Chinmaya Sukumar

Classifying Steel Defects Using Deep Learning

Steel surface defects are a common and longstanding issue in steelmaking. Quality of materials, issues with production and errors caused by workers can cause these steel surface imperfections. The presence and number of defects in rolled steel are a major contributor of a customer's assessment of the quality of rolled steel [1]. Accurately and quickly identifying the type of defects that are present during production would be invaluable to a steel mill in addressing root causes before losses accrue.

Classifying steel surface defects is traditionally a time and resource intensive process requiring a combination of manual inspection and the use of non-destructive tests such as magnetic particle inspection. Recently, advances in imaging technology have automated this process with the use of a high-speed cameras that are placed directly in the production line directed at the steel as it passes through the mill. Conventionally, an individual would be responsible for monitoring the images produced by the camera and identifying then classifying any visible defects. [2]

Manual detection has a few problems. One being the low efficiency of manual image detection and classification compared to the performance of a computer. Secondly, manual techniques are much more prone to error at tasks like this compared to computers. Errors related to incomplete or incorrect detection and classification can lead to major losses for any steel mill. Therefore, automated image defect detection is becoming common place in the production line.

Recently, advances in technology have introduced models that can detect and classify defects, the model built in this project being one of them.

This image classifier deep learning model partly solves the manual detection problem. It has the potential to expedite the process by automatically classifying defects given a flagged image. This is a multistep process. Detecting the presence of a defect in an image is the first step, classifying the image is second, and the last being a detailed analysis of the defect’s attributes such a size, shape, number et cetera that has been explored in other reports [1].

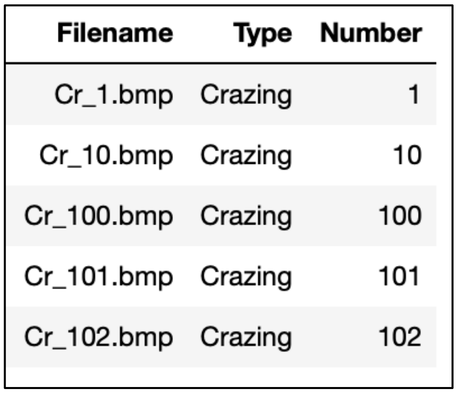
The classifier that has been built would resolve the second step; classification.

Data Wrangling & Cleaning

The dataset was obtained from Kaggle, a data repository, which was extracted from a research paper [2]. The dataset consists of 1800 images in total, with 300 images for each of the following defect classes: Crazing (Cr), Inclusions (In), Scratches (Sc), Rolled in scale (RS), Patches (Pa) and Pitted (PS). A brief explanation of each defect type is as follows [4]:

- Crazing: Formation of microcracks in the surface when exposed to high stress
- Inclusion: Unwanted formation of oxides or other non-metallic particles in the substrate
- Scratches: Abrasions on the surface
- Rolled in scale: Oxidated scale that has been rolled into the steel
- Patches: Visual irregularities in the steel surface
- Pitted: Small areas of corrosion

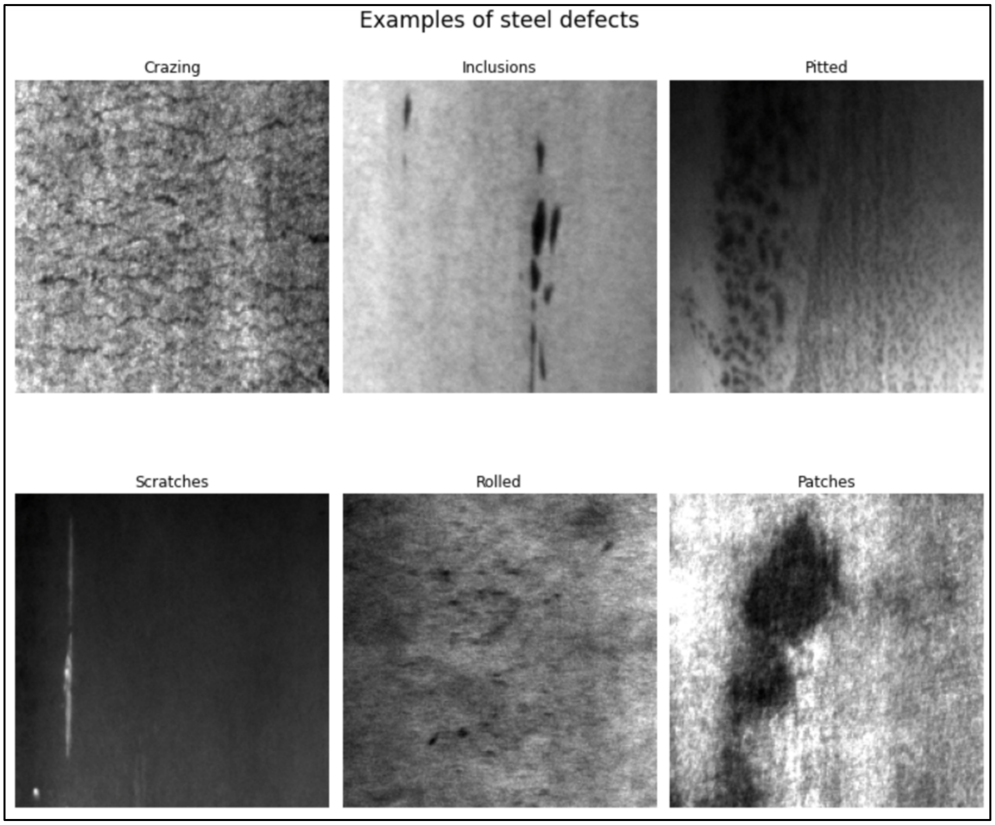
Images of these defects were originally stored as .bmp files in separate folders corresponding to their defect type. Image filenames were extracted from each folder iteratively and consolidated into a DataFrame. The image number as well as the type of defect were added in two separate columns. The data did not contain any null values and required minimal cleaning. The head of the DataFrame is shown below:



Filename	Type	Number
Cr_1.bmp	Crazing	1
Cr_10.bmp	Crazing	10
Cr_100.bmp	Crazing	100
Cr_101.bmp	Crazing	101
Cr_102.bmp	Crazing	102

EDA

The images were all the same size (200x200). Each defect type can be clearly visible A sample of the images is shown in the figure below:

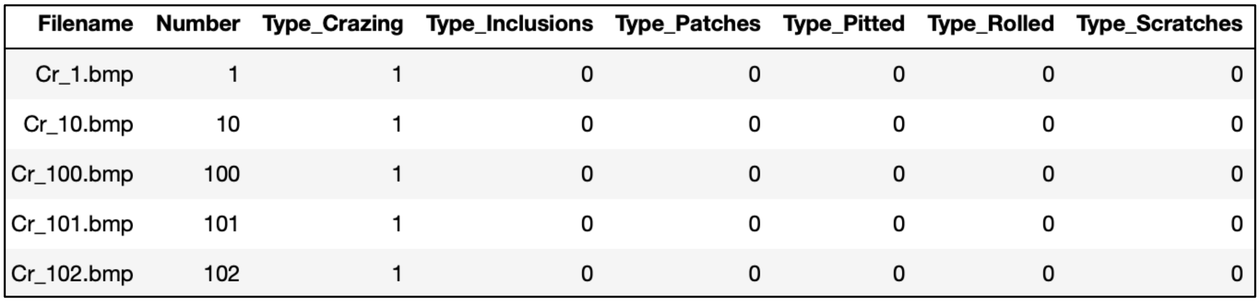

Examples of steel defects

There are some similarities between these images. Inclusion and Scratches are similar but look like negatives of each other. Crazing and Rolled exhibit defects covering the entire area and are of a similar shape and size.

The images and the resulting processed data were too big to be uploaded to GitHub and so were excluded.

Preprocessing

In this stage, the images were opened using the filepath from the DataFrame created above. Images were then stored in a numpy array in the order in which they were retrieved. This array consisted of 1800 grayscale images, each having a 200 x 200 resolution with a shape of (1800, 200, 200) and a grayscale pixel value ranging from 0 to 255. The array was Min-Max Normalized so that the pixel values range between 0 and 1. This manipulation allowed the models built in the next stage to be more efficient and accurate. The defect types, which was the target variable, were one-hot encoded to be suitable for use with Categorical Cross Entropy Loss:



Filename	Number	Type_Crazing	Type_Inclusions	Type_Patches	Type_Pitted	Type_Rolled	Type_Scratches
Cr_1.bmp	1	1	0	0	0	0	0
Cr_10.bmp	10	1	0	0	0	0	0
Cr_100.bmp	100	1	0	0	0	0	0
Cr_101.bmp	101	1	0	0	0	0	0
Cr_102.bmp	102	1	0	0	0	0	0

The images served as the independent variable and the one-hot encoded defect type portion served as the target variable.

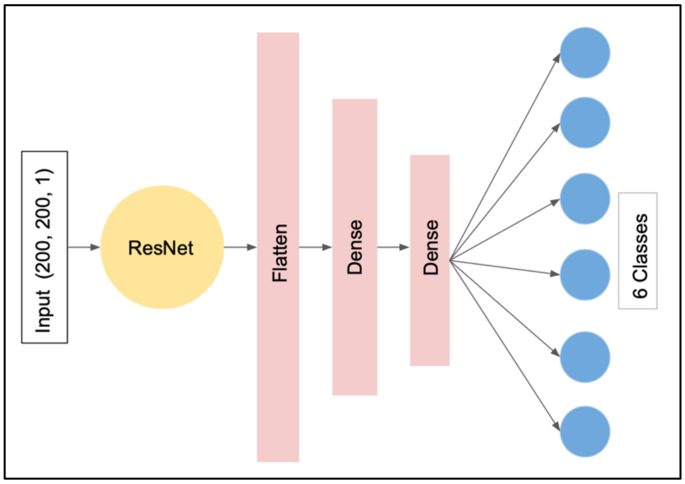
Model Explanation

A Dummy Classifier was first built to serve as a benchmark to evaluate the real models. The model received an F1 macro-average score of 0.14.

Resnet

The first model investigated was a ResNet-50 model. The model was pretrained on the ImageNet dataset which is an expansive library of labeled images. The weights from this training were loaded onto the ResNet model instantiated in this section. Since ResNet only accepts RGB images, the grayscale images were transformed from (200, 200, 1) to (200, 200, 3) with the extra 2 dimensions mimicking RGB channels. The top layer (final layer) of this model was used specifically for the ImageNet dataset which had 1000 classes and thus had 1000 neurons in the final layer. This layer was not included in the model and was replaced with a layer of size 6 corresponding to the 6 classes in this dataset.

Initially, a GlobalAveragePooling2D (GAP) layer was directly connected to the output layer from the ResNet model. The GAP layer takes the average of each feature map in the previous Convolutional layer reducing its size to a one-dimensional array. GAP excels at identifying features - defects in this case - no matter their location. The GAP layer was then connected to a Dense layer to downsample, and then to a final layer with 6 neurons. The results using this architecture were disappointing and will be explained in the subsequent section. The GAP layer was then replaced with a Flatten layer in an effort to improve performance. The reason being that the model was definitely underfitting and a Flatten layer would preserve much of the information from the output of the ResNet model since there is no averaging which would remedy this issue. To illustrate this point, the output shape from the GAP layer was (,2048), while the output from the Flatten layer was (,100352). A representation of the model is shown below:

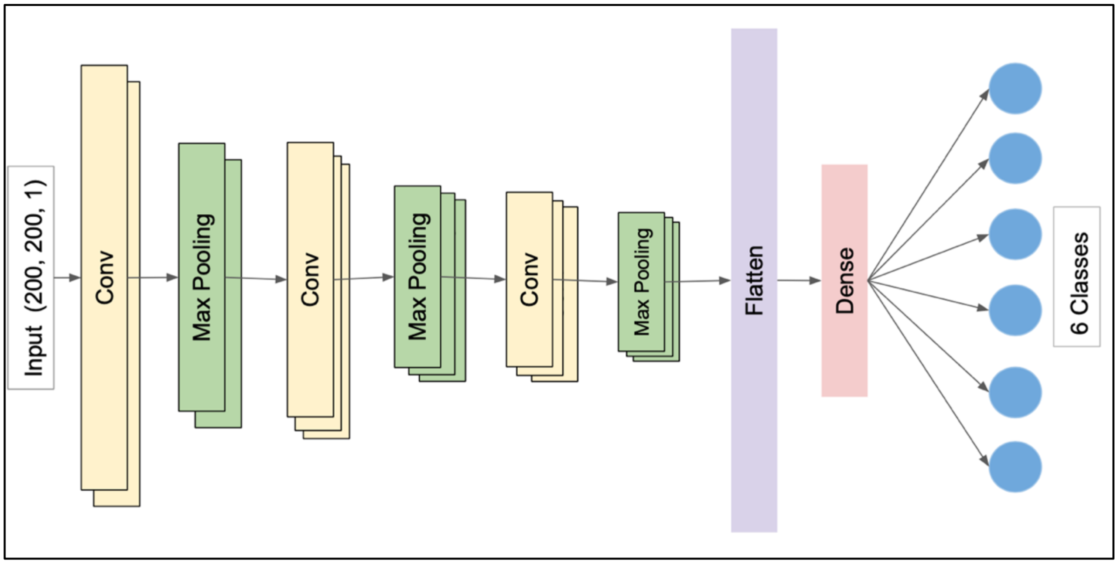


The size of each layer in these images is only representative and does not reflect their actual size.

CNN

A CNN model was also investigated. These models perform superbly in image classification because they excel at detecting image attributes such as lines, colour, shapes and high-level features further on in the network such as data passes through subsequent layers.

A custom CNN was built in this project. The architecture is shown below:



Once again, the layer sizes are relative, and the “3D” layers strictly represent the higher dimensions of the Convolutional and Max Pooling layers.

The filter and kernel sizes chosen of the Convolution and pool size chosen of the Max Pooling layers were a good starting point for any image classifier. The model was ordered in this way so that each successive Convolutional layer would capture patterns that were higher in complexity. Max Pooling was added between layers to reduce dimensionality so that the final Dense layer would not have to deal with too much complexity when classifying. A Flatten layer was used to reduce the outputs from Max Pooling to a 1D array so that the Dense layer that followed could feed into the output 6 neuron Dense layer. Flatten was used instead of GAP due to the verified increase in ResNet score. The activation functions of all layers were ReLU with the last Dense layer being softmax which is typical for multi-class classification.

Modelling

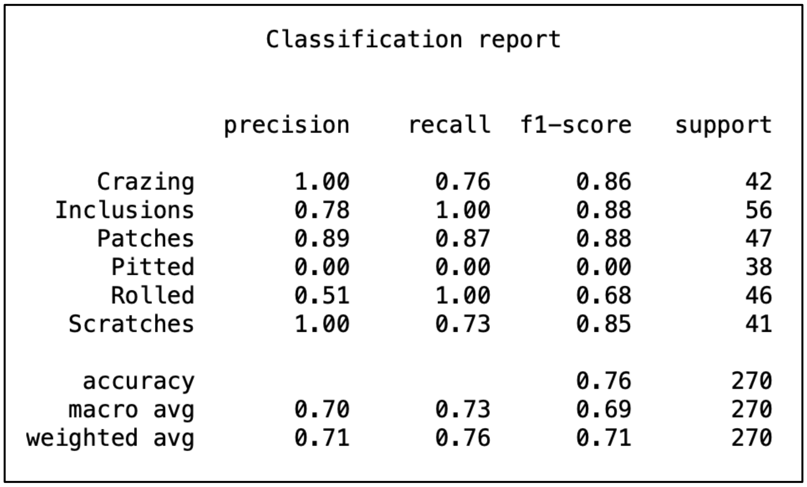
Model selection, including hyperparameter tuning were completed in this stage. A Dummy Classifier was first fit to the model to act as a benchmark for the other models. It’s macro-average F1 score on the validation set was 0.17.

The macro-average F! score was chosen as the evaluation metric since accurate prediction is equally important for all defect classes.

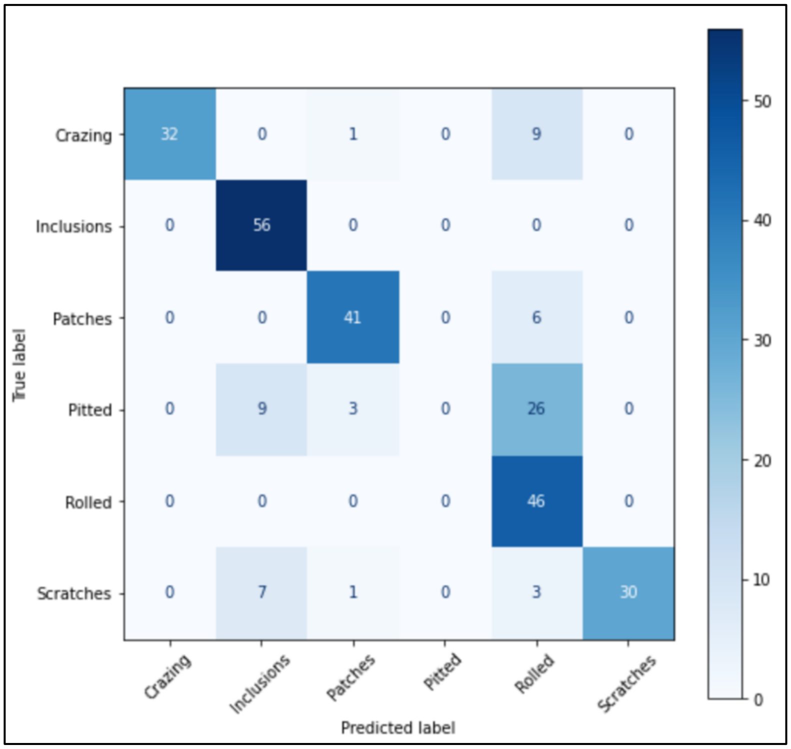
ResNet

The ResNet model that included the GAP layer was trained and its maximum validation score was 0.4148. This permutation of the model was put on hold while the Flatten version of this model was trained and evaluated.

