Classifying Steel Defects Using Deep Learning

Steel surface defects are a common longstanding issue in steelmaking. Quality of materials, issues with production and errors caused by workers can cause these issues with the steel surface. Unfortunately, the presence and number of defects in rolled steel are a major contributor a customer's assessment the quality of rolled steel [1].

Classifying these defects traditionally is a multi-step process. The sample is first obtained to be machined, polished, mounted onto plastic and potentially etched with a strong acid to better distinguish the defect from the base steel. The sample is then viewed under a microscope to classify the defect since many look similar.

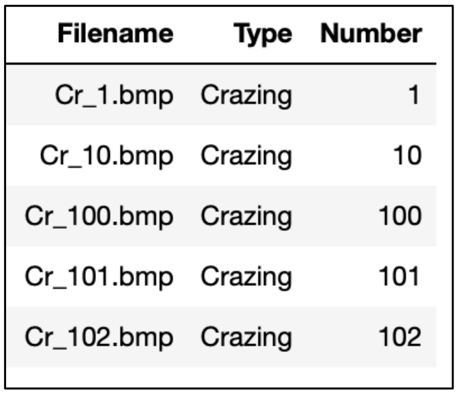
The entire process is quite laborious and requires a significant investment. The classifier that was built in this project solves a part of this problem and has the potential to expedite the process by automatically classifying defects given an image. The solution that is posed could be the addition of a camera with a very high magnification directly into the production line. In this scenario, the camera would take timed pictures of the rolled steel to detect any of irregularities. Any areas that do show defects could be analyzed by an automated microscope that could identify the type of defect using the classifier. This is a multistep process, classifying the images is the first step followed by measuring other attributes of the defect such a size, shape, number et cetera as had been stated in other reports [1]. The classifier that has been built would resolve the first step, classification.

Data Wrangling & Cleaning

The dataset was obtained from Kaggle which was extracted from a research paper [2]. It consisted of 1800 images in total. There were 300 images in each of the following defect classes: Crazing (Cr), Inclusions (In), Scratches (Sc), Rolled (RS), Patches (Pa) and Pitted (PS) [3].

- 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: Abrasion of the surface
- Rolled: Defects associated with the rolling process
- Patches: Visual irregularities in the surface

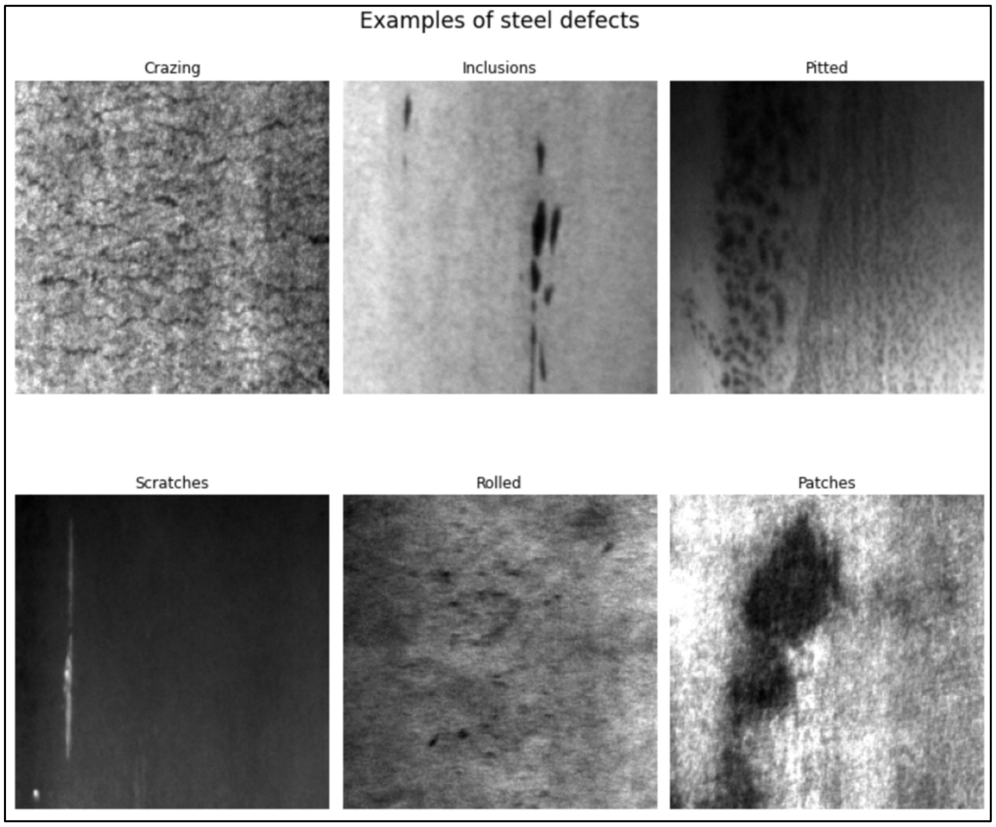
These images were originally stored as .bmp files in separate folders corresponding to their defect type. Image filenames were extracted from each folder via iteratively and consolidated into a DataFrame. The image number as well as the type of defect were added in two separate columns. The data did 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:

