DEEP LEARNING APPROACH FOR CONTROLLING ADDITIVE MANUFACTURING PROCESS

Thesis

Submitted to

School of Computer Science, College of Science and Engineering

Southern University and A&M College

In Partial Fulfillment of the Requirements for

The Degree of

Master of Science in Computer Science

By

Nariman Razavi Arab

Baton Rouge, Louisiana

December 2019

DEEP LEARNING APPROACH FOR CONTROLLING ADDITIVE MANUFACTURING PROCESS

Name: Razavi Arab, Nariman

APPROVED BY:

Yaser Banadaki, 11/4/2019

Yaser Banadaki, Ph.D.

Committee Chair

11/5/19

Ebrahim Khosravi, Ph.D.

Committee Member

11/4/19

Shizhong Yang, Ph.D.

Committee Member

11/5/19

Ebrahim Khosravi, Ph.D.

Chair of Department, Computer Science, Southern University

11/6/19

Habib P. Mohamadian, Ph.D.

Dean of Graduate School, Southern University

ABSTRACT

Name: Razavi Arab, Nariman

Southern University and A&M College

Advisor: Dr. Yaser Banadaki

Additive manufacturing (AM) is a crucial component of smart manufacturing systems that disrupts traditional supply chains. However, the parts built using the state-of-the-art powder-bed 3D printers have noticeable unpredictable mechanical properties. The attempts to improve the geometry and mechanical properties of final products in the 3D printing process were usually limited to the development of a typical control system using feedback from sensor measurement. However, a smarter control system is required to learn and adapt as the AM machine is operating.

In this thesis, I propose a machine learning (ML) algorithm as a promising way of improving the underlying failure phenomena in 3D printing. I employ a Deep Convolutional Neural Network (DCNN) to automatically detect the defects in printing the layers, thereby turning 3D printers into essentially their own inspectors.

We expect that the proposed DCNN model generates a precise feedback signal for a smart 3D printer to recognize any issues with the build itself to make proper adjustments and corrections without operator intervention. This can enhance the quality of the AM process, leading to manufacturing better parts with fewer quality hiccups, limiting waste of time and materials.

ACKNOWLEDGMENTS

This thesis would not have been possible without the constant support of my family, professors and committee members, and friends. Firstly, I would like to express my gratitude to my committee chair Dr. Yaser Banadaki who encouraged me to do this research for the master's thesis. His guidance and constant advice helped me a lot throughout this research. I would also like to thank the rest of my thesis committee: Dr. Ebrahim Khosravi, and Dr. Shizhong Yang for their assistance and suggestions.

I also wish to acknowledge the encouragement and support offered by faculties, students, and the Department of Computer Science while completing this work.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1. Background

Industry 4.0 (factory of the future, smart factory, smart manufacturing, cloud-based manufacturing (CBM), or digital manufacturing) [Figure 1], is the stage of industrial revolution that machines and robots provide a high automation level with the ability to process information, enhance the yield of production [1], [2] visualize the performance in real-time [3], enable intelligent predictive maintenance system [4], and match service providers with customer demands [5].

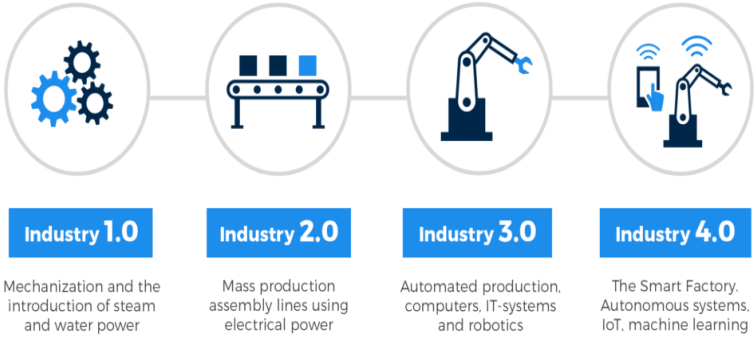

Figure 1: The Four Industrial Revolutions

Source: [6].

Additive Manufacturing (AM) is a crucial component of the smart manufacturing system [Figure 2] to enable flexible configuration and dynamically changing processes [7] to quickly adapt the products to new demands beyond traditional supply chains.

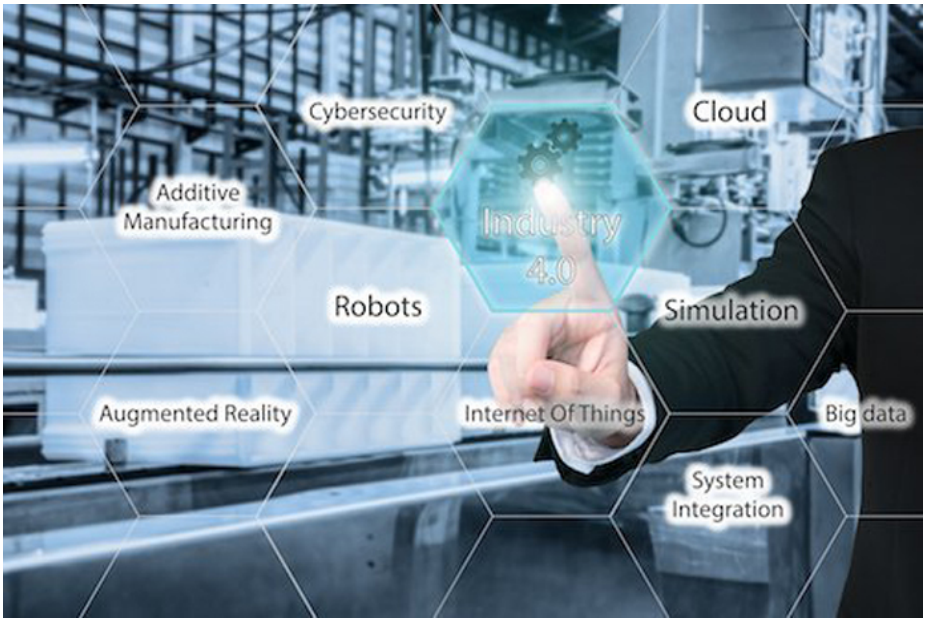

Figure 2: Industry 4.0 among other cutting-edge research fields

Source: [1], [3].

Additive manufacturing (AM) has tremendous potential to make a custom-designed part on-demand and with minimal material, but it is hampered by poor process reliability and throughput due to lack of the condition-awareness of the AM process and automation. The parts built using current state-of-the-art AM machines have noticeable unpredictable mechanical properties. It is a difficult task to build a process–structure–property–performance (PSPP) relationship for AM using traditional numerical and analytical models.

Today, the machine learning (ML) method has been demonstrated to be a valid way to perform complex pattern recognition and regression analysis without an explicit need to construct and solve the underlying physical models. Among ML algorithms, the neural network (NN) is the most widely used model due to the large dataset that is currently available, strong computational power, and sophisticated algorithm architecture [8].

1.2. Significance

Currently, almost all AM machines have only limited sensing capabilities that are mostly inaccessible to the users, or completely "open-loop" system operating without any feedback measurement systems for correction during the process.

However, future AM machines will be a smart system that can perform self-monitoring, self-calibrating, and quality self-controlling in real-time (Figure 3).

The gap between the smart factory and existing manufacturing systems must be bridged with respect to the automation, flexibility, and reconfigurability of AM machines in a computer-integrated manufacturing system.

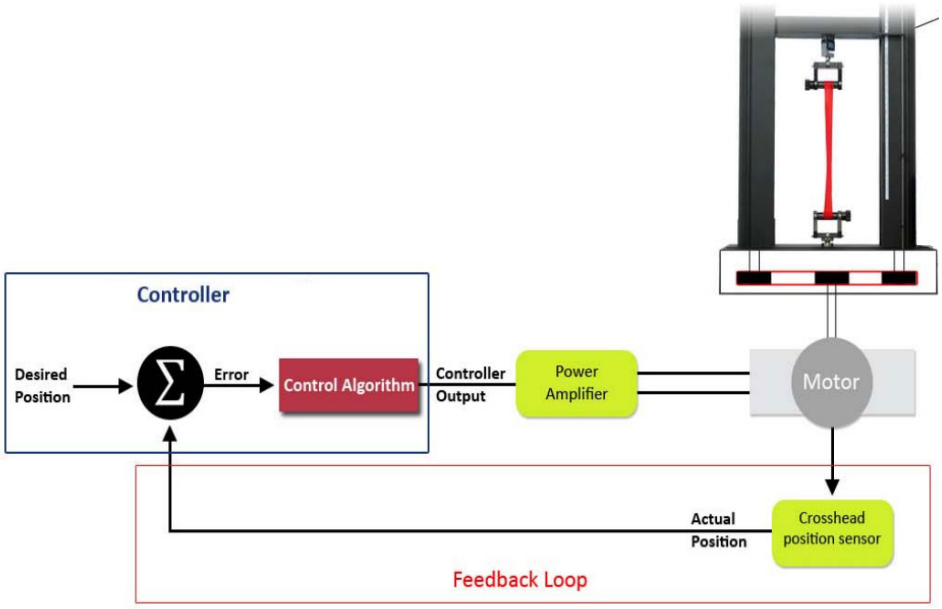

Figure 3: open loop vs closed loop systems materials testing industry

Source: [9].

1.3. Statement of Problem

Additive manufacturing (AM) as an all-important part of smart manufacturing systems that disrupts traditional supply chains. However, the parts built using the state-of the-art powder-bed 3D printers (Figure 4) have noticeable unpredictable mechanical properties.

The attempts to improve the geometry and mechanical properties of final products in the 3D printing process were usually limited to the development of a typical control system using feedback from sensor measurement.

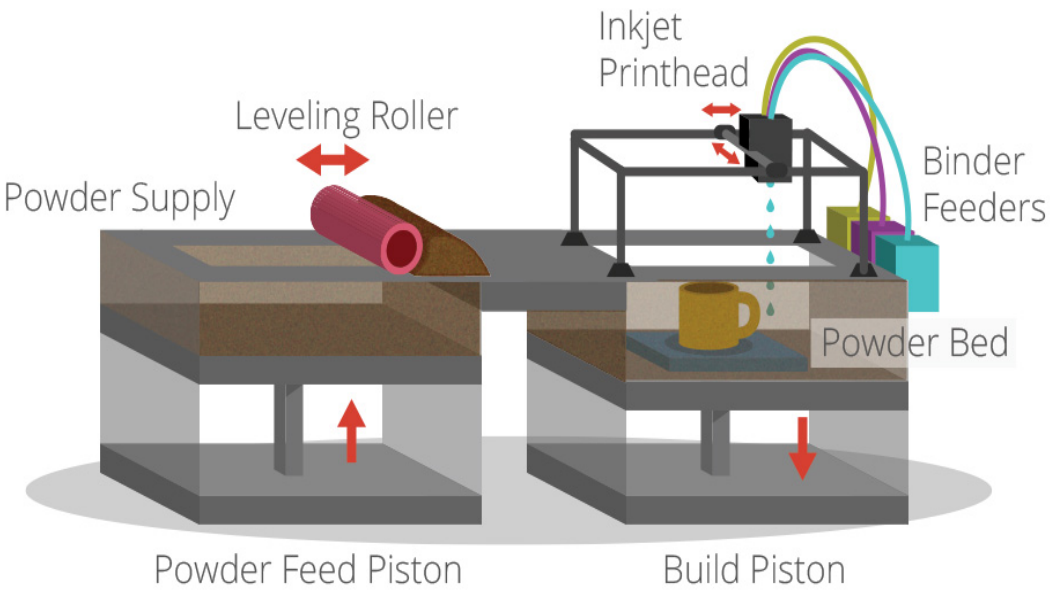

Figure 4: binder jetting 3d printing technology

Source: [10].

However, a smarter control system is required to learn and adapt as AM machine is operating, thereby turning 3D printers into essentially their own inspectors to recognize any issues with the build itself and make proper adjustments and corrections without operator intervention.

This enhances the quality of the AM process, leading to manufacturing better parts with fewer quality hiccups, limiting waste of time and materials.

1.4. Hypothesis

Employing a deep learning-based computer vision system for controlling AM machines will generate a precise feedback signal for a smart 3D printer to recognize any issues.

1.5. Null Hypothesis

Deep learning and computer vision doesn't help to improve the AM process, by recognizing issues in the 3D printing process.

1.6. Research Questions

- What types of feedback could be received during the AM process?
- How can one use computer vision for receiving feedback in the AM process?
- How deep learning and neural networks help to analyze collected image data from the AM process?
- How the parameters of the AM machine can affect the quality of printed products?

1.7. Delimitations

In this research, I applied a deep learning algorithm to train the image processing and classification model for the AM process. The input datasets are generated by collecting images from the all printing layers of 3D objects.

For the plastic 3D printer that has been used as a proof of concept instead of a metal 3D printer in this research, the quality of images from the surface of

objects is limited to plastic material characteristics. The small number of objects are generated and adjusted for the training process.

CHAPTER 2

REVIEW OF LITERATURE

2.1. Smart manufacturing system

Today's challenges for companies and organizations with different operations are real-time access to data and processes, as well as online insights from stakeholders, products, services, and properties. Industry 4.0 can be employed to address these requirements and needs. Industry 4.0 sometimes referred to as smart manufacturing system, has nine building blocks which are technology trends integrated to realize and build up this revolution [Figure 5].

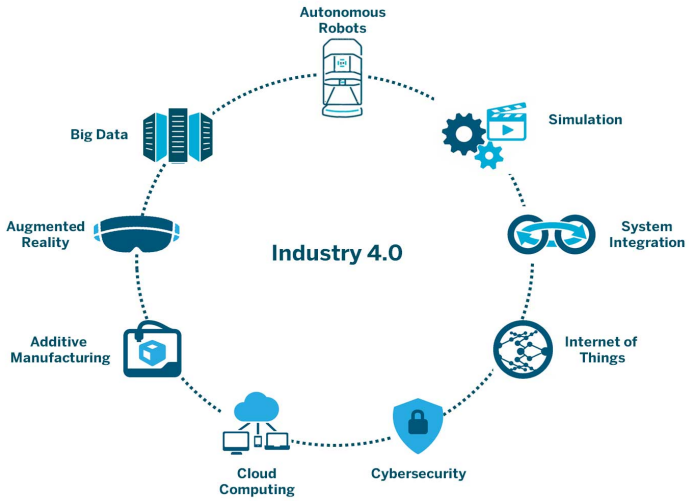

Figure 5: Industry 4.0 building blocks

Source: [11]

2.2. Additive Manufacturing Technologies

Many of the applications best suited for Additive Manufacturing (AM) (e.g. aerospace, biomedical, energy, automotive, and tooling) require levels of part quality assurance and process reliability that are difficult to attain with currently available commercial systems.

In-situ process monitoring and closed-loop control schema can play a critical role in addressing these requirements. In-situ process monitoring of printing builds has become a major research focus for the AM community over the last several years [12].

There are 7 main types of AM technologies defined in the Standard F2792 produced by the American Society for Testing and Materials (ASTM) comprising

the technologies such as “fused deposition modeling”, “ digital light processing”, “stereo-lithography”, “ink-jet and multi-jet printing”, and “selective laser sintering”. Details of the main AM process types are shown in Fig. 6 [13].

Process Type		Description	Related Technologies
#1	Material Extrusion	Material is selectively dispensed through a nozzle or ban orifice	Fused deposition modelling
#2	Material Jetting	Droplets of build material are selectively deposited	Ink-jet printing Multi-jet-modelling
#3	Powder bed fusion	Thermal energy selectively fuses regions of powder bed	Selective laser sintering (SLS) Electron beam melting (EBM)
#4	Directed energy deposition	Focused thermal energy used to fuse materials by melting as material is deposited	Laser metal deposition (LMD)
#5	Vat Photopolymerisation	Liquid photopolymer in a vat is selectively cured by light-activated polymerisation	Stereolithography (SLA) Digital Light Processing (DLP)
#6	Binder jetting	Focused thermal energy used to fuse materials by melting as material is deposited	Powder bed and Ink-jet head (PHID) Plaster-based 3D printing (PP)
#7	Sheet Lamination	Sheets of a material are bonded to form 3D-part	Laminated object Modelling (LOM) Ultrasonic consolidation (UC)

Figure 6: Additive manufacturing process types

Source: [13] .

The materials have different advantages and disadvantages in AM technologies such as “speed of fabrication”, “dimensions”, and “tolerances” that can be processed and achieved during the AM process. In 3D printing and layer-by-layer building of the physical the product, the dimensional/shape accuracy of the printed objects and how much they are compatible with required specifications is the most important quality control feature [14].

Numerous applications have been used fuzzy systems, genetic algorithms, and artificial neural networks (ANNs) as a computational intelligence technique

for predicting the quality but so far a limited use for 3D printing processes have done [14].

2.3. Additive Manufacturing and Machine Learning

Currently, a few companies use additive manufacturing for large-scale production of high-quality industrial parts and the production of finished goods using AM machine is minuscule. Smart AM machine enables scaling down the AM process and improves the geometry and the mechanical properties as shown in Figure 7. AM machines empowered by the ML model can fabricate miniaturized builds with complex geometry and difficulty in fabricating by manufacturing processes such as the subtractive or deformation-based [15]. For instance, hollow objects with thin-shell structures seem more suitable for hybrid fabrication because AM processes can overcome accessibility limitations to the internal features. The structural nodes that all support the same weight, but the part on the right that manufactured by 3D printing and ML algorithms, weight 75% less and is 50% smaller than the original part on the left.

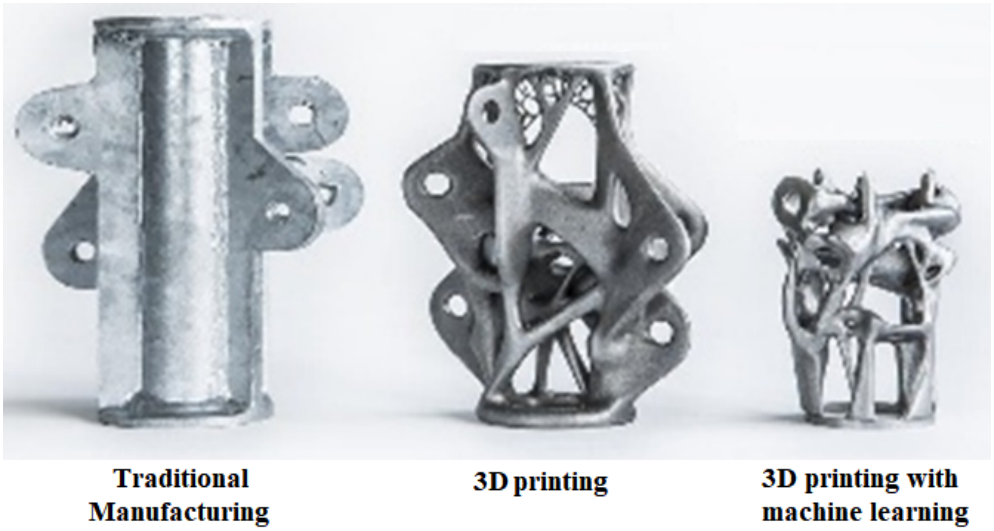

Figure 7: The structural nodes

Source: [16].

2.4. Related Works

Machine learning can play an important role to create a multi-level of predictive models for AM data to the machine operation. However, a few ML models have been explored in existing studies and the application of ML algorithm for improving the AM process has been limited to powder quality, specific processes, and applications. For instance:

- Chen et. al [17] developed an ML model to optimize the parameters of a Binder Jetting (BJ) process without a comprehensive understanding of why the ML model would work better for some particular processes. Some papers are only focused on controlling powder quality using ML models.

- Zhang et. al [3] trained an ML model using computational data of the Discrete Element Method to introduce a powder spreading map for the metal AM process.
- Decost et. al [18] used computer vision and ML methods to characterize, compare, and analyze powder feedstock materials and micrographs for metal AM.
- Stoyanov et. al [14] used an ML algorithm to control the quality of 3D inkjet printing for producing electronic circuits.
- General Electric's labs developed a computer vision technology for detecting microscopic perversion and cracks in printed components by enabling the 3D printer to perform an inspection of printed parts using artificial intelligence and machine learning. [19].

