layers correctly. Moreover, a similar pattern was observed in the predicted outcome. For instance, there was a dip in $P(X)$ for layers four and seven due to stringing. This indicates that both models were sensitive to slight variations in input data and could benefit from regularization. Although CM-M1 performs well in this test, it must be noted that the manual approach took several hours to optimize. In contrast, the automated approach produced CM-A1 and retrained the model with additional epochs in under 15 minutes. Besides, CM-M1 took additional hidden layers to have a similar performance as CM-A1 proving to be computationally inefficient.

The test was then repeated on the second dataset. Table 5 presents the predicted output for a specimen depicted in Figure 17.

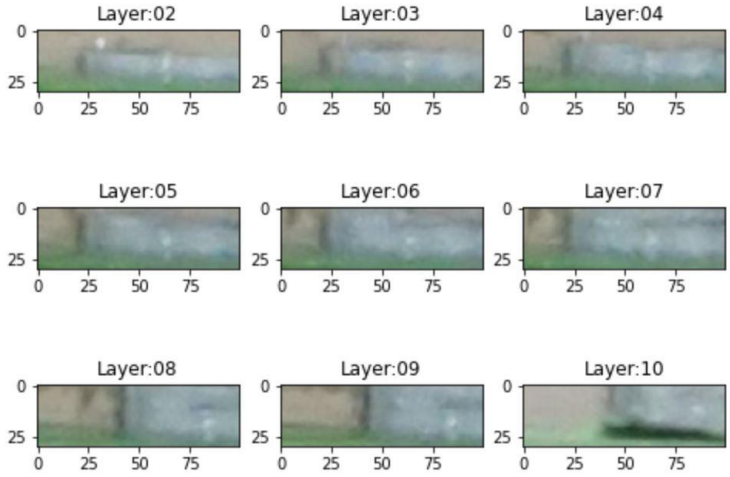


Figure 17: A subplot of different layers of a sample print showing warping in Layer 10

Table 5: The predicted output by CM-M1 and CM-A2 for a specimen printing

CM-M1		**CM-A2**	
Layer #	$P(X = Unwarped\ Corner)$	**Layer #**	$P(X = Unwarped\ Corner)$
2	0.3167	**2**	0.8123
3	0.4812	**3**	0.9543
4	0.4154	**4**	0.9671
5	0.3329	**5**	0.9785
6	0.3864	**6**	0.9792
7	0.4215	**7**	0.9872
8	0.6623	**8**	0.9976
9	0.6349	**9**	0.9983
10	0.0136	**10**	0.0003

As expected, CM-M1 performed poorly in the second test. This is because the hyperparameters were chosen for a specific dataset, and any variations in the test data would affect the accuracy of the underlying model.

As the models are stochastic in nature, the tests were repeated twice to validate the results.

Table 6: Test result summary

Model	Dataset Number	Test Number	Number of Layers	Number of Unwarped Layers	Number of Warped Layers	Layers Correctly Identified	Mean Accuracy
CM-M1	1	1	15	15	0	15	1
		2	15	14	1	15	
		3[8]	9	8	1	9	
	2	1	15	15	0	6	0.36
		2	15	14	1	5	
		3	9	8	1	3	
CM-A1	1	1	15	15	0	15	1
		2	15	14	1	15	
		3	9	8	1	9	
	2	1	15	15	0	7	0.46
		2	15	14	1	9	
		3	9	8	1	2	
CM-A2	1	1	15	15	0	15	1
		2	15	14	1	15	
		3	9	8	1	9	
	2	1	15	15	0	15	1
		2	15	14	1	15	
		3	9	8	1	9	

Overall, the models were able to classify all layers in Dataset 1 correctly. In contrast, CM-M1 and CM-A1 performed poorly with an average accuracy of 0.36 and 0.46, respectively. As CM-A2 was trained on both datasets, the model performed well, having an accuracy of 1. Although the manual approach took longer to optimize, it failed to generalize the problem proving to be inaccurate, time-consuming and computationally inefficient. Theoretically, deeper networks are better at abstracting features leading to a more comprehensive understanding of the input image. The redundant pooling layers and lack of kernels, however, prevented CM-M1 from learning low-level features, as evident from the results. The other approach yielded a much shallower yet efficient model in under 15 minutes, highlighting the impact of hyperparameters and the prowess of black-box optimization algorithms.

[8] The third test for each dataset is depicted in Figures 16 and 17, respectively.

The dataset for this study is relatively small, allowing models to be trained in under an hour. In an industrial environment, however, datasets are complex, with much longer training times necessitating an automated approach. With an increase in hardware accelerators, computational costs are now inconsequential in comparison to the time saved and the accuracy achieved by using black-box optimization algorithms, thus validating the implementation of an automated system.