The Flatten version of the model was evaluated on the validation set and its confusion matrix and classification report can be seen below:



```
                    Classification report

              precision    recall  f1-score   support

     Crazing       1.00      0.76      0.86        42
  Inclusions       0.78      1.00      0.88        56
     Patches       0.89      0.87      0.88        47
      Pitted       0.00      0.00      0.00        38
      Rolled       0.51      1.00      0.68        46
   Scratches       1.00      0.73      0.85        41

    accuracy                           0.76       270
   macro avg       0.70      0.73      0.69       270
weighted avg       0.71      0.76      0.71       270
```

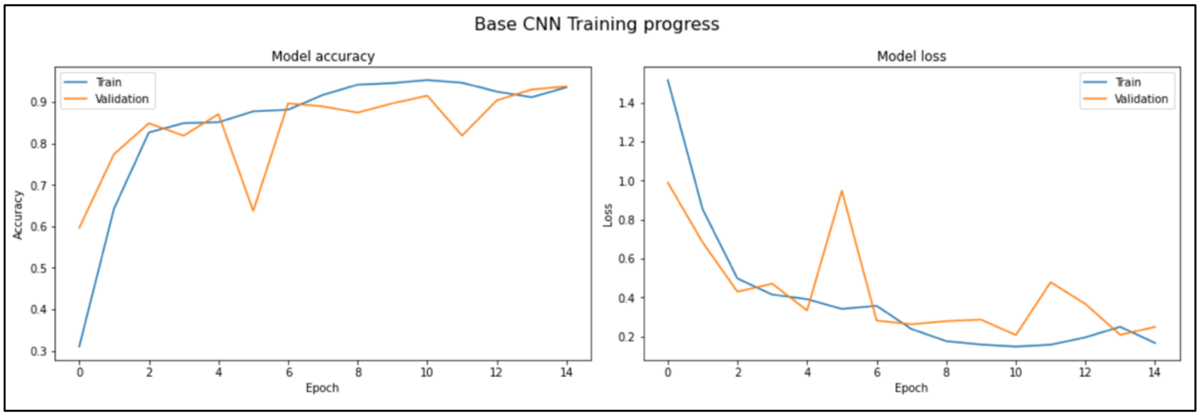


Macro average was 0.73 which is quite good for a untuned model. Precision and recall in most classes were adequate, but ResNet was unable to classify the “Pitted” class at all. It appears the model was confusing “Pitted” with “Rolled”. This can be attributed to the fact that both defect types look similar having small features distributed evenly in both images. The model occasionally misclassified “Scratches” and “Pits” as “Inclusions” which is also understandable since these defects all have a linear form.

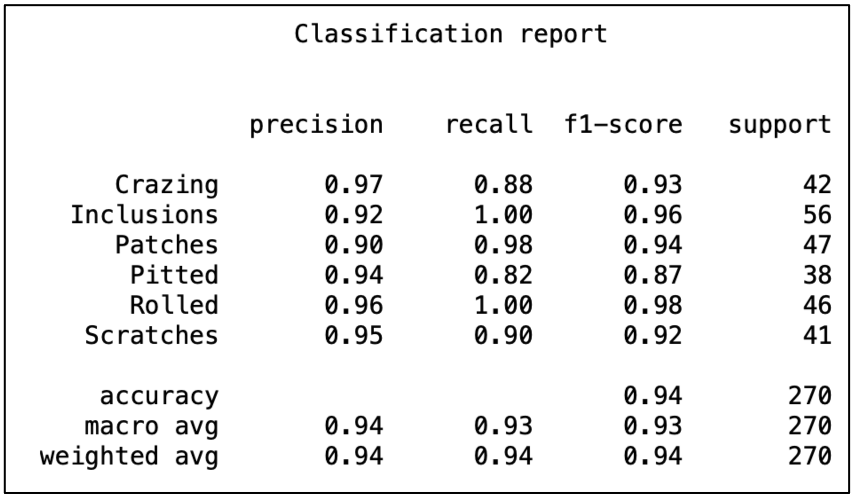
It is important to note that all sets were unfortunately lost midway through the project and had to be re-split. Despite setting the necessary seeds to ensure reproducibility, the original sets could not be reproduced, and so new sets had to be created for the models below.