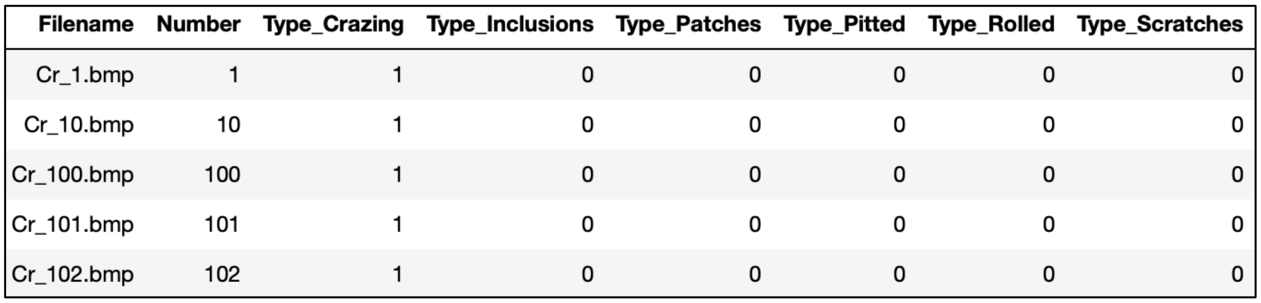


There are some similarities with these images. The Inclusion and Scratches classes look similar but look like negatives of each other. Crazing and Rolled share the similar features with defects covering their entire area and are of a similar shape and size.

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

Preprocessing

In this stage, the images were opened then stored into a numpy array in the order in which they were retrieved. This array had a shape of (1800,200,200) meaning it consisted of 1800 images each with a 200x200 array associated with their grayscale pixel values. The array was then standardized so that the pixel values range between 0 and 1. This manipulation allows the models built in the next stage to be more efficient and accurate. The defect types were also 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 class types portion were one-hot encoded and served as the target variable.

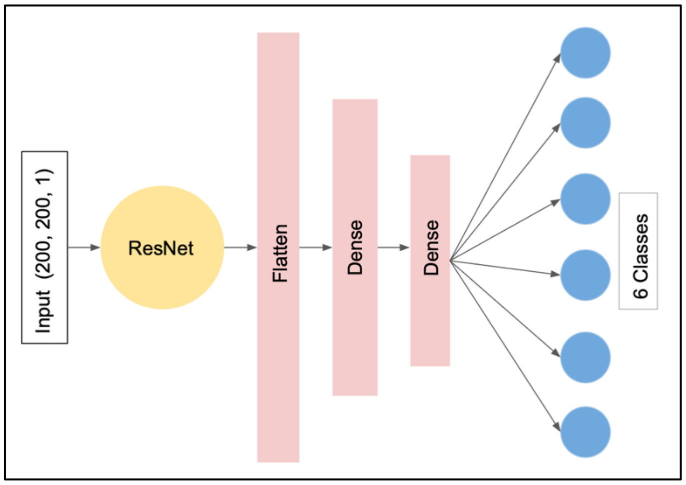
Model Explanation

A Dummy Classifier was first built to serve as a benchmark to evaluate the real models. The model received an F1-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. With 1000 classes, the top layer was not included.

