Machine Learning Engineer Nanodegree

Capstone Report

Automated Optical Inspection using Deep Learning

Sai Tai

July 22th, 2019

Definition

Project Overview

Automated Optical Inspection (AOI) is commonly used in the electronics and manufacturing industry to detect defects in products or its components. The traditional process takes place at regular intervals of time, usually a few months to tune the right parameters for building an ideal product model. It mainly uses the learning-based or rule-based method for defect classification, involving the whole production line for the purpose of quality control.

Since electronics and surface mount devices (SMD) are getting smaller nowadays, the accuracy of AOI is declining, thus an increasing need for manual inspections to prevent unidentified defect products. According to research, visual inspection errors typically range from 20% to 30% [1].

There are lots of different literatures that attempted to leverage machine learning to improve the efficiency of an assembly line - from printed circuit board (PCB) to SMD. As a reference, ‘Classification of incorrectly picked components using Convolutional Neural Networks’[4] reached an accuracy rate of more than 90% by using CNN to detect defects of chip carriers.

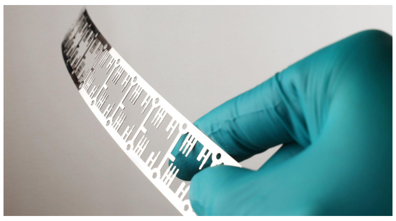


Fig.1 Smd Led Lead Frame[3]

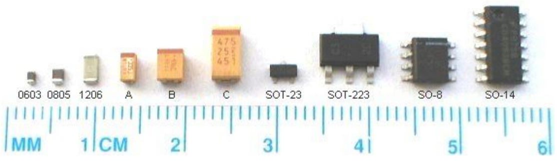


Fig.2 SMD component example[4]

Problem Statement

The goal of this Capstone Project is to create models that can detect different defects; the tasks involved are the following:

1. Preprocess the surface defect data if needed
2. Train classifiers that can determine defects
    a. Machine Learning classifiers
    b. Deep Learning classifiers including transfer learning
3. Install the model into an AOI machine

If the defects can be successfully distinguished by the model, the next step of this project will be detecting defects from more complex electronics components or circuit boards.

[1] Drury C.G., Sinclair M.A., Human and machine performance in an inspection task. Human Factors, 25, 391–399, 1983.
[2] ERIC KOLIBACZ, Classification of incorrectly picked components using Convolutional Neural Networks, 32, 2018.
[3] Figure 1: Smd Led Lead Frame <http://wallsviews.co/smd-led-lead-frame/>
[4] Figure 2: SMD (Surface Mount Device) <https://www.fpga4fun.com/SMD.html>

Metrics

Accuracy

We are dealing with a multilabel classification problem since our goal is detecting different defects. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.
If ŷi is the predicted value of the i-th sample and ŷi is the corresponding true value, then the fraction of correct predictions over $n_{samples}$ is defined as

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

Accuracy equation we use for model training

$$Accuracy = \frac{Number\ of\ correctly\ classified\ Images}{Total\ number\ of input\ images}.$$

Accuracy equation we use for final result

In the dataset, those 6 defects are evenly separated, with 300 images for each defect thus meaning the dataset is balanced. Thus, accuracy is an appropriate metric to evaluate the project.

Confusion matrix

Calculating a confusion matrix can always give us a better idea of what our classification model is getting right and what types of errors it is making. In particular, we can avoid to encounter these 2 problems[1]:

1. When the data has more than 2 classes. With 3 or more classes you may get a classification accuracy of 80%, but you don't know if that is because all classes are being predicted equally well or whether one or two classes are being neglected by the model.
2. When the data does not have an even number of classes. The model may achieve accuracy of 90% or more, but this is not a good score if 90 records for every 100 belong to one class and the model can achieve this score by always predicting the most common class value.

[1] Classification Accuracy and its Limitations
<https://machinelearningmastery.com/confusion-matrix-machine-learning/>

Analysis

Data Exploration

The dataset we use is called NEU surface dataset[1] - it contains 300 pictures (200 × 200 pixels) of each of the six deformities .e., rolled-in scale, patches, crazing, pitted surface, inclusion and scratches. That means there are a total of 1800 images. The images in the dataset are gray-level images of 40.1 KB each.

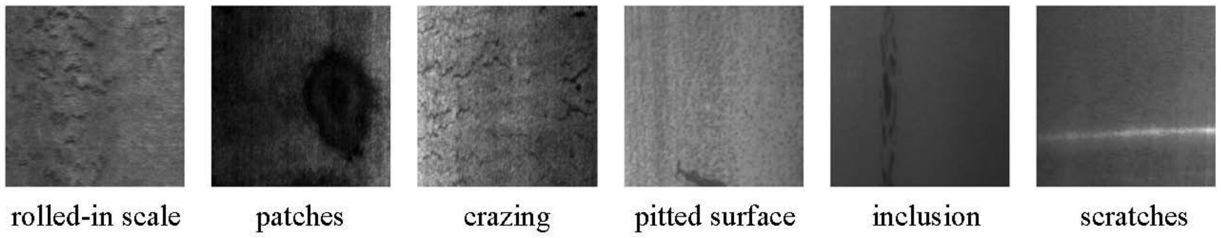


Fig.3 Samples of 6 different defects

Exploratory Visualization

Since our data is in greyscale, we can focus on characteristics that are used for analysing black and white images. In particular, they are levels on contrast, dissimilarity, homogeneity, ASM and energy[1]. After trying those properties I found that the contrast level of each different defect is visually distinguishable, which makes contrast an important feature to be included in the feature vector.

To prove that, below are histograms of contrast level in different defects. Histogram acts as a graphical representation of the tonal distribution in a digital image. The horizontal axis of the graph represents the intensity variations; the vertical axis represents the number of pixels of that particular intensity.

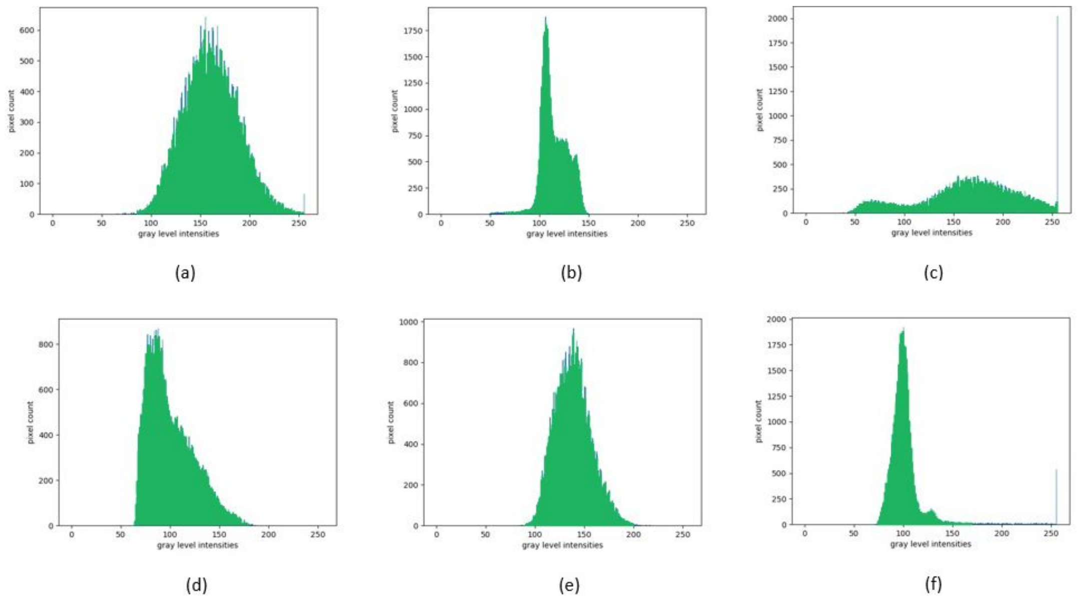


Fig.4 Histogram of 6 defects

[1] Kechen Song and Yunhui Yan - NEU surface defect database
<http://faculty.neu.edu.cn/yunhyan/NEU_surface_defect_database.html>

Algorithms and Techniques

I will use low-level features extraction to train machine learning algorithms and low-level features to train deep learning models, then compare them to find the best suit for final tuning. For the machine learning part, I will use gray-level co-occurrence matrix (GLCM)[1] to extract low-level features of each photo such as contrast, dissimilarity, homogeneity, energy, and asymmetry for training different machine learning classifiers.

- Support Vector Machines (SVM) with RBF kernel
- K-nearest neighbors (KNN)
- Decision Tree
- Ensemble methods - AdaBoost (adaptive boosting)

Also, I will use grid search to find the best hyperparameters for each machine learning model. For example, c in Support Vector Machines, k in k-Nearest Neighbors, max_depth in Decision Tree.