2.5. Deep Learning

Deep Learning is a subset of machine learning that will bring intelligence to the machines. Nowadays, Deep Learning is widely used for object recognition, pattern recognition, natural language processing, and other image processing tasks. But mostly, it is utilized in object recognition tasks such as driver assistance system, autonomous driving system, target detection, to name a few real-world applications. It has also some other interesting applications like "automatic machine translation", "colorization of black and white images", "object

classification in photographs", "image caption generation and automatic game playing", "adding sounds to silent movies", "automatic handwriting generation", and "character text generation" [20] , [21].

2.6. Computer Vision

Computer vision is used to image processing and extracting features from visual images by developing computational methods through the ability of computer science [22]. Facebook uses computer vision for "facial recognition", Google uses it for "image search", and autonomous vehicles use it for "navigation" also in "manufacturing", it is used in "robotics", "process control", and "quality inspection" [18].

2.7. Image Processing

One of the subsets of a computer vision system is image processing and image processing algorithms are used in computer vision to try for simulating in the scale of human vision. In the late 1960s, NASA's Jet Propulsion Laboratory used digital image processing for converting analog signals to digital images with computer systems and today, different applications including Computer-Aided Tomography (CAT) scanning and ultrasounds are using digital imaging. Image Processing applies the mathematical functions and transformations over images

without doing an intelligent complicated calculation over the image itself as well as uses an algorithm for doing transformations on the image such as smoothing, sharpening, contrasting, and stretching on the images. Computer vision is related to use ML techniques for modeling image processing and to recognize patterns for interpretation of images. Computer vision and image processing are able to "distinguish between objects", "classify them", "sort them according to their size". They take images as input and generate the output information such as size, color, and intensity of the images [22].

2.8. Convolutional Neural Networks (CNNs)

Artificial Neural Network is a figuring framework made up of various straightforward, very interconnected processing elements, which processes data by their dynamic state reaction to outer information sources [23]. Artificial Neural Network is a solution for bridging the gap between the capabilities of humans and machines and enabling machines to view the world as humans do in a similar manner. ANNs can use the trained knowledge for a large number of tasks such as "image & video recognition", "image Analysis and classification", "media recreation", "recommendation systems", and natural language processing. Convolutional Neural Networks (CNNs) are outcomes of the advance in combining computer vision and deep learning to construct a particular algorithm

to perfect computer vision applications [24]. “CNNs are drifting territory in deep learning and utilizing in the greater part of the object recognition areas” [20].

CHAPTER 3

METHODOLOGY

Additive manufacturing has tremendous potential to make a custom-designed part on-demand and with minimal material, but it is currently hampered by poor process reliability and throughput. Machine learning was announced as a critical component for continuous growth of AM technology. We propose an ML algorithm for controlling AM process to continuously improve the standard of how the material should be processed in 3D printers. As a first step to implement this system, we will develop an automated AM failure detection by training a deep convolutional neural network. The AM failure detection system is the crucial part of the ML algorithm. The complete ML model enables the AM process with higher throughput (fast) and fewer rejects (accurate).

3.1. Two-phase Strategy

The two-phase strategy to establish the ML algorithm for AM machines is shown in Fig. 8.

The approach presents in two sections: (1) Training an ML based offline detection system to identify correlations and representative model using the images collected from the printed parts and (2) Training an ML based detection system to generate a real-time signal for controlling the AM process.

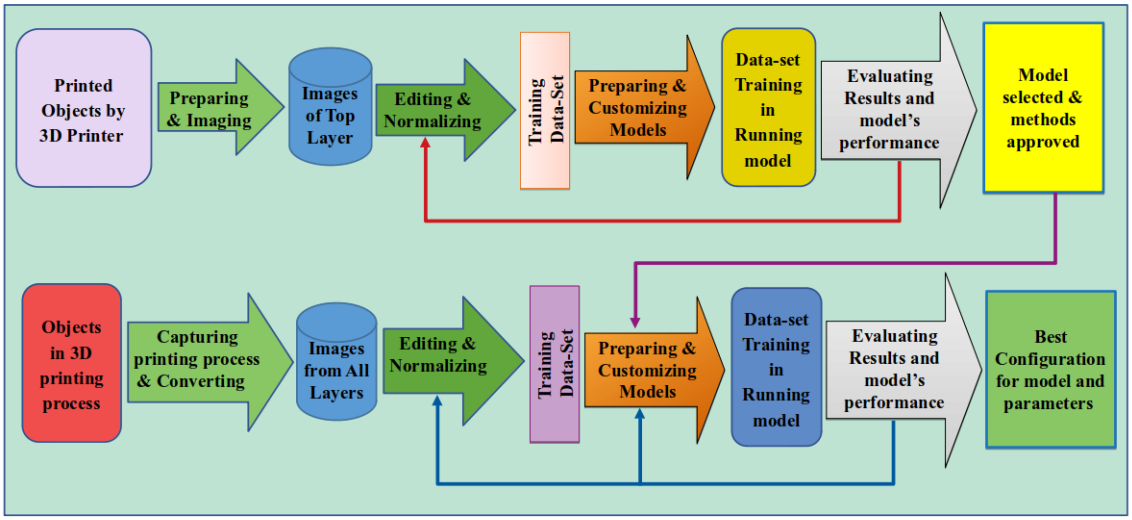

Figure 8: Two-phase strategy for AM process control

3.2. Offline ML Model

Offline Training of ML Model will be the first phase to establish the online ML algorithm for AM machines [Figure 9]. The procedure includes three major steps as follows:

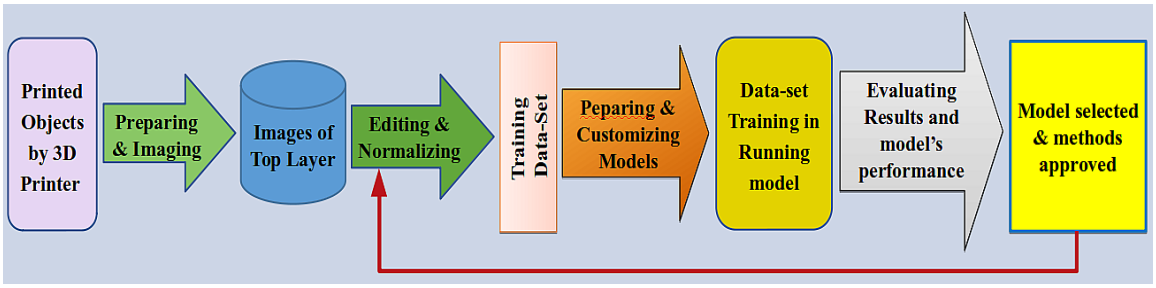

Figure 9: Offline Training of ML Model

3.2.1. Printing 3D objects

In this step, first we design 3D objects in designing tools/software and import them into 3D printer software for adjusting and converting them to acceptable format by 3D printers. We need to print objects in different qualities by changing the printer settings and make a collection of various objects with both perfect and broken surfaces.

3.2.2. Preparing the images from the object surfaces

By Collecting images from the top layer of the printed parts, the required images is prepared for building the training datasets. The images are taken from different angles and different distances and then normalized by editing their size and resolutions. After that, the next step is to label them according to the quality of printing into different classes to use the dataset for a supervised learning technique. This will provide the desired data-set based for training the ML model.

3.3. Training the offline ML Model

we choose the appropriate training model for image processing and image classification. Model trains and validates its parameters in each step and finally results in a trained model that can be applicable to our problem.

One approach for training the offline ML Model is to use traditional shallow neural networks, or support vector machines (SVM) with traditional feature engineering as shown in Fig. 10. The models can be used to predict performance for a relative static environment, e.g., the beginning of the AM process. However, deep models can be used together with rich features to enhance the prediction.

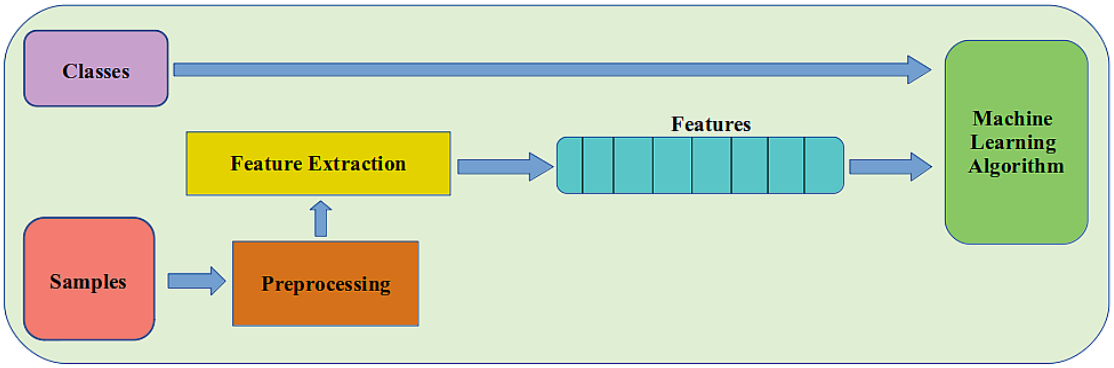

Figure 10: Offline Training of the ML Model

The collected data can be used to train ML algorithm that interpolates between the desired performance in the designed sample and process signatures in the final layers during the printing process [Figure 11].

The well-trained ML software will detect defects in the final parts with final layer patterns in AM process. Once the comprehensive sensing data are trained with printing qualities, one can use the model to control multiple tasks in the AM process in real time.

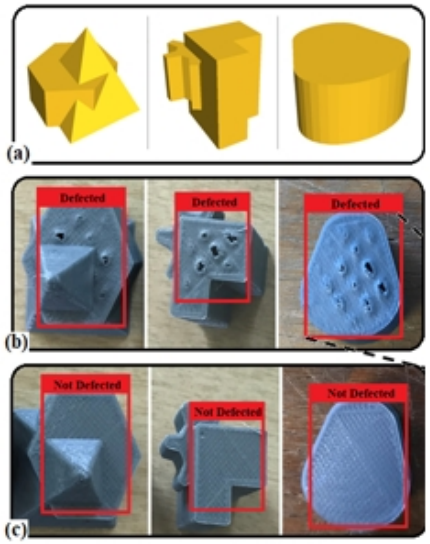

Figure 11: The collected data sample to train ML algorithm

The collected Data sample includes:

- (a) Schematics of the objects in design software;
- (b) Corresponding printed parts with substantial defects on their surfaces that are detected and labeled by the DCNN model as “Broken”;
- (c) Printed parts without defects that are detected and labeled by the DCNN model as “Perfect.”

Highly-dimensional features of the input image can be extracted by passing the input through the convolution filters [Figure 12]. By pooling, you can diminish

dimensionality and brief spatial information. Then, these represented features go over to a classifier or regressor through fully connected layers [25].

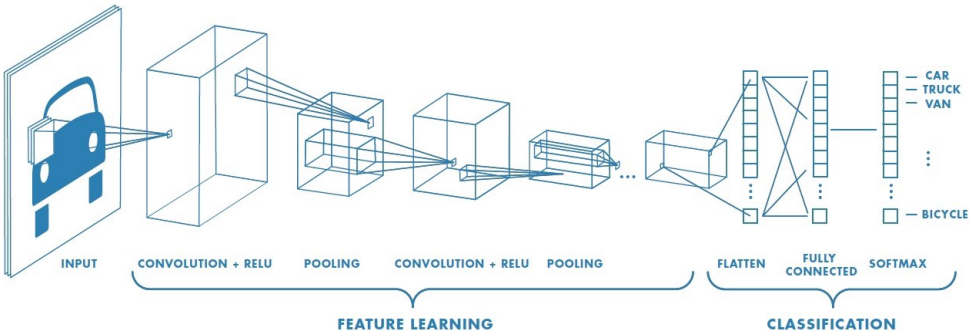

Figure 12: DCNN model

Source: [25]

3.4. Online ML model to control AM Process

As the smart AM machine create a large amount of real-time data, it warrants the use of ML techniques so that the smart control system learns and adapts while the AM machine is operating. Given a collection of training samples with ground truth labels from human experts, multi-modal multi-task learning can be developed to correlate measurements from different sensors and techniques to identify a few representative instruments such as printing properties prediction and defect detection.

an ML model in a smart 3D printer can manage AM operations and optimize the procedure by adjusting the process parameters as they sense certain properties of a build. The more often we print samples, the smarter the ML model

gets and eventually have enough training to automatically predict a problem, and in real time, suggest changes to reach a flawless build. As such, the machine makes better products, faster, with fewer errors, resulting in a breakthrough in productivity. The new AM task can be used to increase the 3D printing database so that AM machine train over time to recognize any issues. The ML model eventually has enough training to automatically predict a problem, provide the derived insights immediately and suggest changes in real time to reach a flawless build. AM machine empowered by machine learning can observe new and different scenarios, takes in new part build data, learns from experience, becomes smarter and more capable, continuously improve the manufacturing process, thereby adds another layer of quality control [2], [5]. This enhances the process and quality of the AM process, thereby produce better parts with fewer quality hiccups, limiting waste of time and materials.

Online Training of ML Model will be the final phase to establish the online ML algorithm for AM machines [Figure 13]. The procedure includes three major steps as follows:

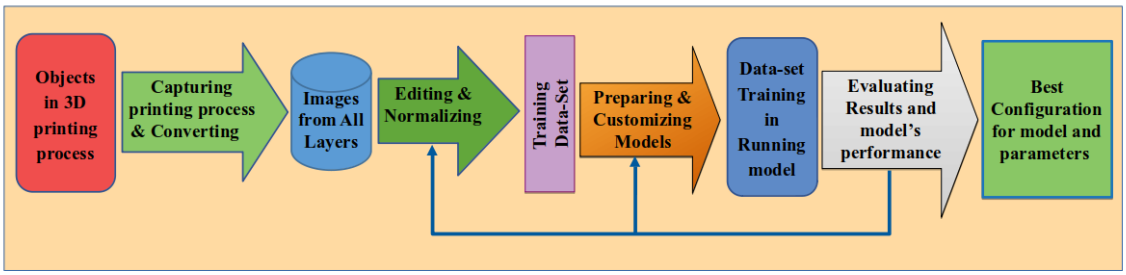

Figure 13: Online Training of ML Model

3.4.1. Printing 3D objects

In this step, first we design 3D objects by designing tools/software and import them into 3D printer software for adjusting and converting them to acceptable format by 3D printer. The objects are printed in different qualities by changing the printer settings and make a collection of various objects with both perfect and broken situations.

3.4.2. Capturing printing process

By capturing video from printing process of objects, and converting the videos to their frames, numerous images are generated from each layer of objects for building the training datasets. The images are normalized by editing their size and number of pixels as well as labeling them according to the printing situation including nuzzle temperature, flow rate and quality classification.

3.4.3. Running Training Model

To train the model, the offline training results are taken into consideration. The model is trained and validated by adjusting its parameters in each step to ensure the final trained model demonstrate the best performance for our problem.

3.4.4. Evaluating the Model

By training the model with the prepared datasets, the model performance such as accuracy is evaluated to adjust the model to achieve better results. In addition, different size of datasets are used for training and validation to evaluate how it turns out and what is the impact of those factors. Also, the model is run by changing the hyper-parameters of the model and comparing the results. The best configuration and parameters of the model are found to serve our goal.

3.4.5. Training a Deep Artificial Neural Network

The traditional image classification techniques are not fully automated classification method so that the feature extraction process requires human intervention, thereby the accuracy of feature selection directly affects the performance of the classification algorithm.

Deep Convolutional Neural Networks and transfer learning approach has recently shown remarkable success in image-based data analysis resulting in a tremendous improvement in automated detection of complex morphologies [26].

High-level steps to build the CNN to classify images are: [27]

- Create convolutional layers by applying kernel or feature maps
- Apply Max pool for translational invariance
- Flatten the inputs
- Create a Fully connected neural network

- Train the model
- Predict the output

Compared with the traditional classification methods, CNNs use multi-layer convolution to extract features and combine the features automatically. The original images can be used as the inputs of the network without the complex preprocessing steps, resulting in great progress in the field of image classification.

The deep neural network extracts spatial features that are then passed to aggregation layers (averaging, pooling, etc.) and additional layers of filters for extracting higher-order features (patterns) that are combined at the top layer for fault detection and classification.

We employed the Deep Artificial Neural Networks (DANN) architecture and a transfer learning technique to retrain the Inception-v3 [28] model of TensorFlow platform [29], Google's deep learning open source software, in detecting and classifying the part printing failure in the AM process. TensorFlow has the advantages of high availability, high flexibility, and high efficiency so that it has ranked first in most of the machine learning and deep learning competitions so far.

Transfer learning is a new machine learning approach which can use the existing knowledge learned from one environment to solve the other new problems often with a small amount of data that have some relation with the old problem with a large amount of data. The large dataset provides the weights of

internal network layers that can be re-adjusted by training with a small amount of data for the new problem. The DCNN models are trained by simple logistic regression [30] and Inception v3.0 platforms [31] using the small number of input images captured from builds with a 2048-dimensional vector size for each image.

3.5. Frameworks, Models, Libraries

In this section, all frameworks, models and libraries that has been used in our approach are explained.

3.5.1. Logistic Regression

Logistic Regression as an ML algorithm is used for solving the classification problems, and known as a predictive analysis algorithm based on the concept of probability [32]. Logistic regression is used when the target variable is a binary variable that is 1 for “Yes” or 0 for “No”. It can directly predict probabilities that are restricted to the (0,1) and keeps the small probabilities of the training data [33].

Statisticians developed the logistic-function/sigmoid-function to describe characteristic of population increase in ecology that is a S-shaped curve for mapping any real-valued number into a value between 0 and 1, but never exactly at those limits. $logistic\,function = 1/(1+e^{-value})$ where e is the base of the natural logarithms and value is the real numerical value that is supposed to transform.

Figure 14 shows a logistic function plot that transforms the numbers between -5 and 5 into the range 0 and 1 [34].

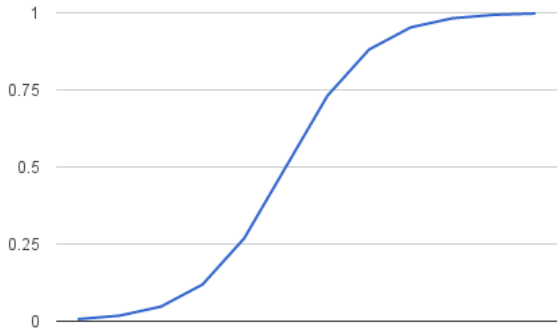

Figure 14: logistic function

3.5.2. Keras

Keras is an high-level API for implementing neural networks that is written in Python. It supports computation engines for developing back-end neural networks [35].

Keras offers consistent and simple APIs for reducing cognitive load that reduces user actions required for common use cases to minimum, and provides explicit and actionable feedback on user error. Integration with lower-level deep learning languages particularly TensorFlow, enables the developers to implement anything buildable in the base language [36].

3.5.3. Tensorflow

Google has developed Tensorflow as an end-to-end open source platform for ML that has a comprehensive, flexible set of tools, libraries and different resources for building and deploying the state-of-the-art applications in ML [37]. One of its applications as an open-source machine learning library is to develop deep neural networks [38]. TensorFlow has a free software and a symbolic math library [39] that ease the process of data gathering, training models, predicting, and correcting future results. TensorFlow can be used for image recognition, handwritten digit classification, natural language processing, word embeddings, recurrent neural networks, PDE (partial differential equation) based simulations, and sequence-to-sequence models for machine translation [40].

3.5.4. Transfer Learning

Transfer learning is an ML method to reuse the trained model of a dataset as the starting point for training model with another dataset. In this situation, transfer learning with domain adaptation improves the second model by exploiting what has been learned in the first configuration [41]. Rapid progress and improved performance are the outcomes of transfer learning as an optimization method through the transfer of knowledge [34]. Retrain the final layer of a pre-trained model, results a considerable decrease in training time of the model, and the size of the dataset for second model [42].

“Develop Model” and “Pre-trained Model” are two common approaches for using transfer learning on predictive modeling problems.