CNN

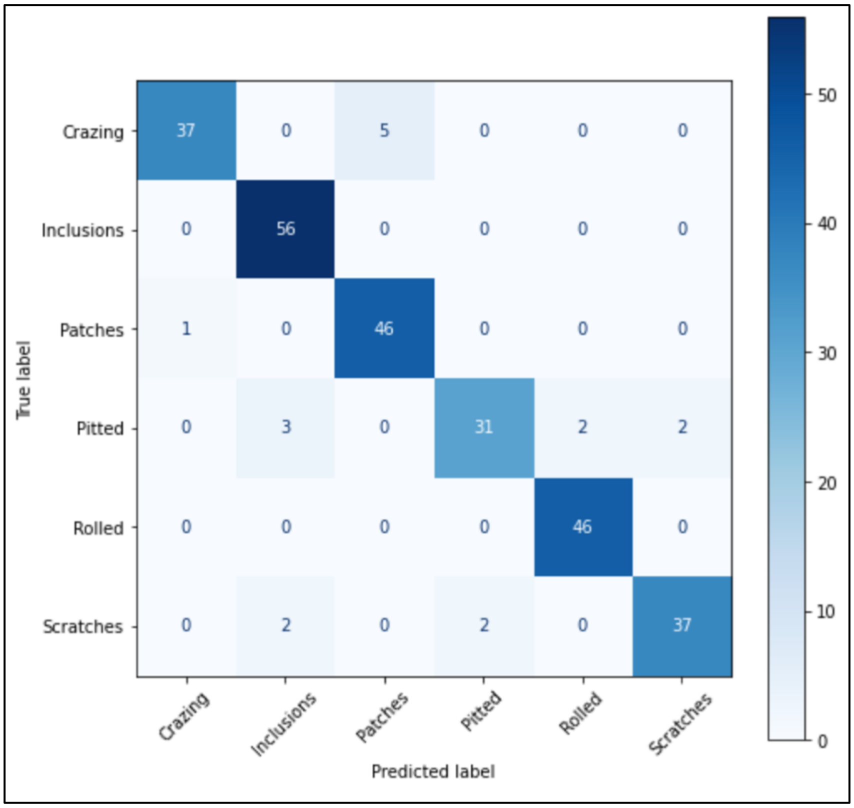
The second model to be evaluated was the custom CNN model with the default optimizer and learning rate (Adam=0.001). As mentioned above, Categorical Cross Entropy Loss was chosen. The learning curves are show below. The model’s best weights out of the 14 epochs were saved using a ModelCheckpoint callback.



The model didn’t overfit the validation set at all and obtains quite a large score and minimal loss during training. The confusion matrix, classification report and learning curves from evaluating on the test set are shown.



```
                    Classification report


              precision    recall  f1-score   support

     Crazing       0.97      0.88      0.93        42
  Inclusions       0.92      1.00      0.96        56
     Patches       0.90      0.98      0.94        47
      Pitted       0.94      0.82      0.87        38
      Rolled       0.96      1.00      0.98        46
   Scratches       0.95      0.90      0.92        41

    accuracy                           0.94       270
   macro avg       0.94      0.93      0.93       270
weighted avg       0.94      0.94      0.94       270
```



	Crazing	Inclusions	Patches	Pitted	Rolled	Scratches
Crazing	37	0	5	0	0	0
Inclusions	0	56	0	0	0	0
Patches	1	0	46	0	0	0
Pitted	0	3	0	31	2	2
Rolled	0	0	0	0	46	0
Scratches	0	2	0	2	0	37

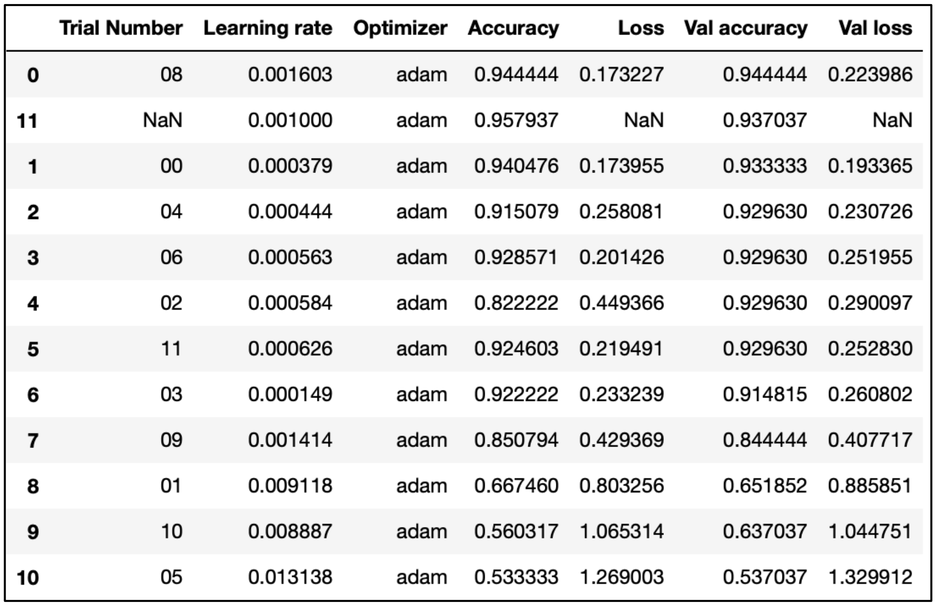
The evaluation metrics from this model are excellent. The model mostly overcame the issue that plagued ResNet which was the misclassification of the “Pitted” class as “Rolled”. Since the model performed well, only one parameter, the learning rate, was chosen as a hyperparameter to tune the model.

