AUTOMATED SURFACE FINISH INSPECTION USING CONVOLUTIONAL NEURAL NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Wafa Louhichi

In Partial Fulfillment
of the Requirements for the
Masters Degree in the
School of Computational Science and Engineering

Georgia Institute of Technology

May 2019

AUTOMATED SURFACE FINISH INSPECTION USING CONVOLUTIONAL NEURAL NETWORKS

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisors Dr. Thomas Kurfess and Dr. Richard Vuduc for their continuous support, patience, motivation, enthusiasm, and immense knowledge. I would've never learnt and grown as much without them.

I would also like to thank my lab mates for their help and support through hard times, especially Michael Isaev and Kyle Saleeby.

I would like to thank my parents and my sisters for their trans-Atlantic support.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

The surface finish of a machined part has an important effect on friction, wear, and aesthetics. The surface finish became a critical quality measure since 1980s mainly due to demands from automotive industry. Visual inspection and quality control have been traditionally done by human experts. Normally, it takes a substantial amount of operators time to stop the process and compare the quality of the produced piece with a surface roughness gauge. This manual process does not guarantee a consistent quality of the surface and is subject to human error and dependent upon the subjective opinion of the expert. Current advances in image processing, computer vision, and machine learning have created a path towards an automated surface finish inspection increasing the automation level of the whole process even further than it is now. In this thesis work, we propose a deep learning approach to replicate human judgment without using a surface roughness gauge. We used a Convolutional Neural Network (CNN) to train a surface finish classifier. Because of data scarcity, we generated our own image dataset of aluminum pieces produced from turning and boring operations on a Computer Numerical Control (CNC) lathe, which consists of a total of 980 training images, 160 validation images, and 140 test images. Considering the limited dataset and the computational cost of training deep neural networks from scratch, we applied transfer learning technique to models pre-trained on the publicly available ImageNet benchmark dataset. We used PyTorch Deep Learning framework and both CPU and GPU to train ResNet18 CNN. The training on CPU took $1h21min55s$ with a test accuracy of 97.14% while the training on GPU took $1min47s$ with a test accuracy = 97.86%. We used Keras API that runs on top of TensorFlow to train a MobileNet model. The training using Colaboratory's GPU took $1h32m14s$ with an accuracy of 98.57%. The deep CNN models provided surprisingly high accuracy missclassifying only a few of 140 testing images. The MobileNet model allowed to run the inference efficiently on mobile devices. The affordable and easy-to-use solution provides a viable new method of automated surface in-

spection systems (ASIS).

CHAPTER 1
INTRODUCTION

Surface roughness is a technical requirement of manufactured parts and is a widely used index of product quality. Achieving the desired surface quality is of great importance for the customer satisfaction, aesthetics and the functional behavior of a part. The surface roughness formation mechanism depends on multiple machining conditions which makes inspecting and predicting the surface finish a complex task. The influence of cutting conditions of machining operation on surface roughness, including the cutting speed, feed rate, depth of cut, tool geometry, choice of coolant, rigidity of workbench, and fixtures, have been reported in [1] [2]. Multiple efforts have been directed to predicting the surface roughness and automating its inspection using computer vision, statistical methods and machine learning. The majority of the work has been focusing on steel as it is the material of choice for a large number and very diverse industrial applications particularly, cold steel strip surfaces which is most sensitive to customers' requirements. Traditionally, surface finish of flat steel products is assessed manually by cutting about $30m$ of a random coil and inspected by an expert. This constitutes typically about 0.05% of the total steel surface produced [3]. Due to human error and the amount of production, the manual inspection process is not sufficient to guarantee a defect-free surface of steel products with reasonable degree of confidence, and thus, the need for automated surface inspection grew. An intelligent surface finish inspection system will give machine operators guidance in selecting the best combination of cutting conditions (i.e. spindle speed, feed rate, and depth of cut) for a specific process. From later half of 1980s, systematic research work on surface inspection of steel products started. Automated surface inspection techniques can be categorized as direct and indirect contact methods. The direct contact methods require a direct contact with the surface to be investigated using stylus instruments (surface roughness profilometer). Using stylus instruments is a slow procedure and has limited flexibility in handling the different geometrical parts. Thus, the direct contact methods are not suitable for fast and large scale manufacturing processes. For indirect contact methods, previous research involved a hardware system set-up composed of multiple cameras, acquisition systems and used computer vision and image processing techniques to assess the surface finish. These methods are not easy to deploy as they usually require particular set-up: consistent lighting, angles of cameras, multiple sources of data and special hardware that is not resistant to the hazardous machine shop conditions: dust, oil, coolant etc ...

In this study the aim is to make the process of automating surface finish inspection as easy and affordable as possible using only an available tool that every mechanic has; mobile devices. Due to unavailable dataset of surface finish images, we create our own dataset. Two OKUMA Genos L250 CNC (Computer Numeric Controlled) Lathes were used for production of aluminum parts with varying surface finishes using facing and turning operations. Our dataset is made of pictures of the machined parts taken using multiple mobile devices cameras. The advances in deep learning and mainly the results shown by Convolutional Neural Networks (CNN) in image classification and object detection allowed for this automated surface finish quality inspection. The model is trained to perform a binary classification of the images based on the surface finish to provide a result of whether the picture corresponds to a good or bad surface finish. Experts opinion was provided to determine which parts corresponded to good or bad surfaces to label the dataset that is then splitted for training, validation and testing. We use transfer learning techniques on pre-trained models for training our CNN. ResNet18 as fixed feature extractor achieved 97.86% accuracy on the test dataset. In order to deploy our model on mobile devices, we use a pre-trained MobileNet model that overcomes the limitations of running such complex models on limited computation resources, power and storage environment. We add two dense layers to the MobileNet model, and one last fully connected layer with softmax activation to output the two classes probabilities. The first 20 layers are frozen, and the rest of the model is trained to correctly classify 98.57% of the test images. The model is used to develop an android app which allows for the classification of surface finish images task to run locally on mobile devices. Our approach performs well to replicate human judgement of surface finish, using uniquely images of the surfaces of the parts as input. This thesis work provides thus a novel approach to automate surface finish inspection providing a cheap, flexible and reliable system suitable for machine shop environment requiring only a mobile device camera.

The thesis starts with a review of the literature and computer-vision based techniques used for automated surface inspection, then gives an overview of CNNs, as well as the transfer learning technique used in this work. Then, we describe the experimental set-up and data collection, the training and testing of the models, and the development of an android app to perform the surface finish quality inspection locally on mobile devices. Finally recommendations for future improvements are provided.

CHAPTER 2
BACKGROUND

Surface Finish is a measure of the overall texture of a surface that is characterized by the lay, surface roughness, and waviness of the surface. The surface finish is most of the time quantified by the deviations in the direction of the normal vector of a real surface from its ideal form. If these deviations are large, the surface is rough; if they are small, the surface is smooth. Most surface finish requirements are noted in arithmetic average deviation of the roughness profile from its mean line noted R_a, expressed as: $R_a = \frac{1}{n}\sum_{i=1}^{n}|y_i|$ where y_i is the vertical distance from the mean line to the i^{th} data point [4] as shown in Figure 2.1 [5].

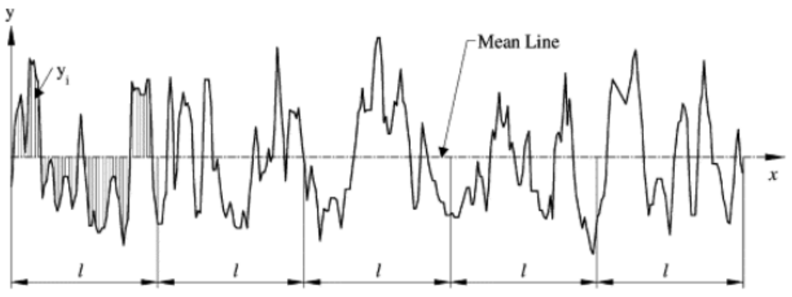


Figure 2.1: Definition of the arithmetic average height R_a.

Consequently, most of surface finish inspection methods target measuring the R_a to decide if the manufactured part satisfies the required quality features. Considering the importance of surface roughness for many fundamental problems such as friction, heat and electricity conduction, tightness and contact joints and positional accuracy etc..., it has been the subject of multiple studies. Traditional stylus techniques have been used to measure the surface roughness, however, due to the limitations of this method, many advanced and sophisticated techniques have been developed. While some research has been made in inspecting and measuring the surface roughness, a lot of research effort has also been done in predicting the surface finish, detecting and classifying defects. The related work in this thesis focuses on techniques based on computer vision used for inspecting the surface finish by measuring the R_a. The most common methods in the literature are reported below.

2.1 Indirect Contact Computer Vision Methods for Surface Finish Inspection

2.1.1 Laser Speckle Images

Earlier optical methods for surface roughness used laser images obtained through a microscope for texture analysis. Speckle images are obtained when a rough surface is illuminated with a partially coherent light forming some random patterns of bright and dark regions. The intensity at a point is the interference of wavelets scattered from different points within the illuminated area, with the phases randomised due to the variations of the surface. An example of a laser speckle image is shown in Figure 2.2 [6].

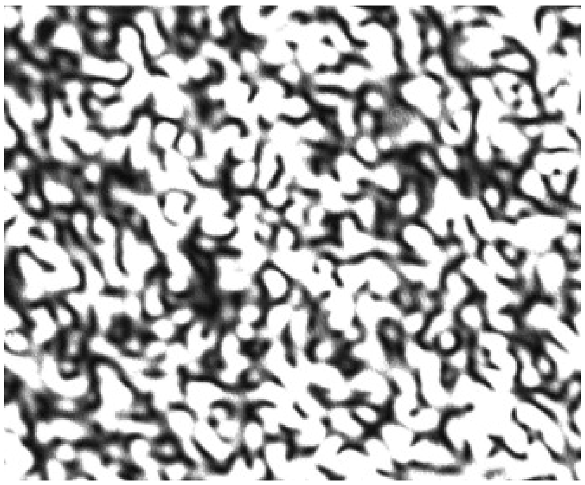


Figure 2.2: Example of speckle image.

Speckle images can be used for surface roughness measurement [6]. A linear relationship between the surface roughness and the standard deviations of the intensity fluctuations of the speckle image was found in Fujii and Asakura [7].

2.1.2 Images Matching using Distance Metrics

T. Jeyapoovan and M. Murugan [8] used the Euclidean and Hamming distances between test images and reference images, inspired by biometric recognition, to calculate the surface roughness. Six parts were machined using a milling operation with different surface roughness ranging between $0.8 - 2.6\mu m$, which is the common range of surface roughness values obtained in milling operations. The average surface roughness (R_a) of these parts were measured using a stylus instrument and recorded. The corresponding surfaces were photographed using a low-incident-angle CCD camera and polychromatic light source. The pictures of the surfaces were normalized to get a pixel intensity in gray scale of the range between 0 and 255, transformed to a 1-D vector and stored in a database to serve as a reference. A database of 6 reference images and their corresponding surface roughness was thus established. To characterize the surface roughness of a new test surface, the Euclidean and Hamming distances between the new test surface and the reference images in the database are calculated. The Euclidean distance (DE) for an n-dimensional space is given by:

$$D_{E(p,q)} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_N - q_N)^2} = \sqrt{\sum_{i=1}^{N}(p_i - q_i)^2} \tag{2.1}$$

where N is the dimension of the feature vector, p_i is the i^{th} value of the feature vector and q_i is the i^{th} value of the reference vector. The lowest Euclidean distance between the reference images and the test image indicates a matching and the corresponding roughness R_a in the database is attributed to the test surface. The Hamming distance represents the number of components that differ in value at corresponding areas of two images. The Hamming distance (D_H) is calculated as follows:

$$D_{H(p,q)} = \frac{1}{N}\sum_{i=1}^{N}(p_i \neq q_i) \tag{2.2}$$

Similarly to the Euclidean distance, the lowest Hamming distance is used to characterize the surface roughness. Six images of parts were used for testing, and it was found that a 0.3 in the value of Hamming distance between the test image and the reference images is a threshold; it is less than 0.30 for the correct matching, it was greater than 0.30 for the other 5 measures. For all the six test images, the surface roughness obtained by matching the test image with the one with the lowest Euclidean distance, is the same as the one obtained with the Hamming distance. Thus, using the Euclidean and Hamming distances to match a new image with an established database of different surface roughness provides good results. It was also observed that the Hamming distance was closer to zero than the Euclidean distance for perfect matching which indicates that using the Hamming distance for surface roughness characterization has larger scope than the Euclidean distance.

2.1.3 Gray Level Co-occurrence Matrix

The the gray level co-occurrence matrix(GLCM) defined by Haralick et al. [9], is a widely used method in texture analysis. The GLCM matrix is a 2-D matrix of the size the number of gray levels in an image. It is constructed by calculating how often a pixel with the intensity (gray-level) value i occurs in a specific spatial relationship to a pixel with the value j. This method is used in Gadelmawla [10] to characterize the surface roughness of 10 machined samples obtained by a facing operation with different feed rates to vary the arithmetic average roughness R_a. Four parameters are calculated from the GLCM matrix to characterize the surface roughness: (1) maximum occurrence of the matrix (MOM), (2) maximum occurrence position (MOP),calculated by searching the GLCM for the maximum value and storing its position in the form of (x,y), (3) standard deviation of the matrix (SDM) and (4) maximum width of the matrix (MWM), which is calculated using a search algorithm below and above the diagonal in a direction normal to it to find the farthest points. The pictures of the surfaces were taken under the same lightening conditions using a Jenavert incident light microscope and a color video camera. It was found that the calculated parameters MOM, MOP, SDM and MWM have a very good correlation (> 0.96) with the arithmetic average roughness (R_a). The three parameters MOM, SDM and MWM are positively correlated with the arithmetic average roughness R_a. However, the MOP parameter decreases by increasing R_a. Thus, all of these parameters could be used as indicators of the surface roughness.

2.1.4 Statistical Analysis

Kiran et al.[11] used a vision system comprised of a Charge-coupled device (CCD) camera and a lighting arrangement to capture images of flat surfaces obtained from different processes: grinding, milling, sandblasting, and shaping. The images were preprocessed to eliminate the gaussian random noise. The gray intensity distribution histograms are used to characterize the surface finish. The variance of the intensity histogram of the images was found to be correlated with the surface roughness R_a. Kumar et al. [12] used a Cubic Convolution interpolation technique to enhance the images of machined surfaces (ground, milled and shaped) captured by a CCD camera while preserving edge details. The arithmetic average of the grey level G_a feature is then calculated as follows:

$$G_a = \frac{1}{n}\sum_{i=1}^{n}|g_i - g_m| \tag{2.3}$$

where g_i is the gray level values of a surface image along the i^{th} line and g_m is the mean of the gray values expressed as :

$$g_m = \frac{1}{n}\sum_{i=1}^{n} g_i. \tag{2.4}$$

Regression analysis were used to establish three expressions of R_a for Grinding, Milling and Shaping as a linear combination of the speed, depth of cut and G_a. The regression equations developed gave a maximum error of 2% in the case of grinding, 6.34% in the case of milling and 8.2% for shaping between the estimated R_a and the R_a measured using a stylus instrument. Kamguem et al.[13] used three features extracted of the captured microscope images of 15 turned parts: the gradient factor of the surface which corresponds to the variation of the material to the surface measured by the change of light, the average cycle of texture corresponding to the number of cycles per unit area and the average gray level G_a. A linear relationship was found between the R_a and each of the three texture features. The models had high correlation with the R_a measured using a stylus instrument. The approach yielded an error in R_a estimation in the range of $9 - 18\%$ for profile surface roughness R_a in the range of $2 - 25\mu m$. Gupta and Raman [14] used a CCD camera to capture the scatter spectrum of the reflection of a HeNe $5mW$ laser light source illuminating a rotating pre-machined workpiece ($140rpm$ and $285rpm$) with R_a in the range of $30 - 120\mu m$. Twelve different features, namely, the mean intensity, standard deviation of the grey level distribution, the root mean square height of the gray level distribution and others were extracted from the resulting gray image. A multiple regression model showed that the features were correlated with the actual surface finish with an accuracy (R^2) approaching 95%. After conducting statistical analysis, it was noticed that the effect of the surface roughness was more significant on two features: the ratio of the standard deviation to the root mean square value and the square of the second moment of the light scatter intensity distribution. Despite the high correlation between the roughness measured using a stylus-profiling instrument and the output of the regression model, the work done by Gupta and Raman [14] presents some limitations mainly due to the surface roughness range used ($30 - 120\mu m$) which is outside the typical range of surface roughness in conventional turning ($1 - 10\mu m$) and the hardware laser-based setup which requires fine adjustments and stringent specifications particularly, of the laser parameters, and location on a vibration-free table.

2.1.5 SVM

Liu et al.[15] evaluated the surface roughness via color information, and proposed a surface roughness measurement method based on a same pixel red and green color difference index. These researchers also performed a comparison experiment on a group of samples before and after surface pollution based on the support vector regression model. These results showed that the color difference index was strongly correlated with the roughness R_a and had such advantages as anti-pollution and a high level of robustness [16]. Wei et al. [15] proposed a Gray Level Co-occurrence Matrix Support Vector Machine (GLCM-SVM) method to measure the surface roughness of a deep hole. The experimental results showed that the GLCM-SVM can have a high level of accuracy and generalization ability to characterize the surface roughness R_a.

2.1.6 Polynomial Networks

In Dhanasekar et al. [17] and B.Y.Lee and Y.S. Tarng [18], polynomial neural networks(PNN) models are used to estimate the R_a of a workpiece. Polynomial networks were proposed by Ivakhnenko [19] which is a Group Method of Data Handling (GMDH) that models non-linear relations between input and output variables. In the PNN model the nodes are a polynomial function of the inputs. The structure of PNN is selected on the basis of the number of input variables, the order of polynomials in each layer and a criterion that balances the model accuracy and the complexity of the fitted polynomials. The loss criterion used is the sum of the mean squared error between the predicted output and the real output and a penalty proportional to the number of coefficients of the polynomial. The input data for [17] are calculated from the improved quality images of surfaces processed using a reconstruction algorithm namely the standard deviation of gray levels G_a and two spectrum parameters (major peak frequency (F1) and principal component magnitude squared value (F2)) obtained by performing a fast Fourier transform FFT. These three inputs are then fed to a three layer polynomial network. In [18], the feed rate, speed and depth of cut along with the arithmetic average of gray levels G_a of images captured by a digital camera are used as inputs to a four layer model. The arithmetic average of the surface roughness R_a is measured using a profilometer and used as the independent variable. For [18], the training dataset consisted of 57 training samples and the model was tested on 16 samples with a reasonable accuracy: the error was no higher than 12% between the real and the predicted R_a . In [17] the optical roughness parameter estimated for the ground and milled surfaces had an high correlation of 0.91 with the measured R_a after applying the super resolution reconstruction algorithm.

