Machine Tool Wear Classification by Time Series Imaging using Convolution Neural Networks

1. Introduction

Tool Wear Monitoring has become an important process in the current industrial developments for cost-effective production. Early prediction of wear can be very useful because the process of machining can be efficient and economical. Tool Wear Monitoring has become an important part of Industry 4.0.

In this Mini-Project Tool Wear Classification is done through the deployment of Convolution Neural Networks and Deep Learning. Time Series data is converted to Images which is further used for classification of Tool Wear.

2. Data

Tool wear data is obtained from public dataset of PHM 2010 Data Challenge. The data contains time series data of six end mill (6mm 3-flute) cutters. These end mills were machined using a high-speed CNC machine under dry milling condition. Seven Time Series data of Force in XYZ direction, Vibrations in XYZ direction and Acoustic Emission data were obtained.

The 6mm end mill was machined line by line with an axial depth of cut of 0.2mm and radial depth of cut of 0.125 mm on a workpiece till the whole surface was machined. Then the tool was removed and its wear was measured under a microscope. Then again it was machined with the above parameters and wear was measured. These cutting tests were done for 315 times. A total of 315 files are present for each cutter with recordings of 315 wear values.

A dynamometer was used to get the cutting forces obtained in the XYZ Direction during machining. An accelerometer was used to get the vibrations obtained in the XYZ direction during machining. An Acoustic Emission Sensor was used to get the acoustic data emitted during machining. In this project, data of cutter C1, C4 and C6 were used.

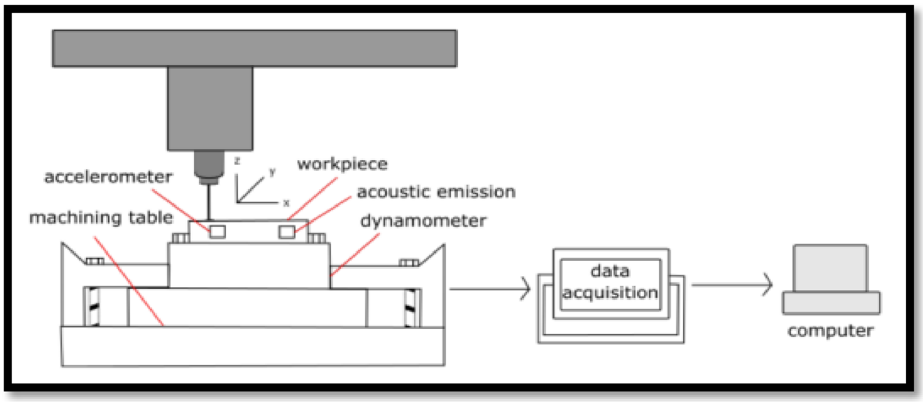


Figure 1. Experiment Setup.

3. Methodology

3.1 Data Pre-Processing/ Imaging

The time series data can be converted to images with the help of a mathematical concept called as Gramian Angular Field Encoding. Here the time series data is normalized and then converted to polar form using inverse cosine transformation. Converting to polar form preserves the temporal information of the data. From this the temporal correlation can be exploited with the different time intervals. Temporal correlation is a matrix of cosines of the converted polar form data.

Here, Gramian Angular Summation Field is used to get images. Time series data of the forces in XYZ direction is used for classification. The process of obtaining images is given below:

1. Step 1: Normalize

First the time series data has to be normalized to the range of -1 to 1. The formula to normalize the time series is given below.

$$X_{normalized} = \frac{(\ x_i - \max(x)\ ) + (\ x_i - \min(x)\ )}{\max(x) - \min\ (x)}$$

Where, *x is the time series data.*

2. Step 2: Piecewise Aggregate Approximation

Since the data contains huge number of recordings (almost 200,000 for each sensor readings and 200,000 for each 315 tests), the data can be compressed by the mathematical operation of Piecewise Aggregate Approximation.

$$\bar{x}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j$$

Where, $\bar{x}_i = mean\ of\ i\ th\ element$

$w = reduced\ total\ number\ of\ dataset\ and\ it\ is\ w < n$

$n = total\ number\ of\ data$

In this project w is 250 and n is 1000. So, for every 1000 data we crush it to 250 averaged data and this will be the size of the image obtained.