For the deep learning part, I will build a convolutional neural network (CNN) from scratch and also use transfer learning in both VGG-16 and ResNet-50 pre-trained model to learn those high-level representations (interpreted by humans). The reason that I want to have 2 pre-trained models is trying more than one architecture / model as a backup plan doesn’t only give me a good relative comparison, it is also a good way to avoid getting stuck when either one model doesn't work well for our specific tasks.

- Build a CNN from scratch
- Transfer learning with VGG-16
- Transfer learning with ResNet-50

Machine learning approaches

SVM

SVM can be used for classification or regression problems. In our case we will use *RBF*, a Non-linear SVM kernel, which means that the boundary that the algorithm calculates doesn't have to be a straight line.

We will also tune the *gamma* parameter to define how much influence a single training example has and *C,* the penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

The main reason I choose SVM is its efficiency which allows the number of features to be greater than the number of samples, but it is crucial to pick regularization term to overcome over-fitting.

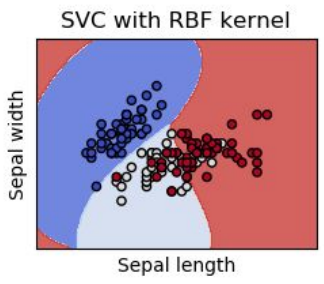


Fig.5 SVM with RBF

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

RBF formula

Decision Tree

The goal of decision tree is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

The reason I choose DTs is because the cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. It also requires fewer data preprocessing from the user, for example, there is no need to normalize columns.

The hyperparameter I will use is *max_depth* to define the maximum depth of the tree.

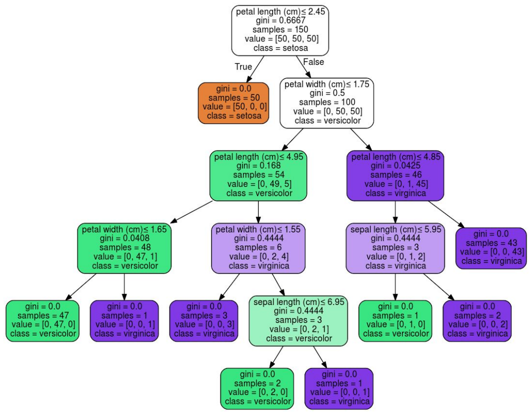


Fig.6 KNN, 3 classes classification

[1] Greycoprops <https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.greycoprops>
2 https://scikit-learn.org/stable/modules/svm.html#svm-kernels
3 https://scikit-learn.org/stable/modules/tree.html#classification

K-nearest neighbors

In k Nearest Neighbors, we try to find the most similar k number of a datapoint as nearest neighbors to another given datapoint, and predict the category by calculating the distance (Euclidean / Manhattan / Minkowski Distance) of its neighbors. The only parameter I will tune is *n_neighbors*, used to define the neighbors to use by default for kneighbors queries.

I choose this algorithm because it doesn't have training step which allows the training time to be fast. It doesn't explicitly build any model, it simply tags the new data entry based on learning from historical data. New data entry would be tagged with majority class in the nearest neighbor.

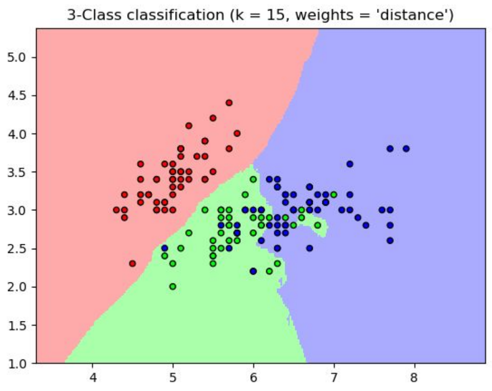
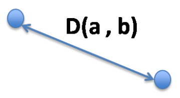


Fig.7 KNN, 3 classes classification[1]

$$D(a,b)=\sqrt{\sum_{i=1}^{n}(b_i-a_i)^2}$$



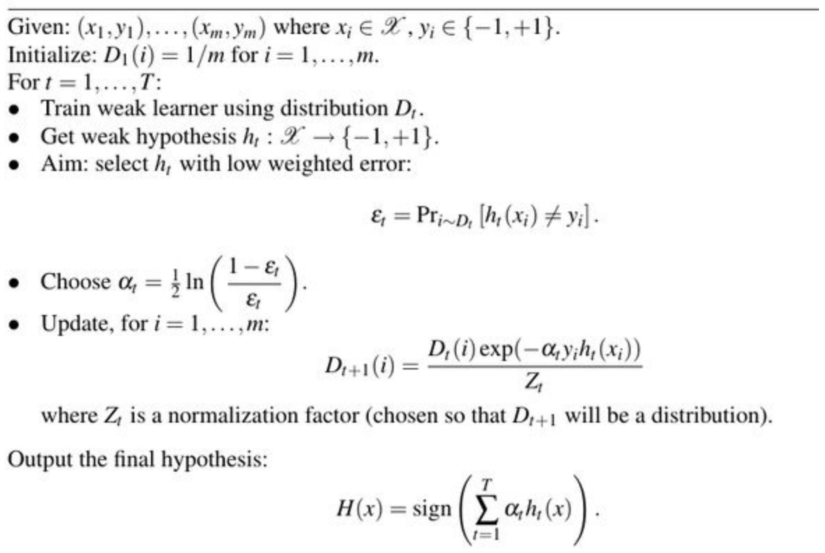
Euclidean Distance[2]

AdaBoost (Adaptive boosting)

Bagging and Boosting are similar in that they are both ensemble techniques, where a set of weak / lazy learners are combined to create a strong learner that obtains better performance than a single one. A 'weak' learner (classifer, predictor, etc) is just one which performs relatively poorly, its accuracy is above chance, but just barely. There is often, but not always, the added implication that it is computationally simple.

The weak learner I will use is Decision Tree. Although, decision trees are usually unstable which means a small change in the data can lead to huge changes in the optimal tree structure. However, their simplicity makes them a strong candidate for a wide range of applications.

For the decision tree, I will tune *max_depth* parameter 2 to make the weak learner even weaker. For the AdaBoost part, I will tune the *n_estimators* to define the maximum number of estimators at which boosting is terminated.



Given: $(x_1,y_1),\dots,(x_m,y_m)$ where $x_i \in \mathscr{X}$, $y_i \in \{-1,+1\}$.
Initialize: $D_1(i)=1/m$ for $i=1,\dots,m$.
For $t=1,\dots,T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathscr{X} \to \{-1,+1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i\sim D_t}\left[h_t(x_i)\neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i=1,\dots,m$:

$$D_{t+1}(i)=\frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x)=\operatorname{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right).$$

Fig.8 Adaboost

1 KNN https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification
2 Euclidean Distanc http://cs.carleton.edu/cs_comps/0910/netflixprize/final_results/knn/index.html
https://towardsdatascience.com/adaboost-for-dummies-breaking-down-the-math-and-its-equations-into-simple-terms-87f439757dcf

Deep learning approaches

Build CNN from Scratch

The illustration on the right shows the computational graph, which includes the convolutional network architecture and also variables that are only present during training.

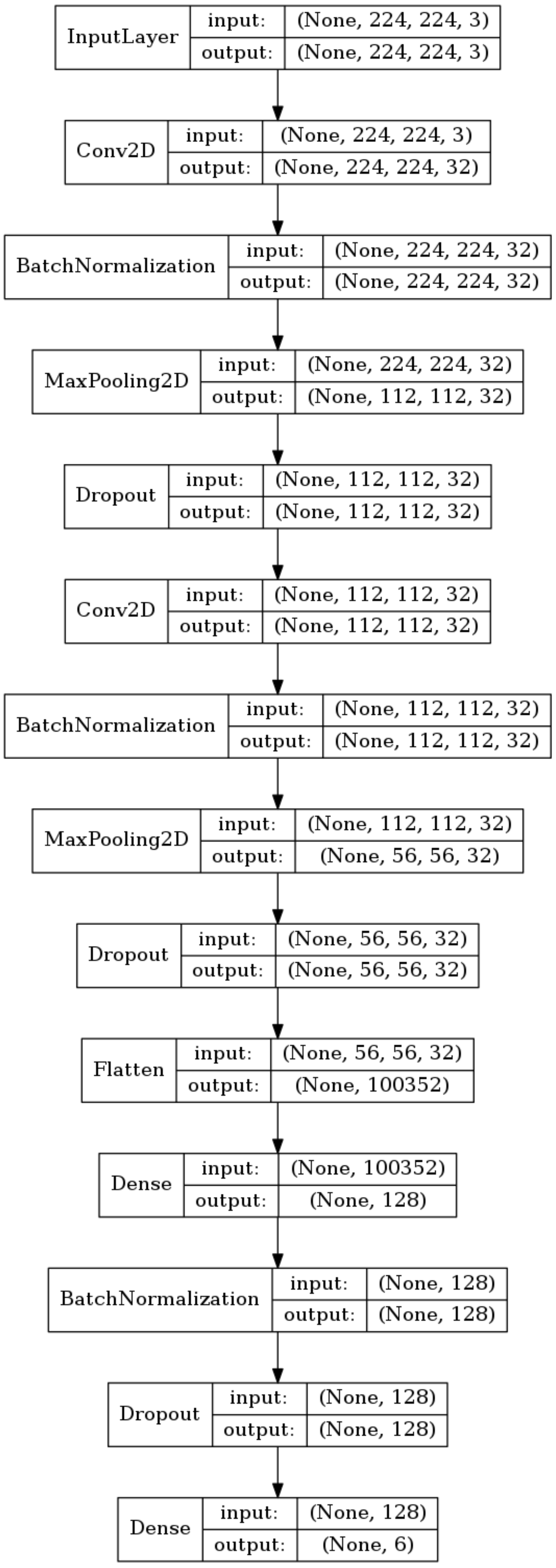
- **Conv2D:** Convolutional layer
- **MaxPooling2D:** Max pooling layer
- **Dense:** Fully connected layer

There are 3 CONV => RELU => POOL blocks, 1 FC layer and 1 softmax classifier at the end.