5.0 Conclusion

The objective of the study was to present the application of Convolutional Neural Networks in additive manufacturing. Fused Filament Fabrication is a commonly used additive manufacturing technique that often suffers from defects, including warp deformation compromising the structural integrity of the component being printed and, in extreme cases damaging the printer. Therefore, a closed-loop in-process monitoring architecture was proposed to pause a print on the onset of warp deformation.

The architecture contained three significant components running in the background, namely a G-code transmission system, image acquisition and processing system, and a CNN classification model. Any Artificial Neural Network, including CNNs, depend on their hyperparameters. Typically, hyperparameters can be optimized using a manual or automated approach. A manual approach, although easier to program, requires multiple trials and error and is often time-consuming, inaccurate and computationally inefficient. In contrast, an automated approach selects an optimal model through black-box optimization algorithms. This thesis utilized a Bayesian-based optimizer to develop an autonomous architecture capable of producing and training models, depending on the problem at hand.

To evaluate the performance of the automated approach and form a comparative study, two datasets were proposed. The first set contained images taken in an ideal environment the second set, however, consisted of a combination of bright, blurry and grainy images to simulate varying lighting conditions. The manual approach produced the CM-M1 classification model, while the automated approach yielded CM-A1 and CM-A2. CM-M1 and CM-A1 were trained on the first dataset while CM-A2 was trained on both datasets. As expected, all datasets performed well on the first set; however, CM-M1 and CM-A1 poorly on Test Set 2, having an accuracy of 0.36 and 0.53, respectively. CM-M1 was tested on the second dataset to highlight the inherent vulnerabilities of this approach. An automated approach is thus required to ensure that models are optimized in an agile manufacturing environment.

In the future, efforts will be made to populate the existing dataset with additional defects to increase the practicality of this system.

6.0 References

1. Wang Y, Chen T, Yeh Y. Advanced 3D printing technologies for the aircraft industry: a fuzzy systematic approach for assessing the critical factors. The International Journal of Advanced Manufacturing Technology. 2018;105(10):4059-4069. doi:10.1007/s00170-018-1927-8
2. 5. Joshi S, Sheikh A. 3D printing in aerospace and its long-term sustainability. Virtual Phys Prototyp. 2015;10(4):175-185. doi:10.1080/17452759.2015.1111519
3. M. S. Alsoufi, M. W. Alhazmi, D. K. Suker, W. K. Hafiz, S. S. Almalki, and R. O. Malibari, "Influence of Multi-Level Printing Process Parameters on 3D Printed Parts in Fused Deposition Molding of Poly(lactic) Acid Plus: A Comprehensive Investigation," *American Journal of Mechanical Engineering*, vol. 7, no. 2, pp. 87–106, 2019.
4. D. Dimitrov, W. V. Wijck, K. Schreve, and N. D. Beer, "Investigating the achievable accuracy of three dimensional printing," *Rapid Prototyping Journal*, vol. 12, no. 1, pp. 42–52, 2006.
5. I. Gibson, B. Stucker, and D. W. Rosen, *Additive manufacturing technologies: rapid prototyping to direct digital manufacturing*. New York: Springer, 2010.
6. W. Gao, Y. Zhang, D. Ramanujan, K. Ramani, Y. Chen, C. B. William, C. Y. Wang, Y. C. Shin, S. Zhang, and P. D. Zavattieri, "The status, challenges, and future of additive manufacturing in engineering," *Computer Aided Design*, pp. 65–89, 2015.
7. R. T. L. Ferreira, I. C. Amatte, T. A. Dutra, and D. Bürger, "Experimental characterization and micrography of 3D printed PLA and PLA reinforced with short carbon fibers," *Composites Part B: Engineering*, vol. 124, pp. 88–100, 2017.
8. T.-M. Wang, J.-T. Xi, and Y. Jin, "A model research for prototype warp deformation in the FDM process," *The International Journal of Advanced Manufacturing Technology*, vol. 33, no. 11-12, pp. 1087–1096, 2006.
9. A. Armillotta, "Assessment of surface quality on textured FDM prototypes," *Rapid Prototyping Journal*, vol. 12, no. 1, pp. 35–41, 2006.
10. A. Peng and X. Xiao, "Investigation on Reasons Inducing Error and Measures Improving Accuracy in Fused Deposition Modeling," *INTERNATIONAL JOURNAL ON Advances in Information Sciences and Service Sciences*, vol. 4, no. 5, pp. 149–157, 2012.
11. A. Guerrero-De-Mier, M. Espinosa, and M. Domínguez, "Bricking: A New Slicing Method to Reduce Warping," *Procedia Engineering*, vol. 132, pp. 126–131, 2015.

12. E. R. Fitzharris, N. Watanabe, D. W. Rosen, and M. L. Shofner, “Effects of material properties on warpage in fused deposition modeling parts,” *The International Journal of Advanced Manufacturing Technology*, vol. 95, no. 5-8, pp. 2059–2070, 2017.