2.1.7 Artificial Neural Networks

Multiple prior work used Neural Networks to assess the surface roughness using different input features (vibrations, feed rate, speed, depth of cut etc ...). This review focuses on the work that used computer vision combined with ANN. Tsai et al.[20] extracted five features from the two-dimensional Fourier transform of images of parts obtained from shaping and milling operations. The obtained features are used as inputs to two NN models that will perform classification in five categories that represent known roughness values of: $6.3, 12.5, 25, 50, 100\mu m$ for the shaped pieces and $1.6, 3.2, 6.3, 12.5, 25, 50\mu m$ for the milled parts. The root mean squared error for the shaped specimens was $1.3177\mu m$ and $0.8311\mu m$ for the milled ones which shows the good results of applying such model to assess the surface roughness. Palani and Natarajan [21] also used a two dimensional Fourier to get the major peak frequency and the squared principal component magnitude. These two features as well as the cutting speed, feed, depth of cut, and G_a are used as input to an ANN to predict the roughness of the end milled parts. The error between the predicted values and stylus based surface roughness was 2.47% for the 10 test images with R_a ranging between $0.3971 - 0.8153\mu m$. P. Priya and B. Ramamoorthy [22] wanted to treat the problem of previous models sensitivity to the inclination angle of the images. They used surface images of samples with different inclination angles. The effect of the inclination angle was eliminated by using a shadow removal algorithm. Five designed frequency domain indexes and sample inclination angles were fed to an ANN. The obtained experimental results showed that machine vision combined with an artificial neural network could achieve high predictive accuracy [22]. In [23], Natarajan et al. used differential evolution algorithm (DEA) as optimisation algorithm for training an ANN to predict surface roughness in turning operations. The DEA is a heuristic method for optimizing nonlinear and non differentiable continuous space functions. For training the ANN, it is applied to global searches within the weight space to minimize the learning error. The cutting speed, feed rate, depth of cut, and average gray level G_a of the surface image of the workpiece were taken as the input parameters and the surface roughness as the output parameter. The results obtained from the DEA-based ANN model were compared with the backpropagation (BP)-based ANN. It is found that the average absolute percentage error is very close 0.62% for the DEA-based ANN and 0.41% for the BP-based ANN. However, the DEA-based is shown to be faster at convergence speed, simpler and more robust at numerical optimization than the BP-based model.

2.2 Challenges and Limitations

The state of the art faces many challenges to perform contact-free automated visual surface finish quality inspection. Among the main difficulties encountered are:

- Hazardeous site: the place for installation of the required equipment for example cameras, illumination, signal processing equipment ... is very hazardous. The illumination systems and cameras require protection from the dust, the high ambient temperature, oil, water, shock and vibration. Thus, there is a necessity for appropriate physical and environmental protection.

- Operational Speed: the speed for the inspection process is very high during production. For flat steel products, speed at the end of rolling, where the inspection equipment has to operate, is typically $20m/s$. Real-time operation at such high speed requires special image processing equipment and software with small execution time.

- Variety of surface finish and defects: for surface roughness measurements, there is a lack of standards that define which surface roughness values are acceptable. This depends on the material, the parts, the applications and human judgement of what is aesthetic for customers.

The previously mentioned approaches to perform indirect surface finish inspection show multiple limitations. The laser scattering and speckle techniques are limited by the wavelength of visible light. These techniques are not physically capable of measuring surfaces whose root mean square roughness is greater than the wavelength of light ($400 - 700$ nanometers, nm)[24]. Since the typical roughness range R_a for conventional machining processes (turning, milling, shaping, drilling) is generally between 0.5 and 10 micrometers (μm) [25], these approaches are more suitable for measurement of precision, sub-micron ($< 1\mu m$) and ultraprecision ($< 50nm$) surface finish, for example those obtained from ultra precision polishing and diamond turning.

An additional restriction in the state of the art is related to the hardware set-up. The special hardware systems as well as stringent and consistent lightning conditions and angles made the suggested systems not easily usable in machine shops. Haralick et al. [9], [13] and [10] used microscopes and frame grabbers to capture the images under consistent lightening conditions. [8] fabricated an adjustable table to hold the specimen and to have the camera at $45°$ to the surface of the part with constant settings. All images were obtained with the same settings and position of the camera. Uniform illumination of the surface was ensured by using a diffuse light source. These consistent and very specific conditions restrict the use of the proposed solutions and undermine the feasibility of the system in machine shop conditions.

CHAPTER 3
PROBLEM STATEMENT

Surface finish inspection is currently, in most cases, still done the traditional way; an expert rotates the part under the light to see if it is shiny, and touches it to see if it is smooth. In order to automate this process using a contact free method and overcome the limitations mentioned in the previous section, we investigate a new approach to replicate the visual human judgement of which surfaces have a good or a bad surface finish. The solution has to be affordable, resistant to hostile machine shop conditions and requiring no special hardware. The first step towards our model consisted in investigating if we can use only images of surfaces, and if based only on the pictures as input, we can see the difference between a good and a bad surface finish. The cameras used to capture parts images are mobile devices cameras considering the availability of these for each mechanic in the machine shop factory. The problem is then assigning a label from a fixed set of categories (bad_surface, good_surface) to an image. This is an image classification problem, which is one of the core problems in computer vision that has a large variety of practical applications. In this work, we have two labels: good and bad. This problem is thus a binary classification problem (2 classes). In the two images below in Figure 3.1, an image classification model, takes a single image as input and assigns probabilities to the two labels, {bad_surface, good_surface}.

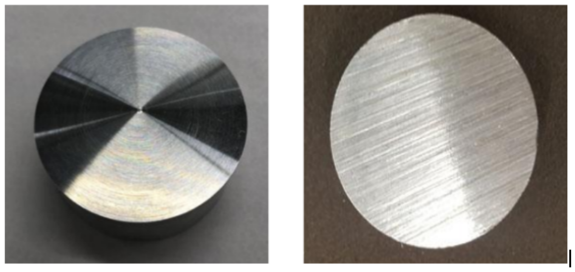


Figure 3.1: Good surface finish image to the left, bad surface image to the right

The image is represented for the computer as one large 3-dimensional array of integers from 0 to 255, of size $Width \times Height \times 3$. For example, The Figure 3.2 of the surface of a yo-yo is 281 pixels wide, 500 pixels tall, and has three color channels Red,Green,Blue (RGB). Therefore, the image consists of $281 \times 500 \times 3$ numbers, or a total of $421,500$ numbers. Each number is an integer that ranges from 0 (black) to 255 (white). Our approach's task is to map these four hundred thousand numbers to a single label: good_surface.

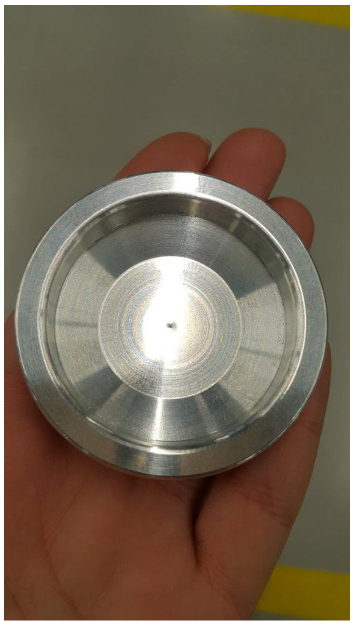


Figure 3.2: This image is a 3-dimensional array of integers between 0 and 255 of size $281 \times 500 \times 3$ ($Width \times Height \times 3$). The 3 represents the RGB color channels

The model that performs this image classification task has to be invariant to the challenges listed below:

- Angle variation: the angles from which the image is taken can vary a lot which changes the orientation of the object. In previous works, this limitations was overcome by fixing the camera angles.

- Scale variation: the objects in the images can have different sizes on the images and in real life.

- Shape: the parts subject of the surface finish inspection can have different shapes while belonging to the same class: good surface for example.

- Occlusion: sometimes the full object is not present in the picture and only a portion of it is visible.

- Illumination conditions: the variations of illumination have a big impact on the pixel level. Under different illumination conditions, the 3-D matrix representing the image can be very different.

- Background: the background can vary a lot from an image to another and sometimes it is hard to distinguish the objects from their background.

The approach taken to solve this problem is a data-driven approach. We provide the model with labeled data which will serve as a training dataset to help the algorithm that looks at these examples learn about the visual appearance and particularities of each class. The pipeline to develop this model consists of 3 main steps as shown in Figure 3.3:

Input
Learning
Evaluation

Figure 3.3: Pipeline of performing image classification

- Input: Training dataset comprised of labeled images.

- Learning: We train the classifier using the training dataset to learn the features of each of the two classes.
- Evaluation: Testing the model on new images that haven't been used before. An accuracy measure is then used to assess how well the model predicted the classes of the test dataset compared to the true known labels.

Once the model is trained and tested, an android app is developed to allow the automated surface finish inspection using only mobile devices.

CHAPTER 4
CONVOLUTIONAL NEURAL NETWORKS (CNN) FOR IMAGE CLASSIFICATION

Convolutional Neural Networks CNN is a class of deep feed-forward artificial neural networks that are very popular, well adapted and effective for visual imagery related problems. One of the first successful applications of CNNs to solve a real-world problem was LeNet for handwritten digits recognition [26], by the computer scientist Yann LeCun. CNNs are multilayer neural networks that are able to automatically learn the important features from labeled data and are also computationally effective thanks to the convolution operation, from which the name is inspired, parameters sharing and the pooling operation. In this section, we will have an overview of how CNNs work.

4.1 Layers

4.1.1 Input Layer

In the case of a fully connected multi-layer neural networks, the input layer is a vector. An image represented as a 3-dimensional array of integers from 0 to 255, of size $Width \times Height \times 3$ would be transformed to a 1-D vector of size: $Width \times Height \times 3$. For example, the Figure 3.2 would be transformed from a 3-D array of size $281 \times 500 \times 3$ to a 1-D vector of size: $281 \times 500 \times 3 = 421,500$. The input of CNNs are $Width \times Heigh$ if the images are in grayscale, otherwise $Width \times Height \times 3$ for RGB.

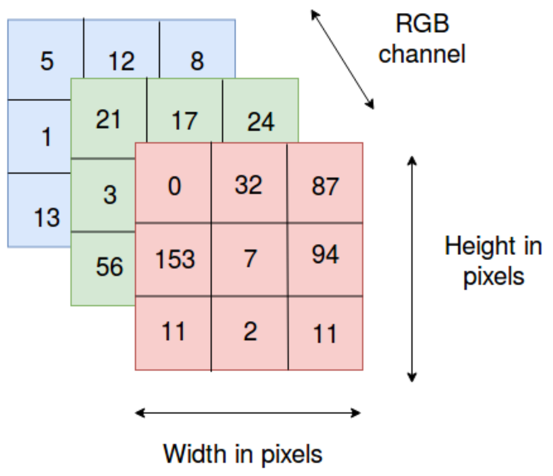


Figure 4.1: CNN input as 3-D array

4.1.2 Linear or Fully Connected Layer

For an input Matrix $X \in R^{N \times D}$, where N is the number of input vectors in a batch, a row vector x_i represents a single input vector with size D. In this case, we are considering that x_i is an input tensor flattened as a 1-D vector. In the case of a classification operation where we have K classes, the linear layer acts as a score function that performs a linear mapping operation from $R^D \rightarrow R^K$ of the form: $f(x_i, W, b) = Wx_i + b$ where W, the weight matrix of size $[K \times D]$, and b, is the bias vector of size $[1 \times K]$, are learnable parameters. The output of this operation is a 1-D vector of size $[1 \times K]$ where each entry is interpreted as a score for the corresponding class. This linear layer is loosely motivated by the connectivity pattern in the brain cells called neuron.

4.1.3 Activation Layer or Non Linearity

The activation function is inspired by the biological neurons that fire when the weighted sum is above a certain threshold. It takes a single number and performs a certain fixed mathematical operation on it ensuring that the output cannot be reproduced by a linear combination of the inputs. This non linearity is crucial and makes the neural network perform complex tasks. There are three main activation functions:

- Sigmoid: the sigmoid function is expressed as : $\sigma(x) = \frac{1}{1+\exp(-x)}$. The graph of the sigmoid function is shown in Figure 4.2 [27] on the left. This function squashes any real value between $-\infty$ and $+\infty$ to be between 0 and 1. The sigmoid activation function shows a major drawback because of saturation at both ends 0 and 1 which zeroes out the gradient and makes its usage unadaptable to the backpropagation algorithm.

- Tanh: the hyperbolic tangent function takes any real value as input and outputs a value between -1 and 1. It is a scaled version of the sigmoid function: $\tanh(x) = 2\sigma(2x) - 1$. In practice, the $\tanh$ is more used than the sigmoid function. The graph of the tanh function is shown in Figure 4.2 [27] on the right.

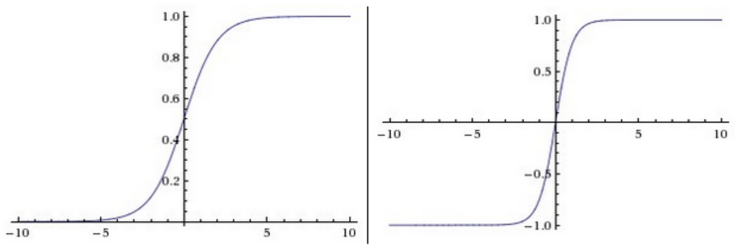


Figure 4.2: Sigmoid function on the left. Tanh function on the right.

- ReLU: The Rectified Linear Unit is the most widely used activation function. It gives an output x if x is positive and 0 otherwise: $f(x) = max(0, x)$. The ReLU function is shown in Figure 4.3. ReLU activations are adapted to multilayer neural networks since a combination of ReLUs is also non linear. The ReLU is computationally efficient due to the fact that it doesnt involve any expensive operation and also yields a sparse output meaning that fewer neurons are firing since any negative value is set to 0. However, because of zeroing any negative number, the gradient will be 0 and the corresponding weights will not be updated during the optimization process. This problem is referred to as dying ReLU problem.

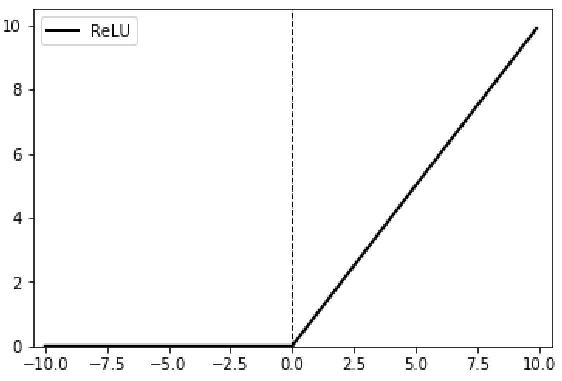


Figure 4.3: ReLU function

The mathematical model of a neuron performing a linear operation followed by an activation function is shown in Figure 4.4 below:

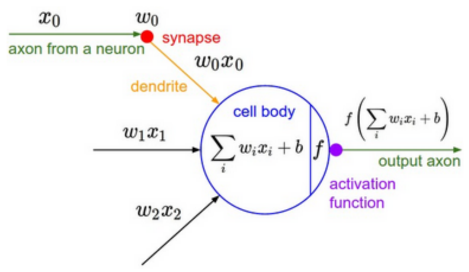


Figure 4.4: Mathematical model of a neuron

4.1.4 Convolution Layer

The ANN models are typically organized into multiple fully-connected layers of neurons in which neurons between two adjacent layers are fully pairwise connected, but neurons within the same layer don't share any connections. These networks are only made of linear and activation layers which don't scale for high dimensional inputs such as images. For a $224 \times 224 \times 3$ image, we will need $150,528$ parameters for a single neuron in the first fully connected layer. An example of a 3-layer Neural Network is shown in the Figure 4.5.

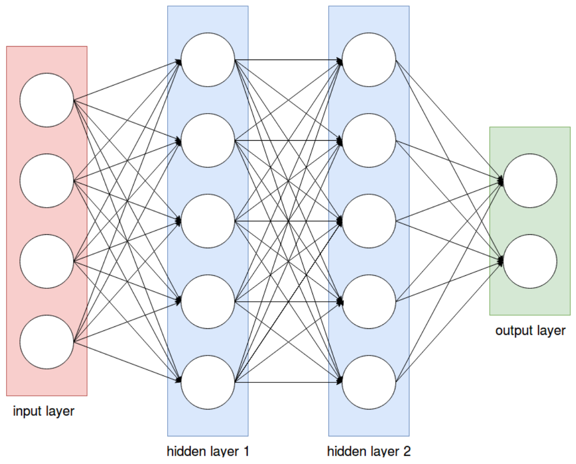
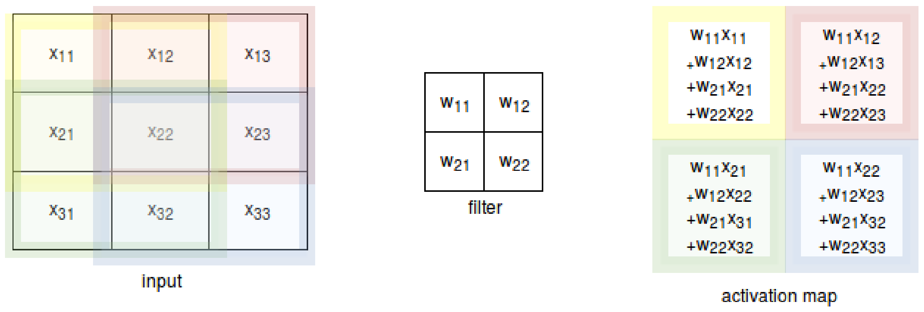


Figure 4.5: A 3-layer neural network with four inputs, two hidden layers of 5 neurons each and one output layer with 2 neurons.