CNN - Hyperparameter Tuning

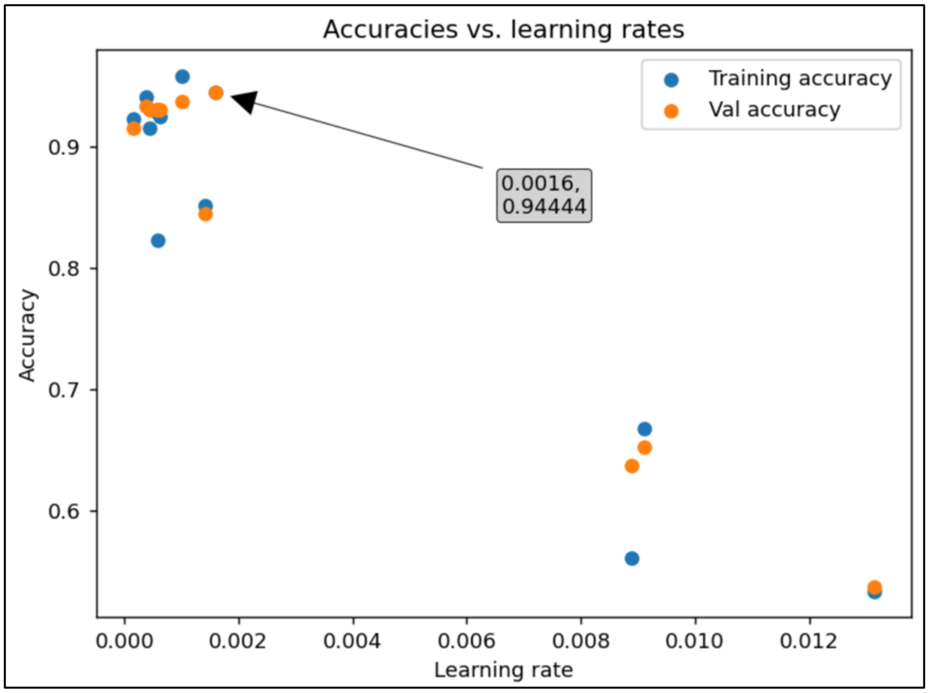
A Randomized Search was performed with a chosen learning rate between 0.0001 and 0.0151. This large range was chosen because a subsequent Grid Search was to be done to fine-tune the learning rate further. However, it proved unnecessary to perform a Grid Search due to the excellent performance obtained from the Random Search.

A Tuner was created that ran a maximum of 12 trials. ModelCheckpoint was unfortunately incompatible with the tuner, so an EarlyStopping callback was used to restore the weights of the best model per trial. The evaluation metrics and learning rates from 11 trials were combined into a DataFrame, and the test score of the 5 best models was added in a separate column. The table below displays all 11 models sorted by test score.

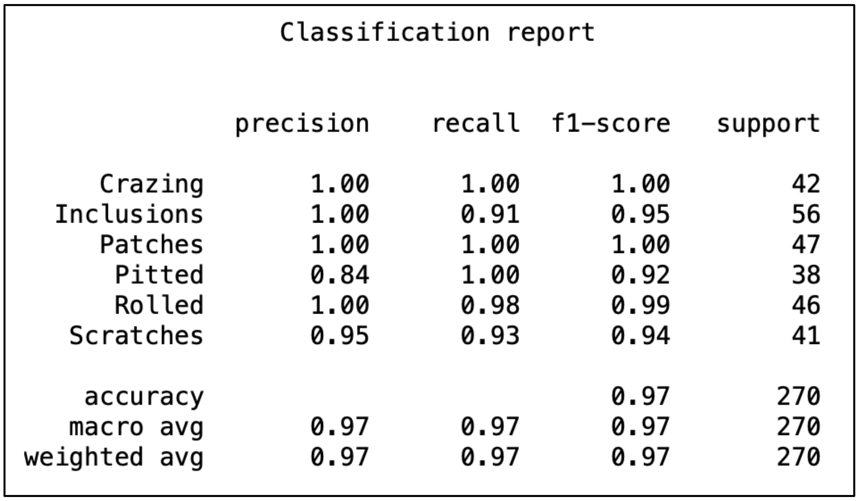
The training and validation set accuracies were plotted:



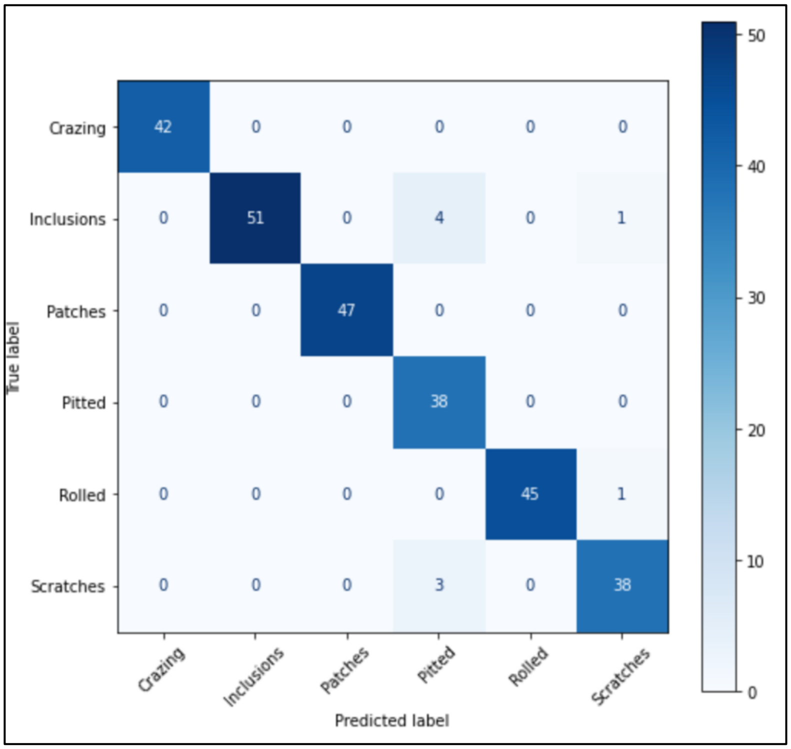
	Trial Number	Learning rate	Optimizer	Accuracy	Loss	Val accuracy	Val loss
0	08	0.001603	adam	0.944444	0.173227	0.944444	0.223986
11	NaN	0.001000	adam	0.957937	NaN	0.937037	NaN
1	00	0.000379	adam	0.940476	0.173955	0.933333	0.193365
2	04	0.000444	adam	0.915079	0.258081	0.929630	0.230726
3	06	0.000563	adam	0.928571	0.201426	0.929630	0.251955
4	02	0.000584	adam	0.822222	0.449366	0.929630	0.290097
5	11	0.000626	adam	0.924603	0.219491	0.929630	0.252830
6	03	0.000149	adam	0.922222	0.233239	0.914815	0.260802
7	09	0.001414	adam	0.850794	0.429369	0.844444	0.407717
8	01	0.009118	adam	0.667460	0.803256	0.651852	0.885851
9	10	0.008887	adam	0.560317	1.065314	0.637037	1.044751
10	05	0.013138	adam	0.533333	1.269003	0.537037	1.329912