Develop Model Approach steps are as follows:

- Selecting a related predictive modeling problem with a plenty of data.
- Developing a proficient model.
- Reusing the model for the second task.
- Tuning the model.

Pre-trained Model Approach steps are as follows:

- Selecting a pre-trained source model released on large and challenging datasets.
- Reusing the pre-trained model as the starting point for second task.
- Tuning the model.

Pre-trained Model Approach of transfer learning is widely-used in the field of deep learning [34] Inception-V3 is the most famous model for transfer learning [42].

3.5.5. Inception-v3

One of the most used of image recognition models is Inception v3 that for training an image recognition system with ImageNet dataset (a common academic dataset in machine learning) gives greater than 78.1% accuracy.

Building blocks of Inception v3 including symmetric and asymmetric structures are convolutions, pooling (average and max), concatenations, dropouts, and fully connected layers. Batch normalization is applied widely all over the model and Softmax is used for computing the Loss. In (Figure 15), a high-level structure of the model is shown [43].

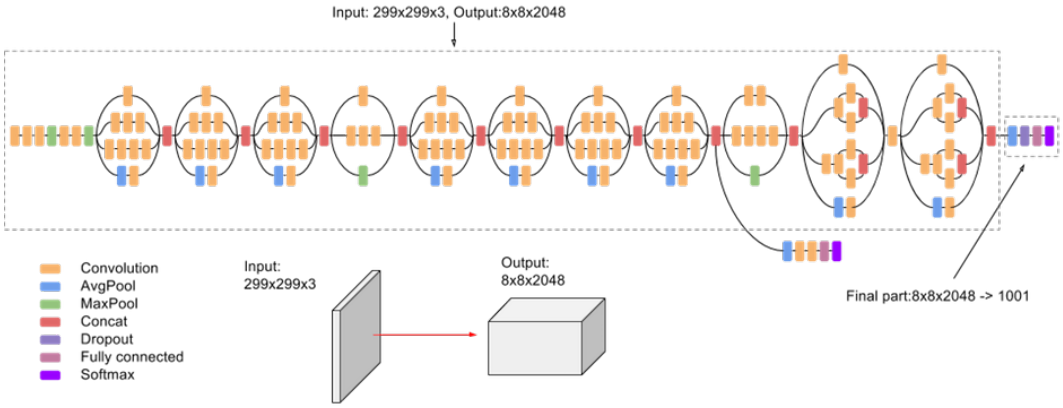

Figure 15: Inception network

Source: [43]

GoogLeNet offered the inception deep convolutional architecture which is called "Inception-V1". The Inception architecture was updated in different aspects, one by introducing the batch normalization in 2015 and it is called Inception-V2 and after a while by adding a factorization layer in the third iteration which is named as Inception-V3 [44].

Inception Layer (Figure 16) compounds the layers of 1 by 1 Convolutional layer, 3 by 3 Convolutional layer, and 5 by 5 Convolutional layer with their output filters to an output vector as an input of the next step [45].

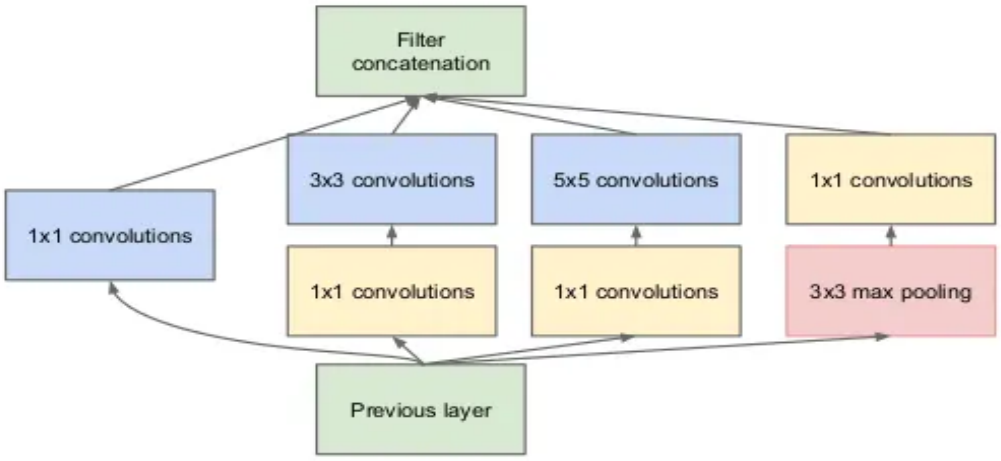

Figure 16: Inception Layer

Source: [45]

CHAPTER 4

Implementation

Implementation of the proposed system, has two approaches. First approach is based on printed objects with Ultimaker 3D printer and images from the surface (top layer) of objects. The second approach is based on objects printed with Ender-3 and images from real-time printing process captured video.

4.1. Running Models with Offline Images

4.1.1. 3D-printed Objects

Objects designed and printed in the following steps

- Make models for the 3d printer using tinkercad.com
- export the created model into Cura 15.04.4
- transport models to the Ultimaker 2 (3D-printer) for printing

4.1.2. Imaging printed Objects

With printed objects, I prepared training and test images in following steps:

- Collecting objects
- Take pictures from the surfaces
- Crop images to desired part
- Resize images to fixed resolution
- Rename and label images to related class

in this approach we had two labels for classification of images: Broken and Perfect. Sample objects has shown in (Figure 17).

Figure 17: Printed Objects

Images from top layer of printed objects from “perfect” class has shown in (Figure 18).

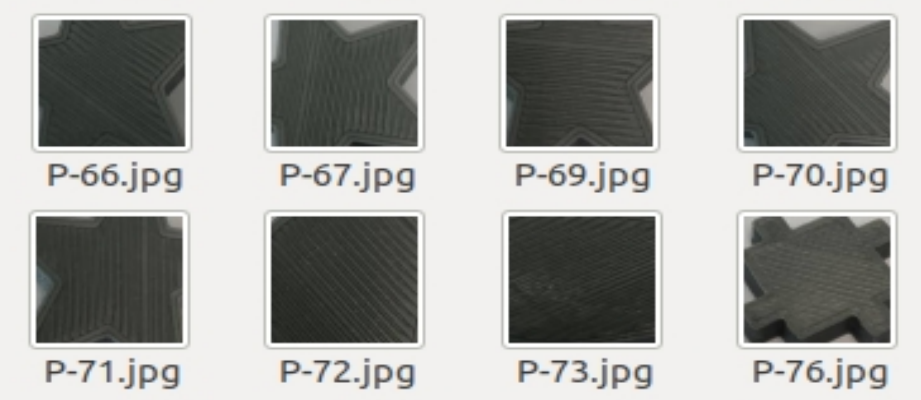

Figure 18: Perfect printed Objects

Images from top layer of printed objects from "broken" class has shown in (Figure 19).

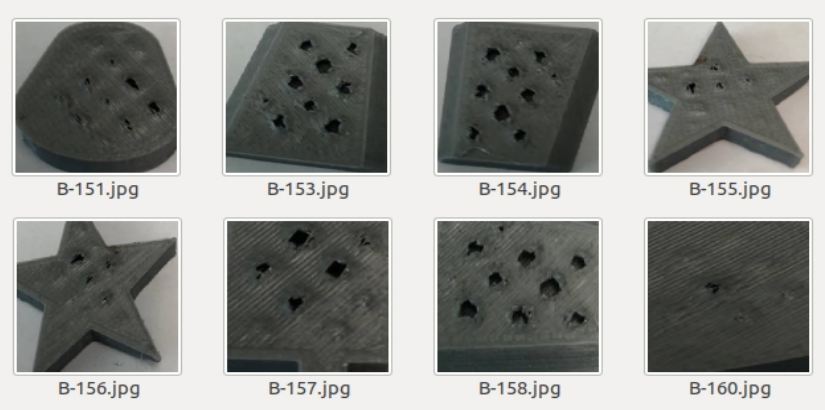

Figure 19: Broken printed Objects

4.1.3. Training Models

In this section, the prepared datasets was used to train with three different models including Logistic Regression (a traditional model for classification in AI and machine learning), Keras (a powerful model and has a strong features for image classification) and Inception (a new approach for transfer learning and for improving the training models accuracy). These are selected for comparing different types of models in performance and results. Choosing the models was based on researches and reviews many surveys and applications done by those models.

4.1.4. Building datasets

Each model has its own structure and needs particular dataset structure for training. Thus the images and each dataset are prepared based on model properties. For example, logistic regression model uses H5 file as an input to train the model. I generated the dataset for logistic regression by preparing images then coding and running a python program to convert them into two H5 files for training and testing images. An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data [46].

4.1.5. Running Models

After preparing the models, it is required to debug the models and troubleshoot any problems and make sure the model is running perfectly. All configuration, libraries and packages are installed and the models are customized to overcome our computational limitations.

4.2. Evaluation of Models

4.2.1. Accuracy in Size of Data-Set

First, 4 groups of images with different number of images is prepared to evaluate the effect of dataseet size in training and test accuracies. Results show that using a larger number of images always increases both training and test accuracies for the all models.

4.2.2. Evaluating Logistic Regression Model

The logistic regression model has best results with 9000 iterations when the learning rate is set to 0.001. The prediction accuracy of the model is %73.4 for detecting defects on the surfaces of the printed objects.

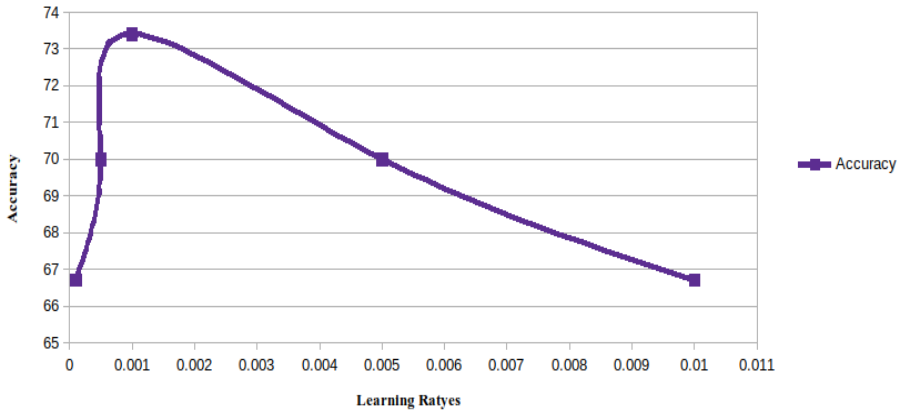

Figure 20: Accuracy in Learning Rates

4.2.3. Evaluating Keras and Inception

The Inception models has the best prediction accuracy outperforming Keras models, and need shorter training time than Keras model. The accuracy of training models has compared in Fig. 21.

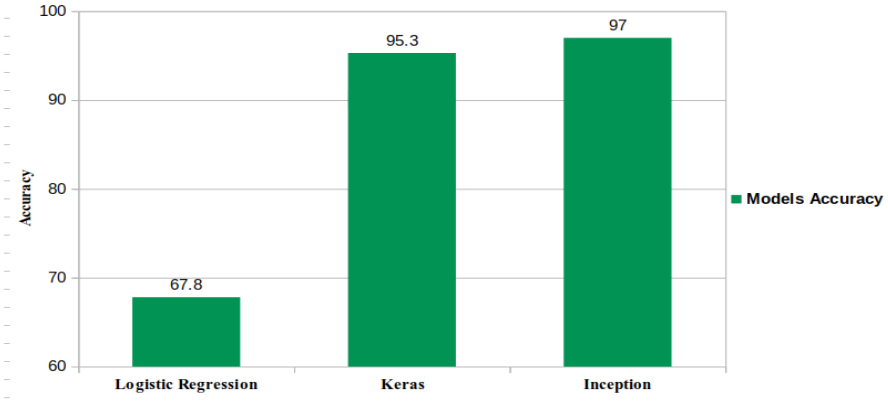

Figure 21: Accuracy of the training Models

4.3. Running models with online images

4.3.1. Objects in 3D-printing process

Objects designed and printed in the following steps:

- Make models for the 3D-printer using Creality Slicer.
- Export the created model into Gcode file.
- Transport models to the Ender-3 (3D-printer) for printing
- Capture and record the video of printing process
    - Camera: Lumenus (Lumenus Digital Optic)
    - Software: Ladibug 3.0

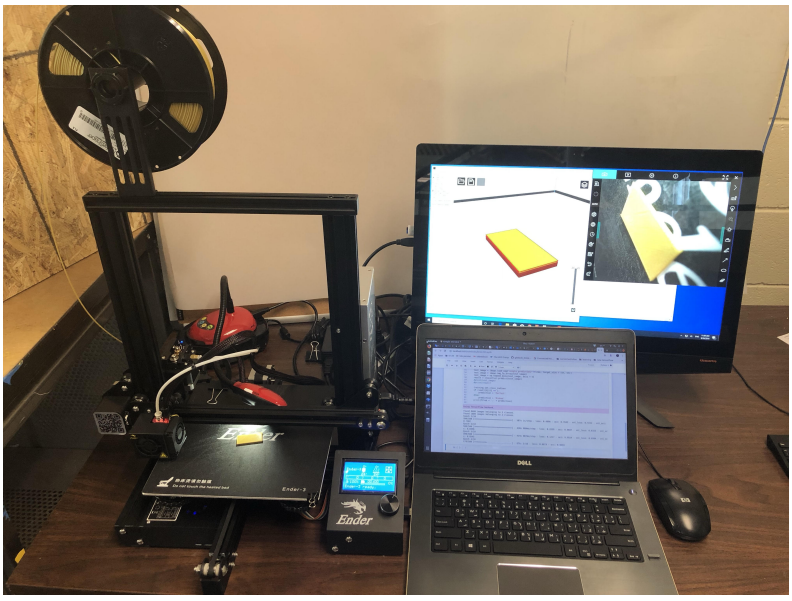

Figure 22: Running Models with Online Images

4.3.2. Capturing the object images in printing process

The training and test images are prepared in following steps:

- Printing objects
- Filming/Capturing the printing process
- Converting Videos to Frames/Images
- Data Augmentation (if needed)
- Crop images to desired part
- Resize images to fixed resolution
- Rename and label images to related class

4.3.3. Examining printing parameters

Sample objects are printed with different printer features, speeds and temperatures and then the captured images are classified in several groups.

- Speeds Range: 50, 100, 200, 400, 800, 999. Speed scale is flow rate of printing material. Feeding flow rate adjust the nuzzle speed.
- Temperature Range: 185, 200, 230, 260. Temperature scale is Centigrade.

The classes are as follows:

Table 1: Printing Parameters

050-185	050-200	050-230	050-260
100-185	100-200	100-230	100-260
200-185	200-200	200-230	200-260
400-185	400-200	400-230	400-260
800-185	800-200	800-230	800-260
999-185	999-200	999-230	999-260

For images in three classes 400-185, 800-185, and 999-185, the printer fails to print properly, and images were very low qualities. Thus, only 21 classes of training datasets were generated.

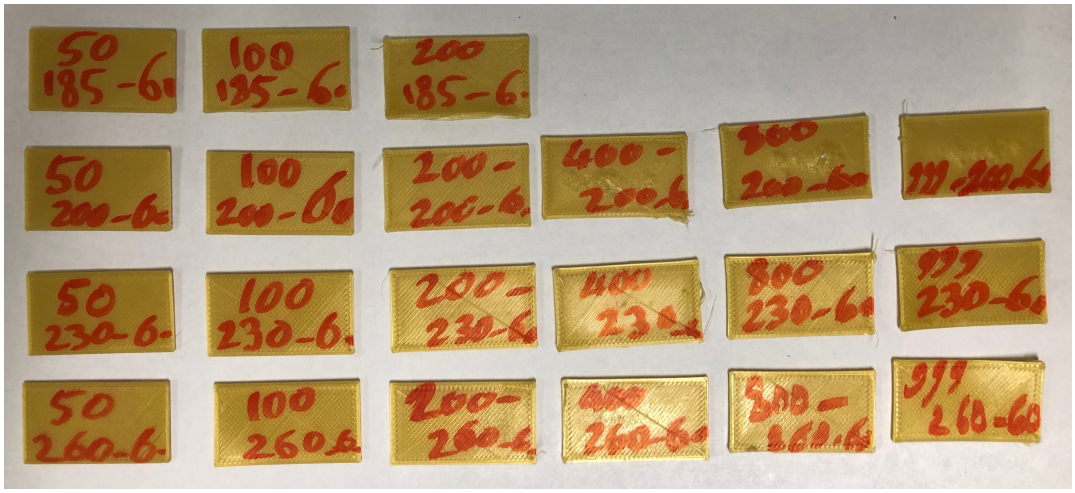

Figure 23: Printed Samples Surface

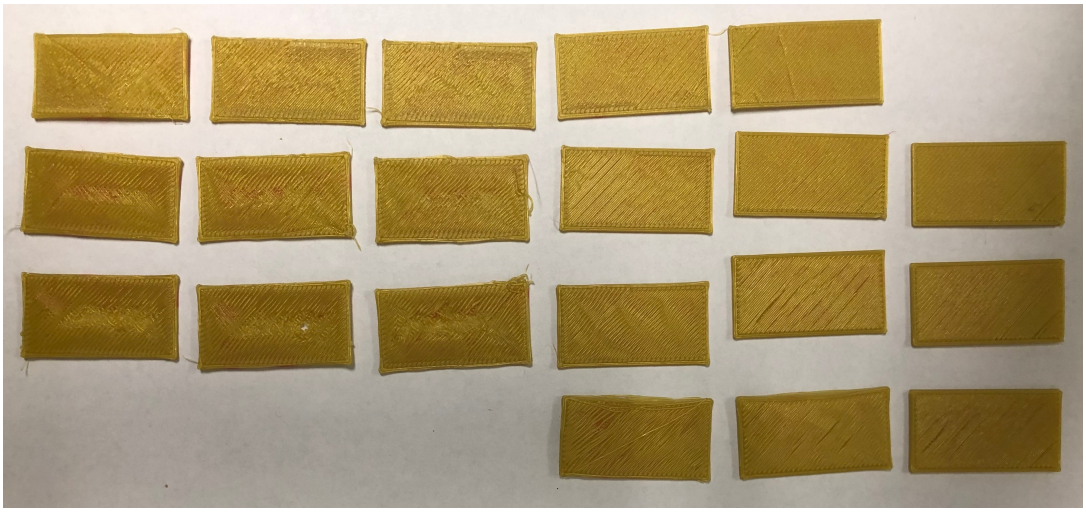

Figure 24: Printed Samples Bottom

4.3.4. Dataset Structure

According to achieved results from previous phase, the Inception-v3 model has been used for training online data and here, the structure of image datasets for Inception model has shown in Fig. 25.

Data-Set includes two folder, Training-Images and Test Images.

- Training-Images folder includes 21 classes as mentioned before and each class contains 5005 images.
- Test-Images folder contains 84 images including 4 images from each class.

All images are resized to 600* by 600 pixels and labeled according to the class name.