Initially, a GlobalAveragePooling2D (GAP) layer was directly connected to the output layer from the ResNet model. This layer takes the average of each feature map in the previous Convolutional layer reducing its size to a one-dimensional array. GAP excels at identification of features, defects in this case, no matter their location. The GAP layer was then connected to a Dense layer to be down sampled and a second Desne layer with 6 neurons corresponding to the 6 defect classes. The results were disappointing using this method which will be explained in the subsequent section. The GAP layer was then replaced with a Flatten layer. This would preserve much of the information from the output of the ResNet model since there is no averaging. To illustrate this point, the ouput shape from the GAP layer was (,2048), while the output from the Flatten layer was (,,100352). A representation of the model is shown below:

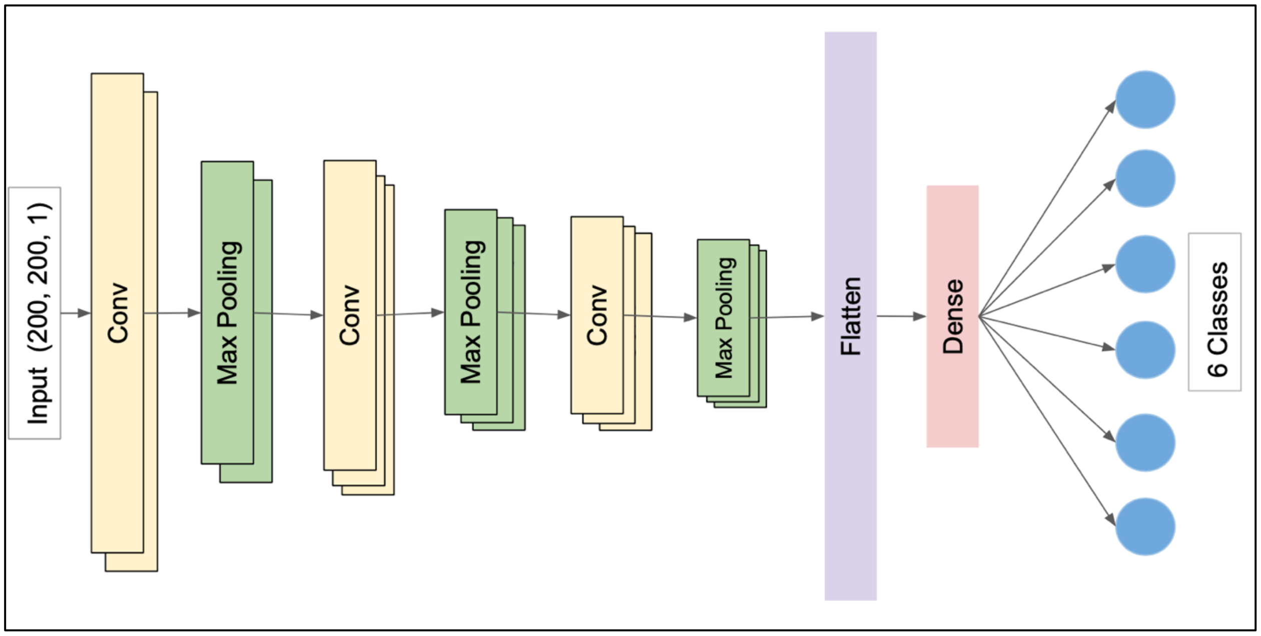


The size of each layer is only representative of the size of the actual layers and do not reflect its actual size.

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 such as objects 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 of the Convolution and pool size of the Max Pooling layers was a good starting point for any image classification. The model was ordered in this way so that each successive Convolutional layer would 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 layer with 6 neurons. Flatten was used instead of GAP due to the predicted increase in ResNet accuracy. The activation functions of all layers were ReLu with the last Dense layer being softmax which is typical for multi-class classification.