Dropout
As you can see from the code block, we'll also be utilizing dropout in our network architecture. Dropout works by randomly disconnecting nodes from the current layer to the next layer. This process of random disconnects during training batches helps to naturally introduce redundancy into the model — no one single node in the layer is responsible for predicting a certain class, object, edge, or corner.

Batch normalization
Batch normalization is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network.

Once implemented, batch normalization has the effect of dramatically accelerating the training process of a neural network, and in some cases improves the performance of the model via a modest regularization effect.


InputLayer input: (None, 224, 224, 3) output: (None, 224, 224, 3)
Conv2D input: (None, 224, 224, 3) output: (None, 224, 224, 32)
BatchNormalization input: (None, 224, 224, 32) output: (None, 224, 224, 32)
MaxPooling2D input: (None, 224, 224, 32) output: (None, 112, 112, 32)
Dropout input: (None, 112, 112, 32) output: (None, 112, 112, 32)
Conv2D input: (None, 112, 112, 32) output: (None, 112, 112, 32)
BatchNormalization input: (None, 112, 112, 32) output: (None, 112, 112, 32)
MaxPooling2D input: (None, 112, 112, 32) output: (None, 56, 56, 32)
Dropout input: (None, 56, 56, 32) output: (None, 56, 56, 32)
Flatten input: (None, 56, 56, 32) output: (None, 100352)
Dense input: (None, 100352) output: (None, 128)
BatchNormalization input: (None, 128) output: (None, 128)
Dropout input: (None, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 6)
Fig.9 CNN from scratch structure

Transfer learning via VGG-16

Though the below example is based on VGG-16, we can still apply the same process into ResNet-50.

Step 1:
Adding new FC layers to the topless model of VGG-16

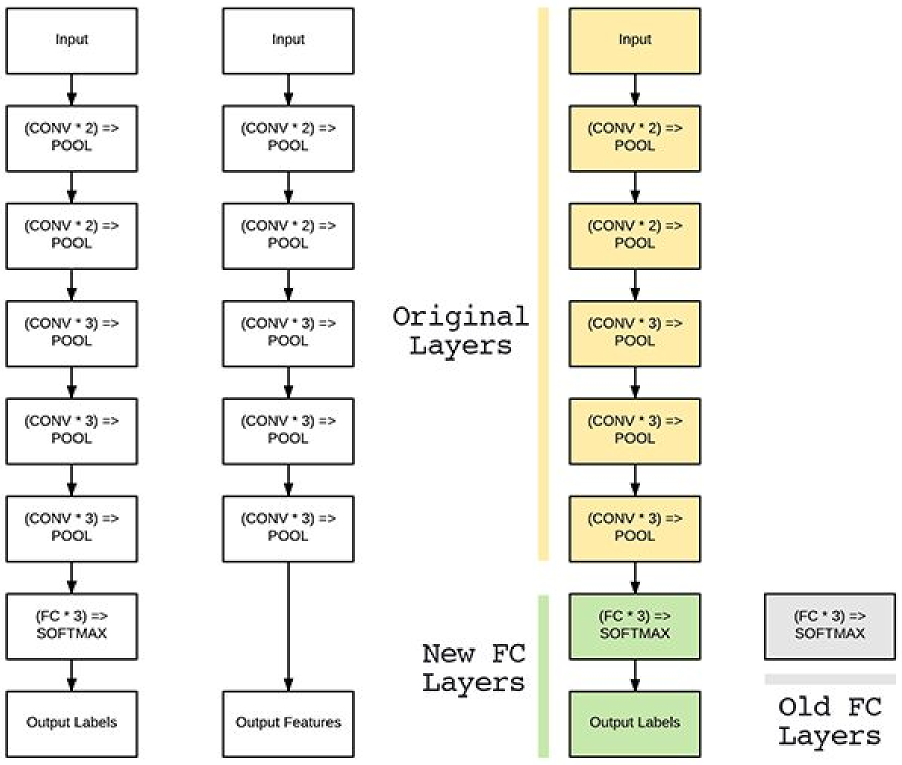


Fig.10 Transfer learning step 1[1]

Left: The original VGG16 network architecture.
Middle: Removing the Fully-connected (FC) layers from VGG16 and treating the final POOL layer as a feature extractor.
Right: Removing the original FC Layers and replacing them with a brand new FC head. These FC layers can then be fine-tuned to a specific dataset (the old FC Layers are no longer used).

Step 2:
Freeze the rest expect FC layer

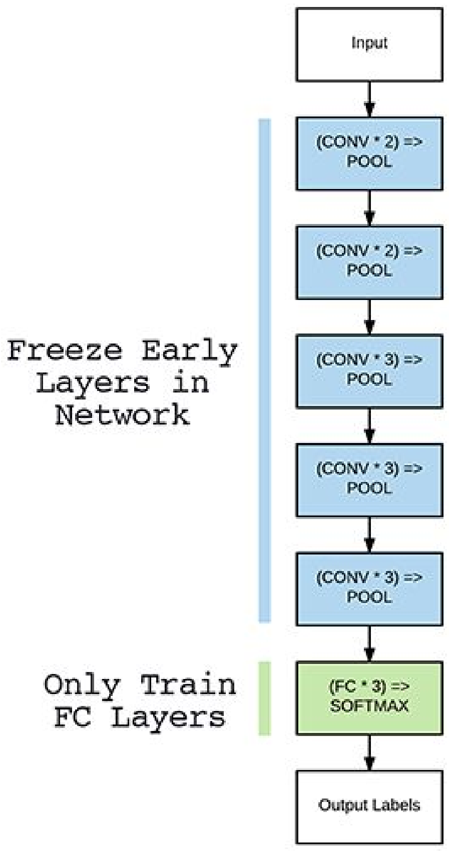


Fig.11 Transfer learning step 2[1]

When we start the transfer learning process, we freeze all CONV layers in the network and only allow the gradient to backpropagate through the FC layers.

Benchmark

K-NN is a lazy learner because it doesn't learn a discriminative function from the training data but "memorizes" the training dataset instead.

For example, an eager learner such as logistic regression algorithm learns its model weights (parameters) during training time. In contrast, there is no training time in K-NN. Although this may sound very convenient, this property doesn't come without a cost: The "prediction" step in K-NN is relatively expensive. Each time we want to make a prediction, K-NN is searching for the nearest neighbor(s) in the entire training set! Due to this characteristic, I will choose this algorithm as our benchmark algorithm.

---

1 Fine-tuning and network surgery
<https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>

Methodology

Data Preprocessing

Histogram Equalization Technique is a technique that preprocesses images to make the CNN model easier to train. Histogram Equalization is the process of taking a low contrast image and increasing the contrast between the image's relative highs and lows in order to bring out subtle differences in shade and create a higher contrast image. The results can be striking, especially for grayscale images. Here are some examples:

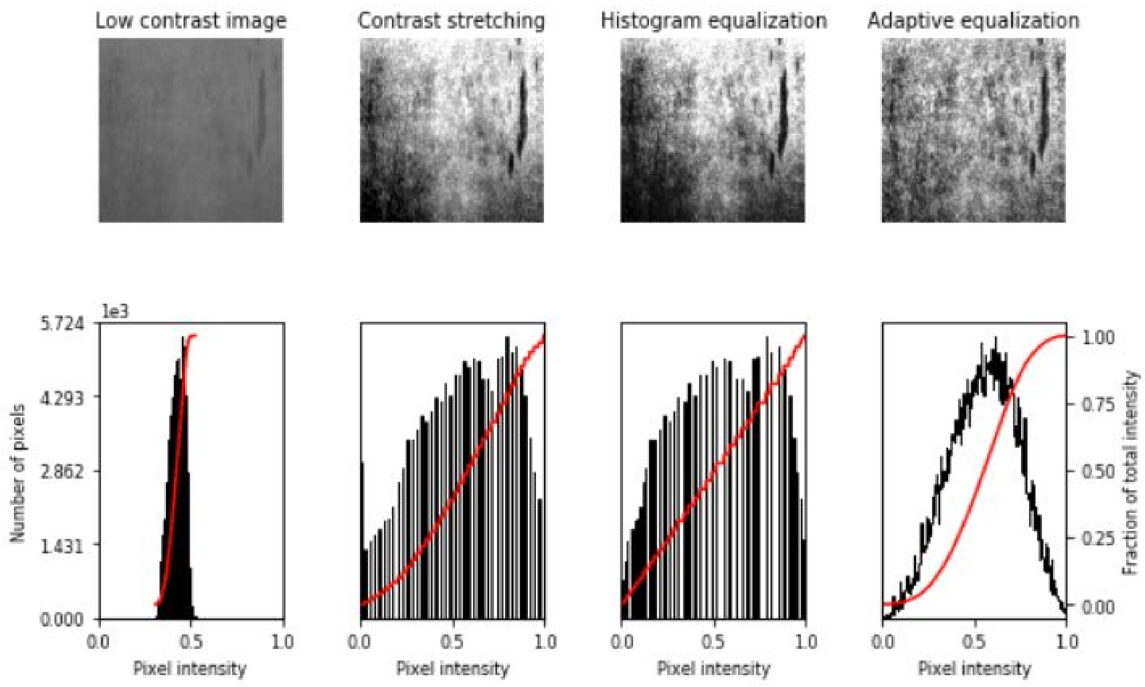


Fig.12 Histogram Equalization

I tried to play around different styles of equalization and found that models are super sensitive to the histogram equalization. It takes lots of time to find the best tuning for models because even a small modification of histogram will cause the performance to be dramatically different. I opted only to slightly preprocess the images by using histogram equalization to save time for the model tuning part later.

We will also use **GLCM** to extract low-features such as contrast, homogeneity, dissimilarity, energy and asymmetry for machine learning algorithm to train.

Features	Formulae
Contrast	Contrast = $\sum_{i,j} \lvert i-j \rvert^2 p(i,j)$
Homogeneity	Homogeneity = $\sum_{i,j} \frac{1}{1+(i-j)^2} p(i,j)$
Dissimilarity	Dissimilarity = $\sum_{i,j} \lvert i-j \rvert p(i,j)$
Energy	Energy = $\sum_{\mathrm{i,j}} \mathrm{p(i,j)}^2$
Asymmetry	Asymmetry= $\sum levels-1 i,j=0 P 2 i,j$

Implementation

The following flow chart shows the process of how to build both machine learning and deep learning models to predict defects.

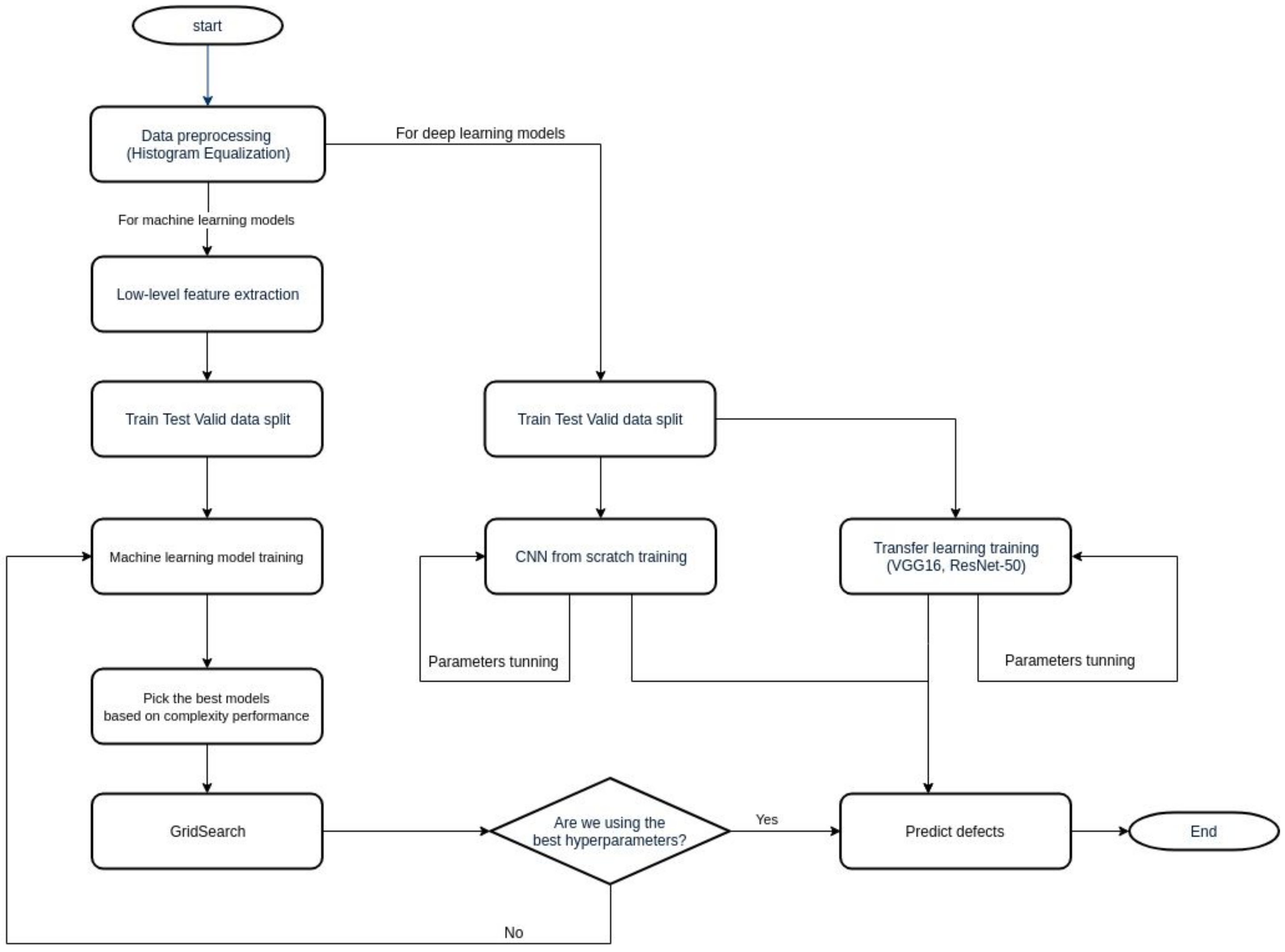

start
Data preprocessing (Histogram Equalization)
For deep learning models
For machine learning models
Low-level feature extraction
Train Test Valid data split
Train Test Valid data split
Machine learning model training
CNN from scratch training
Transfer learning training (VGG16, ResNet-50)
Parameters tunning
Parameters tunning
Pick the best models based on complexity performance
GridSearch
Are we using the best hyperparameters?
Yes
No
Predict defects
End

Fig.13 Flow chart of implementation

The project starts with preprocessing image by using Histogram Equalization, then there are 2 different approaches (machine learning, deep learning) to achieve the same goal - to predict the defect. For low-level features, I used GLCM feature extraction to extract texture features such as contrast, dissimilarity, homogeneity, energy, and asymmetry. These low-level features with the label are then given to test the train split function that is already present in the sci-kit-learn library. The train-test split function splits data and labels. The data is split 80 for training and 20 percent for testing.

Unlike the standard feature extraction algorithms (GLCM, SIFT, HOG), the deep neural networks use several hidden layers to hierarchically learn high-level representation of the image. For instance, the first layer might detect edges in the image, the second layer might detect curves present in the scratch mark, the third layer might detect the whole shape of defects, etc. Then we will split the data - 80 percent for training and 20 percent for testing.

The reason I want to try both machine learning and deep learning algorithm is because I am trying to find the most efficient approaches by experimenting with them. Deep learning models are responsible of learning high-level representations (interpreted by humans) by raw image and machine learning models are responsible of learning low level features (detected by algorithms) for image classification problem.

Machine learning part

The following figures show different machine learning models that have been trained and validated on the training data using different parameters. The graph produces two complexity curves — one for training and one for validation. Similar to the learning curve, the shaded regions of both the complexity curves denote the uncertainty in those curves, and the model is scored on both the training and validation sets using the accuracy function I mentioned at the metrics section.

What I found here is it is worth spending some time to tune the algorithm of decision tree and KNN. Also, the maximum depth 4 results in a model that best generalizes to unseen data because the two lines start separating after the depth of 4. I think that is a good trade off between the score in the test and training sets. If we train our model for a depth of 10, our model will have a high variance, leading to the model overfitting. This is because after the maximum depth is increased to 4, the validation score is gradually decreasing while the training score is increasing.

We can use the same strategy to find the best parameter and to proof the founding here. I used a Grid Search to find the best parameters and the result matches.