CNNs overcome the scaling limitations by exploring the spatial structure of their input (images). The weights of a CNN have the same depth as the input, called input feature maps, meaning if the input is an RGB image, the weights in the first layer are 3-D arrays of dimension $f_w \times f_h \times 3$, with f_w being the weight's size along the width dimension, and f_h being the weight's size along the height dimension. Usually $f_w = f_h = k$. As opposed to the fully connected neural networks, each neuron in the CNN is only connected to a small region in spatial dimensions of the input. The region of the feature maps is covered by a convolution filter, which is a stack of spatial kernels of size $k \times k$. The kernels in the filter are usually not the same, but all of them cover the same spatial "column" of the input tensor. During the forward pass, the layer computes a convolution operation between the region of the input and the convolution filter, producing a single element of output, then slides through the entire image to perform the same operation again and again, from left to right, from top to bottom. A stack of such neurons form a convolutional layer. So the weights of the layer form a 4-D tensor with dimensions that correspond to spatial kernels, input feature size, and output feature size.

Convolutional layers have several important parameters. One of them is the stride S that represents the step-size by which the filter slides over the input. It is typical to use a stride of 1 or 2. Figure 4.6 shows this convolution operation which is an element-wise multiplication with a stride $S = 1$.



x_{11} x_{12} x_{13}
x_{21} x_{22} x_{23}
x_{31} x_{32} x_{33}
input
w_{11} w_{12}
w_{21} w_{22}
filter
$w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$
$w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$
$w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$
$w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$
activation map

Figure 4.6: Convolution operation between image input and filter with stride $S = 1$

Another one is the padding P, which represents how many rows and columns, and sometimes whole input features, will be added to the edges of the input volume. Usually, zero padding is used, meaning that all the values that are added to original input feature maps are zeros.

The output of the convolution with an input image of dimension $h \times w \times d$, is a $(\frac{(h-f_h+2P)}{S} + 1) \times (\frac{(w-f_w+2P)}{S} + 1) \times 1$ tensor. The figure 4.7 shows this convolution operation and the size of the output.

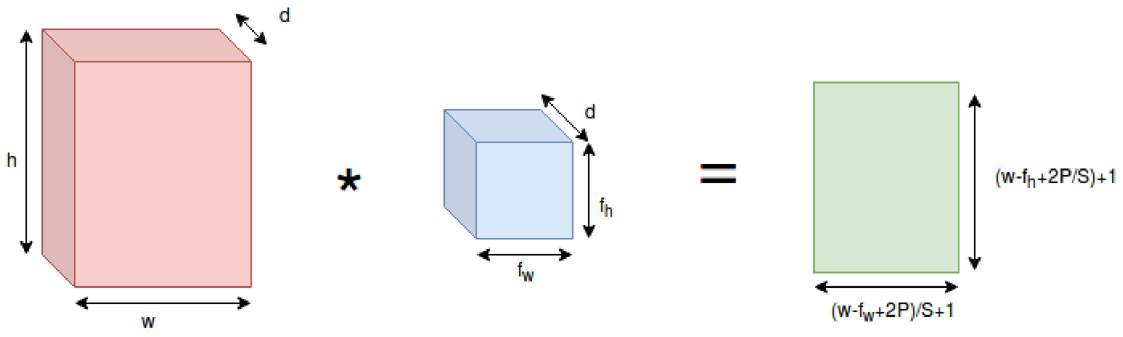


Figure 4.7: Convolution operation between an image input and a filter

The output of the convolution operation between an input volume and a filter corresponding to a single neuron is a 2-D tensor called activation map. In one convolution layer, there are multiple neurons along the output channel dimension. These 2-D activation maps from every neuron are stacked together to form a volume output of the same depth as the number of filters used. As mentioned before, the convolution layer is a volume, and the neurons are organized in depth. At each depth slice, the neurons will share the same weights per the assumption that if a filter is useful to detect features at some position, then it should be relevant to detect others at another position. This weight sharing reduces considerably the number of parameters learned compared to a regular neural networks. Thus a convolutional layer can be defined by the number of filters, their dimensions, the stride and the zero padding.

4.1.5 Pooling Layer

Pooling layers, also referred to as downsampling layers, serve to reduce the dimensions of the activation maps by only retaining the most relevant information. The pooling operation can be Max pooling, Average pooling or Sum pooling. The most widely used operation is the Max pooling operation applied on every depth slice of the activation map, as the max operation avoids cancellation of negative elements and prevents blurring of the activation maps. The Max filter slides through the input with a stride, retaining only the highest value of the covered region, Figure 4.7 shows the pooling operation with a filer of size 2×2 and a stride $S = 2$. It is common to periodically insert pooling layers between convolution layers in order to reduce the amount of parameters, control the size of the neural network and limit overfitting.

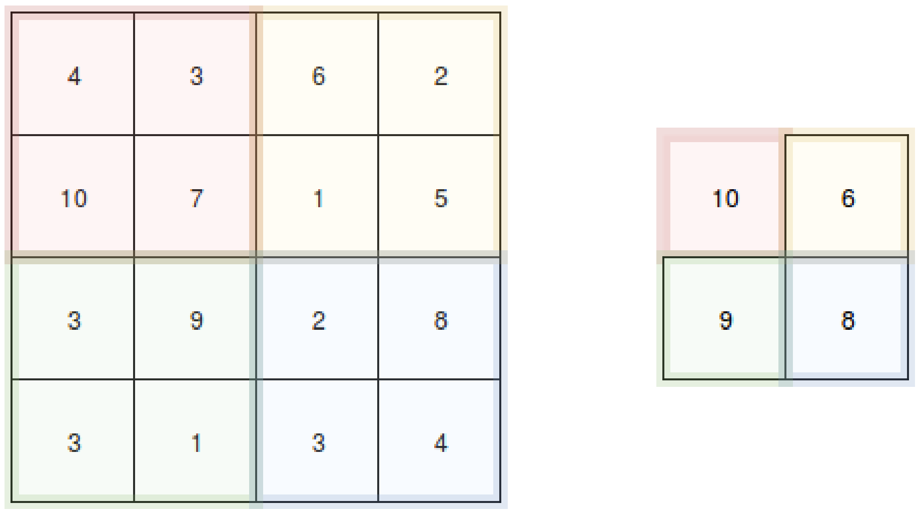


Figure 4.8: Pooling operation on a single slice depth input with a 2 × 2 filter and a stride of 2

4.1.6 Batch Normalization

Batch normalization layer has been shown to speed up convergence [28]. It consists of adjusting and scaling the activations to have a zero mean and unit variance by subtracting the batch mean and dividing by the batch standard deviation. It also helps reducing overfitting since it has a regularization effect on the hidden layers.

4.1.7 Dropout

Dropout, introduced by Srivastava et al. [29], is a very simple and effective regularization technique that consists in randomly setting the neurons in each layer to zero with a certain probability during training.

4.2 CNN Architectures

Most CNNs are a stack of a convolutional layer, followed by a ReLU and optionally a Pooling layer. A fully connected layer is found at the end to output the class scores. Many CNN architectures have been developed since 1990's. The most widely used ones are mentioned below.

4.2.1 LeNet

LeNet was developed by Yann LeCun [26] to recognize hand written digits. This model constitutes the first successful application of CNNs and an inspiration for more recent CNN architectures. The LeNet architecture is shown in Figure 4.9 from the original paper [26].

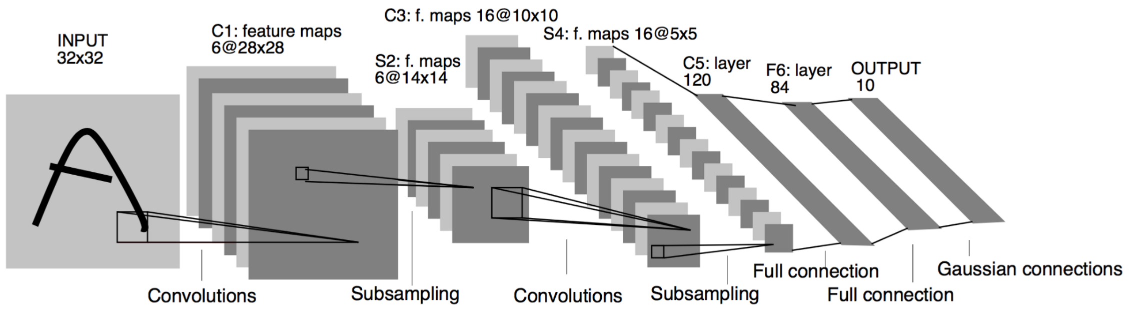


Figure 4.9: Architecture of LeNet-5 used for digits recognition

LeNet has a total of 7 layers: 3 convolutional layers, 2 pooling layers and one fully connected layer followed by the output layer (fully connected layer that outputs the class scores). The input for LeNet are $32 \times 32 \times 1$ images normalized using mean and standard deviation. The activation function used is tanh.

4.2.2 AlexNet

AlexNet was developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [30]. This CNN competed in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [31] achieving top-5 error with 15.3% compared to 26.2% achieved by the second position model. AlexNet is deeper and more complex than LeNet using 5 convolutional layers, 3 max pooling and 3 fully-connected layers for a total of 60 millions parameters. AlexNet was trained on the ImageNet benchmark dataset which contains $10 Million$ labeled images from $10,000+$ categories.The model was implemented in CUDA using two GPUs to make training faster. A dropout, is used on the fully connected layers as regularization technique to reduce overfitting. A novelty introduced in this model is stacking multiple convolutional layers each followed by a non-linearity without having a pooling layer in between. The graph 4.10 from the original paper [30] illustrates AlexNet architecture and the distribution of computation between the two GPUs.

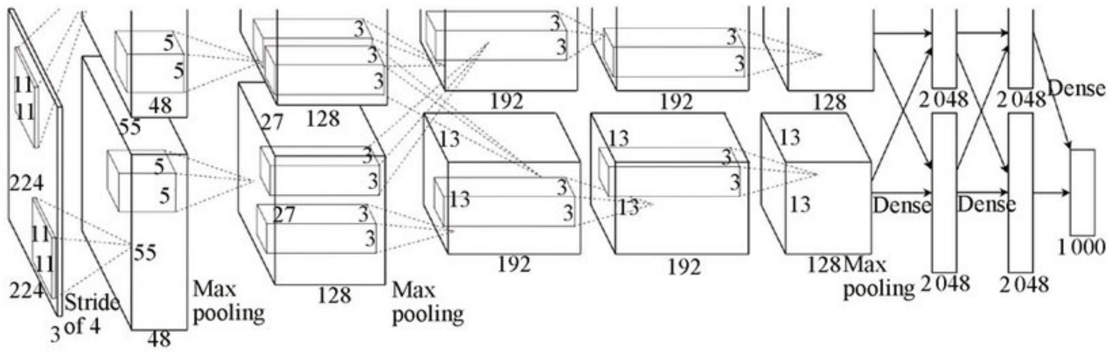


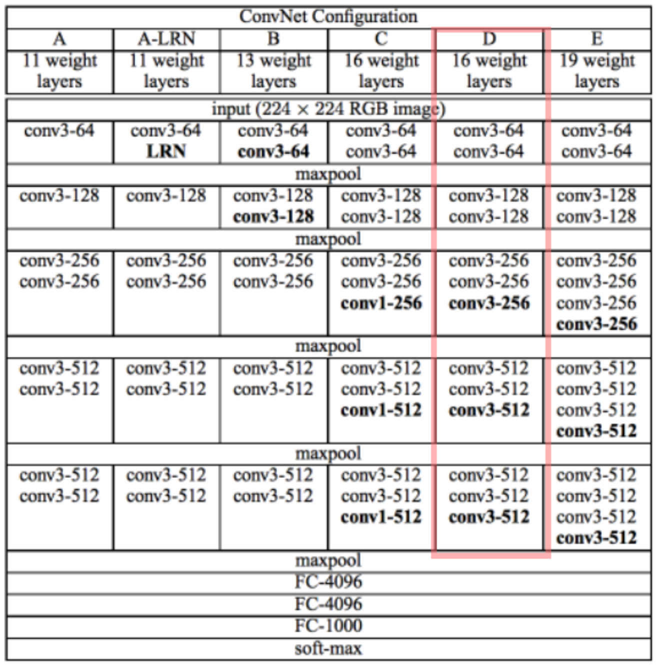
Figure 4.10: Architecture of AlexNet used showing the distribution of responsibilities on the two GPUs

4.2.3 ZF Net

ZF Net designed by Matthew Zeiler and Rob Fergus[32] won the ILSVRC 2013 with 11.2% error rate. The ZF Net is heavily inspired by AlexNet wih few changes mainly reducing the stride and the filter size from 11×11 to 7×7 on the first layer and expanding the middle convolutional layers. ZF Net has almost 140 million parameters.

4.2.4 VGG Net

VGG Net developed by Karen Simonyan and Andrew Zisserman [33] is a very deep 16 layer CNN that was a runner-up in the ILSVRC 2014. The idea of VGG Net is to simplify the structure using only 3×3 filters with stride and pad of 1 and 2×2 max pooling layers with stride 2 but explore the depth. The idea is that a combination of two 3×3 filters in each convolution has an effective receptive field of a 5×5 filter and a combination of 3 has a receptive field of 7×7. This decreases the number of parameters used. The model uses a total of 140 million parameters. This model achieved 7.3% error rate proving that the depth of the CNN is critical for good performance. The Figure 4.11 from the original paper [33] shows the architecture of VGG Net.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 4.11: The different VGG Net architectures with architecture D showing the best performance

4.2.5 GoogLeNet or Inception

GoogLeNet by Szegedy et al.[34] from google is a 22 layer CNN that won ILSVRC 2014 with 6.7% error rate. This paper introduces the inception module where, as opposed to previous models where everything happens sequentially, some parts of this network are computed in parallel. The parts of the network that are parallel are called the inception modules where instead of making a choice of whether to have a pooling operation or a convolution, we perform all of these operations in parallel. This parallel method leads to many outputs, that were limited by applying a 1×1 convolution before the 3×3 and the 5×5 layers. Inception is considered among the very deep CNNs with 100 layers and 9 inception modules. The model reduced significantly the number of parameters by eliminating fully connected layers and using average pooling to have a total of 4 million parameters; $12\times$ less than AlexNet. There are many versions of Inception CNNs, the most recent one is Inception-v4.

4.2.6 ResNet

Residual Networks ResNet developed by Kaiming He et al.[35] is a class of very deep 152 layers CNNs that won LSVRC 2015 with an error rate as low as 3.6%. The main idea of ResNet comes from the authors belief that simply stacking layers in order to increase the depth in network architectures results in an increase in training and testing errors. In ResNet, instead of feeding the output of a sequence of convolution-ReLU to the next convolution-ReLU unit, the residual block has the input x go through a convolution-ReLU-convolution series which gives an output $F(x)$. This output $F(x)$ is then added to the input x to get $H(x) = F(x) + x$ which is passed to the next layer. This means that instead of trying to fit an underlying mapping between the input x and some output $H(x)$ through stacked layers, it is easier to fit a slightly altered representation of the input x since $F(x)+x$ can be seen as adding a Δ to the original input x. The Figure 4.12 from [35] shows the residual bloc. ResNet is also different from the previous architectures by using batch normalization, and similar to Inception, the architecture uses only one fully connected layer at the end of the network to output the class scores. ResNet became the default architecture for current image classification problems.

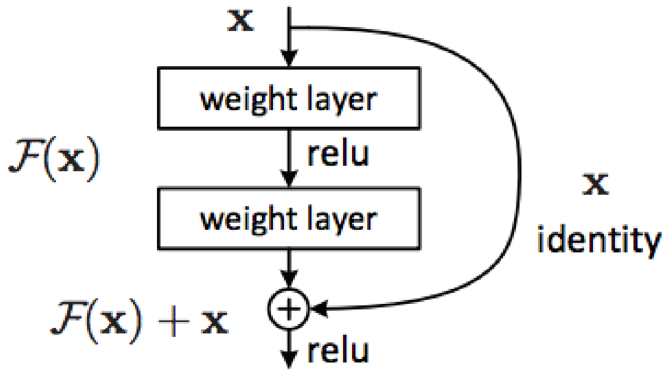

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.12: Residual Block

4.3 Training CNNs

4.3.1 Loss Functions

During training a Neural Network, we define a loss function that will quantify the error of the model with respect to a target. This loss function is crucial for the learning as it is used during training to help tune the weights in a direction that minimizes the loss function (reducing error) at a rate of $\frac{\partial L}{\partial w}$ where L is the cost function and w is the weight. For classification problems, mean squared error(MSE) and cross entropy loss are the most widely used loss functions.

- Mean squared error (MSE): MSE is commonly used in regression and classification problems. For a classification problem $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ where n is the number of training samples, y_i is the true vector output that will have 1 at the index of the correct class and 0 elsewhere, and $\hat{y}_i$ is the output vector of predictions of the model.

- Cross entropy L=loss: is another commonly used loss function. The cross entropy is more sensitive to the accuracy of classification models. A small difference in the confidence of prediction can be shown in cross entropy while MSE might give the same result in some cases. The cross entropy function between an input x and the target output class scores vector y where the ith element corresponds to the score of the correct class is defined as: $L = -log\frac{exp(y_i)}{\sum_j^n exp(y_j)}$, n being the number of classes. For a training set with N inputs, the loss is the average of losses on each individual input $L = \frac{1}{N}\sum_i^N L_i$. The cross entropy function can be interpreted as the normalized probability assigned to the true class, so minimizing the cross entropy loss goes back to maximizing the likelihood of the correct label. In addition to being positive, tending towards zero when the prediction is close to the ground truth which makes it a good candidate for a cost function, the cross entropy loss leads also to quicker learning through gradient descent than the MSE loss which makes it the most widely used in classification problems.

4.3.2 Backpropagation

In regular ANN, we calculate the partial derivative of the loss function with respect to each weight in order to adjust the weights in a way that minimizes this loss function which constitutes the learning process. The first partial derivative is calculated at the final layer since this is what gives the error (the loss between the output of the model and the true result), then using the chain rule, this gradient serves to compute the partial derivative of the loss function with respect to the weights of the previous layer. This calculation is backpropagated till the first layer, which gradient is computed last. $\frac{\partial L(w)}{\partial w}$ expresses by how much the loss function changes if we change the weight. The Figure 4.13 from [27] lecture 4 illustrates the backpropagation operation.