As it can be seen, the ideal learning rate is 0.0016. The training score rapidly diminishes when higher rates are used therefore it seems as though there is a local maximum around the 0.0016 range. The CNN model was evaluated using the validation set. The resulting confusion matrix and classification report are shown below:

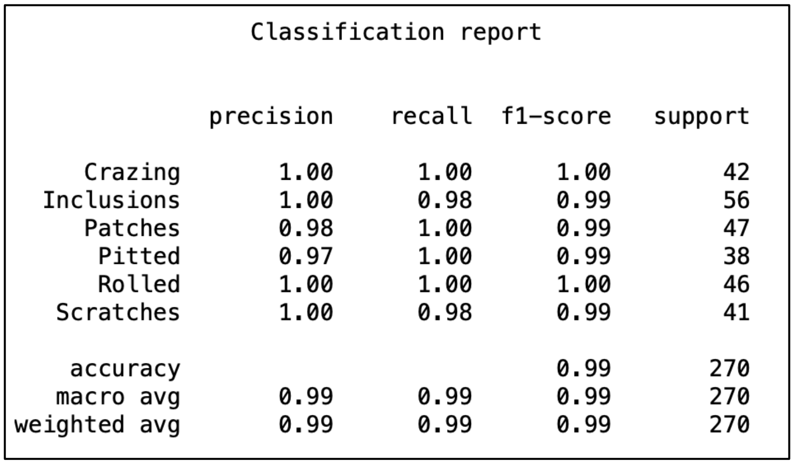


```
                  Classification report


              precision    recall  f1-score   support

     Crazing       1.00      1.00      1.00        42
  Inclusions       1.00      0.91      0.95        56
     Patches       1.00      1.00      1.00        47
      Pitted       0.84      1.00      0.92        38
      Rolled       1.00      0.98      0.99        46
   Scratches       0.95      0.93      0.94        41

    accuracy                           0.97       270
   macro avg       0.97      0.97      0.97       270
weighted avg       0.97      0.97      0.97       270
```

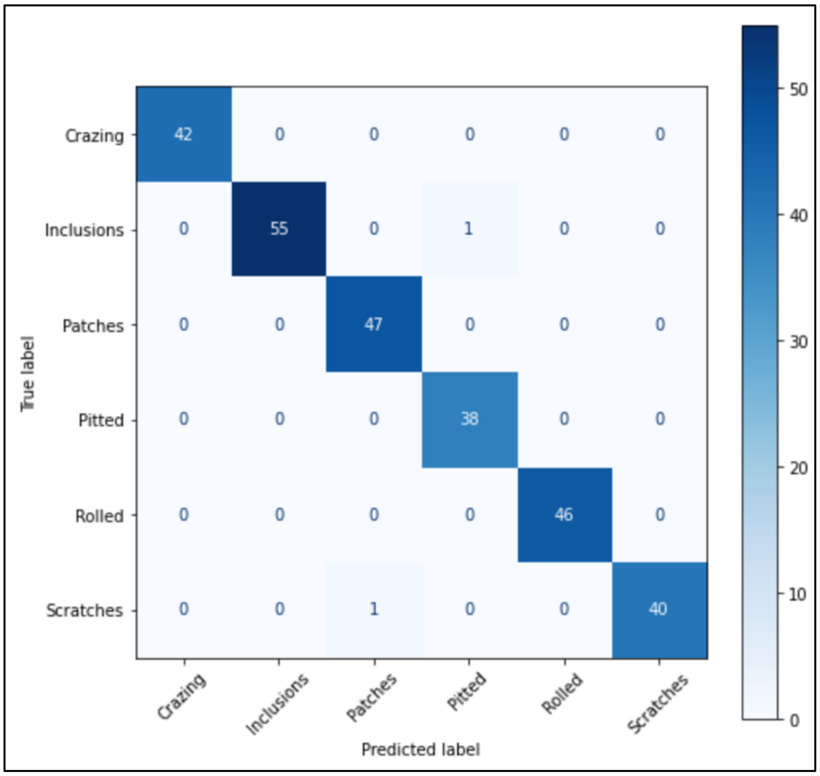


CNN – Image Normalization

The CNN using an Adam optimizer and a learning rate of 0.0016 was chosen to be the final model. However, the images that were used until this point were Min-Max Normalized but not z-score Standardized. Their values ranged from (0,1) but their mean and standard deviation were not 0 and 1 respectively. CNNs perform better when the data is z-score Standardized and so the feature sets standardized and the model was trained using the standardized training set. The results on the standardized validation set are as follows:

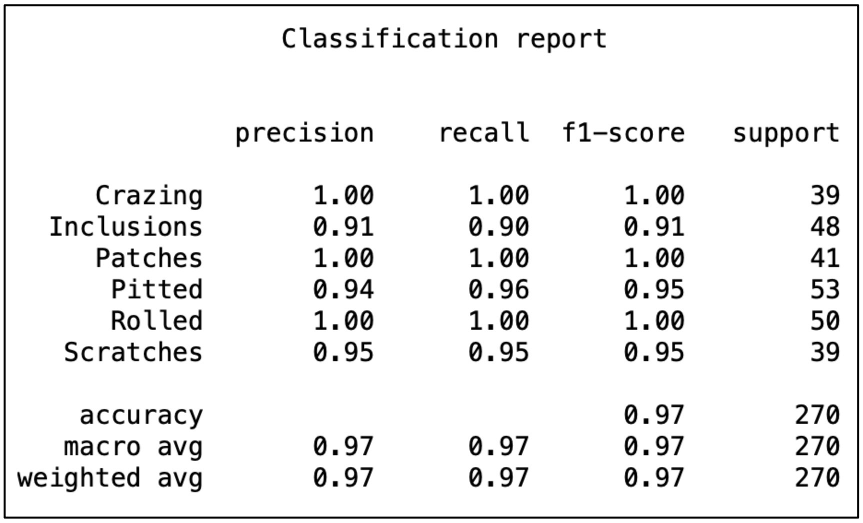


```
                    Classification report

              precision    recall  f1-score   support

     Crazing       1.00      1.00      1.00        42
  Inclusions       1.00      0.98      0.99        56
     Patches       0.98      1.00      0.99        47
      Pitted       0.97      1.00      0.99        38
      Rolled       1.00      1.00      1.00        46
   Scratches       1.00      0.98      0.99        41

    accuracy                           0.99       270
   macro avg       0.99      0.99      0.99       270
weighted avg       0.99      0.99      0.99       270
```