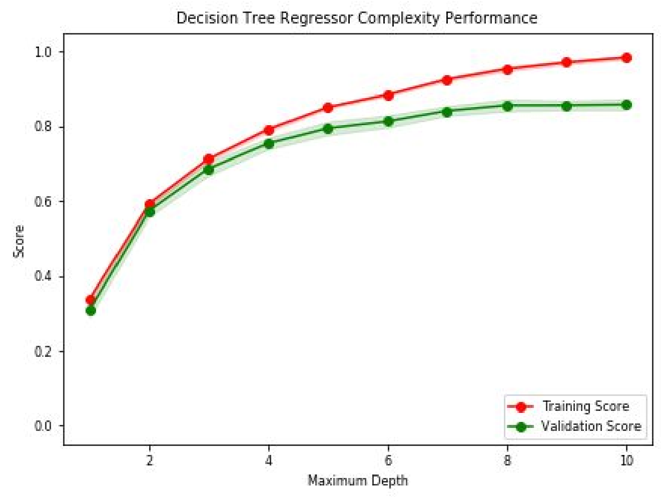


Fig.14 Decision tree complexity performance

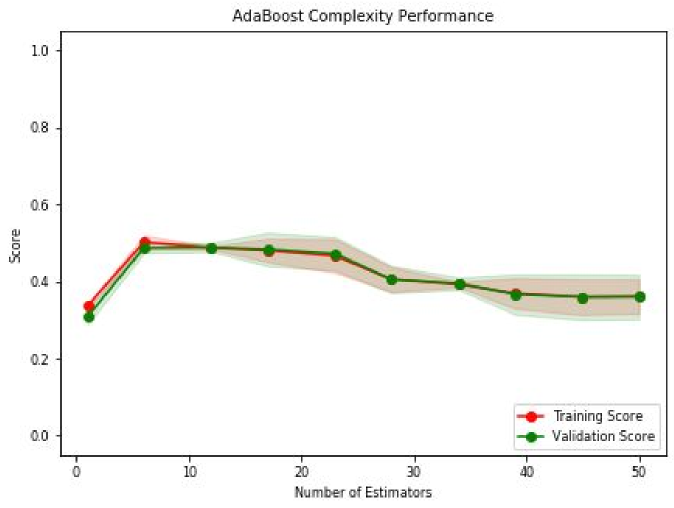


Fig.15 AdaBoost complexity performance

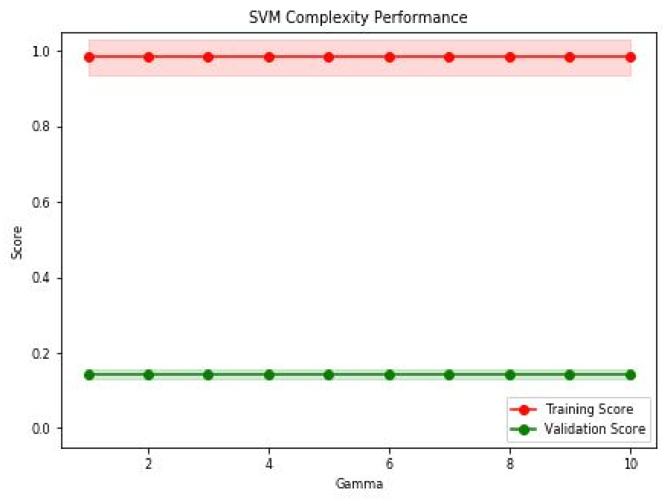


Fig.16 SVM complexity performance

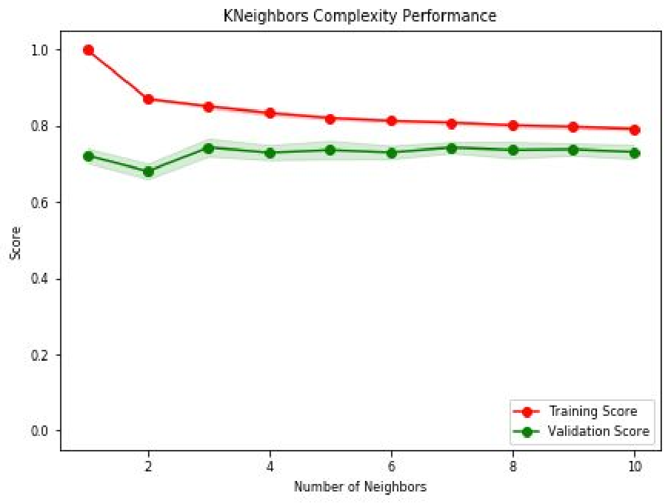


Fig.17 KNN complexity performance

Deep learning part

The below images plot the loss/accuracy curves of each deep learning I used in this project. For example, the transfer learning of VGG16 model shows the model could probably be trained a little more as the trend for accuracy on both datasets is still rising for the last few epochs. We can also see that the model has not yet over-learned the training dataset, showing comparable skill on both datasets.

From the plot of loss, we can see that the model has comparable performance on both train and validation datasets (labeled test). If these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch.

On the other hand, the CNN classifier and transfer learning with ResNet-50 performs poorly during the training time, which means I should tune the parameters on the built-from-scratch CNN more.

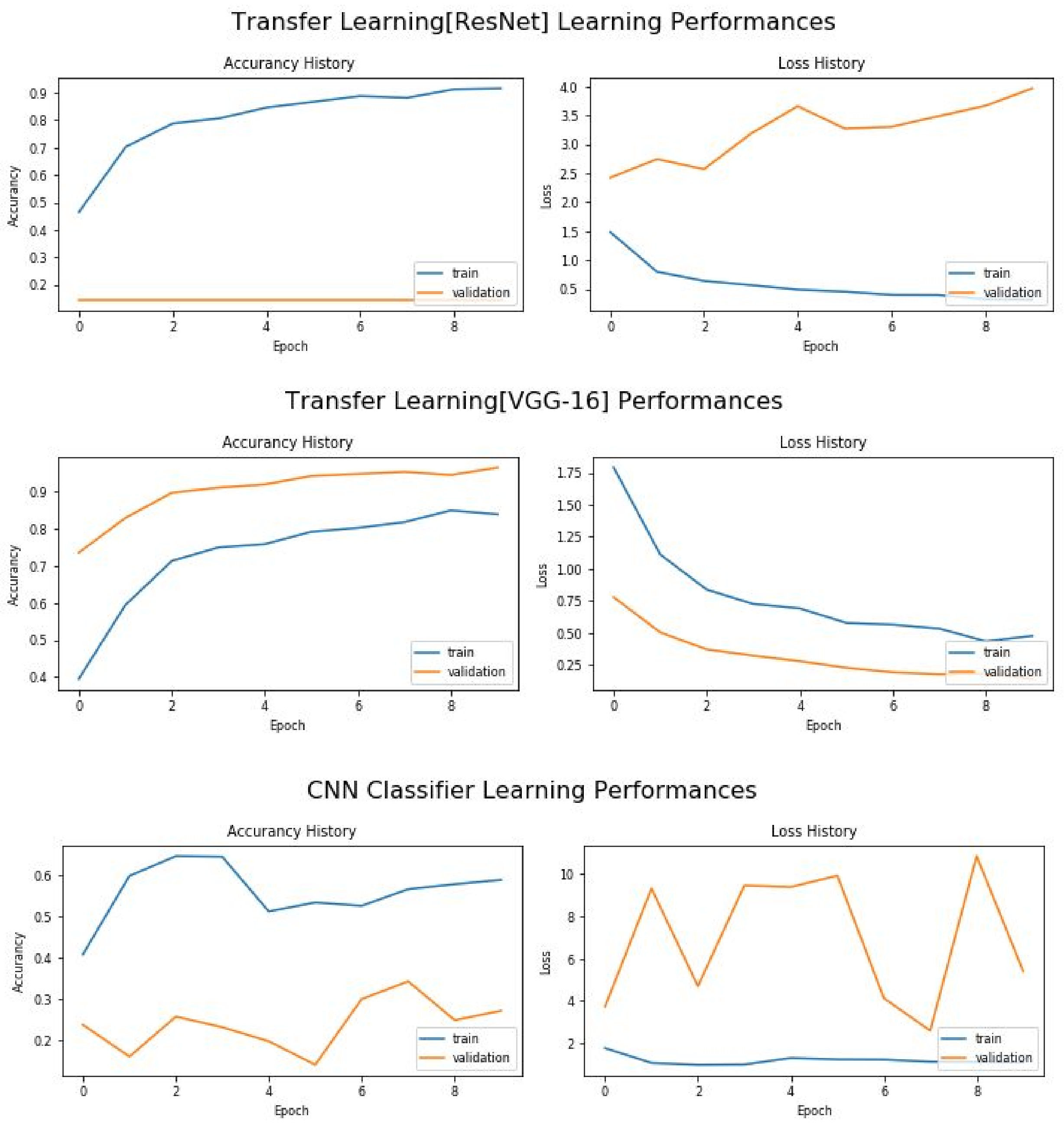


Fig.18 Performance of deep learning classifiers

Refinement

With the help of Grid Search we don't have much work to do on KNN and decision tree this time. Here is the final proof of how those algorithms perform during different training points.