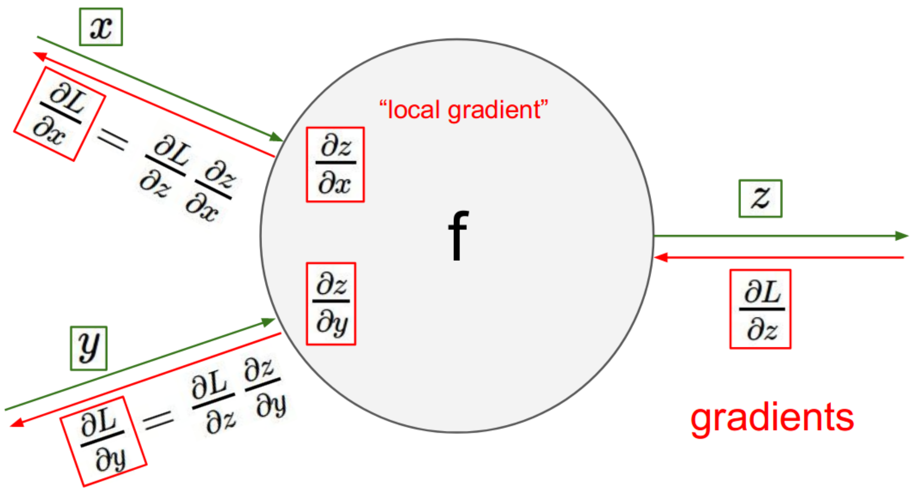

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure 4.13: Illustration of the backpropagation of errors

4.3.3 Optimization

Once the gradients of the loss function with respect to the weights are computed, optimization algorithms are used to perform the weights update in order to minimize the objective function. The training of a neural network consists in performing many iterations of forward pass to compute the output of the model on the training dataset, compute the loss between the output and the real value, then we use backpropagation to compute the gradients of the loss with respect to the weights and finally preform the weights update using optimization algorithms until convergence or achieving the desired accuracy.

- Stochastic Gradient Descent (SGD): SGD is a computationally lighter version of the original gradient descent algorithm. At each iteration, instead of computing the gradient of the objective function across all training data, the SGD randomly selects one training instance, computes the gradient of the objective function with respect to the weights using only the ith training sample (x_i, y_i). At the jth iteration, the weights are updated simultaneously to get the new weights as follows: $W_{j+1} := W_j - \alpha \nabla L(W_j, x_i, y_i)$, where $\nabla L(w_j, x_i, y_i)$ is the gradient of the loss function and α is called the learning rate which defines the step size taken in the direction of the gradient descent. The choice of the learning rate α affects convergence. It is usually set to a small value in order to avoid stepping over the minimum and never converging.

- Mini batch Gradient Descent: in practice, each parameter update in the SGD is computed using a small batch of the training dataset instead of a single one. This uses matrix operations which are in most libraries optimized, reduces variance and ensures a more stable convergence.

- Momentum: when the surface of the objective function curves more steeply in one dimension than another forming a long shallow ravine, the SGD oscillates between the two steep sides and makes very small progress along the ravine towards the local minima [36]. The momentum method [37] multiplies the previous weights with a γ factor < 1 during the update as follows. We define a momentum which is a moving average of the gradients

  $$V_t = \gamma V_{t-1} + \alpha \nabla L(W, x, y)$$

  and use it to update the weights as follows:

  $$W := \gamma W - V_t$$

  The γ term is usually set to 0.9. The momentum helps accelerate gradients and reduces oscillations solving the ravine problem of the SGD.

- Adaptive learning algorithms: in previous gradient descent algorithms, the learning rate parameter is fixed in advance and is applied to update all the weights. Algorithms such as Adagrad, Adadelta, RMSprop, Adam provide an adjusted learning rate for each parameter and are referred to as adaptive gradient descent algorithms.

4.3.4 Regularization

Considering the large number of parameters and the complexity of neural network models, it is easy to overfit meaning modeling very closely the training dataset while being unable to generalize well on new data. In order to control overfitting, some regularization techniques are used. Among the most widely used ones are:

- $L2$ regularization: $L2$ regularization is a very common regularization method that penalizes the squares of the parameters by adding a regularization parameter $\frac{\lambda}{2}\|W\|$ to the loss function where W represents all the weight matrices of the networks and λ is a positive arbitrary number. During the update of the parameters using gradient descent, the regularization term adds a penalization to restrict the values of the individual parameters from being too high.

- $L1$ regularization: is very similar to $L2$ regularization but uses norm-1 for regularization by adding a term $\lambda|W|$ to the loss function. $L1$ regularization leads to a more sparse weight vectors. It is possible to use both $L1$ regularization and $L2$ regularization by adding $\lambda_1|W| + \frac{\lambda}{2}\|W\|$ to the cost function.

- Early stopping: is usually done using a validation dataset and an upper bound on the generalization error. The training is done as usual on the training dataset, and at the end of each epoch(full pass of the dataset) an error is measured using the validation dataset. If the error for the current model is better than the best one so far, the model is saved. The goal is to prompt the training and keep the model with the lowest validation error.

4.3.5 Weights Initialization

If the neural network is trained from scratch, all the parameters have to be initialized. All the weights are initialized with random small numbers in a way that each of them is unique in the beginning in order to learn different features during the training. At each layer,

the parameters are sampled from a multidimensional gaussian distribution with zero mean and standard deviation of $\frac{1}{N}$ with N: the number of inputs. This initialization preserves the variance of the inputs and thus the pixel intensities are obtained after normalizing by substracting the mean and dividing by the standard deviation.
Training deep CNNs from scratch with random initialization requires a lot of computational resources, a very large labeled dataset for supervised models and a lot of tuning efforts. In practice, pre-trained models are used, in which case, the weights are kept and their mean and standard deviation would be the same as their original dataset.

4.4 Transfer Learning

Traditionally, Neural Networks were problem specific and could only solve the particular task they were trained for. When the feature space distribution changes, the NN had to be trained from scratch again with random initialization. Transfer learning aims at reusing models that were trained on a particular dataset to perform new tasks on a similar new dataset. A CNN is usually pre-trained on a very large dataset like ImageNet [38] that has 1.2 million training images with 1000 classes, and that learning is then leveraged to solve a different task using a smaller training dataset. Transfer learning is widely used for computer vision related tasks, since low level features such as detecting shapes, edges and intensities are well transferable and can be used to detect the generic features in images. There are typically two types of transfer learning:

- Feature extractor: as the CNN is formed of layers where the first ones detect generic features and one last usually fully connected layer that performs the specific task of outputting the 1000 class scores of ImageNet, it is possible to use all the layers but the final one as fixed feature extractor for other tasks. The weights of one or more layers are frozen i.e are not updated and are used as generic feature detector. This frozen CNN outputs the activations that are thresholded in the same way as during the training on ImageNet. A final linear classifier layer is then added and trained for the new dataset. This method although only replaces the last fully connected layer works very well in practice.

- Finetuning pre-trained CNNs: in this technique, the pre-trained weights are used for initialization, and then get updated during backpropagation using the new training dataset. It is possible to not finetune all the layers of the CNN and freeze some layers (the weights are not updated during backpropagation) while others are retrained, and their weights are updated to fit the new task. It is common to freeze the first k layers that learn more high level features in order to reduce overfitting, and only retrain the last layers to learn more task specific features.

Many pre-trained models are available to download through many deep learning frameworks such as TensorFlow and PyTorch. It is common to use small learning rates during transfer learning compared to random initialization since the weights already learned some features. Transfer Learning showed good performance and is widely used for NLP, Audio/speech and computer vision.

CHAPTER 5
PROPOSED FRAMEWORK FOR SURFACE FINISH INSPECTION

In order to automate the surface finish inspection using a contact-free affordable and robust method, a CNN is used to classify images of surfaces in good and bad surfaces. The goal is to have the CNN perform a mapping between the image and the class, each image is assigned 0: bad_surface and 1: good_surface. Because of the lack of publicly available dataset of metal surfaces, during this thesis work, a dataset is created and is made publicly available. The images were labeled using an expert's opinion on the surface finish of the corresponding parts. The dataset is made of images of aluminum parts produced using a CNC machine.

5.1 Experimental Set-up and Data Collection

The images of the dataset were taken of multiple parts mainly made of two operations: cylinders from facing operation and yo-yos production in addition to few other shapes. The processes of making the cylinders from the facing operation and the yo-yos are described below:

5.1.1 Facing Operation

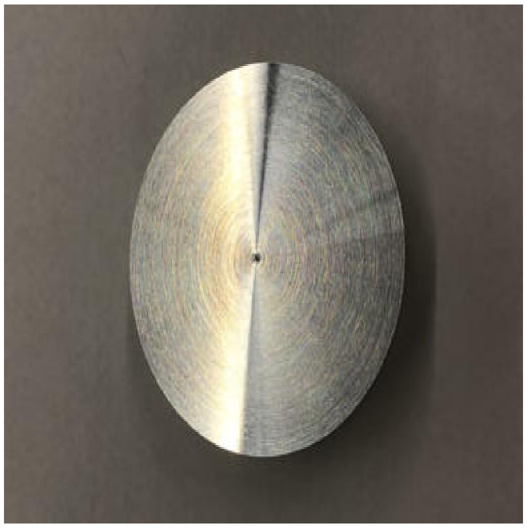
Two OKUMA Genos L250 CNC (Computer Numeric Controlled) Lathes shown in Figure 5.1 were used for production of aluminum parts. Motion was CNC controlled in two axes; the Z-axis, parallel to the axis of rotational symmetry at the center of rotation, and the X-axis, perpendicular to the Z-axis.

Figure 5.1: Two OKUMA Genos L250 CNC Lathes

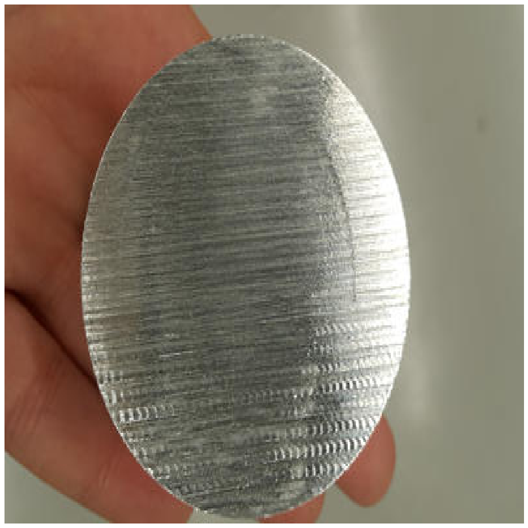
Cylindrical aluminum stock of $2.25in$ in nominal diameter was faced and turned in this process. The aluminum stock was cut into pucks of $0.75 \pm .1$ inches in length with the use of a vertical bandsaw. Both end faces of the pucks were very rough due to the machining marks left by the bandsaw. The pucks were mounted in the lathe spindle with aluminum jaws.

A single point, right-handed turning tool was used for all cutting operations. A sharp carbide insert with a titanium nitride coating was used to make all cuts. The insert was cleaned of any built-up edge and checked for fracture or wear before each part was cut.

The machining process consisted of two operations; a roughing operation and a final finish cut. The first roughing operation consisted of multiple facing passes, where the exposed face of the puck was quickly machined to a known length. Due to the rough faces and relatively large tolerance in initial length of the pucks, multiple facing passes were required to make each puck consistent. This is considered rough machining because the cutting feed rate was set relatively high to decrease overall machining time. After this roughing operation, the exposed face of each puck was consistent and provided a control from which the finishing operations were conducted. The second finishing operation consisted of a single facing pass with a much slower feed rate to produce the desired surface finish. The G-code used to program the machine is included in the appendix A for reference. Figure 5.2 shows pictures of parts resulting from this process.



(a) Good surface



(b) Bad surface

Figure 5.2: Two parts from facing operation

5.1.2 Yo-yo Production

Two OKUMA Genos L250 CNC (Computer Numeric Controlled) Lathes were used for production of yo-yo halves. Motion was CNC controlled in two axes; the Z-axis, parallel to the axis of rotational symmetry at the center of rotation, and the X-axis, perpendicular to the Z-axis.

Cylindrical aluminum stock of $2.25in$ in nominal diameter was faced and turned in this process. The aluminum stock was cut into pucks of 0.75 ± 0.1 inches in length with the use of a vertical bandsaw. Both end faces of the pucks were very rough due to the machining marks left by the bandsaw. These initial cylindrical pucks were used as stock pieces to manufacture the yo-yos. The pucks were mounted in the lathe spindle with aluminum jaws and a single point, right-handed turning tool was used for all cutting operations. A sharp carbide insert with a titanium nitride coating was used to make all cuts. The insert was cleaned of any built-up edge and checked for fracture or wear before each part was cut. Machining process on both machines consisted of two operations; a roughing operation and a final finish cut. Rough machining passes were used to removed large amounts of material in a short amount of time and reveal the general shape of the yoyo. Finishing passes were employed to precisely modify the geometry to match the final part specifications. These operations were completed across two Genos L250 CNC Lathes.

Operations conducted on Genos Lathe #1:

The first roughing operation consisted of multiple facing passes, where the exposed face of the puck was quickly machined to a known length. Due to the rough faces and relatively large tolerance in initial length of the pucks, multiple facing passes were required to make each puck consistent. This is considered rough machining because the cutting feed rate was set relatively high to decrease the overall machining time. After this roughing operation, the exposed face of each puck was consistent and provided a control from which the finishing operations were conducted. The second finishing operation consisted of a single facing pass with a much slower feed rate to produce the desired surface finish.

After the puck was faced to the proper length, an insert drill measuring $0.75in$ in diameter was used to drill the initial hole in the outward face of the yo-yo half. A boring bar was used to widen the hole to the appropriate inner diameter. Finish profile passes, aesthetic fillets, and final slow passes to achieve the desired surface finish were conducted with the same tool used to complete the initial roughing passes, and 80° right handed turning tool. After this process was completed, the aluminum puck was moved to Okuma Genos #2 for machining operation on the other side of the part.

Operations conducted on Genos #2:

The backside of the aluminum puck also needed to be faced in the same manner as the first operation on Genos #1. After the finishing facing pass was completed, the aluminum yo-yo half was at its final length. Following the facing passes, a center drill and a small drill were used to make the initial hole in the backside of the yo-yo, which is ultimately used with a ground steel dowel in a press-fit the yo-yo halves together. This hole was precisely machined to its final press-fit diameter with a small $0.25in$ diameter boring bar. A $0.001in$ interference diameter was used for the press-fit. The final profile of the backside of the yo-yo was machined with an $80\,^\circ$ turning tool, and appropriate aesthetic fillets were added.

A final finishing pass was used to achieve the desired surface finish of the yo-yo. After all machining passes were completed, the yo-yo was removed from the second machine, checked for dimensional accuracy within specified tolerances, and cleaned of cutting fluid.

The G-code used to program the machine is included in the appendix B for reference.

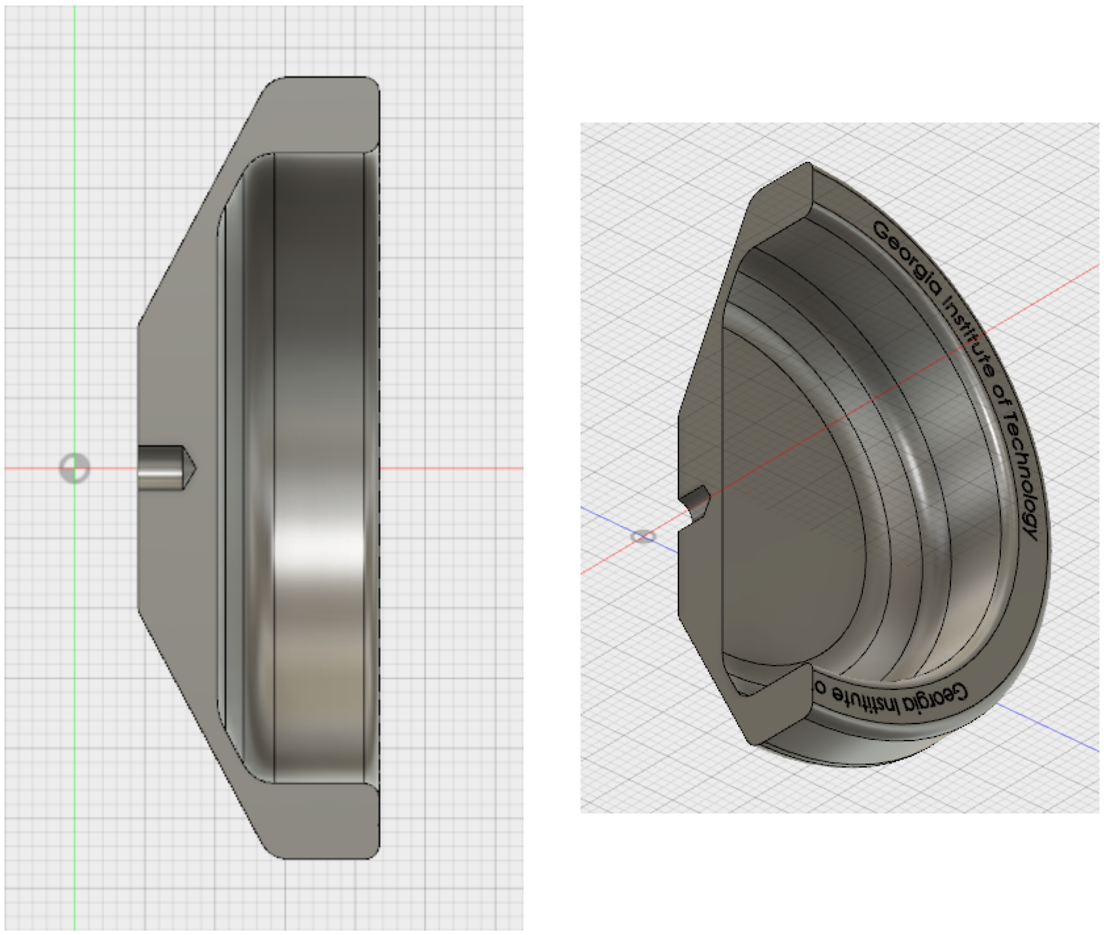


Figure 5.3: Cross section view of the Yo-yo produced from aluminum cylindrical pucks. Features on the right side of the cross-section were completed on Genos #1. Features on the left side of the cross-section were completed with Genos #2.

5.2 Data Acquisition

Once all the parts are made, pictures have been taken of these different parts from different smartphone cameras under varying lightening conditions, orientations and backgrounds. A total of 1280 images were captured. An expert's judgement of the surface finish of the parts resulted in labeling the corresponding images. Two directories of 640 images each are thus created, one for good surface and one for bad surface.