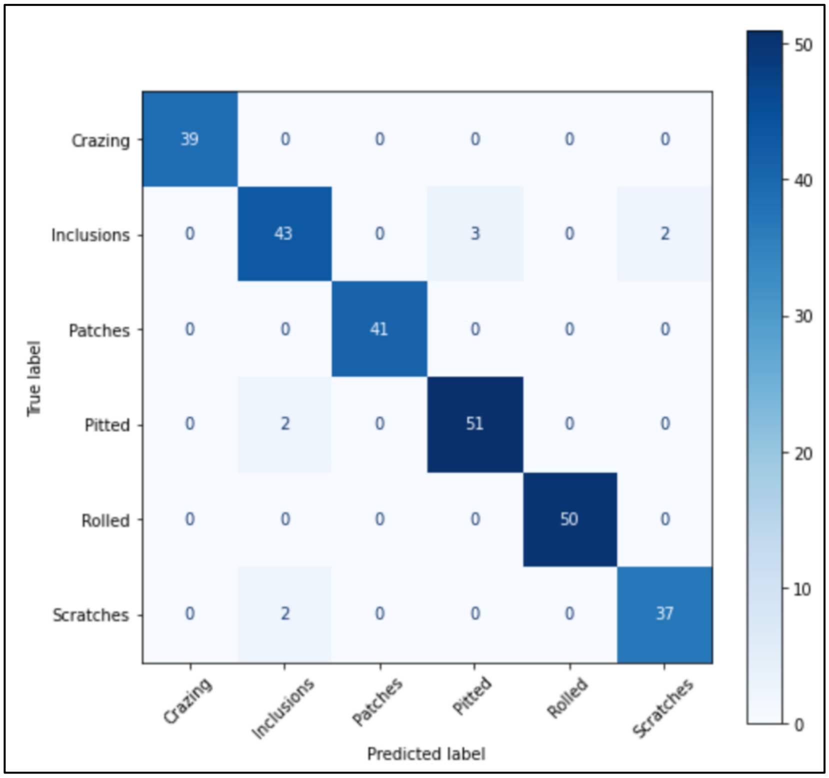


This process resulted only in marginal improvement in its score but was an improvement none the less. This model and the others were saved in the “/models” section of the project.

These are the results on the test set using a standardized X set:



```
                     Classification report


               precision    recall  f1-score   support

     Crazing       1.00      1.00      1.00        39
  Inclusions       0.91      0.90      0.91        48
     Patches       1.00      1.00      1.00        41
      Pitted       0.94      0.96      0.95        53
      Rolled       1.00      1.00      1.00        50
   Scratches       0.95      0.95      0.95        39

    accuracy                           0.97       270
   macro avg       0.97      0.97      0.97       270
weighted avg       0.97      0.97      0.97       270
```



Conclusion

The CNN multi-class classification model presented can be an invaluable tool during the steelmaking process. The model could potentially save time, resources and decrease errors during manual classification. The CNN trained on the standardized data has the best predictive power having an macro-average F1 score of 0.97. A drawback of using the CNN trained on standardized images is that the standardization process could be computationally expensive. The difference in validation score between using CNN-standardized and CNN-unstandardized is 2% however, it would be appropriate to maximize score since incompletely or inaccurately classifying a defect could be quite expensive.

Therefore, the CNN (Adam, learning_rate = 0.0016) trained on a standardized dataset was chosen to be the final model.

It should be noted that though this model is overall excellent at classification, its utility of this model depends heavily on the type of defect that occurs most in production. For example, the precision and recall for “Inclusions” and “Pitted” is good, but is still lower than that of the other defects. A mill that encounters more “Inclusions” and/or “Pitted” defects than most might find this model to be less useful. Another limitation is the fact that there is no “Other” class for defects that do not fit any of the 6 defect classes. This issue could be resolved easily with a quick change to the model.

The CNN model will save time, money and manpower when applied to the automated camera system in a rolling mill. To implement this model, an engineer would have to build a pipeline to capture, access, evaluate and store the images and their predictions. As mentioned earlier, a model would have to be built that can further analyze these defects for other features such as size, shape etc.

Sources

[1] Xu, Yiming, et al. "The Steel Surface Multiple Defect Detection and Size Measurement System Based on Improved Yolov5." Journal of Electrical and Computer Engineering, vol. 2023, 2023, pp. 1–16, https://doi.org/10.1155/2023/5399616.

[2] Song, K., & Yan, Y. (2013). A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. Applied Surface Science, 285, 858-864[1]

[3] Guan, Shengqi & Lei, Ming & Lu, Hao. (2020). A Steel Surface Defect Recognition Algorithm Based on Improved Deep Learning Network Model Using Feature Visualization and Quality Evaluation. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2979755.

[4] Camisani-Calzolari, F.R., et al. "A Review on Causes of Surface Defects in Continuous Casting." IFAC Proceedings Volumes, vol. 36, no. 24, 2003, pp. 113–121, https://doi.org/10.1016/s1474-6670(17)37613-9.