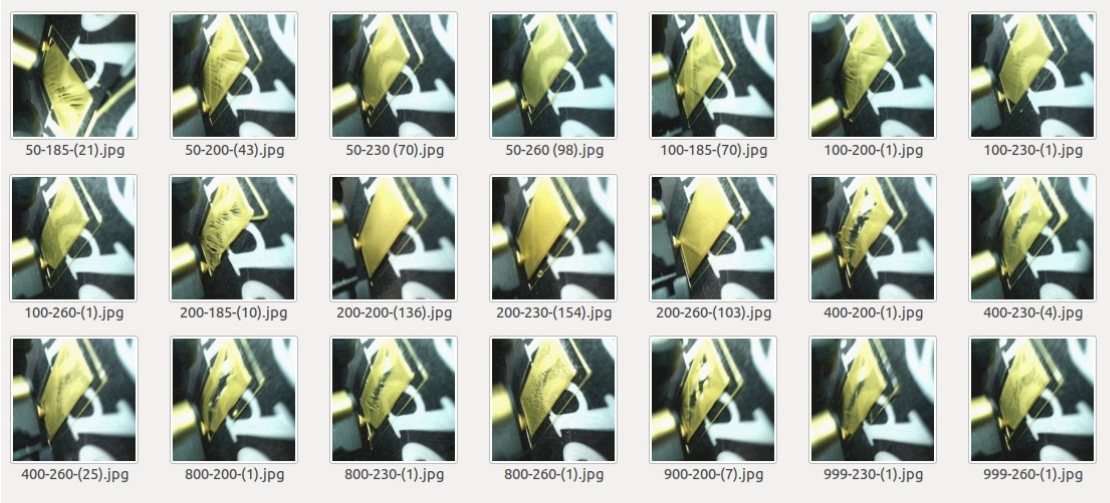

Figure 25: Sample Frames from starting layers

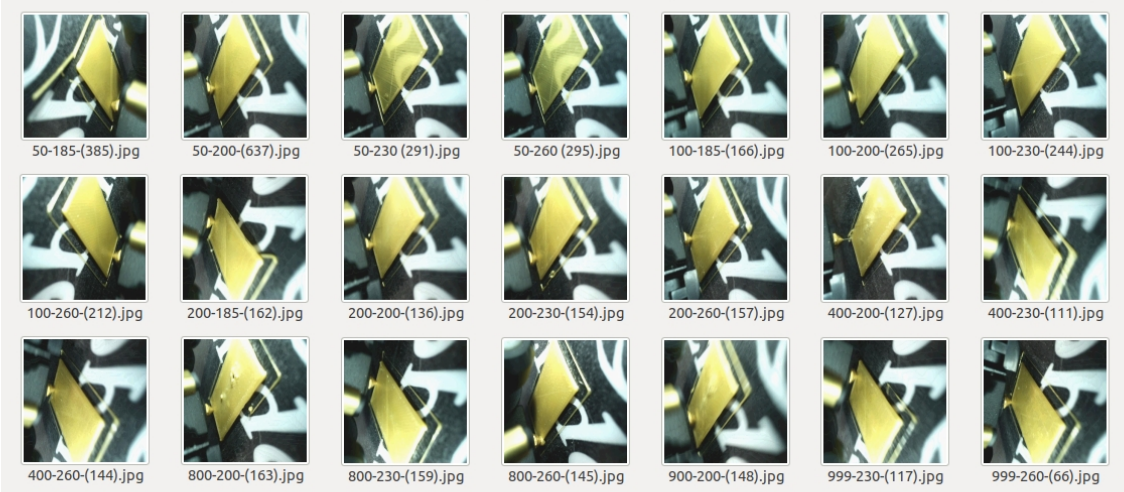

Figure 26: Sample Frames from ending layers

4.4. Hyper-parameters Tuning

A hyperparameter is a parameter that by adjusting it, the results of our models can be improved to reach the best accuracy for our solution. In our case, we have tuned thee hyperparameters including:

4.4.1. Learning Rate

The learning rate is a parameter that scales how large the model can update the weights and biases in every step to move toward the smallest prediction error [47]. In this thesis, the model has been trained with a variety of learning rates from 0.001 to 0.2.

4.4.2. Batch Size

The batch size defines the number of image samples in dataset to use for training the network before the model's internal parameters are updated and refer to the number of training examples utilized in one iteration [48]. The performance of the models is evaluated for changing the batch size in range from 8 to 256 in images.

4.4.3. Training Step

A training step is one gradient update or amount of batch-size processed during one step [49]. In this thesis, the model has been trained with 1000 to 9000 training steps, depending on which parameters are considering and in the interval of 1000 steps.

4.5. Performance evaluation of the models

We trained our model in several configurations of hyper-parameters and compared the results to find the best combination of these parameters and thereby finalize our model according to those settings.

4.5.1. The effect of learning rates in the model accuracy

Training the model with different learning rates shows that how the learning rate effects on accuracy of the models. In our case, the model was trained in different number of training steps and with a range of learning rate. Figure 27 shows the result for low and high learning rates with two different training step sizes, 1000 and 5000. It can be observed that the best performance belongs to learning rate of 0.01.

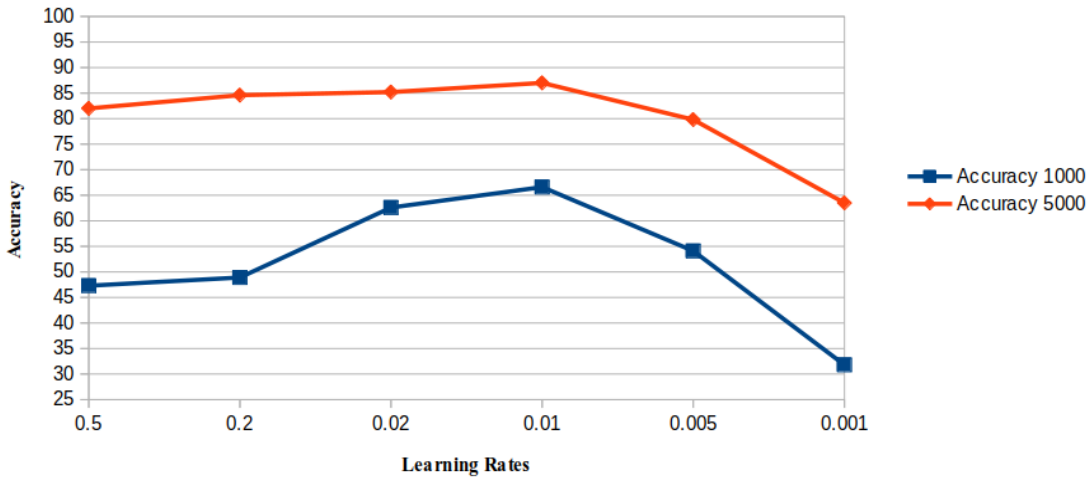

Figure 27: Accuracy in Learning Rate

Batch Size = 8 & Training Steps = 1000, 5000

4.5.2. Effect of number of training steps on accuracy

Training the model with different training steps shows how the number of training steps effects on accuracy of the model. The model was trained with different learning rates and different numbers of training steps. The results show that small number of training steps doesn't have reliable results in accuracy while increasing the number of training steps improves the model performance. The best performance belongs to training steps of 5000 as shown in Fig. 28. In this figure, the results for accuracy in several training steps with two different learning rates, 0.01 and 0.001, has compared.

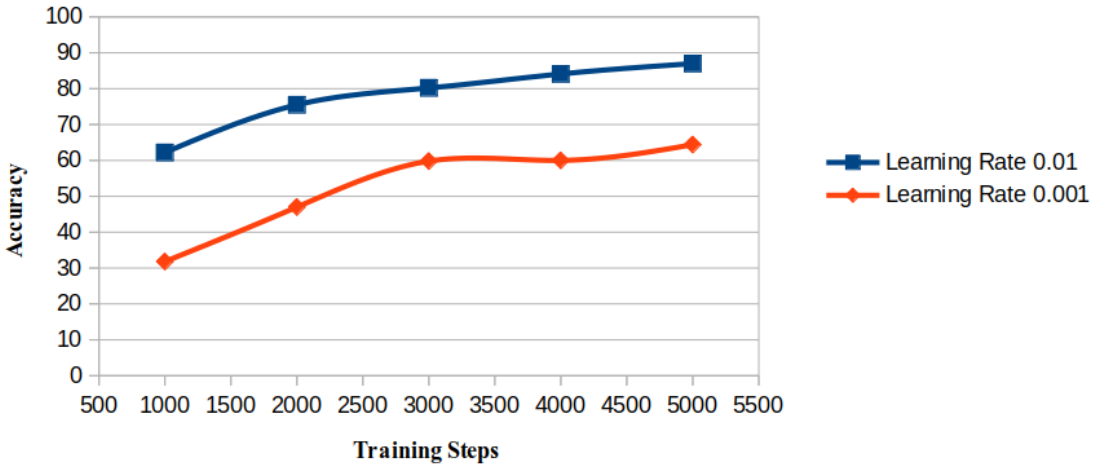

Figure 28: Accuracy in Training Steps

Batch Size = 8 & Learning Rate = 0.01, 0.001

4.5.3. Effect of batch size on the model accuracy

The model is trained for different batch size with two training steps. Figure 29 shows that how the batch size effects on accuracy of the model. The model was trained with learning rates of 0.01 and training steps of 5000 and 9000. The result shows that the effect of batch size is insignificant on accuracy after batch size of 32. Thus, this batch size is selected for our models.

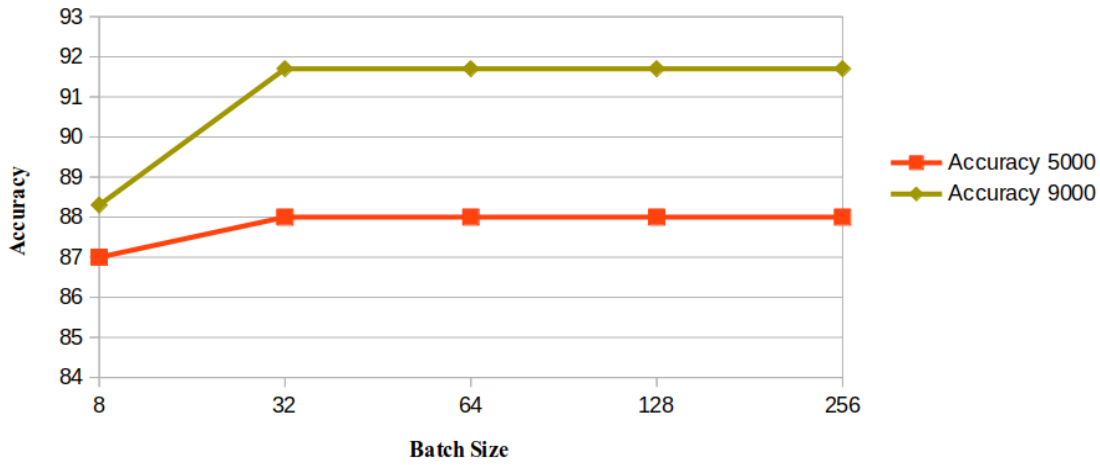

Figure 29: Accuracy in Batch Size

Learning Rate = 0.01 & Training Steps = 5000, 9000

In addition, the outcomes show that with batch size = 32, the larger training steps has the better accuracy in our problem. Figure 30 shows the accuracy of the model with increasing the training steps.

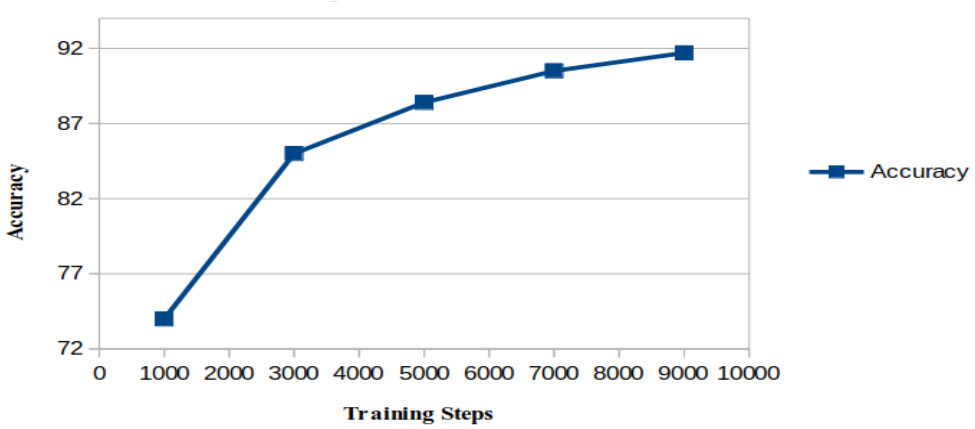

Figure 30: Accuracy in Larger training Steps

Learning Rate = 0.01 & Batch Size = 256

CHAPTER 5

Conclusion

In this chapter, we have run our trained model with test data to determine its accuracy to predict the classification and quality of objects. The test data include 84 images that are 4 images from each class. All numbers and percentages for predictions, are based on these 84 samples.

5.1. Probability of Detection

By applying the trained model to detect the classification and quality of the test data, following results have generated. The columns are our classes and the rows are test data from each class. The 4 samples for each class are averaged in the Table 2. Each row shows the probability of detection [50] for the selected test data in each class. For example, the test data from class 050-185, has been predicted to belong to class 050-185 by %96 on average.

Table 2: Probability of Detection for classifying the objects

		Name of Classes																				
		050-185	050-200	050-230	050-260	100-185	100-200	100-230	100-260	200-185	200-200	200-230	200-260	400-200	400-230	400-260	800-200	800-230	800-260	999-200	999-230	999-260
Test Data	050-185	96	0.5	0.6	0.5	0.2	0.5	0.6	0.3	0	0	0.1	0.1	0	0	0.3	0	0	0	0	0	0.1
	050-200	1.3	36.9	10.9	6.5	15.4	5.9	4.6	3.2	0	0.4	0.1	0.2	0.1	0.2	8.4	0.3	0.7	0.8	0.3	0	3.9
	050-230	2.1	4.8	34.4	21.9	12.1	5.4	6.3	4.2	0.1	0.8	0.8	0.4	0	0.4	0.4	0	0.1	0.8	0.1	0	4.7
	050-260	3.3	1.5	5.8	63.4	7.2	1.3	4.7	7.1	0.1	0.6	0.1	0.8	0	1	1	0.1	0.1	0.4	0.1	0	1.3
	100-185	0.2	1.7	6	8.1	41.9	7	3.2	3.3	0.2	0.7	0	0.5	0.1	0.2	1.4	0.1	0.8	0.2	0.3	0	24
	100-200	2.1	3.1	1.3	8.2	33.5	12.3	20.8	4.2	0.1	0.4	0.1	0.2	0	0.1	1	0.1	0.4	0.2	0.1	0	11.6
	100-230	0.4	1	0.5	6	17.9	11.1	53.3	1.1	0	0.8	0	0.4	0	0	2.1	0.2	0.4	0.2	0	0.6	3.7
	100-260	1	0.3	1.9	28.9	2.2	1.4	8.4	15.4	0	0.5	0.2	0.7	0.1	0.1	17.7	0.2	0.4	0.9	0	0.2	19.3
	200-185	0.1	3.2	0.5	0.3	1.4	8.4	4.3	0.3	78.3	0.4	0	0	1	0.4	0.1	0.7	0.3	0	0.1	0	0.2
	200-200	3.9	0.8	3	3.4	5.7	11.4	15.3	2.5	0.1	3.8	0.1	1.7	1	0.9	27.5	1.5	2.6	2.2	0.3	0.6	11.8
	200-230	1.5	0	1	0.5	1.8	4.5	10.5	1.9	0.2	0.4	14.8	2.2	2	18.6	9.4	3.7	4.1	2.9	0.8	1.2	17.8
	200-260	1.6	0.5	0.7	8	1.6	0.7	3.5	6.3	0.2	1.2	0.5	15.5	0.1	2.9	3	0.3	0.3	1.1	0.1	0.8	51.2
	400-200	0.2	0.7	0.4	1.2	3	1.4	8	0.5	16.2	0.5	0.2	1.4	32.7	2.9	2.5	11.2	2.9	0.5	1.7	2	9.8
	400-230	0.4	0.2	1.1	1.3	4.7	0.3	1.3	1.4	0.3	0.5	4.3	8.4	0.4	35.6	9.7	0.7	2.6	2.5	0.9	3.4	20
	400-260	0.3	0.4	0.3	0.8	3	1	2.3	2.9	0.1	3.8	0.1	0.7	0.2	0.1	65.1	1	0.9	0.9	0.2	0	16
	800-200	0.4	0.1	0.2	0.3	0.4	1	10.6	0.8	0.3	0.2	0.2	0.3	1.7	0.3	2	67.9	2.8	0.4	1.7	2.1	6.4
	800-230	0.5	0.1	0.1	2	0.7	3.5	7.8	3.1	0.3	0.6	0	0.2	0.9	2	11.1	1.9	12.2	0.6	1.4	0.8	50.1
	800-260	0.7	3	1.4	1.7	9.7	5.7	2.1	1.7	0.2	1	2.3	6.4	0.3	3.6	3.7	0.8	6.8	16.9	6	12.9	12.9
	999-200	0.2	0.4	1.3	1.8	14.9	6	8.3	0.5	0.8	0.3	0.7	1.5	1.7	3.4	1.6	10.1	5.2	7.6	17.7	2.2	13.8
	999-230	0.1	0.3	0.6	0.7	2.1	6	19.1	1	6.6	1.6	0.3	5.3	0.8	0.5	10.1	4	1.1	0.5	0.4	32.4	6.6
	999-260	0.1	0	0.1	0.1	0.1	0.1	0.1	0.9	0	0.1	0	0.2	0	0	4.3	0	0.1	1	0	0	92.6

5.2. Correct Predictions

In this section, the percentage of correct predictions in each class for each temperature has been calculated.

Table 3: Correct Prediction Percentage

		Speed					
		50	100	200	400	800	1000
Temperature	185	%100	%75	%75	%100	%100	%100
	200	%75	%0	%0	%75	%75	%25
	230	%100	%75	%25	%50	%25	%25
	260	%100	%25	%50	%100	%0	%100

5.3. Quality Classification

We have graded the quality of printed objects and therefore each class has a grade that shows the quality of that class. Grading has 5 level from 1 for the worst to 5 for the best qualities.

5.3.1. Actual Quality

Figure 31 displays the classification labels that we set for actual quality. Changing temperatures from 185 to 200 results in the same quality grade. The

temperatures of 230 and 260 have better quality grades and in these temperature, we see the lower speed leads to the better quality.

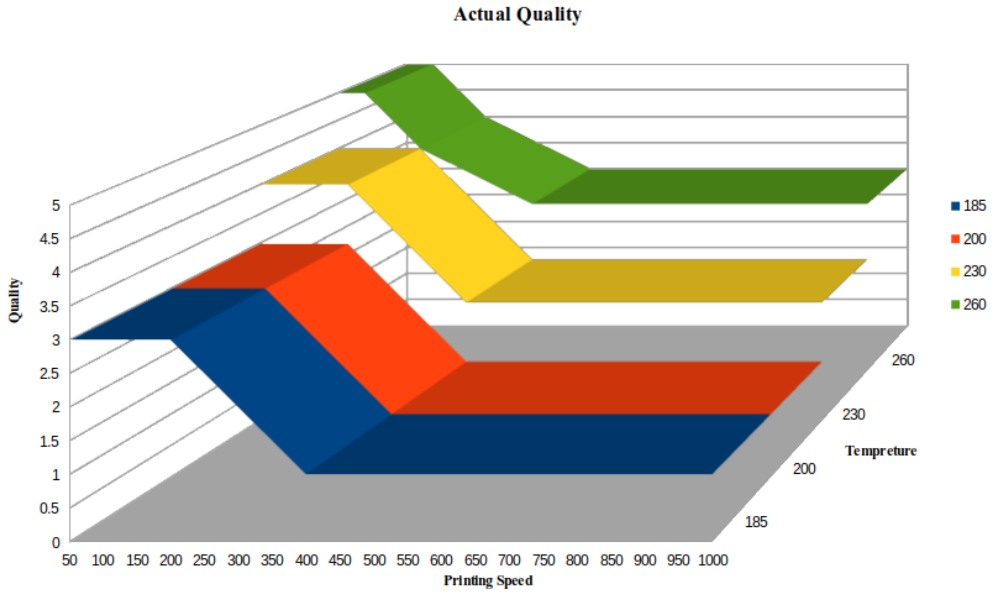

Figure 31: Actual Quality Graph

5.3.2. Predicted Quality

Figure 32 shows how the model has predicted the quality of the parts for higher temperatures, the best quality is for lower speeds and poor qualities belong to high speeds and low temperatures.