5.3 Splitting the Data

Splitting the data into training, validation and testing data is very important for the model performance. The test and the validation dataset are chosen randomly from the same distribution. In order to train the CNN model, the data has to be splitted in training, validation and testing datasets. It is custom to allocate $70 - 80\%$ of the original data for training, $10 - 15\%$ for validation and $10 - 15\%$ for testing. Among the ways to perform this split is to have 3 directories: one for training, one for validation and one for testing. Each of these directories contains two sub-directories: one for images corresponding to a good surface finish and one for bad surface finish. A python script in Appendix C is used to randomly shuffle the data and perform the split into 490 images per class for the training dataset, 80 images per class for validation, and 70 images per class for testing. Once the script to split the data is run, the data is organized in folders as shown in Figure 5.4.

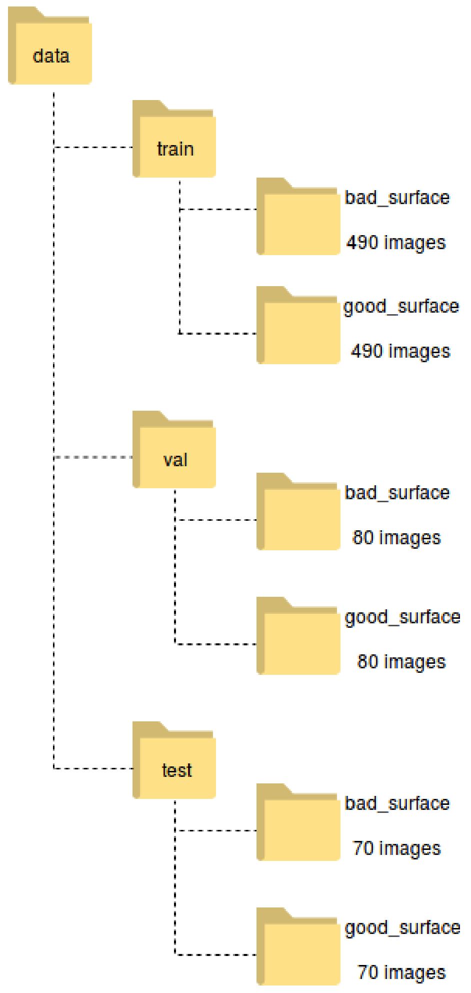


Figure 5.4: Splitting the dataset

5.4 Training and Testing Results using Resnet18 as Fixed Ferature Extractor and PyTorch

5.4.1 PyTorch

PyTorch is a Python package aimed at accelerating deep learning applications, building on the Torch library. PyTorch provides a Numpy-like abstraction for representing tensors, or multiway arrays, and it can take advantage of GPUs for performance. PyTorch provides various tools to load and preprocess the data in an easy way by using the Dataset class and the transforms methods. The models subpackage of PyTorch contains multiple models pre-trained on ImageNet. A pre-trained model is created by calling its constructor from torchvision.models and setting the parameter pretrained to True.

5.4.2 Preprocessing using PyTorch Dataset and Transforms

CNNs require in general few pre-processing. Since the size of the images in the dataset is not consistent, the first pre-processing applied on the dataset is to resize all the samples to $256 \times 256 \times 3$ and then crop the resized image in the center, since most of the images contain the part in the center, in order to have an image of size $224 \times 224 \times 3$. A random horizontal flip is used on the training dataset for data augmentation. The images are then normalized using the means $[0.485, 0.456, 0.406]$ and standard deviations of $[0.229, 0.224, 0.225]$. Figure 5.5 shows a sample (6 images) randomly selected from the training dataset with their corresponding labels.

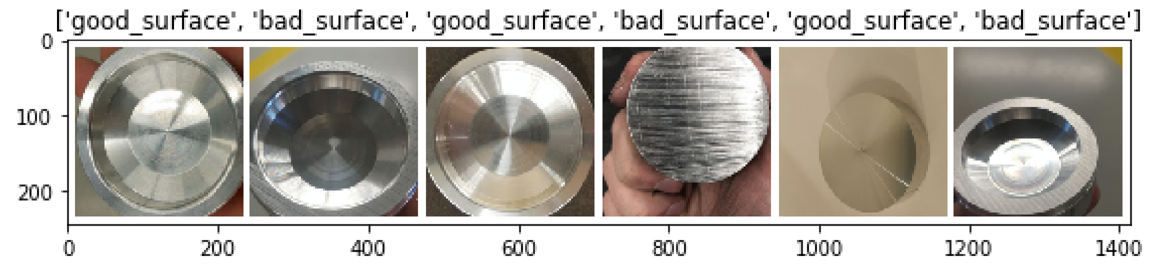

['good_surface', 'bad_surface', 'good_surface', 'bad_surface', 'good_surface', 'bad_surface']
0
100
200
0
200
400
600
800
1000
1200
1400

Figure 5.5: Sample of the training dataset

5.4.3 ResNet18 as Fixed Feature Extractor

ResNet18 model is one of the Residual Networks that became very popular after winning ILSVRC 2015 classification competition with top-5 error rate of 3.57%. ResNet architectures have shown faster convergence and higher accuracy than its counterparts. ResNet18 is 18 layers deep and takes as input an image of size $224 \times 224 \times 3$, and outputs a 1000 vector corresponding the the class scores. In order to use ResNet18 as a fixed feature extractor, the weights are frozen by setting the parameter requires_grad=False in order to prevent the automatic differentiation package from computing the gradients with respect to these weights. In order to change the classifier to perform binary classification, the last fully connected layer of the ResNet18 is reset to output a vector of 2 elements, as shown in the snippet below.

```
model_ft = models.resnet18(pretrained=True)
for param in model_ft.parameters():
    param.requires_grad = False
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
```

The Appendix D shows the architecture of ResNet18 at a layer level with the corresponding parameters used.

5.4.4 Training

All the layers of the ResNet18 model are frozen, and thus only the weights of the last fully connected layer get updated during the training. The mini batch size used for the training is 64; meaning at each iteration(one forward and backward pass of each batch size), 64 samples of the training dataset are used. The cross entropy loss is used to quantify the inaccuracy of predictions of the classifier. The optimization algorithm used is stochastic gradient descent with momentum$= 0.9$ and a learning rate$= 0.001$. The number of epochs (one forward pass and one backward pass of all the training samples) is set to 20. At every epoch, the model obtained is tested on the validation set to monitor the performance of the model. The training is done on CPU took $1h21min55s$. An Nvidia GeForce $GTX1080$ is also used to accelerate training to $1min47s$. Figure 5.6 shows the evolution of the training and validation loss during the training.

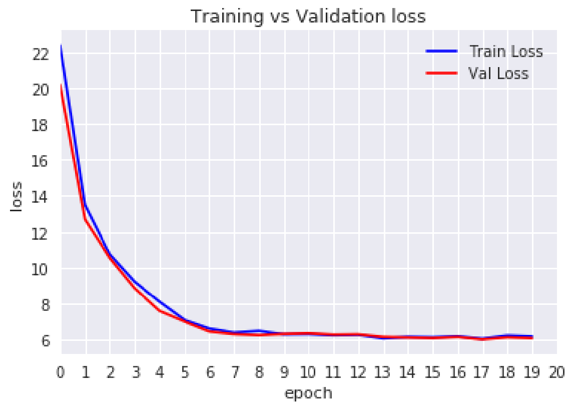


Figure 5.6: Training VS Validation loss

Figure 5.7 shows the evolution of the accuracy of the model on the training and validation dataset at each epoch. Once the training is over, the model with the highest validation accuracy is saved.

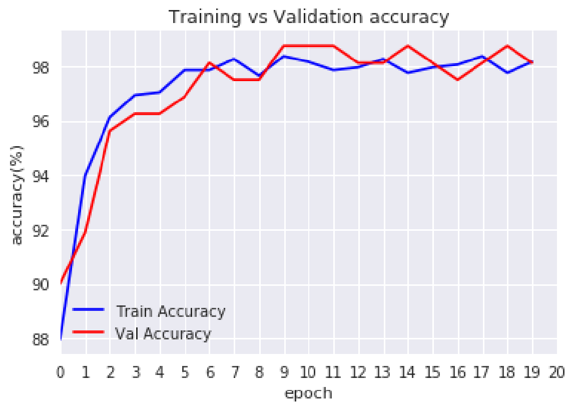
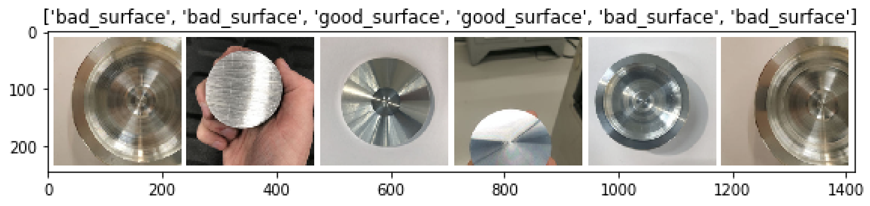


Figure 5.7: Training VS Validation accuracy

5.4.5 Test Results

The trained model is then loaded and the inference is ran on the test dataset in order to assess the performance of the model. The accuracy, defined as as the ratio of correct predictions out of all the predictions made, is appropriate to evaluate the robustness of the model since the classes are balanced. The accuracy on the test dataset is equal to $\frac{137}{140} = 97.86\%$. The model is thus able to correctly label the surface of most of the images of the test dataset, misclassifying only 3 out of 140 samples. Figure 5.8 shows a sample from the test dataset, with the model's predicted classes on top and the actual labels in the bottom.



```
['bad_surface', 'bad_surface', 'good_surface', 'good_surface', 'bad_surface', 'bad_surface']
```

```
Actual labels:
 ['bad_surface', 'bad_surface', 'good_surface', 'good_surface', 'bad_surface', 'bad_surface']
```

Figure 5.8: Sample of the test dataset with the Predicted Vs Actual labels

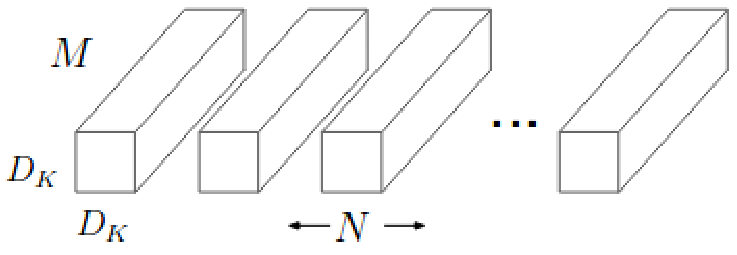
This result shows that the CNN model is a good candidate to automate the surface finish inspection and replicate human judgement in a relatively accurate way.

CHAPTER 6
ANDROID APPLICATION USING MOBILENET AND KERAS

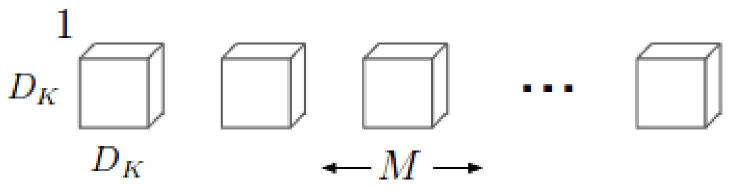
In the goal of providing an affordable, available and robust automated surface finish inspection solution, an android app is designed to allow users to take a picture of a machined part using a mobile device's camera and run the inference to predict whether the image corresponds to a good or bad surface. Running such complex models on mobile devices faces many challenges due to the limited computation resources, power and space. MobileNets [39] overcome these challenges by proposing a small, low latency and accurate CNN for mobile vision applications. Using MobileNet allows to perform inference locally on the device in an efficient way regardless of internet connection.

6.1 MobileNets

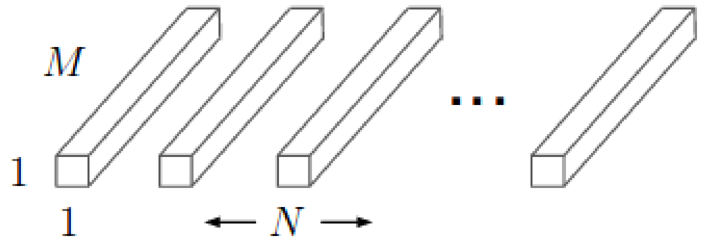
MobileNets are a class of lightweight CNNs developed by a group of researchers at Google [39] to provide an efficient solution to run inference on mobile devices. The main idea behind MobileNets is using depthwise separable convolutions. Instead of performing the convolution operation in the traditional way by multiplying the filter along the full depth of the input and combining all the channels in one output, the MobileNet model performs the convolution operation on each depth slice separately by applying a single filter to each input channel. A depth-wise convolution on a 3 channel image outputs a volume with 3 channels as well. A pointwise operation is then applied on the output volume of the depth-wise convolution, which consists in a regular convolution but with a 1×1 kernel to combine all the channels. A depthwise convolution followed by a pointwise convolution's output is referred to as a depthwise separable convolution. The Figure 6.1 from the original paper shows the depthwise and pointwise convolutional filters in MobileNet compared to typical convolutional filters in standard CNNs.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 6.1: The standard convolutional filters in (a) are replaced by two layers: a depthwise convolution in (b) and a pointwise convolution in (c) to build a depthwise separable filter

Separating the two operations reduces the computation and the model size. MobileNet uses 3×3 depthwise separable convolutions which showed a reduction of computation by a factor of $8 - 9$ compared to standard convolutions with very low loss in accuracy.

6.2 Finetuning the MobileNet CNN

In this section, we use Keras to train the MobileNet CNN. Keras is an open source neural network library written in Python that runs on top of TensorFlow. TensorFlow is an open source deep learning framework developed by Google Brain team within Googles AI organization, that allows the acceleration of the computaion using GPUs and TPUs.

The pre-trained MobileNet model is available in the Keras applications library. The model is imported and the weights are set to 'imagenet' in order to use the model's weights pre-trained on ImageNet dataset. The include_top parameter is set to False in order to discard the last fully connected layer. We add two dense layers (fully connected layers) with ReLU activations to fit the training dataset and a final layer with softmax activation in order to obtain the probabilities. The full 30 layers MobileNet network and the three added layers are shown in Appendix E.

```
base_model=MobileNet(weights='imagenet',include_top=False)

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x=Dense(512,activation='relu')(x)
preds=Dense(2,activation='softmax')(x) #final layer with softmax activation
```

The first 20 layers of MobileNets are frozen to reduce overfitting and the rest of the layers are finetuned during training.

6.2.1 Training

The dataset used for training is the same as the training dataset used in the previous section to train ResNet18. The same pre-processing steps are followed: normalization, shuffling,

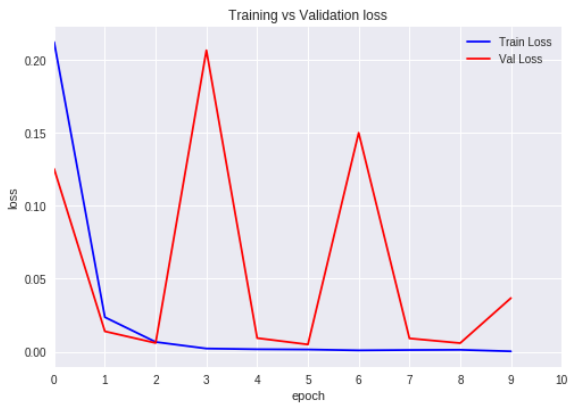


Figure 6.2: Training VS Validation loss

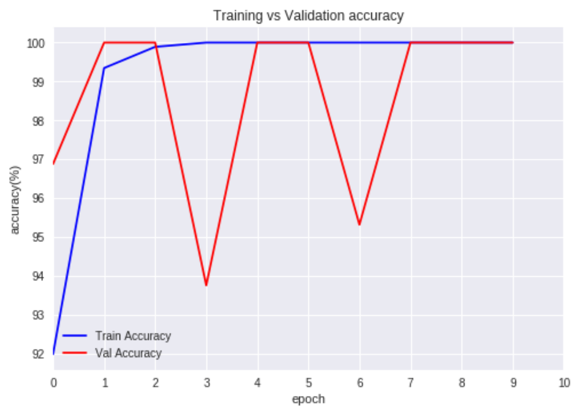


Figure 6.3: Training VS Validation accuracy

random horizontal flipping for the train dataset, resizing to $256 \times 256 \times 3$ and center cropping to obtain a $224 \times 224 \times 3$ input tensor. The data is loaded in batches of 64. The categorical cross-entropy is used as the loss function. In order to update the weights, Adam optimizer is used with an initial learning rate $= 0.0001$. The training is done using 10 epochs. The environment used to perform the training is the online editor Colaboratory, which provides a GPU in order to accelerate the training. The training took $1h32min14s$ using 10 epochs. The hyperparameters are manually tuned in order to get the best model. Figure 6.2 shows the loss on the training and validation datasets of the model at every epoch.

Figure 6.3 shows the accuracy of the trained model at every epoch on the training and the validation dataset.

6.2.2 Test Results

The trained model is saved and the inference is ran on the test dataset. We define here positive as being labelled good surface and negative as bad surface. TP stands for True Positive, FP: False Positive, TN: True Negative and FN: False Negative. We use the test dataset to assess the goodness of the model trained. The results obtained are:

- Accuracy $= 98.57\%$
- True positive(good surface) rate $= \frac{TP}{TP+FN} = \frac{68}{70} = 79.14\%$
- True negative(bad surface) $= \frac{TN}{TN+FP} = \frac{70}{70} = 100\%$

The two test images that were misclassified are:

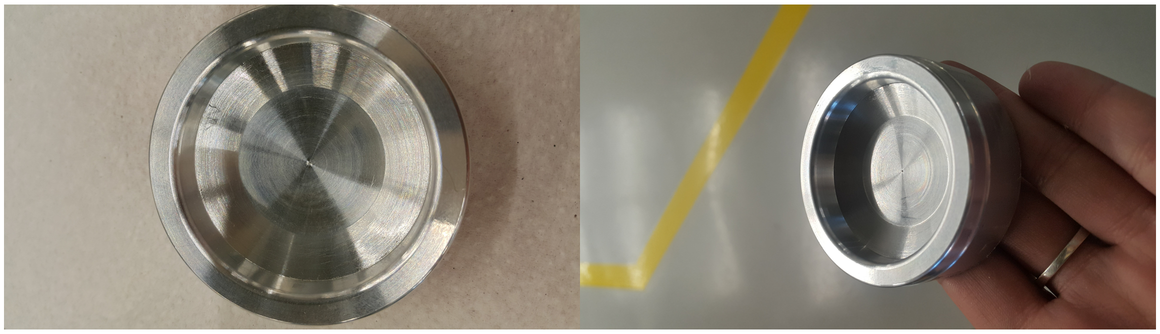


Figure 6.4: Predicted: bad surface
Actual: good surface

6.3 Android Application

6.3.1 Developing the Android App

The Android app consists of two major components. The MainActivity is responsible for taking the photo and the ImageClassifier uses the pretrained model (in form of protobuf file) to classify the photo.