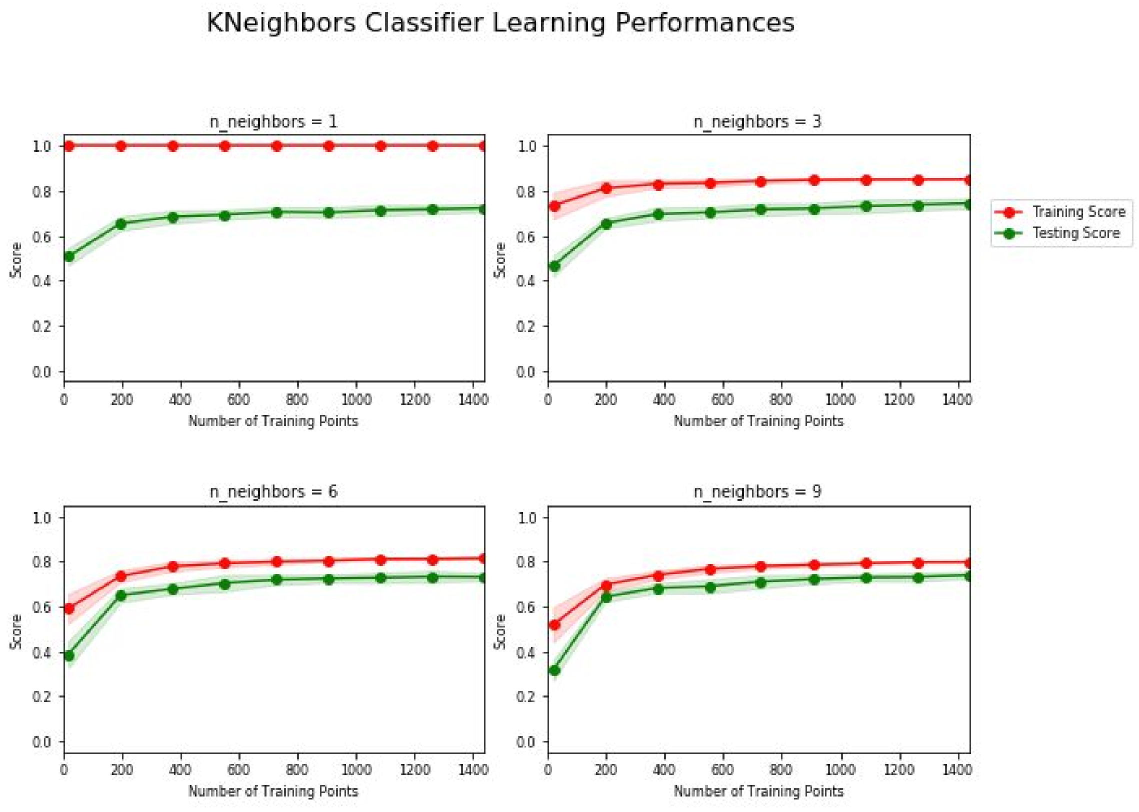


Fig.19 KNN learning performances

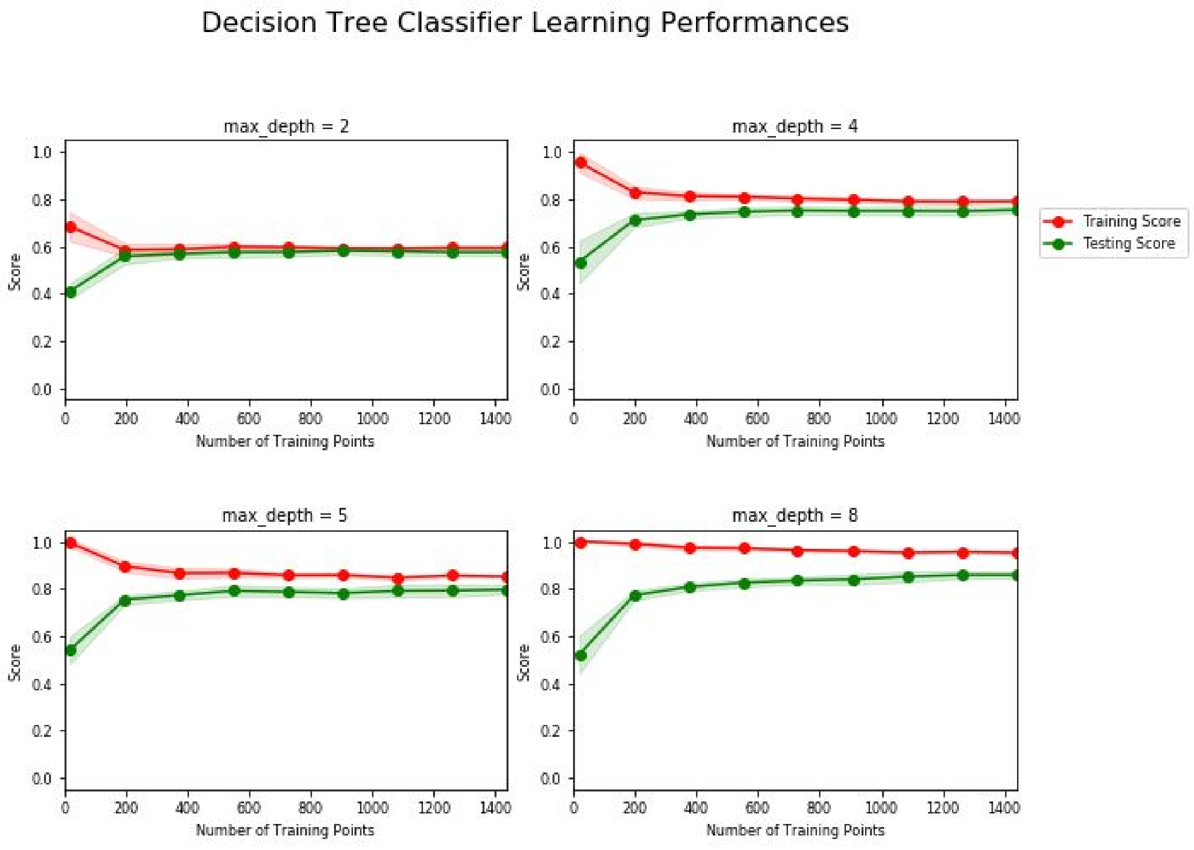
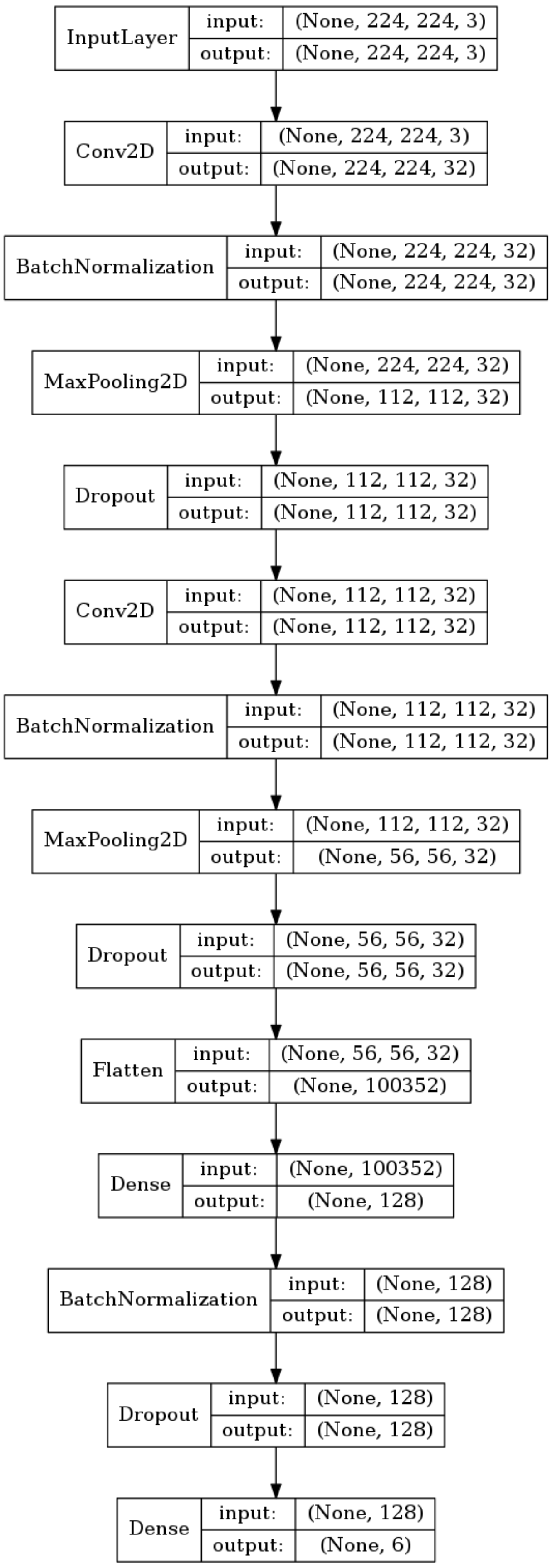


Fig.20 Decision Tree learning performances

The big differences between the version 2 CNN and version 1 CNN architecture is the removed BN and Dropout layer in each CON>ReLu>MaxPooling block and removed BN and dropout layer before the last 2 FC layers.