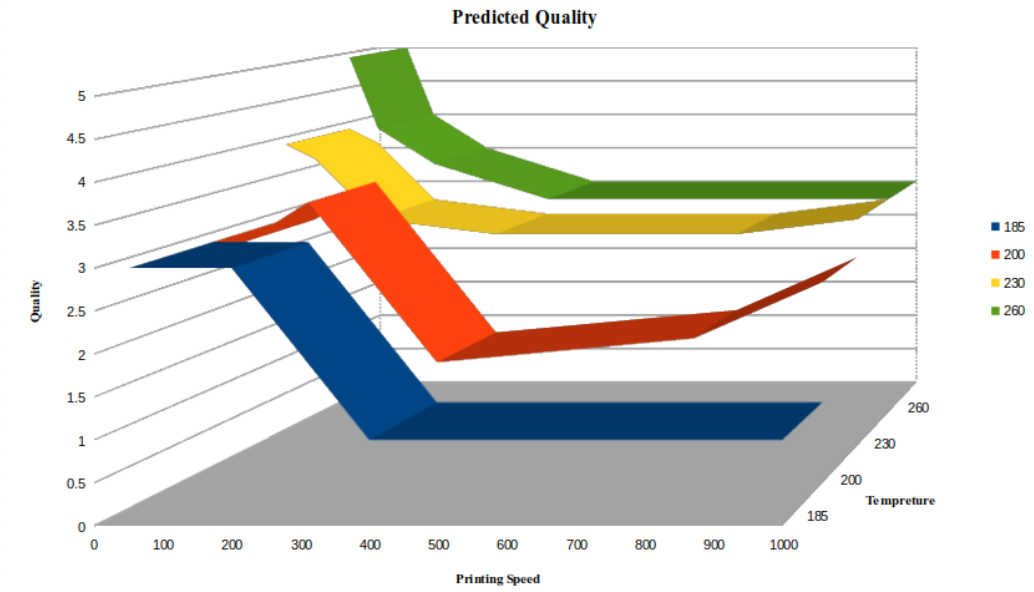

Figure 32: Predicted Quality Graph

5.3.3. Comparing Actual and Predicted Quality

It should be noted, we eliminated the data of temperature 185 because actual and predicted qualities are the same for all speeds. For other temperatures following results were concluded:

- For speed = 50, actual and predicted qualities are the same.
- For speed 100, at first, actual and predicted qualities are the same but with increasing the temperature, the predictions getting worse.
- For speed 200, predictions have around %17 error
- For speed 400, 800, 999 in high temperature, predictions are precise but in low temperature, predictions have around %45 error

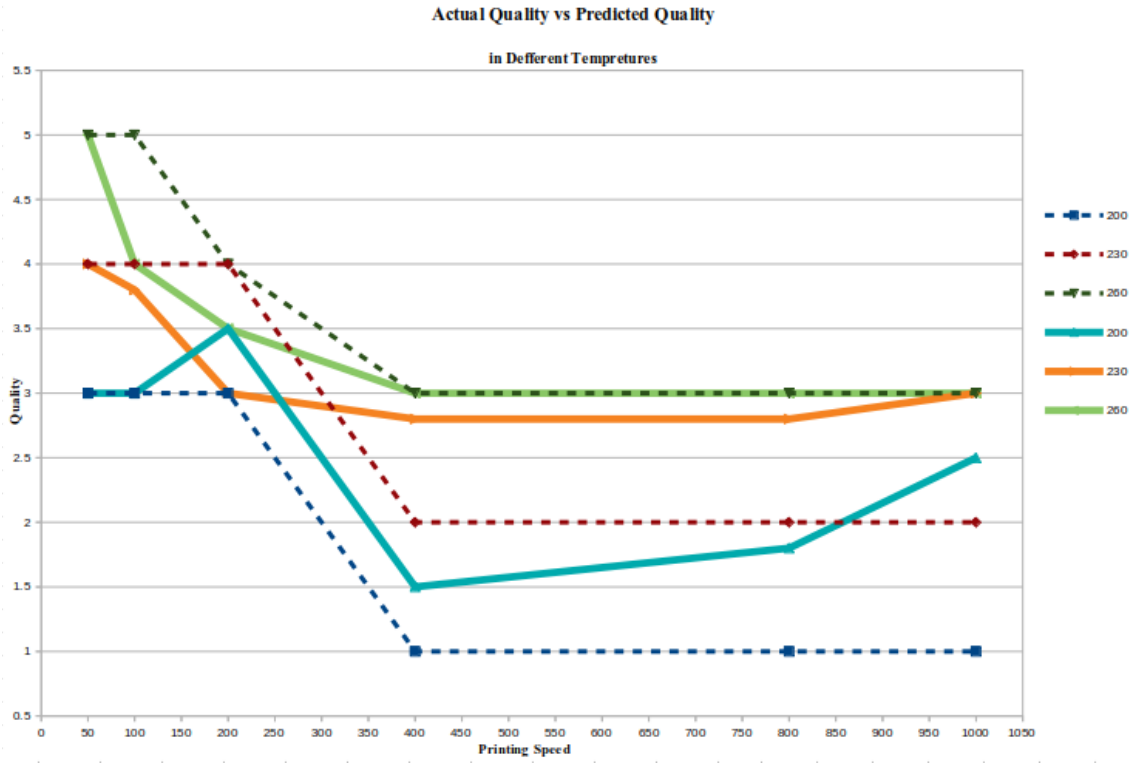

Figure 33: Actual Quality vs Predicted Quality

5.4. Confusion Matrix True Positive False Positive

In binary classification, all dataset instances for test data label as a negative or positive. The results of this prediction or classification process for our 84 test samples summarize in four categories [51] as following:

- True Positive (TP) for Positive actual value and Positive predicted value
- False Positive (FP) for Negative predicted value and Positive actual value
- False Negative (FN) for Positive predicted value and Negative actual value
- True Negative (TN) for Negative predicted value and Negative actual value

Correct predictions are those in TP and TN categories [52].

Table 4: Confusion Matrix for quality of printed objects

		Actual Value	
		Positive	Negative
Predicted Value	Positive	TP = 19	FN = 5
	Negative	FP = 5	TN = 55

REFERENCES

[1] Y. Lin, M. Hung, H. Huang, … C. C.-I. R. and, and undefined 2017, "Development of advanced manufacturing cloud of things (AMCoT)—A smart manufacturing platform," *ieeexplore.ieee.org*.

[2] K. S. Wang, Z. Li, J. Braaten, and Q. Yu, "Interpretation and compensation of backlash error data in machine centers for intelligent predictive maintenance using ANNs," *Adv. Manuf.*, vol. 3, no. 2, pp. 97–104, Jun. 2015.

[3] W. Zhang, A. Mehta, P. S. Desai, and C. Fred Higgs III, "MACHINE LEARNING ENABLED POWDER SPREADING PROCESS MAP FOR METAL ADDITIVE MANUFACTURING (AM)."

[4] P. Xu, H. Mei, L. Ren, W. C.-I. transactions on, and undefined 2016, "ViDX: Visual diagnostics of assembly line performance in smart factories," *ieeexplore.ieee.org*.

[5] X. Xue, Y. Kou, S. Wang, Z. L.-I. T. on, and undefined 2016, "Computational experiment research on the equalization-oriented service strategy in collaborative manufacturing," *ieeexplore.ieee.org*.

[6] Spectralengines, "Industry 4.0 and how smart sensors make the difference," 2018. [Online]. Available: https://www.spectralengines.com/articles/industry-4-0-and-how-smart-sensors-make-the-difference.

[7] B. Scholz-Reiter, D. Weimer, H. T.-C. annals, and undefined 2012, "Automated surface inspection of cold-formed micro-parts," *Elsevier*.

[8] X. Qi, G. Chen, Y. Li, X. Cheng, and C. Li, "Applying Neural-Network-Based Machine Learning to Additive Manufacturing: Current Applications, Challenges, and Future Perspectives," *Engineering*, vol. 5, no. 4, pp. 721–729, Aug. 2019.

[9] Deniz Yalcin, “Open-Loop vs Closed-Loop Systems in the Materials Testing Industry,” 2017. [Online]. Available: https://www.admet.com/open-loop-vs-closed-loop-systems-materials-testing-industry/.

[10] Threeding, “Binder Jetting 3D Printing Technology,” 2016. [Online]. Available: https://www.threeding.com/blog/binder-jetting-3d-printing-technology.

[11] T. Melanson, “What Industry 4.0 Means for Manufacturers,” 2018. [Online]. Available: https://aethon.com/mobile-robots-and-industry4-0/.

[12] L. Scime, J. B.-A. Manufacturing, and undefined 2018, “A multi-scale convolutional neural network for autonomous anomaly detection and classification in a laser powder bed fusion additive manufacturing process,” *Elsevier*.

[13] A. C. F. on A. Manufacturing, “Standard terminology for additive manufacturing technologies,” 2012.

[14] S. Stoyanov and C. Bailey, “Machine Learning for Additive Manufacturing of Electronics.”

[15] D. Deng, Y. Chen, and D. J. Epstein, “Assembled Additive Manufacturing-A Hybrid Fabrication Process Inspired by Origami Design.”

[16] ARKInvest, “ARK Invest disruptive innovation big ideas 2017,” 2017. .

[17] H. Chen, Y. Z.-A. 2015 International, and undefined 2015, “Learning Algorithm Based Modeling and Process Parameters Recommendation System for Binder Jetting Additive Manufacturing Process,” *… .asmedigitalcollection.asme.org*.

[18] B. L. DeCost, H. Jain, A. D. Rollett, and E. A. Holm, “Computer Vision and Machine Learning for Autonomous Characterization of AM Powder Feedstocks,” *JOM*, vol. 69, no. 3, pp. 456–465, Mar. 2017.

[19] R. Bharadwaj, “Artificial Intelligence Applications in Additive Manufacturing (3D Printing) | Emerj,” 2019. [Online]. Available: https://emerj.com/ai-sector-overviews/artificial-intelligence-applications-additive-manufacturing-3d-printing/. [Accessed: 19-Oct-2019].

[20] R. Haridas and R. L. Jyothi, “Convolutional neural networks: A comprehensive survey,” *Int. J. Appl. Eng. Res.*, vol. 14, no. 3, pp. 780–789, 2019.

[21] A. Chowdhury, S. Biswas, S. B.- bioRxiv, and undefined 2017, “Active deep learning reduces annotation burden in automatic cell segmentation,” *biorxiv.org*.

[22] R. Szeliski, “Computer Vision: Algorithms and Applications - Richard Szeliski - Google Books.” [Online]. Available: https://books.google.com/books?hl=en&lr=&id=bXzAlkODwa8C&oi=fnd&pg=PR4&dq=R.+Szeliski,+Computer+Vision:+Algorithms+and+Applications+(New+York:+Springer,+2010&ots=g_234YqDGF&sig=1QpDaEl4Qa8OZE7fsLBgiV7Q8l0#v=onepage&q=R. Szeliski%2C Computer Vision%3A Algo. [Accessed: 13-Oct-2019].

[23] Tutorialspoint, “Artificial Intelligence - Neural Networks.” [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/artificial intelligence neural networks.htm.

[24] M. Akram and H. Tuhin, “Detection and 3D Visualization of Brain Tumor using Deep Learning and Polynomial Interpolation,” 2019.

[25] D. Schumaker, “Deep Learning CNN’s in Tensorflow with GPUs.” [Online]. Available: https://daveschumaker.tumblr.com/post/161325653149/deep-learning-cnns-in-tensorflow-with-gpus.

[26] N. Razaviarab, S. Sharifi, and Y. M. Banadaki, “Smart additive manufacturing empowered by a closed-loop machine learning algorithm,” in *spiedigitallibrary.org*, 2019, p. 17.

[27] R. Khandelwal, “Building Powerful Image Classification Convolutional Neural Network using Keras.” [Online]. Available: https://medium.com/datadriveninvestor/building-powerful-image-classification-convolutional-neural-network-using-keras-a1839d0ff298. [Accessed: 14-Oct-2019].

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, “Rethinking the Inception Architecture for Computer Vision.”

[29] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.”

[30] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng, “Efficient L 1 Regularized Logistic Regression.”

[31] M. Islam, F. Foysal, N. Neehal, … E. K.-P. computer, and undefined 2018, “InceptB: a CNN Based classification approach for recognizing traditional Bengali games,” *Elsevier*.

[32] A. Pant, “Introduction to Logistic Regression.”

[33] R. S. Brid, “Logistic Regression.” [Online]. Available: https://medium.com/greyatom/logistic-regression-89e496433063.

[34] J. Brownlee, “Logistic Regression for Machine Learning,” 2016. [Online]. Available: https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

[35] M. Heller, “What is Keras? The deep neural network API explained,” 2019. [Online]. Available: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html.

[36] Keras.io, “Why use Keras?” [Online]. Available: https://keras.io/why-use-keras/.

[37] J. Poduska, “Top Open Source Tools for Deep Learning - RTInsights,” 2019. [Online]. Available: https://www.rtinsights.com/top-deep-learning-tools/. [Accessed: 19-Oct-2019].

[38] Tensorflow.org, “An end-to-end open source machine learning platform.” [Online]. Available: https://www.tensorflow.org.

[39] A. Moltzau, “AI as a Service? - Towards Data Science,” 2019. [Online]. Available: https://towardsdatascience.com/ai-as-a-service-b465ddc0c7e0. [Accessed: 19-Oct-2019].

[40] Technicalwala, “What is Tensorflow,” 2019. [Online]. Available: http://technicalwala.com/what-is-tensorflow/.

[41] J. Brownlee, “A Gentle Introduction to Transfer Learning for Deep Learning,” 2019. .

[42] A. Milton-Barker, “Inception V3 Deep Convolutional Architecture For Classifying Acute Myeloid/Lymphoblastic Leukemia,” 2019. .

[43] Google.Cloud, “Advanced Guide to Inception v3 on Cloud TPU.” [Online]. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced.

[44] S.-H. Tsang, “Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015,” 2018. [Online]. Available: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c.

[45] F. SHAIKH, “Deep Learning in the Trenches: Understanding Inception Network from Scratch,” 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/.

[46] Fileinfo, “Hierarchical Data Format 5 File.”

[47] J. Jordan, “Setting the learning rate of your neural network.,” 2019. [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/. [Accessed: 20-Oct-2019].

[48] Kevin Shen, “Effect of batch size on training dynamics - Mini Distill - Medium,” 2018. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e. [Accessed: 20-Oct-2019].

[49] T. Samuel, “What is the difference between Step, Batch Size, Epoch, Iteration？ Machine Learning Terminology - Tolotra Samuel,” 2018. [Online]. Available: https://tolotra.com/2018/07/25/what-is-the-difference-between-step-batch-size-epoch-iteration-machine-learning-terminology/. [Accessed: 14-Oct-2019].

[50] C. M. Watson, “Signal Detection and Digital Modulation Classification-Based Spectrum Sensing for Cognitive Radio,” 2013.

[51] Classeval, “Basic evaluation measures from the confusion matrix – Classifier evaluation with imbalanced datasets,” 2016. [Online]. Available:

https://classeval.wordpress.com/introduction/basic-evaluation-measures/. [Accessed: 21-Oct-2019].

[52] OpenEye, “Drawing ROC Curve — OpenEye Python Cookbook vJun 2018,” 2019. [Online]. Available: https://docs.eyesopen.com/toolkits/cookbook/python/plotting/roc.html. [Accessed: 21-Oct-2019].

APPENDIX A

:

Graphs and Tables

Data

The effect of learning rates in the model accuracy

- Batch Size = 8 & Training Steps = 1000, 5000

Table 5: The data for Figure 27		
Learning Rate	Accuracy (TS = 1000)	Accuracy (TS = 5000)
0.5	47.3	82
0.2	48.9	84.6
0.02	62.6	85.2
0.01	66.6	87
0.005	54.1	79.8
0.001	31.8	63.5

Effect of number of training steps on accuracy

- Batch Size = 8 & Learning Rate = 0.01, 0.001

Table 6: The data for Figure 28		
Steps	Learning Rate 0.01	Learning Rate 0.001
1000	62.2	31.8
2000	75.5	47
3000	80.2	59.8
4000	84.1	60
5000	87	64.4

Effect of Batch Size on the model accuracy

- Learning Rate = 0.01 & Training Steps = 5000, 9000

Table 7: The data for Figure 29		
Batch Size	Accuracy (TS = 5000)	Accuracy (TS = 9000)
8	87	88.3
32	88	91.7
64	88	91.7
128	88.6	91.7
256	88.4	91.7

Accuracy in Larger Training Steps

- Learning Rate = 0.01 & Batch Size = 256

Table 8: The data for Figure 30	
Steps	Accuracy
1000	74
3000	85
5000	88.4
7000	90.5
9000	91.7

Table 9: Predictions for each Sample test Data																					
	050-185	050-200	050-230	050-260	100-185	100-200	100-230	100-260	200-185	200-200	200-230	200-260	400-200	400-230	400-260	800-200	800-230	800-260	999-200	999-230	999-260
050-185	92.4	1.8	0.7	0.3	0.2	1.4	0.9	0.8	0.0	0.1	0.0	0.0	0.0	0.0	0.9	0.0	0.1	0.0	0.0	0.0	0.3
050-185	98.0	0.1	0.0	0.4	0.4	0.6	0.1	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
050-185	96.7	0.2	1.6	0.0	0.0	0.1	0.3	0.3	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
050-185	97.0	0.0	0.0	1.5	0.0	0.1	1.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