The workflow of the Android App is shown in Figure 6.5:

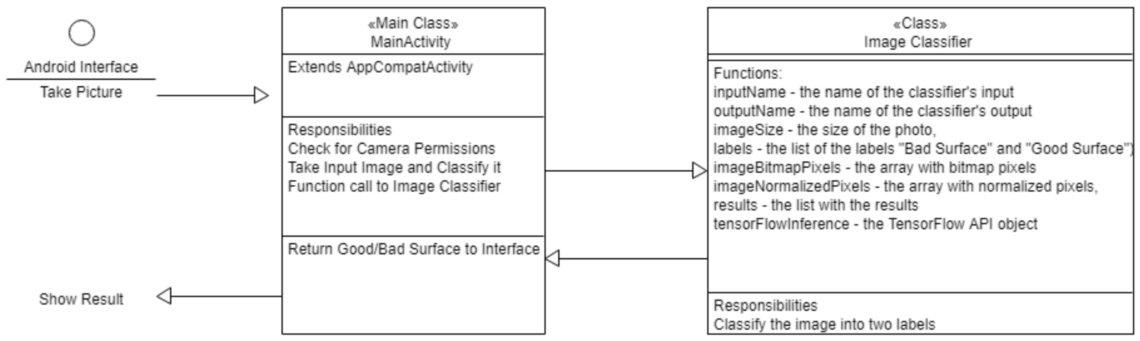

Android Interface
Take Picture
Show Result
«Main Class»
MainActivity
Extends AppCompatActivity
Responsibilities
Check for Camera Permissions
Take Input Image and Classify it
Function call to Image Classifier
Return Good/Bad Surface to Interface
«Class»
Image Classifier
Functions:
inputName - the name of the classifier's input
outputName - the name of the classifier's output
imageSize - the size of the photo,
labels - the list of the labels "Bad Surface" and "Good Surface")
imageBitmapPixels - the array with bitmap pixels
imageNormalizedPixels - the array with normalized pixels,
results - the list with the results
tensorFlowInference - the TensorFlow API object
Responsibilities
Classify the image into two labels

Figure 6.5: Workflow of the Android app

The Android app takes a photo then using the onActivityResult() we make a call to classify it. The interface method takes a Bitmap parameter as input which is our image and returns a Result. The Result class has two attributes: a result string (Good Surface or Bad Surface) and a confidence vector (the output scores of the softmax function of the model as probabilities of both classes). Once the inference is done, we use the probabilities vector to decide which class has the highest probability and attribute the corresponding label. ImageClassifierFactory is created inside the createClassifier() function of the MainActivity class. The assets parameter is an AssetManager instance. The other parameters are placed in the Constants class.
The app uses two files which paths are provided as inputs: the labels file and the protobuf (the model) file.

- GRAPH_FILE_PATH: the path to our classifier file in the assets folder which has our saved neural network graph/model. (output_graph.pb)
- LABELS_FILE_PATH: the path to our labels file in the assets folder. The labels for us are just 'good_surface' and 'bad_surface'. (output.txt)
- GRAPH_INPUT_NAME: the name of the input layer of our graph. (input_10)
- GRAPH_OUTPUT_NAME: name of the output layer of our graph. (k2tfout_0)

- IMAGE_SIZE: the model was trained with image pixels 224×224.
- COLOR_CHANNELS: 3 (RGB).

Once the image is captured by the device's camera, the image pixels are passed to the classifyPhoto() method. Inside this method, we crop the bitmap to fit $224 \times 224 \times 3$ pixels. After this operation we call recognizeImage() method on our classifier instance and we get the results. To crop the image, we use ImageUtils.getCroppedBitmap(). The Factory creates the instance of the classifier. We pass the following parameters from the Activity class:

- Labels: FileUtils class is used to provide the list of strings representing the labels. In our case, as we are performing a binary classification, we have 2 labels.
- ImageBitmapPixels: the array that we allocate based on the image size.
- ImageNormalizedPixels: based on the image size and color channels needed for the inference, we allocate the array. In our app, the dimensions are $224 \times 224 \times 3$. Results: Classifier will assign the probability to each of the classes.

The classification process consists of three steps:

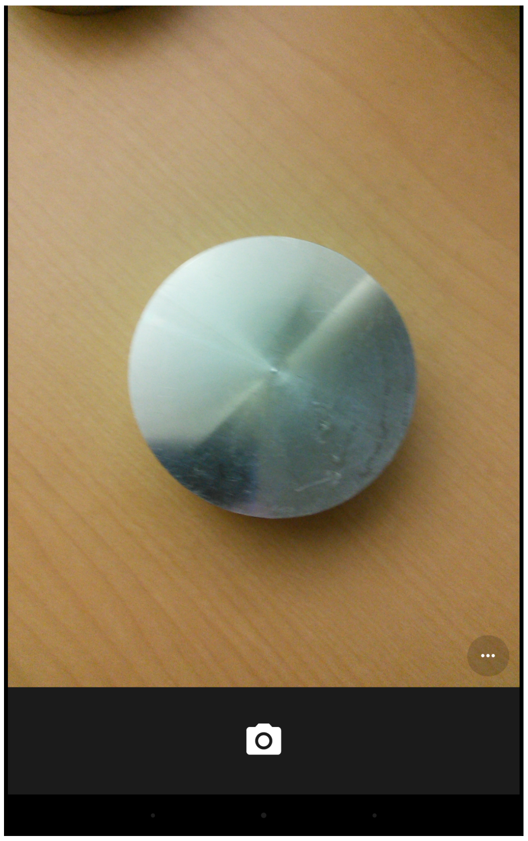
Step 1: feed the classifier's feed() Method: We pass three parameters namely input name, array with normalized values and dimensions of the input.

Step 2: feed the classifier's therun() method: We pass two parameters: an array with two possible outputs and a flag to enable stats.

Step 3: get Results using the fetch() method: The output name from which we want to take the results and the float array which has probabilities assigned to the class.

6.3.2 Testing the Android App

The Android app is tested on a Nexus 7 tablet with android version 6.0.1. The layout of the app is shown below in Figure 6.6. When the app is launched, the camera starts and the layout on the left-side is seen. The small camera logo in the center bottom of the screen allows to capture the image. Once the image is taken and the user is ready to classify it, the check button can be clicked to process the image.


(a) Camera on to capture the image

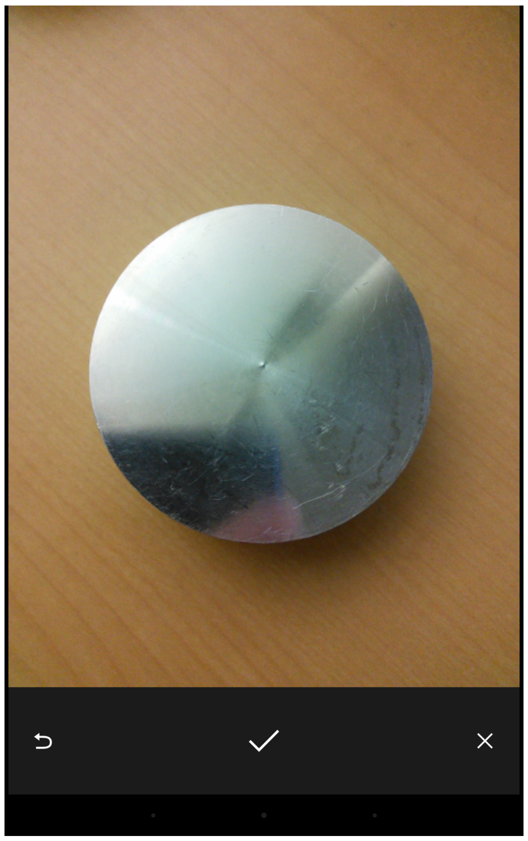
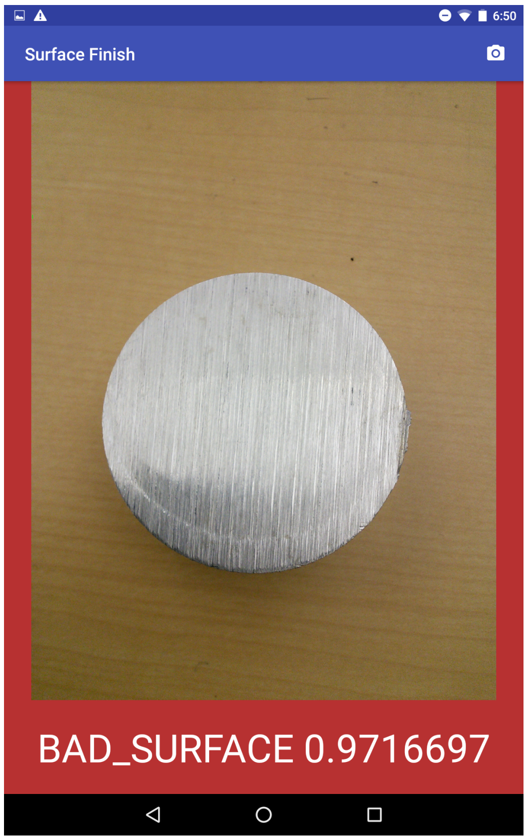
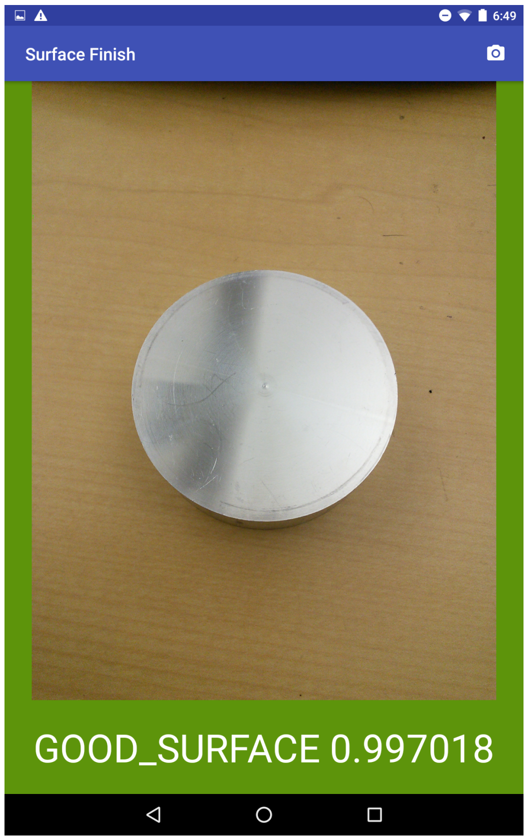

(b) Validate and proceed to classify

Figure 6.6: Android App

The inference takes in average 1.90 ± 0.27 seconds to output the result of the classification method, which confirms the inference speed of MobileNet models on mobile devices. The Figure 6.12 shows the output of the app for a bad surface image on the left, and a good surface on the right.



(a) Bad Surface

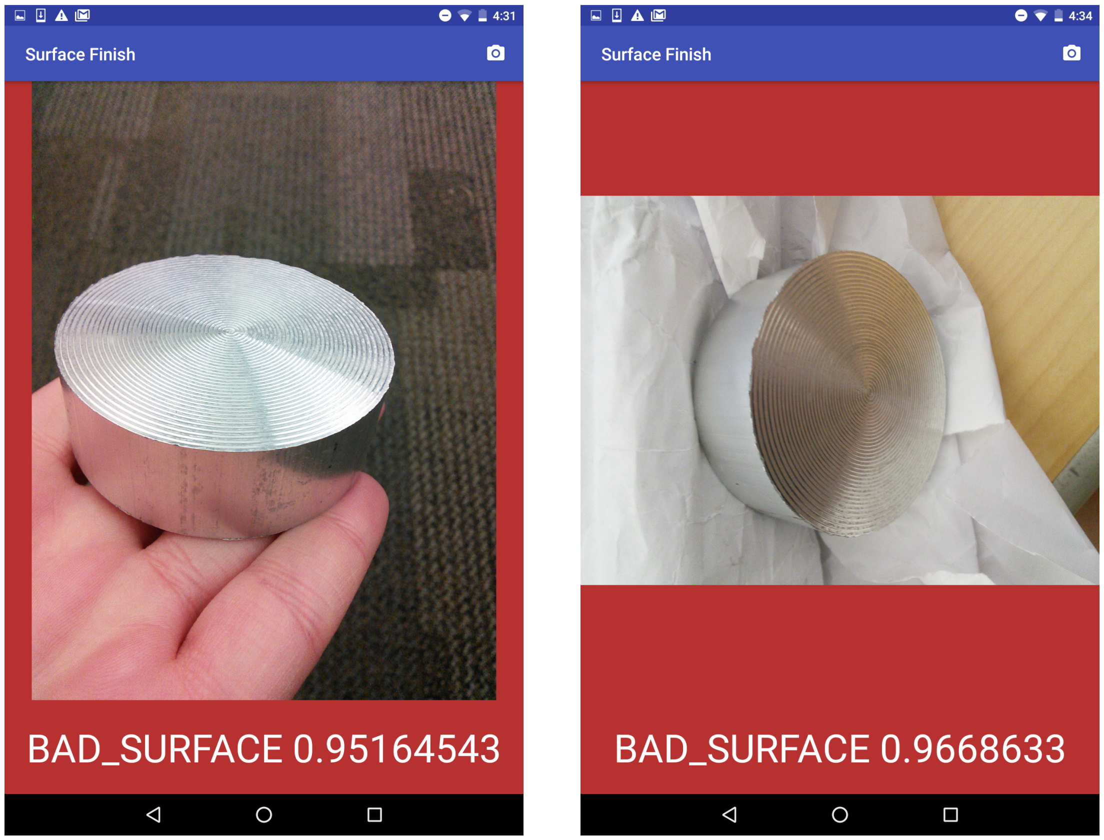


(b) Good Surface

Figure 6.7: Android App

Varying the angles:

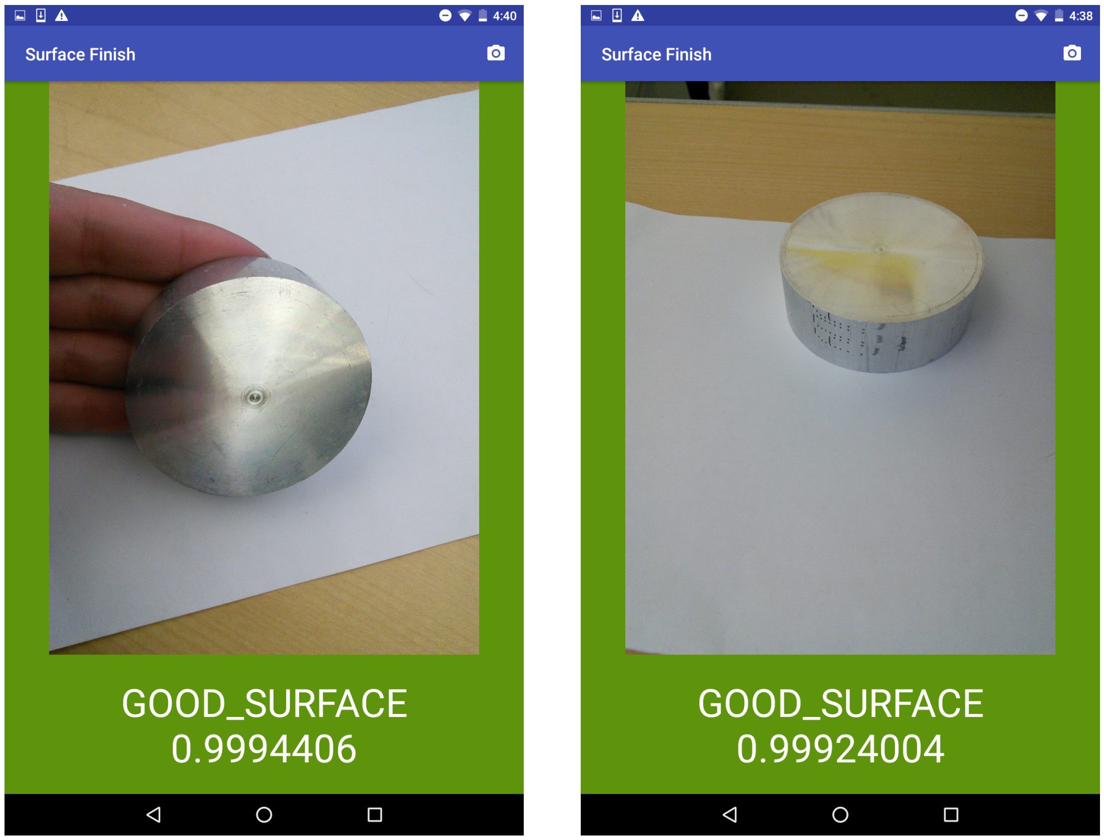
Using the same Nexus 7 tablet, we take pictures of parts from different angles. The results are shown below:



```
BAD_SURFACE 0.95164543
```

```
BAD_SURFACE 0.9668633
```

Figure 6.8: Bad surface finish detected from different angles



GOOD_SURFACE
0.9994406

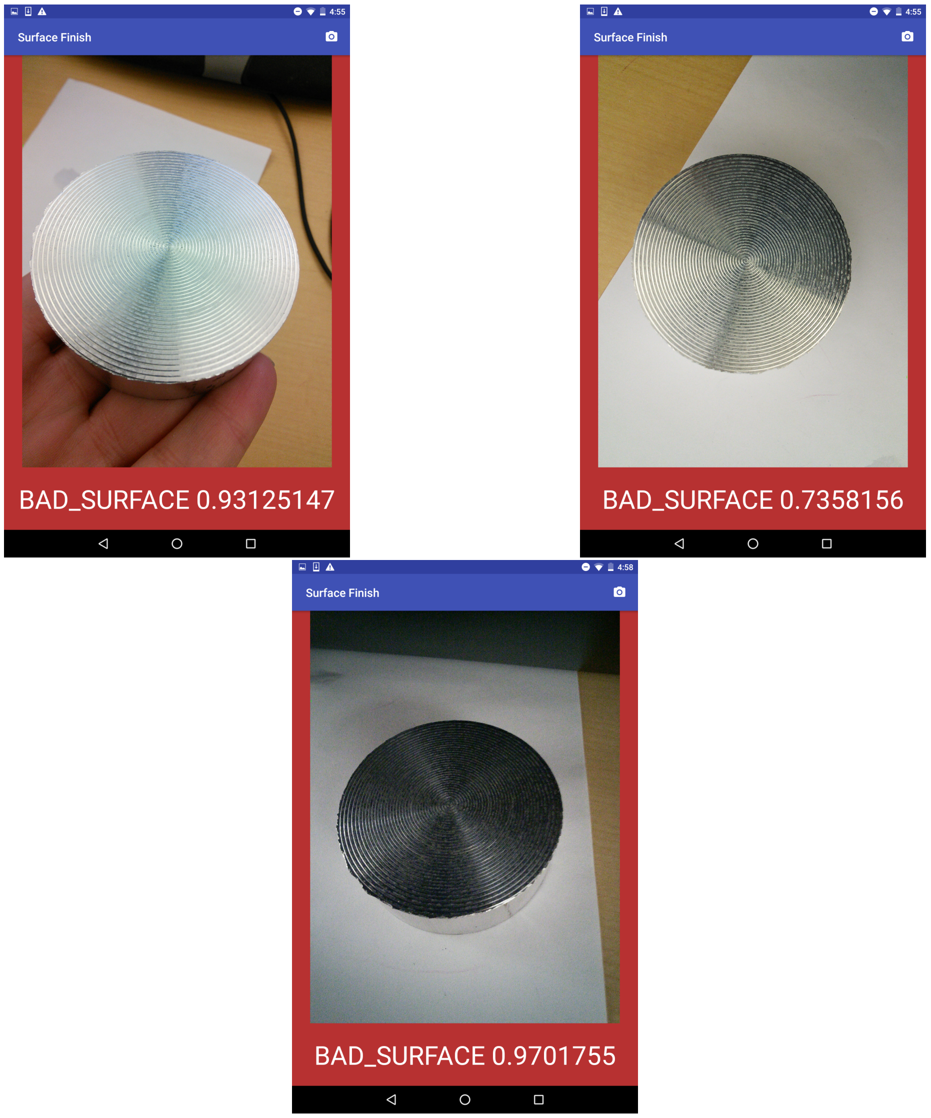
GOOD_SURFACE
0.99924004

Figure 6.9: Good surface finish detected from different angles

These tests show that the app is, in most cases, resistant to varying the angles and the positions of the parts.