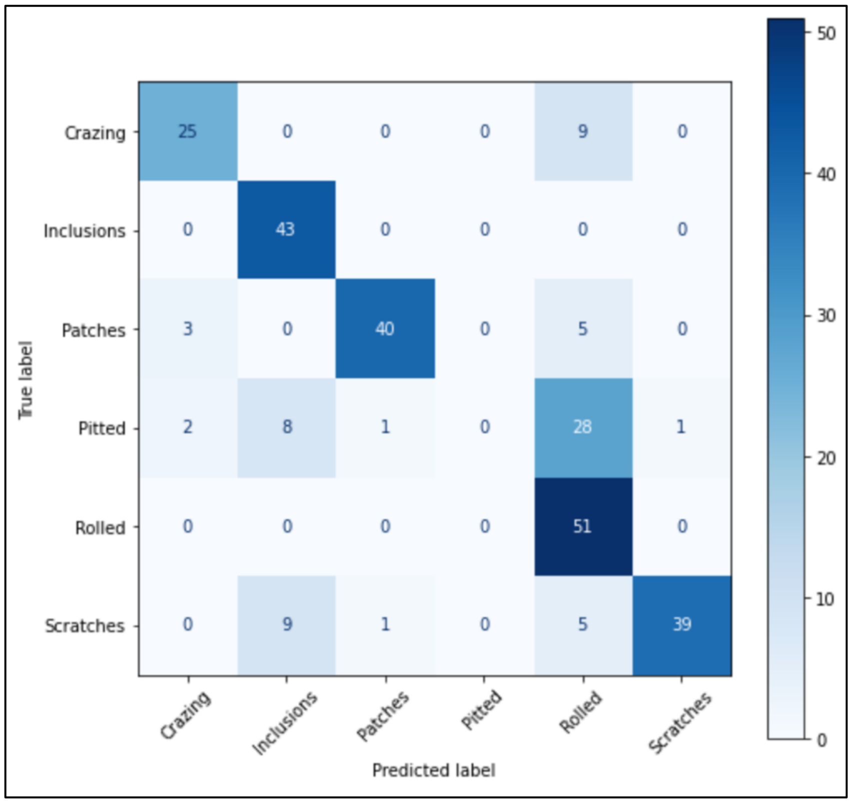
Modelling

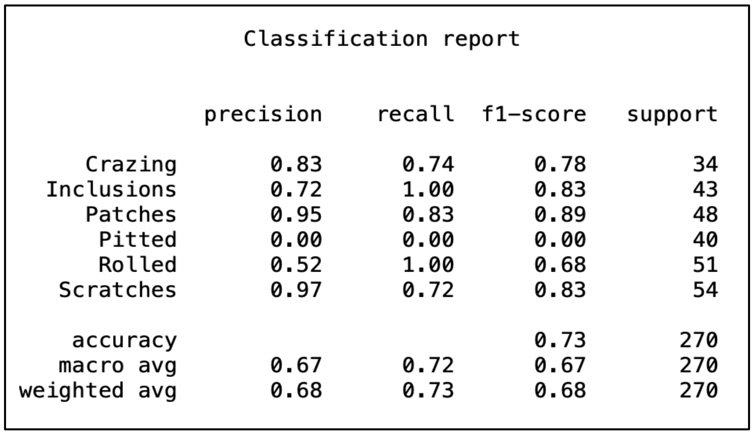
Model selection, including hyperparameter tuning were completed in this stage. A Dummy Classifier was first fit to the model to act as a benchmark for the other models. It's F1-score only the test set was 0.14.

ResNet

The ResNet model which included the GAP layer was trained and outputted a maximum F1-score accuracy on the validation set of 0.418. 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 test set and its confusion matrix and classification report can be seen below:





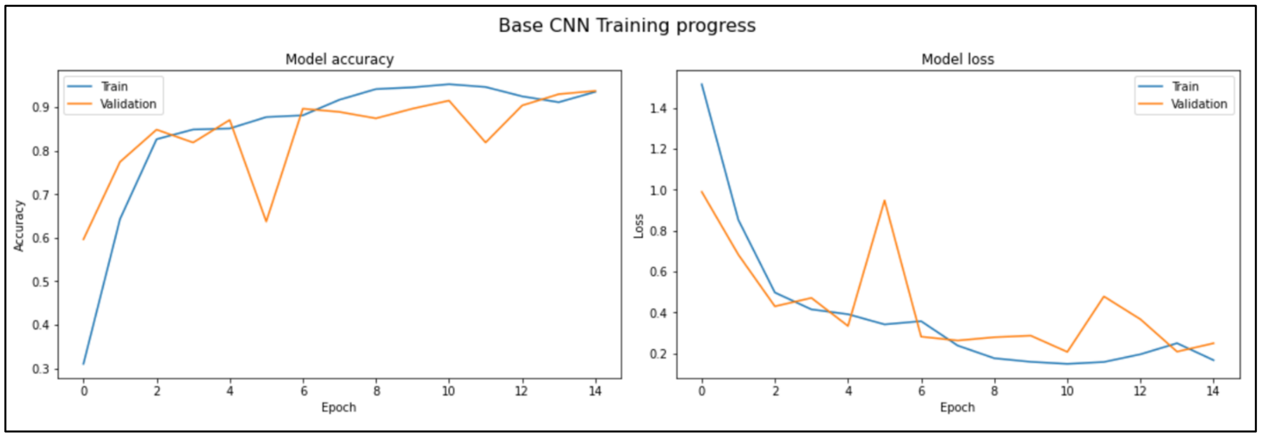
```
                 Classification report

              precision    recall  f1-score   support

     Crazing       0.83      0.74      0.78        34
  Inclusions       0.72      1.00      0.83        43
     Patches       0.95      0.83      0.89        48
      Pitted       0.00      0.00      0.00        40
      Rolled       0.52      1.00      0.68        51
   Scratches       0.97      0.72      0.83        54

    accuracy                           0.73       270
   macro avg       0.67      0.72      0.67       270
weighted avg       0.68      0.73      0.68       270
```

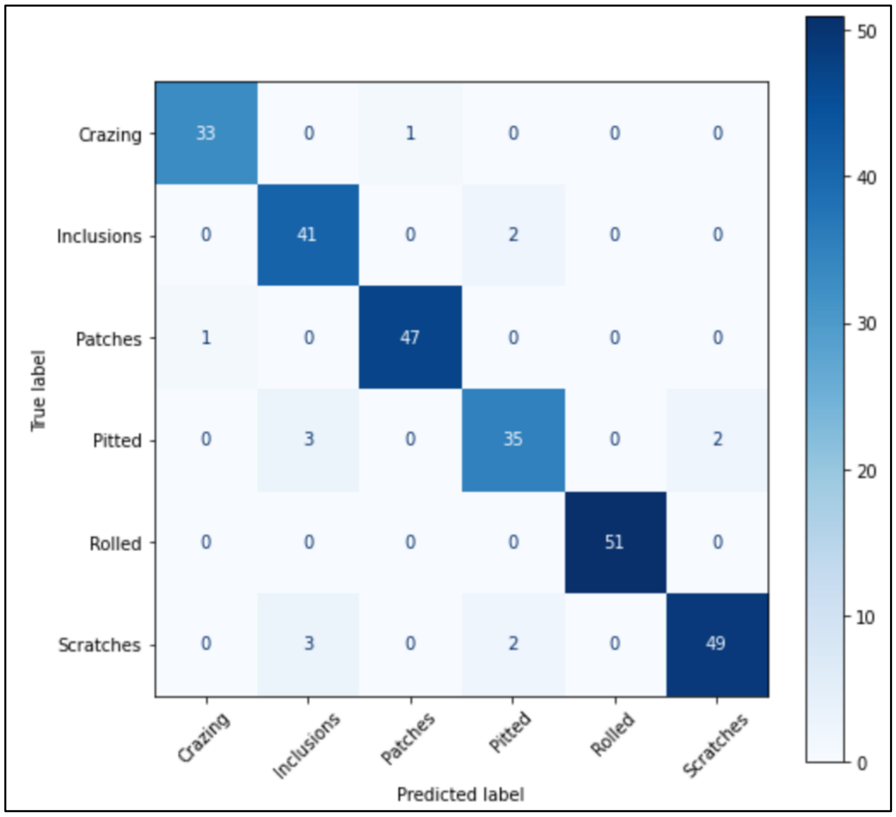
Overall accuracy was 0.73 which is quite good for a untuned model. Precision and recall in most classes were decent however it was unable to classify the “Pitted” class at all. It appears the model was confusing “Pitted” with “Rolled”. This can be attributed to the fact that both defect types look similar having small features distributed evenly in both images. The model sometimes misclassifies “Scratches” as “Inclusions” which is also understandable since they also look similar.