050-200	0.6	10.1	19.0	13.1	8.2	4.2	3.3	1.8	0.0	0.5	0.1	0.3	0.2	0.3	30.3	0.9	1.9	0.4	0.6	0.0	4.2
050-200	2.2	61.2	1.4	4.8	8.1	5.0	1.1	5.1	0.0	0.0	0.0	0.1	0.0	0.0	0.2	0.1	0.0	1.9	0.6	0.0	8.0
050-200	0.2	48.0	1.3	0.7	39.8	6.4	2.0	1.3	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
050-200	2.1	28.4	21.8	7.5	5.4	8.0	11.9	4.5	0.0	1.1	0.1	0.2	0.0	0.2	3.2	0.1	0.9	0.9	0.1	0.1	3.4
050-230	0.1	1.1	45.1	24.9	4.2	5.6	15.7	1.6	0.0	0.1	0.0	0.0	0.0	0.0	0.3	0.0	0.1	0.0	0.0	0.0	1.0
050-230	4.0	1.2	30.8	8.7	7.4	8.7	2.7	11.2	0.0	2.5	2.4	1.1	0.0	0.3	1.1	0.0	0.2	3.0	0.1	0.0	14.5
050-230	1.7	2.1	31.6	29.6	23.2	3.2	2.3	2.5	0.1	0.4	0.8	0.3	0.0	0.7	0.2	0.0	0.1	0.1	0.1	0.0	0.6
050-230	2.4	14.9	30.1	24.1	13.7	4.1	4.4	1.4	0.2	0.3	0.1	0.2	0.0	0.8	0.1	0.0	0.1	0.2	0.1	0.1	2.7
050-260	3.8	3.0	5.8	36.9	14.6	3.3	5.2	14.7	0.2	1.4	0.3	1.8	0.0	3.9	2.1	0.4	0.3	0.3	0.2	0.1	1.8
050-260	0.1	0.1	3.0	83.0	1.2	0.3	7.9	1.8	0.0	0.1	0.0	0.0	0.0	0.0	1.0	0.0	0.1	0.0	0.0	0.1	1.2
050-260	8.6	0.6	13.2	61.7	0.6	1.3	2.9	8.6	0.0	0.3	0.1	0.1	0.0	0.0	0.6	0.1	0.1	0.1	0.0	0.1	1.0
050-260	0.8	2.3	1.4	72.1	12.3	0.4	2.7	3.2	0.0	0.6	0.1	1.3	0.0	0.1	0.2	0.0	0.1	1.1	0.0	0.0	1.3
100-185	0.4	3.0	20.0	3.2	62.0	5.9	0.4	2.5	0.1	0.2	0.1	0.6	0.0	0.2	0.1	0.0	0.1	0.3	0.2	0.0	0.5
100-185	0.0	1.8	1.7	2.5	9.2	2.3	2.0	2.6	0.0	0.5	0.0	0.2	0.0	0.0	2.3	0.0	0.1	0.1	0.1	0.0	74.4
100-185	0.1	0.3	1.1	1.9	56.5	9.1	2.5	5.1	0.1	1.5	0.1	0.5	0.2	0.3	1.5	0.3	2.4	0.2	0.9	0.0	15.2
100-185	0.2	1.5	1.0	24.9	40.0	10.6	8.0	3.0	0.7	0.8	0.0	0.5	0.0	0.2	1.5	0.2	0.5	0.3	0.0	0.1	6.0
100-200	0.2	8.9	0.7	3.7	69.2	15.0	0.1	0.8	0.0	0.1	0.1	0.3	0.0	0.0	0.0	0.1	0.0	0.3	0.3	0.0	0.2
100-200	0.2	0.2	0.4	0.4	10.0	14.6	25.1	2.4	0.1	1.0	0.0	0.2	0.1	0.0	3.7	0.2	0.8	0.1	0.1	0.0	40.3
100-200	0.1	1.6	2.8	9.8	8.9	5.3	55.8	9.2	0.0	0.2	0.0	0.2	0.0	0.0	0.2	0.1	0.1	0.2	0.0	0.0	5.4
100-200	7.9	1.9	1.3	19.0	46.1	14.4	2.3	4.5	0.4	0.1	0.1	0.2	0.0	0.5	0.2	0.2	0.5	0.1	0.0	0.0	0.2
100-230	0.3	0.4	0.1	0.8	9.0	30.8	57.5	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.0	0.1	0.0	0.1
100-230	0.3	0.8	0.9	4.9	60.3	9.3	15.5	1.7	0.1	0.2	0.0	0.0	0.0	0.0	0.4	0.1	0.1	0.2	0.0	0.0	5.2
100-230	0.5	2.8	0.7	11.3	1.2	3.8	55.0	0.7	0.0	2.6	0.0	1.4	0.2	0.2	7.3	0.4	1.3	0.5	0.0	1.7	8.4
100-230	0.6	0.1	0.1	7.0	1.1	0.6	85.1	1.2	0.0	0.6	0.0	0.1	0.0	0.0	0.9	0.2	0.1	0.0	0.0	0.8	1.2
100-260	0.7	0.5	1.4	26.7	5.9	4.6	11.9	12.9	0.1	1.0	0.0	0.2	0.2	0.1	6.0	0.2	0.0	0.2	0.0	0.0	26.9
100-260	2.3	0.2	2.9	32.4	1.1	0.2	3.8	37.9	0.0	0.2	0.6	2.3	0.0	0.2	4.0	0.1	0.1	2.3	0.0	0.1	9.3
100-260	0.8	0.4	2.3	51.2	1.4	0.4	10.3	4.3	0.0	0.1	0.0	0.2	0.0	0.0	2.7	0.1	0.1	0.9	0.0	0.6	24.2
100-260	0.1	0.3	1.0	5.4	0.6	0.3	7.6	6.5	0.0	0.9	0.0	0.2	0.1	0.2	58.2	0.2	1.3	0.1	0.0	0.0	16.9
200-185	0.0	0.2	0.1	0.0	0.3	0.0	0.2	0.1	98.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200-185	0.2	11.9	1.8	1.0	4.8	32.7	16.0	0.9	26.2	1.2	0.0	0.0	0.3	0.1	0.1	0.3	0.8	0.1	0.0	0.1	0.5
200-185	0.0	0.3	0.0	0.1	0.5	0.7	0.6	0.0	95.9	0.0	0.1	0.0	0.1	0.3	0.1	1.1	0.1	0.0	0.1	0.0	0.0
200-185	0.0	0.2	0.1	0.0	0.2	0.1	0.2	0.1	92.4	0.1	0.0	0.1	3.4	1.0	0.2	1.6	0.1	0.0	0.3	0.0	0.0
200-200	0.8	1.3	7.8	5.6	9.9	6.1	21.6	1.1	0.2	6.1	0.1	1.2	2.4	2.1	10.4	2.1	6.2	2.5	0.4	0.1	11.5
200-200	12.6	0.9	2.5	1.0	7.4	19.1	34.8	2.7	0.4	0.7	0.1	0.2	0.2	1.3	2.3	2.4	2.1	0.6	0.3	1.9	6.4
200-200	2.0	0.5	1.3	6.8	5.2	20.3	3.3	6.0	0.0	7.1	0.1	5.3	0.5	0.1	3.1	1.2	1.7	5.8	0.2	0.3	29.2

200-230	1.5	0.1	1.1	0.8	5.2	5.6	20.3	5.9	0.6	1.4	0.4	1.1	0.9	2.2	3.2	0.9	2.6	0.8	0.1	0.5	45.0
200-230	3.3	0.1	2.1	0.4	1.1	6.6	2.0	0.7	0.3	0.2	29.1	3.2	1.1	33.2	1.4	2.9	5.3	1.5	2.0	1.3	2.2
200-230	0.1	0.0	0.1	0.1	0.1	0.3	5.5	0.2	0.0	0.2	1.1	0.6	5.2	23.0	29.3	2.6	7.4	0.8	0.9	2.0	20.5
200-230	1.1	0.0	0.6	0.7	1.1	5.4	14.0	0.5	0.0	0.0	28.8	3.7	0.9	16.1	3.6	8.4	1.1	8.3	0.3	1.1	3.7
200-260	0.1	0.0	0.0	0.9	0.0	0.0	2.9	1.0	0.0	0.1	0.0	1.9	0.0	0.0	5.4	0.0	0.2	1.3	0.0	0.0	86.1
200-260	0.3	0.1	0.5	3.7	0.3	0.5	2.0	8.4	0.0	0.2	0.0	2.9	0.0	0.3	0.3	0.0	0.1	0.1	0.0	0.0	80.1
200-260	0.8	1.4	1.3	3.8	2.2	1.6	8.3	5.9	0.5	4.3	0.1	32.4	0.2	0.6	4.3	1.2	0.6	0.5	0.1	2.8	27.3
200-260	5.3	0.3	0.8	23.6	4.0	0.7	0.7	9.8	0.3	0.4	1.9	24.9	0.1	10.9	2.0	0.0	0.1	2.7	0.1	0.2	11.1
400-200	0.3	1.1	0.1	0.9	2.5	1.8	13.8	0.3	0.5	0.6	0.0	0.2	28.0	0.3	5.6	9.0	8.0	0.2	2.3	1.6	22.7
400-200	0.4	1.2	0.3	2.9	2.4	2.5	4.5	0.1	51.3	0.6	0.1	2.2	18.4	1.1	1.3	4.3	1.0	0.8	1.1	3.0	0.5
400-200	0.2	0.3	0.9	0.6	6.1	1.1	8.9	1.4	0.7	0.5	0.4	1.9	56.5	1.7	1.3	12.8	0.3	0.3	2.3	0.0	1.6
400-200	0.1	0.0	0.1	0.4	1.0	0.1	5.0	0.3	12.4	0.2	0.3	1.2	27.9	8.3	1.8	18.6	2.4	0.9	1.3	3.4	14.4
400-230	0.1	0.6	2.6	2.4	0.8	0.3	0.0	0.6	0.0	1.2	13.7	32.4	0.8	17.7	9.5	0.3	0.5	4.2	2.5	0.3	9.6
400-230	0.0	0.1	0.5	0.2	1.4	0.2	2.9	1.3	0.3	0.6	0.0	0.3	0.3	6.9	27.8	0.3	3.0	1.1	0.1	2.4	50.3
400-230	1.4	0.1	0.4	1.0	1.1	0.6	1.7	0.8	0.2	0.1	0.2	0.7	0.2	49.5	1.2	0.3	6.6	3.7	0.3	10.9	19.0
400-230	0.0	0.0	0.7	1.4	15.5	0.1	0.8	3.0	0.6	0.1	3.5	0.3	0.4	68.3	0.4	2.0	0.2	1.0	0.6	0.0	1.1
400-260	0.5	0.1	0.0	0.1	0.1	0.0	0.7	0.7	0.0	4.1	0.0	0.0	0.0	0.0	74.9	0.1	0.0	0.1	0.0	0.0	18.4
400-260	0.2	1.3	0.9	2.1	10.0	3.4	1.9	7.9	0.1	6.2	0.2	0.9	0.3	0.0	32.3	1.8	0.5	1.1	0.1	0.0	28.7
400-260	0.4	0.1	0.2	1.1	0.9	0.3	6.3	2.7	0.0	3.9	0.2	1.8	0.1	0.2	65.1	1.0	2.0	2.4	0.1	0.1	11.4
400-260	0.0	0.0	0.1	0.1	0.9	0.1	0.2	0.5	0.2	1.0	0.0	0.1	0.3	0.1	88.0	1.2	0.9	0.1	0.6	0.0	5.6
800-200	0.4	0.1	0.4	0.1	0.3	0.7	1.2	0.2	0.3	0.1	0.3	0.1	1.9	0.1	0.8	91.9	0.3	0.2	0.3	0.1	0.1
800-200	0.0	0.1	0.1	0.0	0.0	0.1	0.8	0.1	0.1	0.1	0.0	0.1	1.4	0.0	2.8	92.9	0.1	0.1	0.9	0.1	0.1
800-200	0.3	0.0	0.0	0.4	0.6	2.1	35.7	2.1	0.2	0.4	0.0	1.0	3.3	0.5	3.1	9.0	8.9	0.8	0.3	7.4	23.9
800-200	0.7	0.0	0.3	0.5	0.5	1.1	4.8	0.8	0.6	0.2	0.3	0.1	0.4	0.6	1.3	77.6	2.0	0.4	5.4	0.8	1.5
800-230	0.3	0.1	0.1	0.1	0.6	1.5	1.0	2.4	0.0	0.1	0.0	0.5	0.8	1.8	0.4	0.2	11.8	0.9	3.3	0.8	73.4
800-230	0.2	0.2	0.1	3.9	0.0	0.2	0.6	0.2	0.0	0.1	0.0	0.0	0.0	0.0	3.5	0.0	0.1	0.2	0.0	0.1	90.6
800-230	0.1	0.0	0.0	3.9	1.0	1.9	5.9	6.6	0.2	1.9	0.0	0.2	0.3	1.0	39.5	0.5	5.6	0.1	0.1	2.1	29.0
800-230	1.6	0.2	0.1	0.1	1.3	10.6	23.7	3.1	0.8	0.4	0.0	0.1	2.6	5.2	1.0	7.0	31.1	1.1	2.3	0.2	7.3
800-260	0.8	0.2	1.2	0.9	0.6	2.5	4.4	0.1	0.4	0.5	1.0	0.2	0.9	9.8	0.4	1.0	24.0	0.5	0.1	50.6	0.1
800-260	0.4	10.5	2.1	0.1	36.0	16.9	0.2	0.3	0.1	1.7	0.5	0.1	0.2	3.1	0.2	0.2	2.0	10.1	13.9	0.1	1.3
800-260	0.2	1.2	1.7	3.8	0.7	0.6	1.8	2.1	0.1	0.9	0.2	0.6	0.1	0.1	11.6	0.1	0.3	35.3	1.3	0.6	36.0
800-260	1.2	0.2	0.6	2.0	1.4	2.9	1.7	4.4	0.2	0.8	7.5	24.8	0.2	1.6	2.8	1.9	1.0	21.7	8.7	0.5	14.1
900-200	0.2	0.0	0.1	0.3	0.6	0.9	4.7	0.5	0.0	0.1	0.5	3.6	0.5	8.9	1.2	6.0	17.7	21.6	9.9	4.3	18.3
900-200	0.1	0.2	0.3	0.2	10.7	4.8	0.8	0.3	0.4	0.7	0.0	1.3	1.1	0.4	1.5	17.5	1.6	6.0	19.4	0.2	32.5
900-200	0.3	0.2	3.0	1.0	22.4	4.4	13.5	0.6	1.5	0.2	2.0	0.1	1.4	3.1	3.0	1.0	1.3	2.4	31.4	4.2	3.2

900-200	0.2	1.2	2.0	5.6	25.9	14.2	14.2	0.5	1.2	0.1	0.4	1.1	3.6	1.3	0.6	16.1	0.2	0.3	10.0	0.3	1.2
999-230	0.3	0.8	2.0	2.1	7.3	23.1	16.3	2.1	0.4	2.1	1.0	19.2	1.2	0.5	2.5	10.9	1.4	0.7	0.5	2.1	3.4
999-230	0.0	0.0	0.0	0.0	0.0	0.0	6.2	0.1	0.9	0.0	0.0	0.1	0.7	0.4	3.5	1.8	0.5	0.2	0.0	81.1	4.5
999-230	0.1	0.1	0.1	0.7	0.6	0.8	11.8	0.4	24.4	4.1	0.1	1.5	1.3	0.4	31.4	2.2	0.4	0.4	0.1	9.1	9.9
999-230	0.1	0.0	0.1	0.0	0.5	0.2	42.1	1.3	0.7	0.2	0.1	0.2	0.1	0.8	2.9	1.2	2.1	0.5	0.9	37.5	8.5
999-260	0.2	0.1	0.0	0.1	0.3	0.2	0.0	1.2	0.0	0.3	0.0	0.2	0.0	0.0	3.2	0.0	0.1	3.7	0.1	0.0	90.2
999-260	0.0	0.1	0.3	0.4	0.0	0.0	0.1	0.7	0.0	0.0	0.0	0.1	0.0	0.0	2.8	0.0	0.1	0.4	0.0	0.0	94.8
999-260	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.1	0.0	0.3	0.0	0.0	0.4	0.0	0.0	0.1	0.0	0.0	98.5
999-260	0.0	0.0	0.0	0.0	0.0	0.0	0.4	1.4	0.0	0.1	0.0	0.0	0.0	0.0	10.9	0.0	0.0	0.1	0.0	0.0	86.9

Table 10: Predictions with Analytical Data

Actual	**N**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Quality Yes/No**	
	3	**3**	**4**	**5**	**3**	**3**	**4**	**5**	**3**	**3**	**4**	**4**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**Quality Score**	
	050-185	050-200	050-230	050-260	100-185	100-200	100-230	100-260	200-185	200-200	200-230	200-260	400-200	400-230	400-260	800-200	800-230	800-260	999-200	999-230	999-260	**Predicted**	
050-185	92.4	1.8	0.7	0.3	0.2	1.4	0.9	0.8	0.0	0.1	0.0	0.0	0.0	0.0	0.9	0.0	0.1	0.0	0.0	0.0	0.3	3	N
050-185	98.0	0.1	0.0	0.4	0.4	0.6	0.1	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	3	N
050-185	96.7	0.2	1.6	0.0	0.0	0.1	0.3	0.3	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	N
050-185	97.0	0.0	0.0	1.5	0.0	0.1	1.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	N
Average	96.0	0.5	0.6	0.5	0.2	0.5	0.6	0.3	0.0	0.0	0.1	0.1	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.1	3.0	
050-200	0.6	10.1	19.0	13.1	8.2	4.2	3.3	1.8	0.0	0.5	0.1	0.3	0.2	0.3	30.3	0.9	1.9	0.4	0.6	0.0	4.2	3	N
050-200	2.2	61.2	1.4	4.8	8.1	5.0	1.1	5.1	0.0	0.0	0.0	0.1	0.0	0.0	0.2	0.1	0.0	1.9	0.6	0.0	8.0	3	N
050-200	0.2	48.0	1.3	0.7	39.8	6.4	2.0	1.3	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	N
050-200	2.1	28.4	21.8	7.5	5.4	8.0	11.9	4.5	0.0	1.1	0.1	0.2	0.0	0.2	3.2	0.1	0.9	0.9	0.1	0.1	3.4	3	N
Average	1.3	36.9	10.9	6.5	15.4	5.9	4.6	3.2	0.0	0.4	0.1	0.2	0.1	0.2	8.4	0.3	0.7	0.8	0.3	0.0	3.9	3.0	
050-230	0.1	1.1	45.1	24.9	4.2	5.6	15.7	1.6	0.0	0.1	0.0	0.0	0.0	0.0	0.3	0.0	0.1	0.0	0.0	0.0	1.0	4	Y
050-230	4.0	1.2	30.8	8.7	7.4	8.7	2.7	11.2	0.0	2.5	2.4	1.1	0.0	0.3	1.1	0.0	0.2	3.0	0.1	0.0	14.5	4	Y
050-230	1.7	2.1	31.6	29.6	23.2	3.2	2.3	2.5	0.1	0.4	0.8	0.3	0.0	0.7	0.2	0.0	0.1	0.1	0.1	0.0	0.6	4	Y
050-230	2.4	14.9	30.1	24.1	13.7	4.1	4.4	1.4	0.2	0.3	0.1	0.2	0.0	0.8	0.1	0.0	0.1	0.2	0.1	0.1	2.7	4	Y
Average	2.1	4.8	34.4	21.9	12.1	5.4	6.3	4.2	0.1	0.8	0.8	0.4	0.0	0.4	0.4	0.0	0.1	0.8	0.1	0.0	4.7	4.0	
050-260	3.8	3.0	5.8	36.9	14.6	3.3	5.2	14.7	0.2	1.4	0.3	1.8	0.0	3.9	2.1	0.4	0.3	0.3	0.2	0.1	1.8	5	Y
050-260	0.1	0.1	3.0	83.0	1.2	0.3	7.9	1.8	0.0	0.1	0.0	0.0	0.0	0.0	1.0	0.0	0.1	0.0	0.0	0.1	1.2	5	Y