Varying the lighting:

We vary the shadows and the lighting conditions under which the pictures are taken to test to app.

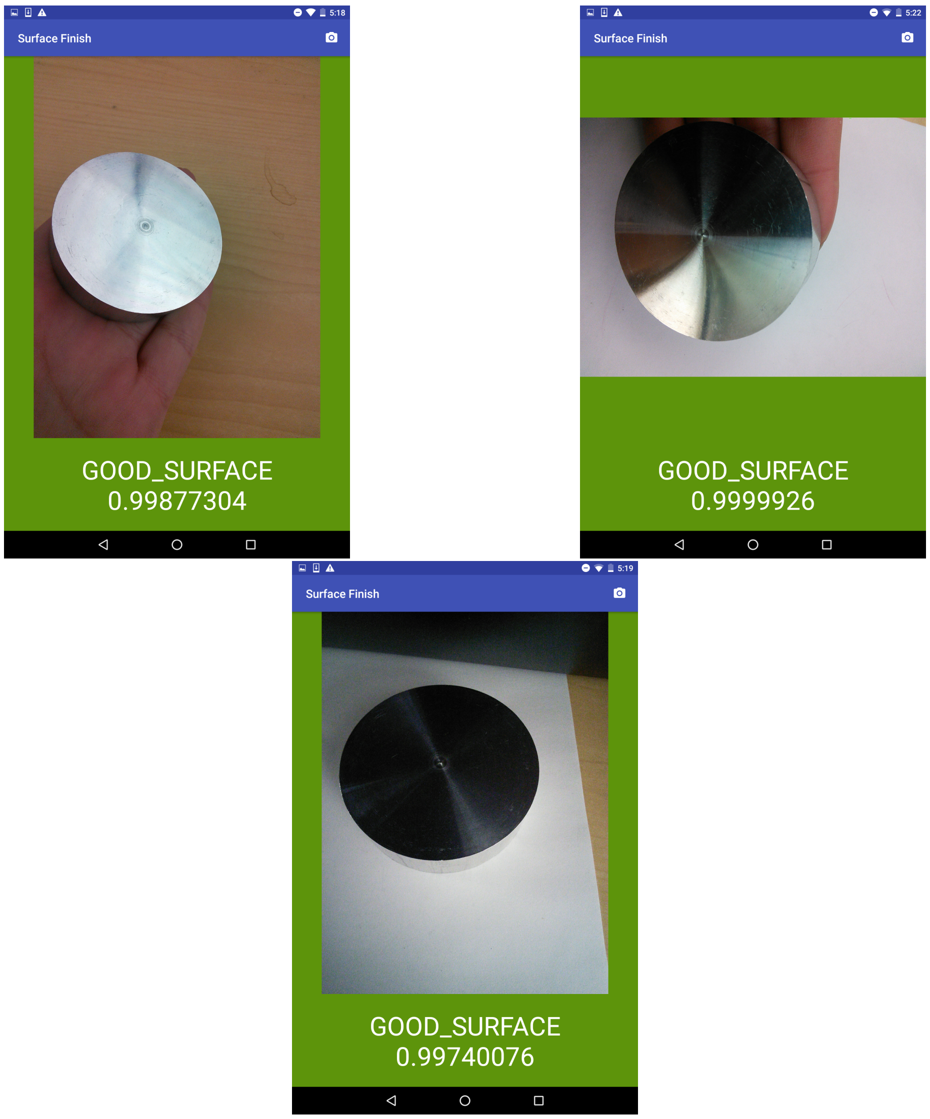


BAD_SURFACE 0.93125147

BAD_SURFACE 0.7358156

BAD_SURFACE 0.9701755

Figure 6.10: Bad surface finish detected under different lighting conditions



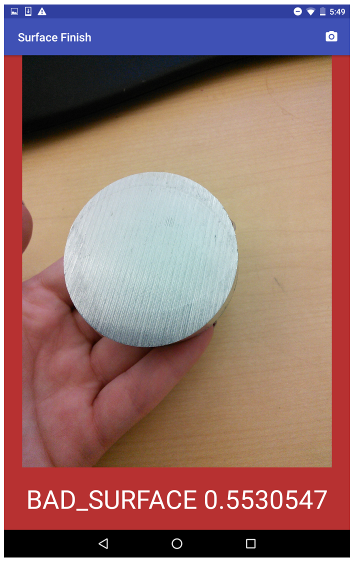
GOOD_SURFACE
0.99877304

GOOD_SURFACE
0.9999926

GOOD_SURFACE
0.99740076

Figure 6.11: Good surface finish detected under different lighting conditions

These tests show that the app is able to correctly classify the images of the surfaces in most of the cases under varying lighting conditions. However, in some cases, if the surface is bad but it looks shiny on the camera the probability of it being a bad surface drops and is sometimes misclassified. The two images below show that the probability is low when the surface is bad, but is put in an angle where it appears shiny, and is misclassified when it is reflecting light and seems like a shiny surface.



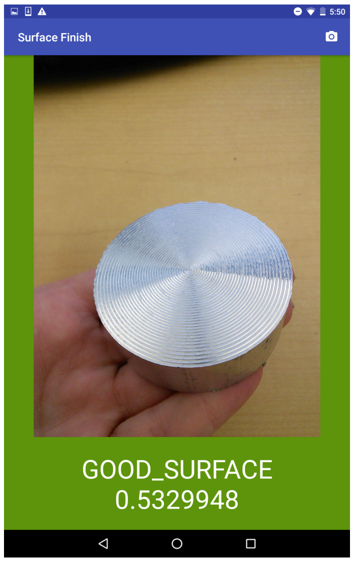


Figure 6.12: Bad surface finish parts misclassified under bright light

CHAPTER 7
CONCLUSION AND RECOMMENDATIONS

In this thesis work, we propose a framework to automate the surface finish inspection of aluminum parts. The CNN models used ResNet18 as well as MobileNet show the power of CNN to classify images and perform a surface finish inspection with only pictures of the surfaces as input. Our approach doesn't require any special hardware or data collection from the manufacturing process. Both models achieved a high classification accuracy and were resistant to variations in lightening conditions, backgrounds, shapes, sizes, angles and positions. The framework proposed in this work overcomes the limitations of the manual approach: not consistent, subject to human errors, not adapted to fast-paced and mass production environments; as well as the current automated surface inspections methods: not limited to a range of surface roughness, requires no special hardware set-up, resistant to hostile machine shop environment. A camera of a mobile device is sufficient to capture an image and perform the classification locally with no need for internet connection. This work thus aims at providing an affordable and easy solution to use in machine shops.

Some of the limitations of this work are related to the restricted dataset used to train and test. The dataset can be enriched to include more complex shapes of parts and tested on a variety of surfaces to ensure the model's robustness. Based on the high accuracy achieved by the current models on this dataset, we presume that the model can be successfully transferable to richer datasets. All the parts used in this work are aluminum, surface finish inspection of other materials can be included as a future direction. This work can be carried further to assigning the exact surface roughness R_a to the corresponding images instead of simply performing a binary classification. A feedback loop can be used to optimize the manufacturing process based on the surface finish inspection of the manufactured parts.

Appendices

APPENDIX A
G-CODE OF THE FACING OPERATION

```
(FACING OPERATION)

G50 S3000          (RPM, MAX SPINDLE SPEED)
G0X[VPVLX]
Z[VPVLZ]

(CHANGE TO TOOL 5)
T050505

G95 (SET FEED UNITS FOR MM/REV)

G96 S250 M3 (SET CONSTANT SURFACE SPEED ON)

FCTA=0.05

(COOLANT ON)
M8

(APPROACH)
G0 X80 Z15

M1   (OPTIONAL STOP, VERIFY LOCATION)
G0 Z5
M1   (OPTIONAL STOP, VERIFY LOCATION)

(BEGIN AT Z=5 MM AWAY FROM PART)
(FACE TO Z=-2MM INTO PART)
(CUT DEPTH = 1MM PER PASS)
(VARY CUTTING FEED IF NEEDED)

(APPROACH)

G1 Z0.25 F[FCTA]
G1 X-1 F[FCTA]
G0 Z1.25
```

```
X80

G1 Z0 F[FCTA]
G1 X-1 F[FCTA]
G0 Z1
X80

G1 Z-0.25 F[FCTA]
G1 X-1 F[FCTA]
G0 Z0.75
X80

G1 Z-0.5F[FCTA]
G1 X-1 F[FCTA]
G0 Z0.5
X80

(MOVE AWAY FROM PART)
G0 X150
Z50

(GO TO HOME POSITION)
G0 X[VPVLX]
Z[VPVLZ]

M30
(END OF PROGRAM)
```