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

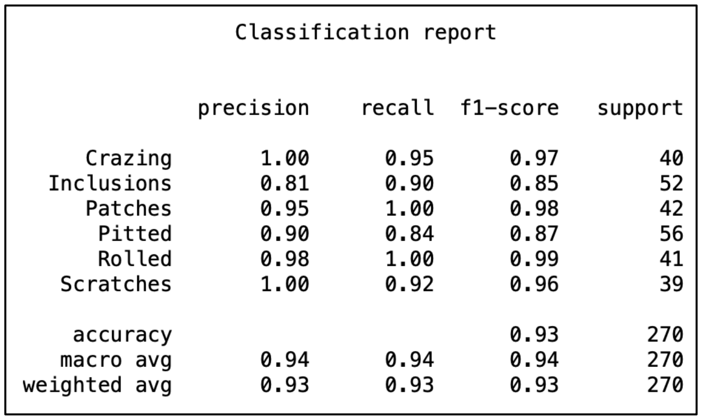
The second model to be evaluated was the custom CNN model with the default optimizer and learning rate (Adam=0.001). As mentioned above, Categorical Cross Entropy Loss was chosen. The loss and accuracy graph during training are show below. The model's best weights out of the 14 epochs were saved using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





True label \ Predicted label	Crazing	Inclusions	Patches	Pitted	Rolled	Scratches
Crazing	33	0	1	0	0	0
Inclusions	0	41	0	2	0	0
Patches	1	0	47	0	0	0
Pitted	0	3	0	35	0	2
Rolled	0	0	0	0	51	0
Scratches	0	3	0	2	0	49