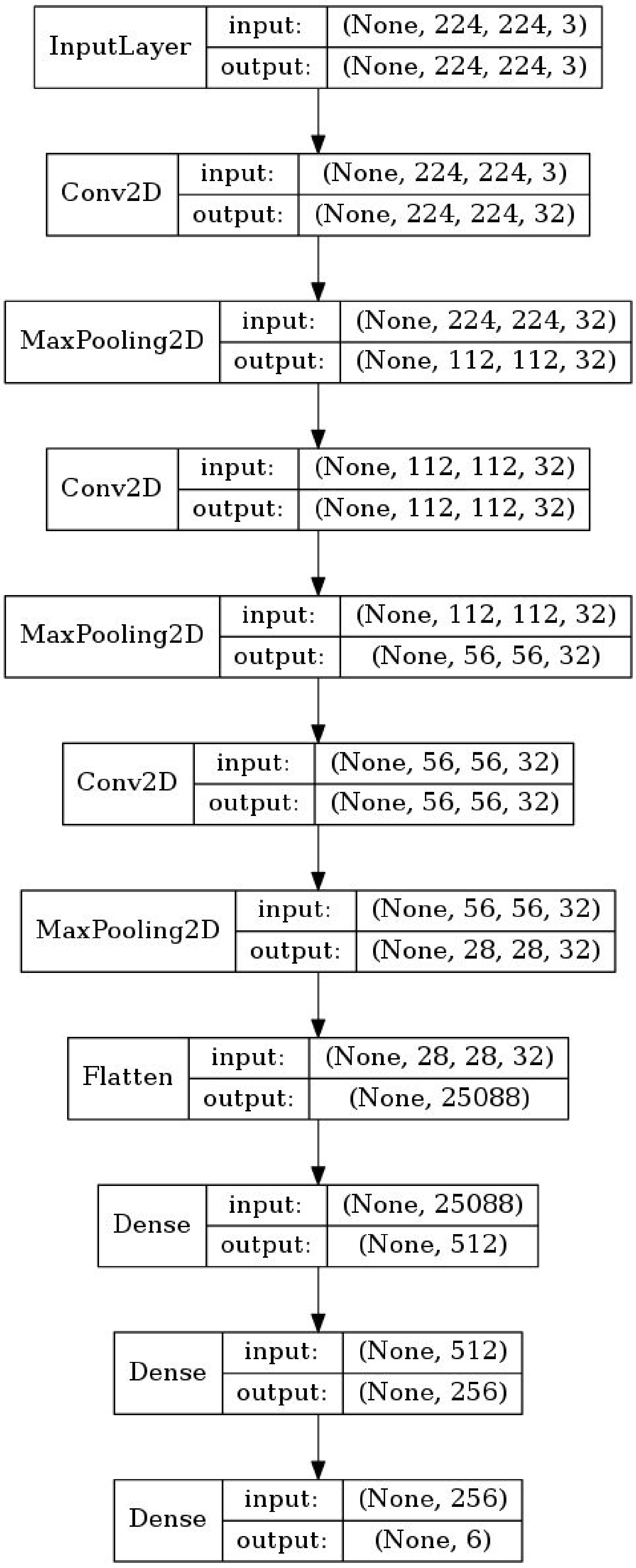
Version 1 - CNN architecture



InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 32)
BatchNormalization | input: (None, 224, 224, 32) | output: (None, 224, 224, 32)
MaxPooling2D | input: (None, 224, 224, 32) | output: (None, 112, 112, 32)
Dropout | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
Conv2D | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
BatchNormalization | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
MaxPooling2D | input: (None, 112, 112, 32) | output: (None, 56, 56, 32)
Dropout | input: (None, 56, 56, 32) | output: (None, 56, 56, 32)
Flatten | input: (None, 56, 56, 32) | output: (None, 100352)
Dense | input: (None, 100352) | output: (None, 128)
BatchNormalization | input: (None, 128) | output: (None, 128)
Dropout | input: (None, 128) | output: (None, 128)
Dense | input: (None, 128) | output: (None, 6)

Fig.9 CNN from scratch structure

Version 2 - CNN architecture



InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 32)
MaxPooling2D | input: (None, 224, 224, 32) | output: (None, 112, 112, 32)
Conv2D | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
MaxPooling2D | input: (None, 112, 112, 32) | output: (None, 56, 56, 32)
Conv2D | input: (None, 56, 56, 32) | output: (None, 56, 56, 32)
MaxPooling2D | input: (None, 56, 56, 32) | output: (None, 28, 28, 32)
Flatten | input: (None, 28, 28, 32) | output: (None, 25088)
Dense | input: (None, 25088) | output: (None, 512)
Dense | input: (None, 512) | output: (None, 256)
Dense | input: (None, 256) | output: (None, 6)

Fig.22 Decision Tree learning performances

After updating the CNN architecture to a simpler version, the learning performance of the version 2 CNN is better than the version 1 CNN.

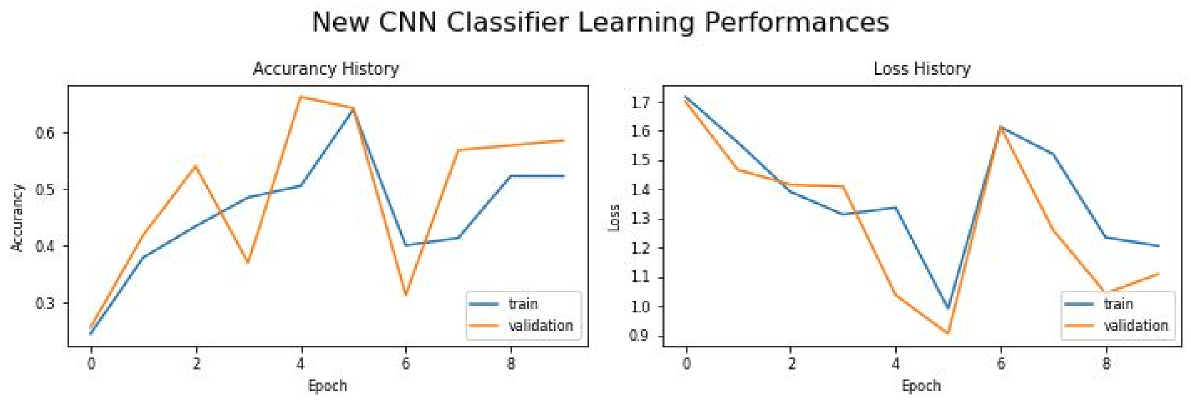


Fig.23 Performance of new CNN classifiers

Result

Model Evaluation and Validation

Accuracy rate:

Algorithms	**Mean accuracy (cross-validation 20%)**	**Accuracy (testing set)**	**Prediction Time (testing set)**
KNN	73.05%	77.3333%	0.8s
AdaBoost with 2 depths decision tree	48.77%	59.5556%	6.4s
Decision Tree Classifier	79.58%	83.5556%	0.4s
SVC	14.30%	14.8889%	4.7s
Version 1 CNN	30.00%	53.3334%	1.6s
Version 2 CNN	70.45%	71.6667%	1.1s
Transfer learning ResNet50	13.92%	15.5556%	5.3s
Transfer learning VGG-16	96.59%	95.5556%	4.2s

Model robustness is also an important aspect to be discussed when working on model evaluation and validation. It can be understood as - If a model has a testing error (on a new test set) equal to the training error, then the model is said to be robust i.e the model generalises well and doesn't overfit.