APPENDIX B

G-CODE OF THE YO-YO PRODUCTION

```
(OKONEV.MIN)
(T00 D=0. CR=0. – ZMIN=–11.75 – BORING TURNING)
(T01 D=0. CR=0. – ZMIN=–11.75 – GENERAL TURNING)
(T04 D=0. CR=0. – ZMIN=–11.496 – BORING TURNING)

G0 Z[VPVLZ]
X[VPVLX]

G50 S3000


(FACE5)
M1
G0 Z[VPVLZ]
X[VPVLX]

T050505

M8
G95
G50 S3000
G96 S200 M3
G0 X77.15 Z5.9
G0 Z2.314
G1 X59.978 F0.3
X57.15 Z0.9
X–2.381
X0.447 Z2.314
G0 X77.15
Z1.814
G1 X59.978 F0.3
X57.15 Z0.4
X–2.381
X0.447 Z1.814
G0 X77.15
```

```
Z1.514
G1 X59.978 F0.3
X57.15 Z0.1
X-2.381
X0.447 Z1.514
G0 X77.15
Z5.9

G0 Z[VPVLZ]
X[VPVLX]


(DRILL1)
M1
G0 Z[VPVLZ]
X[VPVLX]
T070717  (T7 DRILL OFFSET)

G95
G97 S1000 M3
G0 Z13.9
X0.

G0 Z5.9

G74 X0. Z-11.75 D1. L1. K4 I0 F0.1

Z13.9

G0 Z[VPVLZ]
X[VPVLX]

(PROFILE1)
M1
G0 Z[VPVLZ]
X[VPVLX]
T020202

M8
G95
G50 S3000
```

```
G96 S300 M3

G0 X23.8 Z5.9
G0 Z1.864
G1 X23.914 F0.1
X25.8 Z0.1
Z−11.496
X23.8
G0 Z1.514
X24.972
G1 X27.8 Z0.1 F0.1
Z−11.496
X24.8
G0 Z1.514
X26.972
G1 X29.8 Z0.1 F0.1
Z−11.216
X28.696 Z−11.496
X26.8
X23.972 Z−10.082
G0 Z1.514
X28.972
G1 X31.8 Z0.1 F0.1
Z−10.709
X28.8 Z−11.47
X25.972 Z−10.055
G0 Z1.514
X30.972
G1 X33.8 Z0.1 F0.1
Z−10.202
X30.8 Z−10.963
X27.972 Z−9.548
G0 Z1.514
X32.972
G1 X35.8 Z0.1 F0.1
Z−9.695
X32.8 Z−10.456
X29.972 Z−9.041
G0 Z1.514
X34.972
G1 X37.8 Z0.1 F0.1
```

```
Z−9.188
X34.8 Z−9.949
X31.972 Z−8.534
G0 Z1.514
X36.972
G1 X39.8 Z0.1 F0.1
Z−8.681
X36.8 Z−9.442
X33.972 Z−8.027
G0 Z1.514
X38.972
G1 X41.8 Z0.1 F0.1
Z−8.174
X38.8 Z−8.935
X35.972 Z−7.52
G0 Z1.514
X40.972
G1 X43.8 Z0.1 F0.1
Z−1.1
G2 X43.492 Z−1.9 L2.154
G1 Z−7.745
X40.8 Z−8.427
X37.972 Z−7.013
G0 Z1.514
X42.972
G1 X45.8 Z0.1 F0.1
Z0.008
G2 X43.492 Z−1.9 L2.154
G1 X40.664 Z−0.486
G0 Z1.514
X43.069
G1 X43.372 F0.1
X46.201 Z0.1
G2 X44.8 Z−0.354 L2.154
G1 X41.972 Z1.06
G0 X23.8
Z5.9
G0 Z[VPVLZ]
X[VPVLX]

(BORE–F)
```

```
M1
G0 Z[VPVLZ]
X[VPVLX]

T020202

G95
G50 S3000
G96 S200 M3
G0 X0. Z5.9
G0 Z1.414
X42.012
G1 X54.302 F0.09
X57.13 Z0.
X47.8
G2 X44. Z-1.9 L1.9
G1 Z-7.901
X28.818 Z-11.75
X0.
G0 Z5.9

G0 Z[VPVLZ]
X[VPVLX]

(PROFILE6)
M1
G0 Z[VPVLZ]
X[VPVLX]

T050505

G95
G50 S3000
G96 S250 M3
G0 X77.15 Z5.9
G0 Z1.414
X58.419
G1 X54.251 F0.3
X51.423 Z0.
G3 X56. Z-2.291 L2.291 F0.1
G1 Z-6.191
```

```
X58.828 Z-4.776 F0.3
X60.
G0 X77.15
Z5.9

G0 Z[VPVLZ]
X[VPVLX]

M9

M30
%
```

```
 (1001.MIN)
(T01 D=0. CR=0. - ZMIN=-14.148 - GENERAL TURNING)
(T02 D=6.325 CR=0. - ZMIN=-2. - REAMER)
(T08 D=0. CR=0. - ZMIN=-16.647 - GENERAL TURNING)

G0 Z[VPVLZ]
X[VPVLX]

G50 S3000


(ROUGHFACE)
M1
G0 Z[VPVLZ]
X[VPVLX]

T060606

M8
G95
G50 S3000
G96 S250 M3

G0 X77.15 Z7.43
G0 Z2.832
G1 X58.564 F0.35
X57.15 Z2.125
X-2.381
X-0.967 Z2.832
```

```
G0 X77.15
Z1.832
G1 X58.564 F0.35
X57.15 Z1.125
X−2.381
X−0.967 Z1.832
G0 X77.15
Z0.832
G1 X58.564 F0.35
X57.15 Z0.125
X−2.381
X−0.967 Z0.832
G0 X77.15
Z7.43

G0 Z[VPVLZ]
X[VPVLX]

(CENTERDRILL)
M1
G0 Z[VPVLZ]
X[VPVLX]

T101010

M8
G95
G97 S2400 M3

G0 X0. Z17.43

G0 Z7.43

G1 Z−2. F0.15

G0 Z7.43

G0 Z[VPVLZ]
X[VPVLX]

(DRILL2 D DRILL)
```

```
M1
G0 Z[VPVLZ]
X[VPVLX]

T080808

G95
G97 S2400 M3

G0 X0. Z17.43
G0 Z7.43

G74 X0. Z-5.5 D2 L1 K4 I0 F0.06

G0 Z7.43

G0 Z[VPVLZ]
X[VPVLX]

(ROUGHPROF)
M1
G0 Z[VPVLZ]
X[VPVLX]

T060606

G95
G50 S3000
G96 S300 M3

G0 X77.15 Z7.43
G0 Z2.916
X60.174
G1 X57.564 F0.3
X56.15 Z2.209
Z-11.088
G3 X57.15 Z-12.025 L3.976
G1 X58.564 Z-12.732
G0 Z3.122
X56.564
G1 X55.15 Z2.415 F0.3
```

```
Z-10.518
G3 X56.65 Z-11.482 L3.976
G1 X58.064 Z-12.189
G0 Z3.137
X55.564
G1 X54.15 Z2.43 F0.3
Z-10.118
G3 X55.65 Z-10.776 L3.976
G1 X57.064 Z-11.483
G0 Z3.137
X54.564
G1 X53.15 Z2.43 F0.3
Z-9.83
X53.295 Z-9.867
G3 X54.65 Z-10.302 L3.976
G1 X56.064 Z-9.594
G0 Z3.137
X53.564
G1 X52.15 Z2.43 F0.3
Z-9.576
X53.295 Z-9.867
G3 X53.65 Z-9.962 L3.976
G1 X55.064 Z-9.255
G0 Z3.137
X52.564
G1 X51.15 Z2.43 F0.3
Z-9.323
X52.65 Z-9.703
X54.064 Z-8.996
G0 Z3.137
X51.564
G1 X50.15 Z2.43 F0.3
Z-9.069
X51.65 Z-9.45
X53.064 Z-8.742
G0 Z3.137
X50.564
G1 X49.15 Z2.43 F0.3
Z-8.816
X50.65 Z-9.196
X52.064 Z-8.489
```

```
G0 Z3.137
X49.564
G1 X48.15 Z2.43 F0.3
Z−8.562
X49.65 Z−8.943
X51.064 Z−8.235
G0 Z3.137
X48.564
G1 X47.15 Z2.43 F0.3
Z−8.309
X48.65 Z−8.689
X50.064 Z−7.982
G0 Z3.137
X47.564
G1 X46.15 Z2.43 F0.3
Z−8.055
X47.65 Z−8.436
X49.064 Z−7.728
G0 Z3.137
X46.564
G1 X45.15 Z2.43 F0.3
Z−7.802
X46.65 Z−8.182
X48.064 Z−7.475
G0 Z3.137
X45.564
G1 X44.15 Z2.43 F0.3
Z−7.548
X45.65 Z−7.928
X47.064 Z−7.221
G0 Z3.137
X44.564
G1 X43.15 Z2.43 F0.3
Z−7.295
X44.65 Z−7.675
X46.064 Z−6.968
G0 Z3.137
X43.564
G1 X42.15 Z2.43 F0.3
Z−7.041
X43.65 Z−7.421
```

```
X45.064 Z-6.714
G0 Z3.137
X42.564
G1 X41.15 Z2.43 F0.3
Z-6.788
X42.65 Z-7.168
X44.064 Z-6.461
G0 Z3.137
X41.564
G1 X40.15 Z2.43 F0.3
Z-6.534
X41.65 Z-6.914
X43.064 Z-6.207
G0 Z3.137
X40.564
G1 X39.15 Z2.43 F0.3
Z-6.281
X40.65 Z-6.661
X42.064 Z-5.954
G0 Z3.137
X39.564
G1 X38.15 Z2.43 F0.3
Z-6.027
X39.65 Z-6.407
X41.064 Z-5.7
G0 Z3.137
X38.564
G1 X37.15 Z2.43 F0.3
Z-5.774
X38.65 Z-6.154
X40.064 Z-5.447
G0 Z3.137
X37.564
G1 X36.15 Z2.43 F0.3
Z-5.52
X37.65 Z-5.9
X39.064 Z-5.193
G0 Z3.137
X36.564
G1 X35.15 Z2.43 F0.3
Z-5.267
```

```
X36.65 Z-5.647
X38.064 Z-4.94
G0 Z3.137
X35.564
G1 X34.15 Z2.43 F0.3
Z-5.013
X35.65 Z-5.393
X37.064 Z-4.686
G0 Z3.137
X34.564
G1 X33.15 Z2.43 F0.3
Z-4.76
X34.65 Z-5.14
X36.064 Z-4.433
G0 Z3.137
X33.564
G1 X32.15 Z2.43 F0.3
Z-4.506
X33.65 Z-4.886
X35.064 Z-4.179
G0 Z3.137
X32.564
G1 X31.15 Z2.43 F0.3
Z-4.253
X32.65 Z-4.633
X34.064 Z-3.926
G0 Z3.137
X31.564
G1 X30.15 Z2.43 F0.3
Z-3.999
X31.65 Z-4.379
X33.064 Z-3.672
G0 Z3.137
X30.564
G1 X29.15 Z2.43 F0.3
Z-3.746
X30.65 Z-4.126
X32.064 Z-3.419
G0 Z3.137
X29.564
G1 X28.15 Z2.43 F0.3
```

```
Z−3.492
X29.65 Z−3.872
X31.064 Z−3.165
G0 Z3.137
X28.564
G1 X27.15 Z2.43 F0.3
Z−3.238
X28.65 Z−3.619
X30.064 Z−2.912
G0 Z3.137
X27.564
G1 X26.15 Z2.43 F0.3
Z−2.985
X27.65 Z−3.365
X29.064 Z−2.658
G0 Z3.137
X26.564
G1 X25.15 Z2.43 F0.3
Z−2.731
X26.65 Z−3.112
X28.064 Z−2.405
G0 Z3.137
X25.564
G1 X24.15 Z2.43 F0.3
Z−2.478
X25.65 Z−2.858
X27.064 Z−2.151
G0 Z3.137
X24.564
G1 X23.15 Z2.43 F0.3
Z−2.224
X24.65 Z−2.605
X26.064 Z−1.898
G0 Z3.137
X23.564
G1 X22.15 Z2.43 F0.3
Z−1.971
X23.65 Z−2.351
X25.064 Z−1.644
G0 Z3.137
X22.564
```

```
G1 X21.15 Z2.43 F0.3
Z-1.717
X22.65 Z-2.098
X24.064 Z-1.391
G0 Z3.137
X21.564
G1 X20.15 Z2.43 F0.3
Z-1.464
X21.65 Z-1.844
X23.064 Z-1.137
G0 Z3.137
X20.564
G1 X19.15 Z2.43 F0.3
Z-1.21
X20.65 Z-1.591
X22.064 Z-0.884
G0 Z3.137
X19.564
G1 X18.15 Z2.43 F0.3
Z-0.957
X19.65 Z-1.337
X21.064 Z-0.63
G0 Z3.137
X18.564
G1 X17.15 Z2.43 F0.3
Z-0.703
X18.65 Z-1.084
X20.064 Z-0.377
G0 Z3.137
X17.564
G1 X16.15 Z2.43 F0.3
Z-0.45
X17.65 Z-0.83
X19.064 Z-0.123
G0 Z3.137
X16.564
G1 X15.15 Z2.43 F0.3
Z-0.196
X16.65 Z-0.577
X18.064 Z0.131
G0 Z3.137
```

```
X15.689
G1 X14.275 Z2.43 F0.3
Z0.025
X15.65 Z-0.323
X17.064 Z0.384
G0 Z3.137
X14.814
G1 X13.4 Z2.43 F0.3
Z0.247
X14.775 Z-0.101
X16.189 Z0.606
G0 Z0.832
G1 X14.814 F0.3
X13.4 Z0.125
Z-0.538 F0.097
X52.662 Z-10.491
G3 X56.25 Z-13.412 L3.276
G1 Z-13.441
X57.664 Z-14.148 F0.3
X59.991
G0 X77.15
Z7.43

G0 Z[VPVLZ]
X[VPVLX]


(FINFACE)
M1
G0 Z[VPVLZ]
X[VPVLX]
T040404

G95
G50 S3000
G96 S250 M3

G0 X87.15 Z7.43
G0 Z0.707
G1 X58.564 F0.35
X57.15 Z0.
```

```
X-0.794
X0.62 Z0.707
G0 X87.15
Z7.43

G0 Z[VPVLZ]
X[VPVLX]


(FINPROF)
M1
G0 Z[VPVLZ]
X[VPVLX]

G95
G50 S3000
G96 S500 M3

G0 Z7.43
X77.15
Z1.414
X15.644
G1 X14.828 F0.12
X12. Z0.
Z-0.021
G3 X12.104 Z-0.043 L0.397
G1 X53.418 Z-10.517
G3 X56. Z-12.619 L2.357
G1 Z-16.547
G3 X55.993 Z-16.647 L1.497
G1 X58.822 Z-15.233
X60.
G0 X77.15
Z7.43

G0 Z[VPVLZ]
X[VPVLX]

(HOLE BORING OPERATION)
M1
G0 Z[VPVLZ]
```

```
X[VPVLX]

T121212

M8
G96 S200 M3
G95

(APPROACH)
G0 X6. Z15.
G0 Z2.
G0 X6.473  (BORE DIAMETER = .2489IN)
M1 (CHECK BORE PLACEMENT)

G1 Z1 F0.02

(BORE PRESSFIT DIA)
G1 Z−4.5 F0.02

(MOVE SLIGHTLY INWARD BEFORE RETRACT)
X6.35

G0 Z2 (CHANGE TO RAPID OUT)

G0 Z[VPVLZ]
X[VPVLX]

M9
M30
%
```

APPENDIX C
PYTHON SCRIPT FOR SPLITTING THE DATA

```
import os
import random
from PIL import Image

#Original directory with two subfolders:
#       good_surface: containing all 640 images of good surfaces
#       bad_surface: containing all 640 images of bad surfaces
orig_dir = "original"
good_dir = os.path.join(orig_dir,'good_surface')
bad_dir = os.path.join(orig_dir, 'bad_surface')

good_surface_files = os.listdir(good_dir)
bad_surface_files = os.listdir(bad_dir)
random.shuffle(good_surface_files)
random.shuffle(bad_surface_files)

# test dataset
test_good = good_surface_files[:70]
test_bad = bad_surface_files[:70]

#validation dataset
val_good = good_surface_files[70:150]
val_bad = bad_surface_files[70:150]

#train dataset
train_good = good_surface_files[150:]
train_bad = bad_surface_files[150:]

def save(surface,dataset,listfiles):
    source = os.path.join('original','{}_surface'.format(surface))
    target_path = os.path.join('data',dataset)
    target = os.path.join(target_path,'{}_surface'.format(surface))
    if not os.path.exists(target):
        os.mkdir(target)
    for filename in listfiles:
        image = Image.open(os.path.join(source, filename))
        image.save(os.path.join(target, filename.split('/')[-1]))

save('good','test',test_good)
save('bad','test',test_bad)
save('good','val',val_good)
save('bad','val',val_bad)
save('good','train',train_good)
save('bad','train',train_bad)
```

APPENDIX D
RESNET18 LAYERS

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
     padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=
     True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1,
     dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1,
          1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine
         =True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1,
          1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine
         =True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1,
          1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine
         =True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1,
          1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine
         =True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride
         =(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2,
           2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride
         =(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2,
            2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
        )
    )
    (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride
            =(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride
            =(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
    )
)
(layer4): Sequential(
    (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride
            =(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride
            =(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
        (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2,
            2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride
         =(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

APPENDIX E
MOBILENETS LAYERS

```
0 input_2
1 conv1_pad
2 conv1
3 conv1_bn
4 conv1_relu
5 conv_dw_1
6 conv_dw_1_bn
7 conv_dw_1_relu
8 conv_pw_1
9 conv_pw_1_bn
10 conv_pw_1_relu
11 conv_pad_2
12 conv_dw_2
13 conv_dw_2_bn
14 conv_dw_2_relu
15 conv_pw_2
16 conv_pw_2_bn
17 conv_pw_2_relu
18 conv_dw_3
19 conv_dw_3_bn
20 conv_dw_3_relu
21 conv_pw_3
22 conv_pw_3_bn
```

```
23 conv_pw_3_relu
24 conv_pad_4
25 conv_dw_4
26 conv_dw_4_bn
27 conv_dw_4_relu
28 conv_pw_4
29 conv_pw_4_bn
30 conv_pw_4_relu
31 conv_dw_5
32 conv_dw_5_bn
33 conv_dw_5_relu
34 conv_pw_5
35 conv_pw_5_bn
36 conv_pw_5_relu
37 conv_pad_6
38 conv_dw_6
39 conv_dw_6_bn
40 conv_dw_6_relu
41 conv_pw_6
42 conv_pw_6_bn
43 conv_pw_6_relu
44 conv_dw_7
45 conv_dw_7_bn
46 conv_dw_7_relu
47 conv_pw_7
48 conv_pw_7_bn
49 conv_pw_7_relu
```

```
50 conv_dw_8
51 conv_dw_8_bn
52 conv_dw_8_relu
53 conv_pw_8
54 conv_pw_8_bn
55 conv_pw_8_relu
56 conv_dw_9
57 conv_dw_9_bn
58 conv_dw_9_relu
59 conv_pw_9
60 conv_pw_9_bn
61 conv_pw_9_relu
62 conv_dw_10
63 conv_dw_10_bn
64 conv_dw_10_relu
65 conv_pw_10
66 conv_pw_10_bn
67 conv_pw_10_relu
68 conv_dw_11
69 conv_dw_11_bn
70 conv_dw_11_relu
71 conv_pw_11
72 conv_pw_11_bn
73 conv_pw_11_relu
74 conv_pad_12
75 conv_dw_12
76 conv_dw_12_bn
```

```
77 conv_dw_12_relu
78 conv_pw_12
79 conv_pw_12_bn
80 conv_pw_12_relu
81 conv_dw_13
82 conv_dw_13_bn
83 conv_dw_13_relu
84 conv_pw_13
85 conv_pw_13_bn
86 conv_pw_13_relu
87 global_average_pooling2d_2
88 dense_5
89 dense_6
90 dense_7
```

REFERENCES

[1] D. B. Ioan D. Marinescu Constantin Ispas, *Handbook of machine tool analysis*. Boca Raton: CRC Press, 2002.

[2] S. Abainia and N. Ouelaa, “Experimental study of the combined influence of the tool geometry parameters on the cutting forces and tool vibrations,” *The International Journal of Advanced Manufacturing Technology*, vol. 79, no. 5, pp. 1127–1138, 2015.

[3] N. Neogi, D. K. Mohanta, and P. K. Dutta, “Review of vision-based steel surface inspection systems,” *EURASIP Journal on Image and Video Processing*, vol. 2014, no. 1, p. 50, 2014.

[4] J. K.R. A. Degarmo E. Paul; Black, *Materials and Processes in Manufacturing (9th ed.)* Wiley., 2003.

[5] E. Gadelmawla, M. Koura, T. Maksoud, I. Elewa, and H. Soliman, “Roughness parameters,” *Journal of Materials Processing Technology*, vol. 123, no. 1, pp. 133 – 145, 2002.

[6] T. Vorburger and E. Teague, “Optical techniques for on-line measurement of surface topography,” *Precision Engineering*, vol. 3, pp. 61–83, 1981.

[7] H. Fujii and T. Asakura, “Effect of surface roughness on the statistical distribution of image speckle intensity,” *Optics Communications*, vol. 11, no. 1, pp. 35 –38, 1974.

[8] T. Jeyapoovan and M. Murugan, “Surface roughness classification using image processing,” *Measurement*, vol. 46, no. 7, pp. 2065 –2072, 2013.

[9] R. Haralick, K Shanmugam, and I. Dinstein, “Texture features for image classification,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 3, Jan. 1975.

[10] E. Gadelmawla, “A vision system for surface roughness characterization using the gray level co-occurrence matrix,” *NDT E International*, vol. 37, no. 7, pp. 577 –588, 2004.

[11] M. Kiran, B. Ramamoorthy, and V. Radhakrishnan, “Evaluation of surface roughness by vision system,” *International Journal of Machine Tools and Manufacture*, vol. 38, no. 5, pp. 685 –690, 1998, International Conference on Metrology and Properties of Engineering Surfaces.

[12] R. Kumar, P. Kulashekar, B. Dhanasekar, and B. Ramamoorthy, “Application of digital image magnification for surface roughness evaluation using machine vision,” *International Journal of Machine Tools and Manufacture*, vol. 45, no. 2, pp. 228 –234, 2005.

[13] R. Kamguem, S. A. Tahan, and V. Songmene, “Evaluation of machined part surface roughness using image texture gradient factor,” *International Journal of Precision Engineering and Manufacturing*, vol. 14, no. 2, pp. 183–190, 2013.

[14] M. G. S. Raman, “Machine vision assisted characterization of machined surfaces,” *International Journal of Production Research*, vol. 39, no. 4, pp. 759 –784, 2001.

[15] W. Liu, X. Tu, Z. Jia, W. Wang, X. Ma, and X. Bi, “An improved surface roughness measurement method for micro-heterogeneous texture in deep hole based on gray-level co-occurrence matrix and support vector machine,” *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 1, pp. 583–593, 2013.

[16] H. Yi, J. Liu, P. Ao, E. Lu, and H. Zhang, “Visual method for measuring the roughness of a grinding piece based on color indices,” *Opt. Express*, vol. 24, no. 15, pp. 17 215–17 233, 2016.

[17] B. Dhanasekar and B. Ramamoorthy, “Assessment of surface roughness based on super resolution reconstruction algorithm,” *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 11, pp. 1191–1205, 2008.

[18] B. Lee and Y. Tarng, “Surface roughness inspection by computer vision in turning operations,” *International Journal of Machine Tools and Manufacture*, vol. 41, no. 9, pp. 1251 –1263, 2001.

[19] A. G. Ivakhnenko, “Polynomial theory of complex systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 4, pp. 364–378, 1971.

[20] D.-M. Tsai, J.-J. Chen, and J.-F. Chen, “A vision system for surface roughness assessment using neural networks,” *The International Journal of Advanced Manufacturing Technology*, vol. 14, no. 6, pp. 412–422, 1998.

[21] S. Palani and U. Natarajan, “Prediction of surface roughness in cnc end milling by machine vision system using artificial neural network based on 2d fourier transform,” *The International Journal of Advanced Manufacturing Technology*, vol. 54, no. 9, pp. 1033–1042, 2011.

[22] P. Priya and B. Ramamoorthy, “The influence of component inclination on surface finish evaluation using digital image processing,” *International Journal of Machine Tools and Manufacture*, vol. 47, no. 3, pp. 570 –579, 2007.

[23] S. H. Yang, U. Natarajan, M. Sekar, and S. Palani, "Prediction of surface roughness in turning operations by computer vision using neural network trained by differential evolution algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 9, pp. 965–971, 2010.

[24] J. M. Bennett, "Recent developments in surface roughness characterization," *Measurement Science and Technology*, vol. 3, no. 12, pp. 1119–1127, 1992.

[25] R. K. E. DeGarmo J.T. Black, *Material and processing in manufacturing*. John Wiley and Sons, New York, NY, 2003.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] A. Karpathy, "Stanford University CS231n: Convolutional Neural Networks for Visual Recognition,"

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: `1502.03167`.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[32] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. arXiv: `1311.2901`.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. arXiv: `1409.4842`.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: `1512.03385`.

[36] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: Erlbaum, 1986.

[37] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145 –151, 1999.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: `1704.04861`.