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



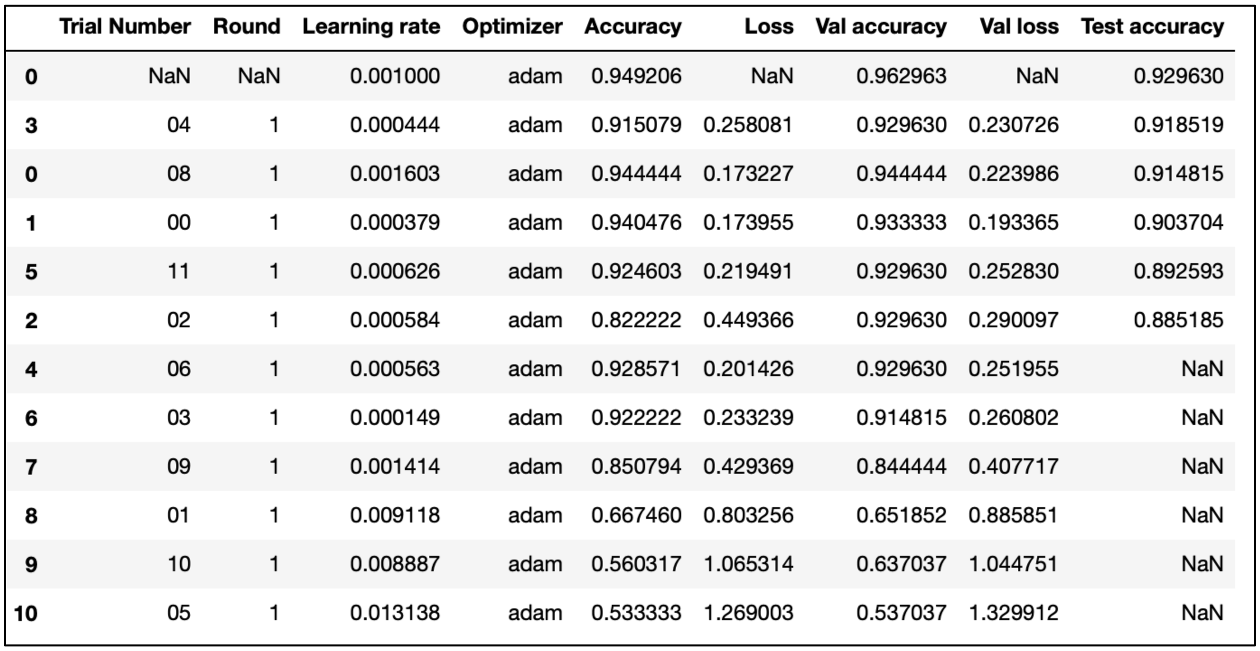
```
                  Classification report


              precision    recall  f1-score   support

     Crazing       1.00      0.95      0.97        40
  Inclusions       0.81      0.90      0.85        52
     Patches       0.95      1.00      0.98        42
      Pitted       0.90      0.84      0.87        56
      Rolled       0.98      1.00      0.99        41
   Scratches       1.00      0.92      0.96        39

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

Evaluation metrics from this model are excellent. The model 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

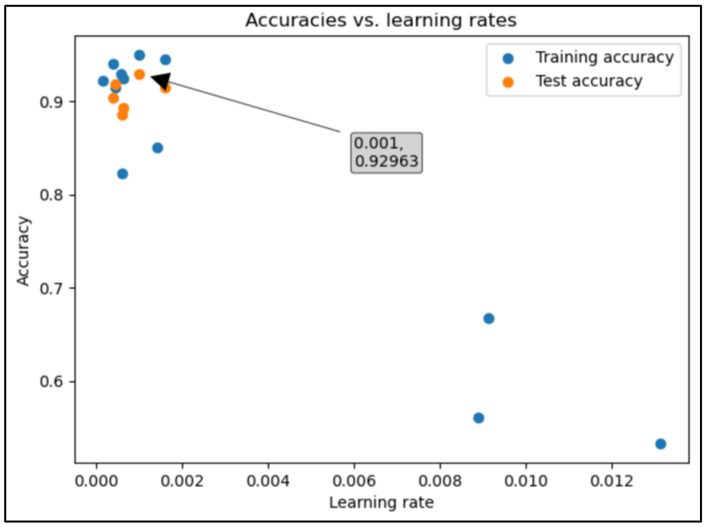
A Randomized Search was performed with a chosen learning rate between 0.0001 and 0.0151. This large range was chosen because a subsequent Grid Search was to be done to fine-tune the learning rate further but was not done due to the excellent performance of the tuned model.

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 accuracy of the 5 best models was added in a separate column. The table below displays all 11 models sorted by test accuracy.



	Trial Number	Round	Learning rate	Optimizer	Accuracy	Loss	Val accuracy	Val loss	Test accuracy
0	NaN	NaN	0.001000	adam	0.949206	NaN	0.962963	NaN	0.929630
3	04	1	0.000444	adam	0.915079	0.258081	0.929630	0.230726	0.918519
0	08	1	0.001603	adam	0.944444	0.173227	0.944444	0.223986	0.914815
1	00	1	0.000379	adam	0.940476	0.173955	0.933333	0.193365	0.903704
5	11	1	0.000626	adam	0.924603	0.219491	0.929630	0.252830	0.892593
2	02	1	0.000584	adam	0.822222	0.449366	0.929630	0.290097	0.885185
4	06	1	0.000563	adam	0.928571	0.201426	0.929630	0.251955	NaN
6	03	1	0.000149	adam	0.922222	0.233239	0.914815	0.260802	NaN
7	09	1	0.001414	adam	0.850794	0.429369	0.844444	0.407717	NaN
8	01	1	0.009118	adam	0.667460	0.803256	0.651852	0.885851	NaN
9	10	1	0.008887	adam	0.560317	1.065314	0.637037	1.044751	NaN
10	05	1	0.013138	adam	0.533333	1.269003	0.537037	1.329912	NaN

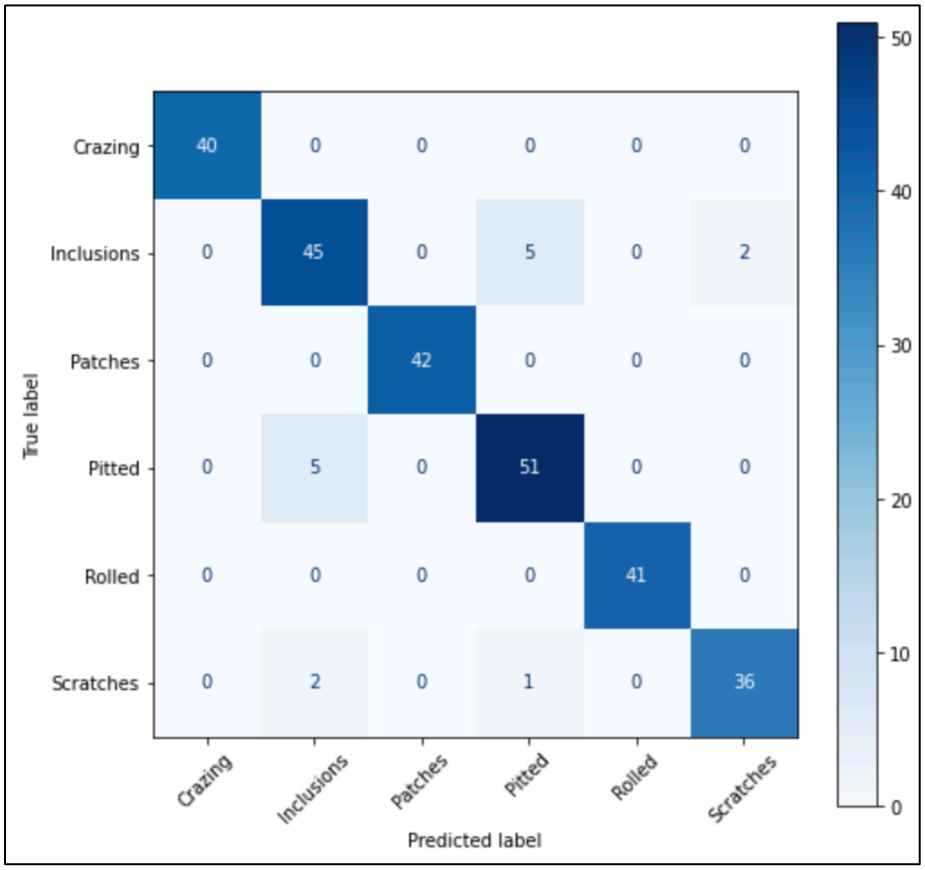
The training and test accuracies were then plotted:

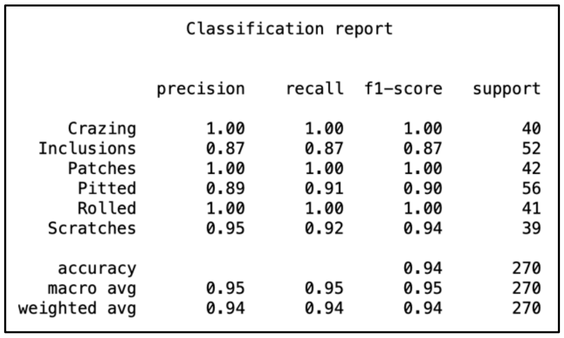