13. M. S. Alsoufi and A. E. Elsayed, “Warping Deformation of Desktop 3D Printed Parts Manufactured by Open Source Fused Deposition Modeling (FDM) System,” *International Journal of Mechanical & Mechatronics Engineering*, 2016.

14. A. Armillotta, M. Bellotti, and M. Cavallaro, “Warpage of FDM parts: Experimental tests and analytic model,” *Robotics and Computer-Integrated Manufacturing*, vol. 50, pp. 140–152, 2018.

15. Q. Y. Lu and C. H. Wong, “Additive manufacturing process monitoring and control by non-destructive testing techniques: challenges and in-process monitoring,” *Virtual and Physical Prototyping*, vol. 13, no. 2, pp. 39–48, 2017.

16. Y. Li, W. Zhao, Q. Li, T. Wang, and G. Wang, “In-Situ Monitoring and Diagnosing for Fused Filament Fabrication Process Based on Vibration Sensors,” *Sensors*, vol. 19, no. 11, p. 2589, 2019.

17. F. Baumann and D. Roller, “Vision based error detection for 3D printing processes,” *MATEC Web of Conferences*, vol. 59, p. 06003, 2016.

18. Z. Zhang, G. Wen and S. Chen, "Weld image deep learning-based on-line defects detection using convolutional neural networks for Al alloy in robotic arc welding", *Journal of Manufacturing Processes*, vol. 45, pp. 208-216, 2019. Available: 10.1016/j.jmapro.2019.06.023.

19. B. Zhang, P. Jaiswal, R. Rai, P. Guerrier and G. Baggs, "Convolutional neural network-based inspection of metal additive manufacturing parts", *Rapid Prototyping Journal*, vol. 25, no. 3, pp. 530-540, 2019. Available: 10.1108/rpj-04-2018-0096.

20. 2. Yamashita R, Nishio M, Do R, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights Imaging. 2018;9(4):611-629. doi:10.1007/s13244-018-0639-9

21. Feurer M, Hutter F. Hyperparameter Optimization. Automated Machine Learning. 2019:3-33. doi:10.1007/978-3-030-05318-5_1

22. Enyinna Nwankpa C, Ijomah W, Gachagan A, Marshall S. Activation Functions: Comparison Of Trends In Practice And Research For Deep Learning.; 2018.

23. W. Yi, H. Ketai, Z. Xiaomin, and D. Wenying, “Machine vision based statistical process control in fused deposition modeling,” 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2017.

24. Delli U, Chang S. Automated process monitoring in 3D printing using supervised machine learning. Procedia Manufacturing. 2018;26:865-70.

25. Wu M, Phoha VV, Moon YB, Belman AK. Detecting malicious defects in 3d printing process using machine learning and image classification. ASME 2016 International Mechanical Engineering Congress and Exposition: American Society of Mechanical Engineers Digital Collection; 2016.
26. T. S. Srivatsan and T. S. Sudarshan, *Additive manufacturing: innovations, advances, and applications*. Boca Raton: CRC Press, 2016.
27. What Is a Neural Network?. Mathworks.com. https://www.mathworks.com/discovery/neural-network.html. Accessed April 12, 2020.
28. Raschka S. What is Softmax Regression and How is it Related to Logistic Regression? - KDnuggets. KDnuggets. https://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html. Published 2020. Accessed April 12, 2020.
29. Jalali A, Azimi J, Fern X, Zhang R. A Lipschitz Exploration-Exploitation Scheme for Bayesian Optimization. Machine Learning and Knowledge Discovery in Databases. 2013:210-224. doi:10.1007/978-3-642-40988-2_14
30. A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in ISVC, pages 867-877, 2015

Ryerson University Library
NON-EXCLUSIVE LICENSE

Family name: SALUJA Given name: ADITYA

Title of paper: The Implementation of Convolutional Neural Networks for Warp Detection in 3D Printed Components manufactured via Fused Filament Fabrication: A Bayesian-Based Automated Approach

In consideration of Ryerson University Library making my paper available to interested persons,
I, Aditya Saluja

hereby grant a non-exclusive license, for the full term of copyright protection, to Ryerson University:

to preserve, perform, produce, reproduce, translate in any format, and to make available in print or online by telecommunication to the public for non-commercial purposes.

I undertake to submit my paper to the Ryerson University Library. Any abstract submitted with the paper will be considered to form part of the paper.

I represent and promise that my paper is my original work, does not infringe any rights of others, and that I have the right to make the grant conferred by this non-exclusive license.

If third-party copyrighted material was included in my paper for which, under the terms of the Copyright Act, written permission from the copyright owners is required I have obtained such permission from the copyright owners to do the acts mentioned in paragraph (a) above for the full term of copyright protection.

I retain copyright ownership and moral rights in my paper, and may deal with the copyright in my paper, in any way consistent with the rights granted by me to The Ryerson Library in this non-exclusive license.

Aditya Saluja 04/13/2020

Signature **Date**

Ryerson University Library
Get connected. Be transformed