050-260	8.6	0.6	13.2	61.7	0.6	1.3	2.9	8.6	0.0	0.3	0.1	0.1	0.0	0.0	0.6	0.1	0.1	0.1	0.0	0.1	1.0	5	Y
050-260	0.8	2.3	1.4	72.1	12.3	0.4	2.7	3.2	0.0	0.6	0.1	1.3	0.0	0.1	0.2	0.0	0.1	1.1	0.0	0.0	1.3	5	Y
Average	3.3	1.5	5.8	63.4	7.2	1.3	4.7	7.1	0.1	0.6	0.1	0.8	0.0	1.0	1.0	0.1	0.1	0.4	0.1	0.0	1.3	5.0	
100-185	0.4	3.0	20.0	3.2	62.0	5.9	0.4	2.5	0.1	0.2	0.1	0.6	0.0	0.2	0.1	0.0	0.1	0.3	0.2	0.0	0.5	3	N
100-185	0.0	1.8	1.7	2.5	9.2	2.3	2.0	2.6	0.0	0.5	0.0	0.2	0.0	0.0	2.3	0.0	0.1	0.1	0.1	0.0	74.4	3	N
100-185	0.1	0.3	1.1	1.9	56.5	9.1	2.5	5.1	0.1	1.5	0.1	0.5	0.2	0.3	1.5	0.3	2.4	0.2	0.9	0.0	15.2	3	N
100-185	0.2	1.5	1.0	24.9	40.0	10.6	8.0	3.0	0.7	0.8	0.0	0.5	0.0	0.2	1.5	0.2	0.5	0.3	0.0	0.1	6.0	3	N
Average	0.2	1.7	6.0	8.1	41.9	7.0	3.2	3.3	0.2	0.7	0.0	0.5	0.1	0.2	1.4	0.1	0.8	0.2	0.3	0.0	24.0	3.0	
100-200	0.2	8.9	0.7	3.7	69.2	15.0	0.1	0.8	0.0	0.1	0.1	0.3	0.0	0.0	0.0	0.1	0.0	0.3	0.3	0.0	0.2	3	N
100-200	0.2	0.2	0.4	0.4	10.0	14.6	25.1	2.4	0.1	1.0	0.0	0.2	0.1	0.0	3.7	0.2	0.8	0.1	0.1	0.0	40.3	3	N
100-200	0.1	1.6	2.8	9.8	8.9	5.3	55.8	9.2	0.0	0.2	0.0	0.2	0.0	0.0	0.2	0.1	0.1	0.2	0.0	0.0	5.4	4	Y
100-200	7.9	1.9	1.3	19.0	46.1	14.4	2.3	4.5	0.4	0.1	0.1	0.2	0.0	0.5	0.2	0.2	0.5	0.1	0.0	0.0	0.2	3	N
Average	2.1	3.1	1.3	8.2	33.5	12.3	20.8	4.2	0.1	0.4	0.1	0.2	0.0	0.1	1.0	0.1	0.4	0.2	0.1	0.0	11.6	3.3	
100-230	0.3	0.4	0.1	0.8	9.0	30.8	57.5	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.0	0.1	0.0	0.1	4	Y
100-230	0.3	0.8	0.9	4.9	60.3	9.3	15.5	1.7	0.1	0.2	0.0	0.0	0.0	0.0	0.4	0.1	0.1	0.2	0.0	0.0	5.2	3	N
100-230	0.5	2.8	0.7	11.3	1.2	3.8	55.0	0.7	0.0	2.6	0.0	1.4	0.2	0.2	7.3	0.4	1.3	0.5	0.0	1.7	8.4	4	Y
100-230	0.6	0.1	0.1	7.0	1.1	0.6	85.1	1.2	0.0	0.6	0.0	0.1	0.0	0.0	0.9	0.2	0.1	0.0	0.0	0.8	1.2	4	Y
Average	0.4	1.0	0.5	6.0	17.9	11.1	53.3	1.1	0.0	0.8	0.0	0.4	0.0	0.0	2.1	0.2	0.4	0.2	0.0	0.6	3.7	3.8	
100-260	0.7	0.5	1.4	26.7	5.9	4.6	11.9	12.9	0.1	1.0	0.0	0.2	0.2	0.1	6.0	0.2	0.0	0.2	0.0	0.0	26.9	5	Y
100-260	2.3	0.2	2.9	32.4	1.1	0.2	3.8	37.9	0.0	0.2	0.6	2.3	0.0	0.2	4.0	0.1	0.1	2.3	0.0	0.1	9.3	5	Y
100-260	0.8	0.4	2.3	51.2	1.4	0.4	10.3	4.3	0.0	0.1	0.0	0.2	0.0	0.0	2.7	0.1	0.1	0.9	0.0	0.6	24.2	5	Y
100-260	0.1	0.3	1.0	5.4	0.6	0.3	7.6	6.5	0.0	0.9	0.0	0.2	0.1	0.2	58.2	0.2	1.3	0.1	0.0	0.0	16.9	3	N
Average	1.0	0.3	1.9	28.9	2.2	1.4	8.4	15.4	0.0	0.5	0.2	0.7	0.1	0.1	17.7	0.2	0.4	0.9	0.0	0.2	19.3	4.5	
200-185	0.0	0.2	0.1	0.0	0.3	0.0	0.2	0.1	98.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	N
200-185	0.2	11.9	1.8	1.0	4.8	32.7	16.0	0.9	26.2	1.2	0.0	0.0	0.3	0.1	0.1	0.3	0.8	0.1	0.0	0.1	0.5	3	N
200-185	0.0	0.3	0.0	0.1	0.5	0.7	0.6	0.0	95.9	0.0	0.1	0.0	0.1	0.3	0.1	1.1	0.1	0.0	0.1	0.0	0.0	3	N
200-185	0.0	0.2	0.1	0.0	0.2	0.1	0.2	0.1	92.4	0.1	0.0	0.1	3.4	1.0	0.2	1.6	0.1	0.0	0.3	0.0	0.0	3	N
Average	0.1	3.2	0.5	0.3	1.4	8.4	4.3	0.3	78.3	0.4	0.0	0.0	1.0	0.4	0.1	0.7	0.3	0.0	0.1	0.0	0.2	3.0	
200-200	0.8	1.3	7.8	5.6	9.9	6.1	21.6	1.1	0.2	6.1	0.1	1.2	2.4	2.1	10.4	2.1	6.2	2.5	0.4	0.1	11.5	4	Y
200-200	12.6	0.9	2.5	1.0	7.4	19.1	34.8	2.7	0.4	0.7	0.1	0.2	0.2	1.3	2.3	2.4	2.1	0.6	0.3	1.9	6.4	4	Y
200-200	2.0	0.5	1.3	6.8	5.2	20.3	3.3	6.0	0.0	7.1	0.1	5.3	0.5	0.1	3.1	1.2	1.7	5.8	0.2	0.3	29.2	3	N
200-200	0.0	0.5	0.2	0.4	0.3	0.1	1.4	0.1	0.0	1.2	0.0	0.0	0.7	0.2	94.2	0.1	0.2	0.0	0.1	0.0	0.2	3	N
Average	3.9	0.8	3.0	3.4	5.7	11.4	15.3	2.5	0.1	3.8	0.1	1.7	1.0	0.9	27.5	1.5	2.6	2.2	0.3	0.6	11.8	3.5	
200-230	1.5	0.1	1.1	0.8	5.2	5.6	20.3	5.9	0.6	1.4	0.4	1.1	0.9	2.2	3.2	0.9	2.6	0.8	0.1	0.5	45.0	3	N
200-230	3.3	0.1	2.1	0.4	1.1	6.6	2.0	0.7	0.3	0.2	29.1	3.2	1.1	33.2	1.4	2.9	5.3	1.5	2.0	1.3	2.2	4	Y

200-230	0.1	0.0	0.1	0.1	0.1	0.3	5.5	0.2	0.0	0.2	1.1	0.6	5.2	23.0	29.3	2.6	7.4	0.8	0.9	2.0	20.5	3	N
200-230	1.1	0.0	0.6	0.7	1.1	5.4	14.0	0.5	0.0	0.0	28.8	3.7	0.9	16.1	3.6	8.4	1.1	8.3	0.3	1.1	3.7	4	Y
Average	1.5	0.0	1.0	0.5	1.8	4.5	10.5	1.9	0.2	0.4	14.8	2.2	2.0	18.6	9.4	3.7	4.1	2.9	0.8	1.2	17.8	3.5	
200-260	0.1	0.0	0.0	0.9	0.0	0.0	2.9	1.0	0.0	0.1	0.0	1.9	0.0	0.0	5.4	0.0	0.2	1.3	0.0	0.0	86.1	3	N
200-260	0.3	0.1	0.5	3.7	0.3	0.5	2.0	8.4	0.0	0.2	0.0	2.9	0.0	0.3	0.3	0.0	0.1	0.1	0.0	0.0	80.1	3	N
200-260	0.8	1.4	1.3	3.8	2.2	1.6	8.3	5.9	0.5	4.3	0.1	32.4	0.2	0.6	4.3	1.2	0.6	0.5	0.1	2.8	27.3	4	Y
200-260	5.3	0.3	0.8	23.6	4.0	0.7	0.7	9.8	0.3	0.4	1.9	24.9	0.1	10.9	2.0	0.0	0.1	2.7	0.1	0.2	11.1	4	Y
Average	1.6	0.5	0.7	8.0	1.6	0.7	3.5	6.3	0.2	1.2	0.5	15.5	0.1	2.9	3.0	0.3	0.3	1.1	0.1	0.8	51.2	3.5	
400-200	0.3	1.1	0.1	0.9	2.5	1.8	13.8	0.3	0.5	0.6	0.0	0.2	28.0	0.3	5.6	9.0	8.0	0.2	2.3	1.6	22.7	1	N
400-200	0.4	1.2	0.3	2.9	2.4	2.5	4.5	0.1	51.3	0.6	0.1	2.2	18.4	1.1	1.3	4.3	1.0	0.8	1.1	3.0	0.5	3	N
400-200	0.2	0.3	0.9	0.6	6.1	1.1	8.9	1.4	0.7	0.5	0.4	1.9	56.5	1.7	1.3	12.8	0.3	0.3	2.3	0.0	1.6	1	N
400-200	0.1	0.0	0.1	0.4	1.0	0.1	5.0	0.3	12.4	0.2	0.3	1.2	27.9	8.3	1.8	18.6	2.4	0.9	1.3	3.4	14.4	1	N
Average	0.2	0.7	0.4	1.2	3.0	1.4	8.0	0.5	16.2	0.5	0.2	1.4	32.7	2.9	2.5	11.2	2.9	0.5	1.7	2.0	9.8	1.5	
400-230	0.1	0.6	2.6	2.4	0.8	0.3	0.0	0.6	0.0	1.2	13.7	32.4	0.8	17.7	9.5	0.3	0.5	4.2	2.5	0.3	9.6	4	Y
400-230	0.0	0.1	0.5	0.2	1.4	0.2	2.9	1.3	0.3	0.6	0.0	0.3	0.3	6.9	27.8	0.3	3.0	1.1	0.1	2.4	50.3	3	N
400-230	1.4	0.1	0.4	1.0	1.1	0.6	1.7	0.8	0.2	0.1	0.2	0.7	0.2	49.5	1.2	0.3	6.6	3.7	0.3	10.9	19.0	2	N
400-230	0.0	0.0	0.7	1.4	15.5	0.1	0.8	3.0	0.6	0.1	3.5	0.3	0.4	68.3	0.4	2.0	0.2	1.0	0.6	0.0	1.1	2	N
Average	0.4	0.2	1.1	1.3	4.7	0.3	1.3	1.4	0.3	0.5	4.3	8.4	0.4	35.6	9.7	0.7	2.6	2.5	0.9	3.4	20.0	2.8	
400-260	0.5	0.1	0.0	0.1	0.1	0.0	0.7	0.7	0.0	4.1	0.0	0.0	0.0	0.0	74.9	0.1	0.0	0.1	0.0	0.0	18.4	3	N
400-260	0.2	1.3	0.9	2.1	10.0	3.4	1.9	7.9	0.1	6.2	0.2	0.9	0.3	0.0	32.3	1.8	0.5	1.1	0.1	0.0	28.7	3	N
400-260	0.4	0.1	0.2	1.1	0.9	0.3	6.3	2.7	0.0	3.9	0.2	1.8	0.1	0.2	65.1	1.0	2.0	2.4	0.1	0.1	11.4	3	N
400-260	0.0	0.0	0.1	0.1	0.9	0.1	0.2	0.5	0.2	1.0	0.0	0.1	0.3	0.1	88.0	1.2	0.9	0.1	0.6	0.0	5.6	3	N
Average	0.3	0.4	0.3	0.8	3.0	1.0	2.3	2.9	0.1	3.8	0.1	0.7	0.2	0.1	65.1	1.0	0.9	0.9	0.2	0.0	16.0	3.0	
800-200	0.4	0.1	0.4	0.1	0.3	0.7	1.2	0.2	0.3	0.1	0.3	0.1	1.9	0.1	0.8	91.9	0.3	0.2	0.3	0.1	0.1	1	N
800-200	0.0	0.1	0.1	0.0	0.0	0.1	0.8	0.1	0.1	0.1	0.0	0.1	1.4	0.0	2.8	92.9	0.1	0.1	0.9	0.1	0.1	1	N
800-200	0.3	0.0	0.0	0.4	0.6	2.1	35.7	2.1	0.2	0.4	0.0	1.0	3.3	0.5	3.1	9.0	8.9	0.8	0.3	7.4	23.9	4	Y
800-200	0.7	0.0	0.3	0.5	0.5	1.1	4.8	0.8	0.6	0.2	0.3	0.1	0.4	0.6	1.3	77.6	2.0	0.4	5.4	0.8	1.5	1	N
Average	0.4	0.1	0.2	0.3	0.4	1.0	10.6	0.8	0.3	0.2	0.2	0.3	1.7	0.3	2.0	67.9	2.8	0.4	1.7	2.1	6.4	1.8	
800-230	0.3	0.1	0.1	0.1	0.6	1.5	1.0	2.4	0.0	0.1	0.0	0.5	0.8	1.8	0.4	0.2	11.8	0.9	3.3	0.8	73.4	3	N
800-230	0.2	0.2	0.1	3.9	0.0	0.2	0.6	0.2	0.0	0.1	0.0	0.0	0.0	0.0	3.5	0.0	0.1	0.2	0.0	0.1	90.6	3	N
800-230	0.1	0.0	0.0	3.9	1.0	1.9	5.9	6.6	0.2	1.9	0.0	0.2	0.3	1.0	39.5	0.5	5.6	0.1	0.1	2.1	29.0	3	N
800-230	1.6	0.2	0.1	0.1	1.3	10.6	23.7	3.1	0.8	0.4	0.0	0.1	2.6	5.2	1.0	7.0	31.1	1.1	2.3	0.2	7.3	2	N
Average	0.5	0.1	0.1	2.0	0.7	3.5	7.8	3.1	0.3	0.6	0.0	0.2	0.9	2.0	11.1	1.9	12.2	0.6	1.4	0.8	50.1	2.8	
800-260	0.8	0.2	1.2	0.9	0.6	2.5	4.4	0.1	0.4	0.5	1.0	0.2	0.9	9.8	0.4	1.0	24.0	0.5	0.1	50.6	0.1	2	N
800-260	0.4	10.5	2.1	0.1	36.0	16.9	0.2	0.3	0.1	1.7	0.5	0.1	0.2	3.1	0.2	0.2	2.0	10.1	13.9	0.1	1.3	3	N

800-260	0.2	1.2	1.7	3.8	0.7	0.6	1.8	2.1	0.1	0.9	0.2	0.6	0.1	0.1	11.6	0.1	0.3	35.3	1.3	0.6	36.0	3	N
800-260	1.2	0.2	0.6	2.0	1.4	2.9	1.7	4.4	0.2	0.8	7.5	24.8	0.2	1.6	2.8	1.9	1.0	21.7	8.7	0.5	14.1	3	Y
Average	0.7	3.0	1.4	1.7	9.7	5.7	2.1	1.7	0.2	1.0	2.3	6.4	0.3	3.6	3.7	0.8	6.8	16.9	6.0	12.9	12.9	2.8	
900-200	0.2	0.0	0.1	0.3	0.6	0.9	4.7	0.5	0.0	0.1	0.5	3.6	0.5	8.9	1.2	6.0	17.7	21.6	9.9	4.3	18.3	3	N
900-200	0.1	0.2	0.3	0.2	10.7	4.8	0.8	0.3	0.4	0.7	0.0	1.3	1.1	0.4	1.5	17.5	1.6	6.0	19.4	0.2	32.5	3	N
900-200	0.3	0.2	3.0	1.0	22.4	4.4	13.5	0.6	1.5	0.2	2.0	0.1	1.4	3.1	3.0	1.0	1.3	2.4	31.4	4.2	3.2	1	N
900-200	0.2	1.2	2.0	5.6	25.9	14.2	14.2	0.5	1.2	0.1	0.4	1.1	3.6	1.3	0.6	16.1	0.2	0.3	10.0	0.3	1.2	3	N
Average	0.2	0.4	1.3	1.8	14.9	6.0	8.3	0.5	0.8	0.3	0.7	1.5	1.7	3.4	1.6	10.1	5.2	7.6	17.7	2.2	13.8	2.5	
999-230	0.3	0.8	2.0	2.1	7.3	23.1	16.3	2.1	0.4	2.1	1.0	19.2	1.2	0.5	2.5	10.9	1.4	0.7	0.5	2.1	3.4	3	N
999-230	0.0	0.0	0.0	0.0	0.0	0.0	6.2	0.1	0.9	0.0	0.0	0.1	0.7	0.4	3.5	1.8	0.5	0.2	0.0	81.1	4.5	2	N
999-230	0.1	0.1	0.1	0.7	0.6	0.8	11.8	0.4	24.4	4.1	0.1	1.5	1.3	0.4	31.4	2.2	0.4	0.4	0.1	9.1	9.9	3	N
999-230	0.1	0.0	0.1	0.0	0.5	0.2	42.1	1.3	0.7	0.2	0.1	0.2	0.1	0.8	2.9	1.2	2.1	0.5	0.9	37.5	8.5	4	N
Average	0.1	0.3	0.6	0.7	2.1	6.0	19.1	1.0	6.6	1.6	0.3	5.3	0.8	0.5	10.1	4.0	1.1	0.5	0.4	32.4	6.6	3.0	
999-260	0.2	0.1	0.0	0.1	0.3	0.2	0.0	1.2	0.0	0.3	0.0	0.2	0.0	0.0	3.2	0.0	0.1	3.7	0.1	0.0	90.2	3	N
999-260	0.0	0.1	0.3	0.4	0.0	0.0	0.1	0.7	0.0	0.0	0.0	0.1	0.0	0.0	2.8	0.0	0.1	0.4	0.0	0.0	94.8	3	N
999-260	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.1	0.0	0.3	0.0	0.0	0.4	0.0	0.0	0.1	0.0	0.0	98.5	3	N
999-260	0.0	0.0	0.0	0.0	0.0	0.0	0.4	1.4	0.0	0.1	0.0	0.0	0.0	0.0	10.9	0.0	0.0	0.1	0.0	0.0	86.9	3	N
Average	0.1	0.0	0.1	0.1	0.1	0.1	0.1	0.9	0.0	0.1	0.0	0.2	0.0	0.0	4.3	0.0	0.1	1.0	0.0	0.0	92.6	3.0	

Table 11: Actual Quality Classification							
		Printing Speeds					
		50	100	200	400	800	1000
Temperature	185	3	3	3	1	1	1
	200	3	3	3	1	1	1
	230	4	4	4	2	2	2
	260	5	5	4	3	3	3

Table 12: Predicted Quality classification							
		Printing Speeds					
		50	100	200	400	800	1000
Temperature	185	3	3	3	1	1	1
	200	3	3	3.5	1.5	1.8	2.5
	230	4	3.8	3	2.8	2.8	3
	260	5	4	3.5	3	3	3

APPENDIX B

:

Model's Source Code

Using Inception-V3

```
# retrain.py
# original file by Google:
"""
```

Transfer learning with Inception v3 or Mobilenet models. This example shows how to take a Inception v3 or Mobilenet model trained on ImageNet images, and train a new top layer that can recognize other classes of images.

*The top layer receives as input a 2048-dimensional vector (1001-dimensional for Mobilenet) for each image. We train a softmax layer on top of this representation. Assuming the softmax layer contains N labels, this corresponds to learning N + 2048*N (or 1001*N) model parameters corresponding to the learned biases and weights.*

You can replace the image_dir argument with any folder containing subfolders of images. The label for each image is taken from the name of the subfolder it's in.

This produces a new model file that can be loaded and run by any TensorFlow program, for example the label_image sample code.