As it can be seen, the ideal learning rate corresponds to the default CNN, 0.001. It is apparent that there is a pattern here with a maximum accuracy when the learning rate is around this default value. The training accuracy rapidly diminishes when higher rates are used. Testing evaluation wasn't performed on these models since their training accuracy was so low.

CNN – Image Tuning

The CNN with an Adam optimizer and a learning rate of 0.001 was chosen to be the final model. However, the images were not z-score normalized. Their values ranged from (0,1) but their mean and variance were not 0 and 1 respectively. CNNs work best when data is z-score normalized, therefore the feature sets were z-score normalized and the model was trained using these adjusted sets. The target variable was left as is. The results are as follows.





```
                  Classification report


              precision    recall  f1-score   support

     Crazing       1.00      1.00      1.00        40
  Inclusions       0.87      0.87      0.87        52
     Patches       1.00      1.00      1.00        42
      Pitted       0.89      0.91      0.90        56
      Rolled       1.00      1.00      1.00        41
   Scratches       0.95      0.92      0.94        39

    accuracy                           0.94       270
   macro avg       0.95      0.95      0.95       270
weighted avg       0.94      0.94      0.94       270
```

This process did result in a marginal improvement in accuracy but an improvement none the less. This model and the others were saved in the /models section of the project.

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 normalized data is the model with the best predictive power, however it will be computationally expensive to normalize all images before classification. The baseline CNN trained on the non-normalized images performs almost as well as the CNN (normalized) with a difference of 0.02%. Therefore, the default CNN is chosen to be the final model.

It should be noted that this model is overall excellent at classification but the 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 notable 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.

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 characterise these defects.

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] 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.

[3] 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.