According to above accuracy table, all the deep learning algorithms are slightly overfitted by around 1-2% and the machine learning algorithms are a bit overfitted by around 1-2%. One improvement is to add noise to an underconstrained neural network model with a small training dataset, this can have a regularising effect and reduce overfitting. Overall, in general I would say the result is acceptable but still have room to improve. [1]

1 How to Improve Deep Learning Model Robustness by Adding Noise - <https://machinelearningmastery.com/how-to-improve-deep-learning-model-robustness-by-adding-noise/>

Normalized confusion matrix:

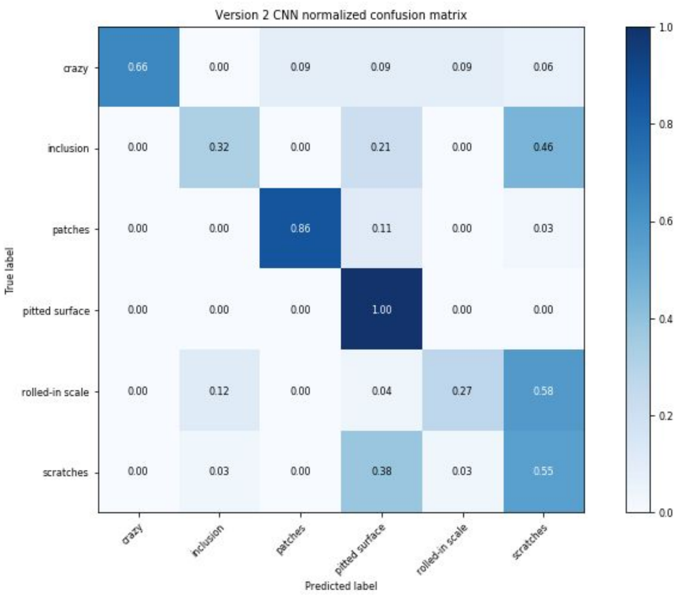


Fig.24 Version 2 CNN normalized confusion matrix

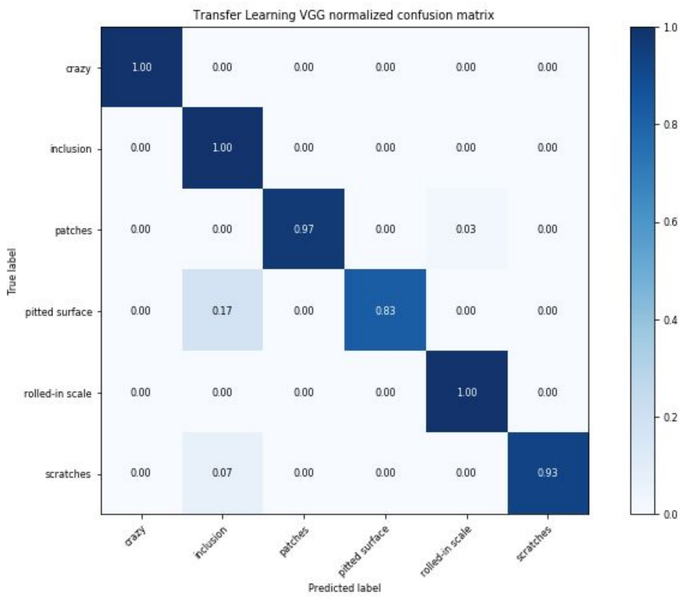


Fig.25 VGG normalized confusion matrix

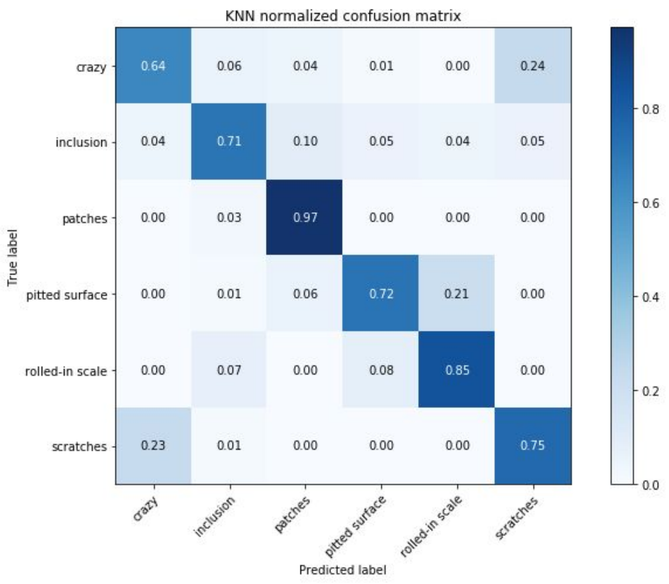


Fig.26 KNN normalized confusion matrix

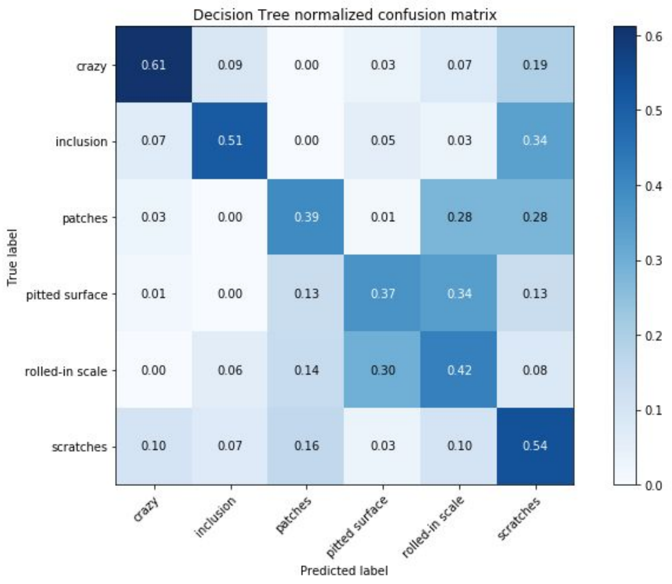


Fig.27 Decision Tree normalized confusion matrix

Justification

The decision tree classifier and VGG-16 transfer learning classifier achieved an accuracy of 83.56 and 95.56 percent, which is more than the KNN benchmark model with 77.33 percent accuracy.

Conclusion

Free-Form Visualization

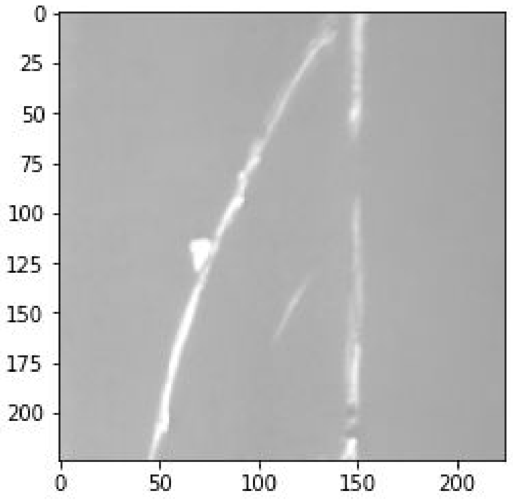


Fig.27 Raw scratch mark image

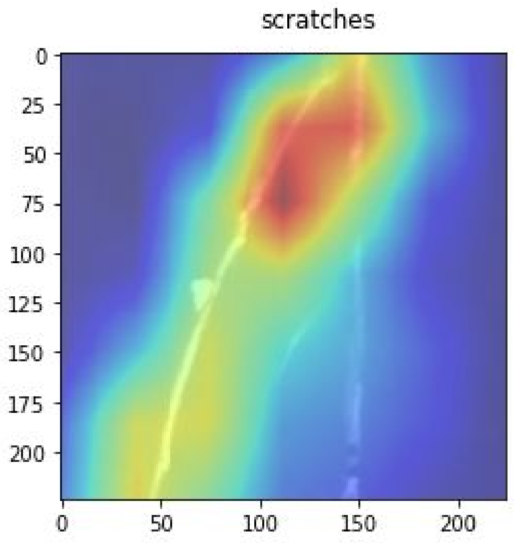


Fig.26 Heatmap scratch mark image

Reflection

The intent of this project is to detect surface defects by using as many training approaches as possible. At first I tried to research if there is any feature I can extract from the photos to make the model's performance better. Then I found there are lots of feature extraction approaches such as PCA, LDA, GW, LBPand Direct GLCM. I thought that is a good opportunity to choose one of these texture analysis algorithms, then pass the feature into different machine learning algorithms and compare them with deep learning algorithms trained by raw images.

There are 2 interesting aspects I want to highlight. Firstly, I didn't expect SVC to perform poorly compared to other machine learning classifiers. Secondly, I thought the accuracy rate must be more than 50% when using transfer learning but unfortunately the result of ResNet-50 only reached 15% accuracy.

With deep learning based computer vision we achieved beyond human level accuracy with the approaches. Our solution is unique — we not only used deep learning and machine learning for classification but also tried to interpret the defect area with heat maps on the image itself.

Human factor cannot be completely dissociated but we can substantially reduce human intervention. We substantially reduced the human review rate. With our case study we proved that we can automate material inspection with deep learning & reduce human review rate.

Improvement

There are 2 aspects that I think may help to improve the performance of this project. Our dataset (NEU Surface Defect Dataset), there are only 300 images per class for training and validation which is smaller than textbook datasets (e.g., MNIST or CIFAR) and popular datasets (e.g., ImageNet or COCO.) We may build a software to let workers in factories to label different kinds of defects to extend our dataset continuously. Also, I can use those ML and DL models which have relatively good performance to detect defects from more complex objects such as electrical components or circuit boards.