To use with TensorBoard:
tensorboard --logdir /path/to/tensorboard_logs
"""

```
from datetime import datetime
import hashlib
import os
import os.path
import random
import re
import sys
import tarfile

import numpy as np
```

```
from six.moves import urllib
import tensorflow as tf

from tensorflow.contrib.quantize.python import quant_ops
from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat

# module level variables
##############################################################################
MIN_NUM_IMAGES_REQUIRED_FOR_TRAINING = 10
MIN_NUM_IMAGES_SUGGESTED_FOR_TRAINING = 100

MIN_NUM_IMAGES_REQUIRED_FOR_TESTING = 3

MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M

# path to folders of labeled images
TRAINING_IMAGES_DIR = os.getcwd() + '/training_images'

TEST_IMAGES_DIR = os.getcwd() + "/test_images/"

# where to save the trained graph
OUTPUT_GRAPH = os.getcwd() + '/' + 'retrained_graph.pb'

# where to save the intermediate graphs
INTERMEDIATE_OUTPUT_GRAPHS_DIR = os.getcwd() + '/intermediate_graph'

# how many steps to store intermediate graph, if "0" then will not store
INTERMEDIATE_STORE_FREQUENCY = 0

# where to save the trained graph's labels
OUTPUT_LABELS = os.getcwd() + '/' + 'retrained_labels.txt'

# where to save summary logs for TensorBoard
TENSORBOARD_DIR = os.getcwd() + '/' + 'tensorboard_logs'

# how many training steps to run before ending
```

```
# NOTE: original Google default is 4000, use 4000 (or possibly higher) for production grade results
HOW_MANY_TRAINING_STEPS=9000

# how large a learning rate to use when training
LEARNING_RATE = 0.01

# what percentage of images to use as a test set
TESTING_PERCENTAGE = 10

# what percentage of images to use as a validation set
VALIDATION_PERCENTAGE = 10

# how often to evaluate the training results
EVAL_STEP_INTERVAL = 30

# how many images to train on at a time
TRAIN_BATCH_SIZE = 32

# How many images to test on. This test set is only used once, to evaluate the final accuracy of the model after
# training completes.  A value of -1 causes the entire test set to be used, which leads to more stable results across runs.
TEST_BATCH_SIZE = -1

# How many images to use in an evaluation batch. This validation set is used much more often than the test set, and is an early indicator of how
# accurate the model is during training. A value of -1 causes the entire validation set to be used, which leads to
# more stable results across training iterations, but may be slower on large training sets.
VALIDATION_BATCH_SIZE = -1

# whether to print out a list of all misclassified test images
PRINT_MISCLASSIFIED_TEST_IMAGES = False

# Path to classify_image_graph_def.pb, imagenet_synset_to_human_label_map.txt, and imagenet_2012_challenge_label_map_proto.pbtxt
MODEL_DIR = os.getcwd() + "/" + "model"

# Path to cache bottleneck layer values as files
```

```
BOTTLENECK_DIR = os.getcwd() + '/' + 'bottleneck_data'

# the name of the output classification layer in the retrained graph
FINAL_TENSOR_NAME = 'final_result'

# whether to randomly flip half of the training images horizontally
FLIP_LEFT_RIGHT = False

# a percentage determining how much of a margin to randomly crop off the training images
RANDOM_CROP = 0

# a percentage determining how much to randomly scale up the size of the training images by
RANDOM_SCALE = 0

# a percentage determining how much to randomly multiply the training image input pixels up or down by
RANDOM_BRIGHTNESS = 0

# Which model architecture to use. 'inception_v3' is the most accurate, but also the slowest. For faster or smaller models, chose a MobileNet with
# the form 'mobilenet_<parameter size>_<input_size>[_quantized]'. For example, 'mobilenet_1.0_224' will pick a model that is 17 MB in size and takes
# 224 pixel input images, while 'mobilenet_0.25_128_quantized' will choose a much less accurate, but smaller and faster network that's 920 KB
# on disk and takes 128x128 images. See https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html for more information on Mobilenet.

# By default this script will use the high accuracy, but comparatively large and slow Inception v3 model architecture.
# It's recommended that you start with this to validate that you have gathered good training data, but if you want to deploy
# on resource-limited platforms, you can try the `--architecture` flag with a Mobilenet model. For example:

# example command to run floating-point version of mobilenet:
# ARCHITECTURE = 'mobilenet_1.0_224'
```

```
# example command to run quantized version of mobilenet:
# ARCHITECTURE = 'mobilenet_1.0_224_quantized'

# There are 32 different Mobilenet models to choose from, with a variety of file size and latency options. The first
# number can be '1.0', '0.75', '0.50', or '0.25' to control the size, and the second controls the input image size,
# either '224', '192', '160', or '128', with smaller sizes running faster.
# See https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html for more information on Mobilenet.
ARCHITECTURE = 'inception_v3'

######################################################################
def main():
    print("starting program . . .")
    # make sure the logging output is visible, see https://github.com/tensorflow/tensorflow/issues/3047
    tf.logging.set_verbosity(tf.logging.INFO)

    if not checkIfNecessaryPathsAndFilesExist():
        return
    # end if

    # prepare necessary directories that can be used during training
    prepare_file_system()

    # Gather information about the model architecture we'll be using.
    model_info = create_model_info(ARCHITECTURE)
    if not model_info:
        tf.logging.error('Did not recognize architecture flag')
        return -1
    # end if

    # download the model if necessary, then create the model graph
    print("downloading model (if necessary) . . .")
    downloadModelIfNotAlreadyPresent(model_info['data_url'])
    print("creating model graph . . .")
    graph, bottleneck_tensor, resized_image_tensor = (create_model_graph(model_info))

    # Look at the folder structure, and create lists of all the images.
```

```
   print("creating image lists . . .")
   image_lists = create_image_lists(TRAINING_IMAGES_DIR,
TESTING_PERCENTAGE, VALIDATION_PERCENTAGE)
   class_count = len(image_lists.keys())
   if class_count == 0:
      tf.logging.error('No valid folders of images found at ' +
TRAINING_IMAGES_DIR)
      return -1
   # end if
   if class_count == 1:
      tf.logging.error('Only one valid folder of images found at ' +
TRAINING_IMAGES_DIR + ' - multiple classes are needed for classification.')
      return -1
   # end if

   # determinf if any of the distortion command line flags have been set
   doDistortImages = False
   if (FLIP_LEFT_RIGHT == True or RANDOM_CROP != 0 or RANDOM_SCALE !=
0 or RANDOM_BRIGHTNESS != 0):
      doDistortImages = True
   # end if

   print("starting session . . .")
   with tf.Session(graph=graph) as sess:
      # Set up the image decoding sub-graph.
      print("performing jpeg decoding . . .")
      jpeg_data_tensor, decoded_image_tensor =
add_jpeg_decoding( model_info['input_width'],
                                          model_info['input_height'],
                                          model_info['input_depth'],
                                          model_info['input_mean'],
                                          model_info['input_std'])
      print("caching bottlenecks . . .")
      distorted_jpeg_data_tensor = None
      distorted_image_tensor = None
      if doDistortImages:
         # We will be applying distortions, so setup the operations we'll need.
         (distorted_jpeg_data_tensor, distorted_image_tensor) =
add_input_distortions(FLIP_LEFT_RIGHT, RANDOM_CROP, RANDOM_SCALE,
```

```
 RANDOM_BRIGHTNESS, model_info['input_width'],
model_info['input_height'], model_info['input_depth'],
model_info['input_mean'], model_info['input_std'])
      else:
         # We'll make sure we've calculated the 'bottleneck' image summaries and
         # cached them on disk.
         cache_bottlenecks(sess, image_lists, TRAINING_IMAGES_DIR,
BOTTLENECK_DIR, jpeg_data_tensor, decoded_image_tensor,
                           resized_image_tensor, bottleneck_tensor, ARCHITECTURE)
      # end if

      # Add the new layer that we'll be training.
      print("adding final training layer . . .")
      (train_step, cross_entropy, bottleneck_input, ground_truth_input, final_tensor) =
add_final_training_ops(len(image_lists.keys()),  FINAL_TENSOR_NAME,
 bottleneck_tensor, model_info['bottleneck_tensor_size'], model_info['quantize_layer'])
      # Create the operations we need to evaluate the accuracy of our new layer.
      print("adding eval ops for final training layer . . .")
      evaluation_step, prediction = add_evaluation_step(final_tensor, ground_truth_input)

      # Merge all the summaries and write them out to the tensorboard_dir
      print("writing TensorBoard info . . .")
      merged = tf.summary.merge_all()
      train_writer = tf.summary.FileWriter(TENSORBOARD_DIR + '/train', sess.graph)
      validation_writer = tf.summary.FileWriter(TENSORBOARD_DIR + '/validation')

      # Set up all our weights to their initial default values.
      init = tf.global_variables_initializer()
      sess.run(init)

      # Run the training for as many cycles as requested on the command line.
      print("performing training . . .")
      for i in range(HOW_MANY_TRAINING_STEPS):
         # Get a batch of input bottleneck values, either calculated fresh every
         # time with distortions applied, or from the cache stored on disk.
         if doDistortImages:
            (train_bottlenecks, train_ground_truth) =
get_random_distorted_bottlenecks(sess, image_lists, TRAIN_BATCH_SIZE, 'training',
TRAINING_IMAGES_DIR, distorted_jpeg_data_tensor,  distorted_image_tensor,
resized_image_tensor, bottleneck_tensor)
```

```
        else:
            (train_bottlenecks, train_ground_truth, _) =
get_random_cached_bottlenecks(sess, image_lists, TRAIN_BATCH_SIZE, 'training',
 BOTTLENECK_DIR, TRAINING_IMAGES_DIR, jpeg_data_tensor,
decoded_image_tensor, resized_image_tensor, bottleneck_tensor, ARCHITECTURE)
        # end if

        # Feed the bottlenecks and ground truth into the graph, and run a training
        # step. Capture training summaries for TensorBoard with the `merged` op.
        train_summary, _ = sess.run([merged, train_step], feed_dict={bottleneck_input:
train_bottlenecks, ground_truth_input: train_ground_truth})
        train_writer.add_summary(train_summary, i)

        # Every so often, print out how well the graph is training.
        is_last_step = (i + 1 == HOW_MANY_TRAINING_STEPS)
        if (i % EVAL_STEP_INTERVAL) == 0 or is_last_step:
            train_accuracy, cross_entropy_value = sess.run([evaluation_step,
cross_entropy], feed_dict={bottleneck_input: train_bottlenecks, ground_truth_input:
train_ground_truth})
            tf.logging.info('%s: Step %d: Train accuracy = %.1f%%' % (datetime.now(), i,
train_accuracy * 100))
            tf.logging.info('%s: Step %d: Cross entropy = %f' % (datetime.now(), i,
cross_entropy_value))
            validation_bottlenecks, validation_ground_truth, _ =
(get_random_cached_bottlenecks(sess, image_lists, VALIDATION_BATCH_SIZE,
'validation', BOTTLENECK_DIR, TRAINING_IMAGES_DIR, jpeg_data_tensor,
decoded_image_tensor, resized_image_tensor, bottleneck_tensor, ARCHITECTURE))
            # Run a validation step and capture training summaries for TensorBoard with
the `merged` op.
            validation_summary, validation_accuracy = sess.run(
                [merged, evaluation_step], feed_dict={bottleneck_input:
validation_bottlenecks, ground_truth_input: validation_ground_truth})
            validation_writer.add_summary(validation_summary, i)
            tf.logging.info('%s: Step %d: Validation accuracy = %.1f%% (N=%d)' %
(datetime.now(), i, validation_accuracy * 100, len(validation_bottlenecks)))
        # end if

        # Store intermediate results
        intermediate_frequency = INTERMEDIATE_STORE_FREQUENCY
```

```
            if (intermediate_frequency > 0 and (i % intermediate_frequency == 0) and i > 0):
                intermediate_file_name = (INTERMEDIATE_OUTPUT_GRAPHS_DIR +
'intermediate_' + str(i) + '.pb')
                tf.logging.info('Save intermediate result to : ' + intermediate_file_name)
                save_graph_to_file(sess, graph, intermediate_file_name)
            # end if
        # end for

        # We've completed all our training, so run a final test evaluation on some new
images we haven't used before
        print("running testing . . .")
        test_bottlenecks, test_ground_truth, test_filenames =
(get_random_cached_bottlenecks(sess, image_lists, TEST_BATCH_SIZE, 'testing',
BOTTLENECK_DIR,  TRAINING_IMAGES_DIR, jpeg_data_tensor,
decoded_image_tensor, resized_image_tensor,  bottleneck_tensor, ARCHITECTURE))
        test_accuracy, predictions = sess.run([evaluation_step, prediction],
feed_dict={bottleneck_input: test_bottlenecks, ground_truth_input: test_ground_truth})
        tf.logging.info('Final test accuracy = %.1f%% (N=%d)' % (test_accuracy * 100,
len(test_bottlenecks)))

        if PRINT_MISCLASSIFIED_TEST_IMAGES:
            tf.logging.info('=== MISCLASSIFIED TEST IMAGES ===')
            for i, test_filename in enumerate(test_filenames):
                if predictions[i] != test_ground_truth[i]:
                    tf.logging.info('%70s  %s' % (test_filename, list(image_lists.keys())
[predictions[i]]))
                # end if
            # end for
        # end if

        # write out the trained graph and labels with the weights stored as constants
        print("writing trained graph and labbels with weights")
        save_graph_to_file(sess, graph, OUTPUT_GRAPH)
        with gfile.FastGFile(OUTPUT_LABELS, 'w') as f:
            f.write('\n'.join(image_lists.keys()) + '\n')
        # end with

        print("done !!")
# end function
```

APPENDIX C

:

Model's Source Code

Using Keras

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
# Initialising the CNN
classifier = Sequential()
# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Step 3 - Flattening
classifier.add(Flatten())
# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
# Part 2 - Fitting the CNN to the images
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory('training_set',
target_size = (64, 64),
batch_size = 32,
class_mode = 'binary')
test_set = test_datagen.flow_from_directory('test_set',
target_size = (64, 64),
batch_size = 32,
class_mode = 'binary')

classifier.fit_generator(training_set,
steps_per_epoch = 240,
```

```
epochs = 10,
validation_data = test_set,
validation_steps = 120)
# Part 3 - Making new predictions

from os import listdir
from os.path import isfile, join

dirName = 'single_prediction/'
fileNames = [f for f in listdir(dirName) if isfile(join(dirName, f))]

import numpy as np
from keras.preprocessing import image
for files in fileNames:
    #print('single_prediction/'+files)
    test_image = image.load_img('single_prediction/'+files, target_size = (64, 64))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis = 0)
    result = classifier.predict(test_image)
    #print(test_image)
    #print(result)

    training_set.class_indices
    if result[0][0] == 1:
        prediction = 'Perfect'
    else:
        prediction = 'Broken'
    print(files + ':' + prediction)
```

APPENDIX D

:

Model's Source Code

Using Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
%matplotlib inline

def load_dataset():
    train_dataset = h5py.File("B_P_train.h5", "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels
    test_dataset = h5py.File("B_P_test.h5", "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels
    classes = ['Perfect','Broken'] # the list of classes
    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))
    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes

# Loading the data (cat/non-cat)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
# Example of a picture
index = 3
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:, index]) + ", it's a '" + classes[np.squeeze(train_set_y[:,
index])] +  "' picture.")
m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = test_set_x_orig.shape[1]
print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("Height/Width of each image: num_px = " + str(num_px))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_set_x shape: " + str(train_set_x_orig.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x shape: " + str(test_set_x_orig.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
# Reshape the training and test examples
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
```

```
test_set_x_flatten =  test_set_x_orig.reshape(test_set_x_orig.shape[0], -1).T
print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
train_set_x = train_set_x_flatten/255.
test_set_x = test_set_x_flatten/255.
# GRADED FUNCTION: sigmoid

def sigmoid(z):
    """
    Compute the sigmoid of z
    Arguments:
    z -- A scalar or numpy array of any size.
    Return:
    s -- sigmoid(z)
    """
   s = 1/(1 + np.exp(-z))
  return s
print ("sigmoid([0, 2]) = " + str(sigmoid(np.array([0,2]))))
# GRADED FUNCTION: initialize_with_zeros

def initialize_with_zeros(dim):
    """
    This function creates a vector of zeros of shape (dim, 1) for w and initializes b to 0.
    Argument:
    dim -- size of the w vector we want (or number of parameters in this case)
    Returns:
    w -- initialized vector of shape (dim, 1)
    b -- initialized scalar (corresponds to the bias)
    """
    w = np.zeros((dim, 1))
    b = 0
    assert(w.shape == (dim, 1))
    assert(isinstance(b, float) or isinstance(b, int))
    return w, b

dim = 2
w, b = initialize_with_zeros(dim)
```

```
print ("w = " + str(w))
print ("b = " + str(b))

# GRADED FUNCTION: propagate
def propagate(w, b, X, Y):
    """
    Implement the cost function and its gradient for the propagation explained above
    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat) of size (1, number of
examples)
    Return:
    cost -- negative log-likelihood cost for logistic regression
    dw -- gradient of the loss with respect to w, thus same shape as w
    db -- gradient of the loss with respect to b, thus same shape as b
    Tips:
    - Write your code step by step for the propagation. np.log(), np.dot()
    """
    m = X.shape[1]
    # FORWARD PROPAGATION (FROM X TO COST)
    A = sigmoid(np.dot(w.T, X) + b)                                    # compute activation
    cost = -1./m* np.sum(Y*np.log(A) + (1-Y)*np.log(1-A))                                 #
compute cost
     # BACKWARD PROPAGATION (TO FIND GRAD)
    dw = 1./m*np.dot(X, (A-Y).T)
    db = 1./m*np.sum(A-Y)
    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    #cost = np.squeeze(cost)
    #assert(cost.shape == ())
    grads = {"dw": dw,
             "db": db}
        return grads, cost
w, b, X, Y = np.array([[1],[2]]), 2, np.array([[1,2],[3,4]]), np.array([[1,0]])
grads, cost = propagate(w, b, X, Y)
print ("dw = " + str(grads["dw"]))
print ("db = " + str(grads["db"]))
print ("cost = " + str(cost))
```

```
# GRADED FUNCTION: optimize
def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):
    """
    This function optimizes w and b by running a gradient descent algorithm
    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of shape (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat), of shape (1, number of examples)
    num_iterations -- number of iterations of the optimization loop
    learning_rate -- learning rate of the gradient descent update rule
    print_cost -- True to print the loss every 100 steps
    Returns:
    params -- dictionary containing the weights w and bias b
    grads -- dictionary containing the gradients of the weights and bias with respect to the cost function
    costs -- list of all the costs computed during the optimization, this will be used to plot the learning curve.
    Tips:
    You basically need to write down two steps and iterate through them:
        1) Calculate the cost and the gradient for the current parameters. Use propagate().
        2) Update the parameters using gradient descent rule for w and b.
    """
    costs = []
    for i in range(num_iterations):
        # Cost and gradient calculation (≈ 1-4 lines of code)
        grads, cost =propagate(w, b, X, Y)
        # Retrieve derivatives from grads
        dw = grads["dw"]
        db = grads["db"]
        # update rule (≈ 2 lines of code)
        w = w - learning_rate * dw
        b = b - learning_rate * db
        # Record the costs
        if i % 100 == 0:
            costs.append(cost)
        # Print the cost every 100 training examples
        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))
```

```
    params = {"w": w,
              "b": b}
    grads = {"dw": dw,
             "db": db}
    return params, grads, costs
params, grads, costs = optimize(w, b, X, Y, num_iterations= 100, learning_rate = 0.009,
print_cost = False)
print ("w = " + str(params["w"]))
print ("b = " + str(params["b"]))
print ("dw = " + str(grads["dw"]))
print ("db = " + str(grads["db"]))
# GRADED FUNCTION: predict
def predict(w, b, X):
    '''
    Predict whether the label is 0 or 1 using learned logistic regression parameters (w, b)
    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)
    Returns:
    Y_prediction -- a numpy array (vector) containing all predictions (0/1) for the
examples in X
    '''
    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0], 1)
    # Compute vector "A" predicting the probabilities of a cat being present in the picture
     A =   sigmoid(np.dot(w.T, X) + b)
   for i in range(A.shape[1]):
        # Convert probabilities A[0,i] to actual predictions p[0,i]
        if A[0, i] > 0.5:
            Y_prediction[0, i] = 1
        else:
            Y_prediction[0, i] = 0
    assert(Y_prediction.shape == (1, m))
    return Y_prediction
print ("predictions = " + str(predict(w, b, X)))
# GRADED FUNCTION: model
def model(X_train, Y_train, X_test, Y_test, num_iterations, learning_rate, print_cost):
```

```
    """
    Builds the logistic regression model by calling the function you've implemented previously
    Arguments:
    X_train -- training set represented by a numpy array of shape (num_px * num_px * 3, m_train)
    Y_train -- training labels represented by a numpy array (vector) of shape (1, m_train)
    X_test -- test set represented by a numpy array of shape (num_px * num_px * 3, m_test)
    Y_test -- test labels represented by a numpy array (vector) of shape (1, m_test)
    num_iterations -- hyperparameter representing the number of iterations to optimize the parameters
    learning_rate -- hyperparameter representing the learning rate used in the update rule of optimize()
    print_cost -- Set to true to print the cost every 100 iterations
    Returns:
    d -- dictionary containing information about the model.
    """
    # initialize parameters with zeros (≈ 1 line of code)
    w, b = initialize_with_zeros(X_train.shape[0])
    # Gradient descent (≈ 1 line of code)
    parameters, grads, costs = optimize(w, b, X_train, Y_train, num_iterations, learning_rate, print_cost = print_cost)
    # Retrieve parameters w and b from dictionary "parameters"
    w = parameters["w"]
    b = parameters["b"]
    # Predict test/train set examples (≈ 2 lines of code)
    Y_prediction_test = predict(w, b, X_test)
    Y_prediction_train = predict(w, b, X_train)
    # Print train/test Errors
    print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test)) * 100))
    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train" : Y_prediction_train,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
```

```
        "num_iterations": num_iterations}
        return d
d = model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 9000,
learning_rate = 0.001, print_cost = True)
```