3. Step 3: Convert to Polar Form

Next, the PAA data is converted to polar form by inducing a inverse cosine transform of each data.

$$\emptyset = arcos(x) \quad where -1 \leq x \leq 1$$

4. Step 4: Gramian Angular Summation Field Encoding

GASF Encoding is done through getting the Temporal correlation of the polar form data. The correlation matrix is obtained by:

$$GASF = \begin{bmatrix} \cos(\emptyset_1 + \emptyset_1) & \cos(\emptyset_1 + \emptyset_2) & \dots\dots\dots \cos(\emptyset_1 + \emptyset_n) \\ \vdots & \vdots & \vdots \qquad \vdots \\ \cos(\emptyset_n + \emptyset_1) & \cos(\emptyset_n + \emptyset_2) & \dots\dots \quad \cos(\emptyset_n + \emptyset_n) \end{bmatrix}$$

5. Step 5: Convert to Image

Convert the GASF Matrix to an image for passing through the deep learning networks.

Note: in this project, all three-force readings were converted to images and merged together as one image for each set of 1000 readings in the data file. Totally getting more than 63000 images for each cutter.

3.1.1 Data Split

For each file, 75% were train data and 25% were validation data. The split graph is given below:

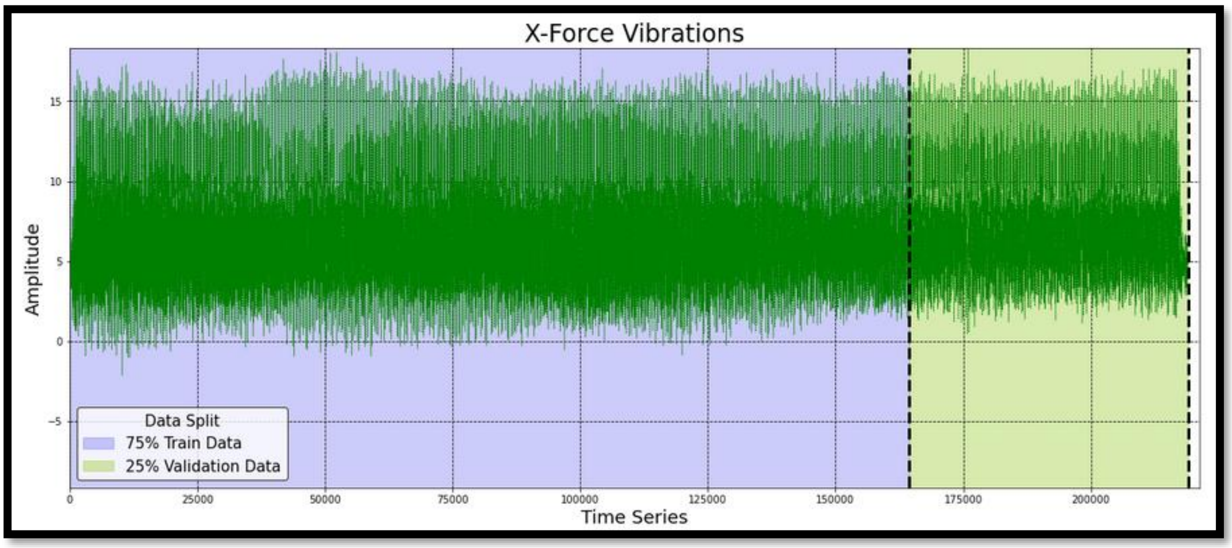


Figure 2. Data Train-Valid-Split

3.2 Deep Learning

Convolution Neural Network is normally used to classify images. After convolution the data is fed to layers of deep neural networks for classification. The architecture used is given below.

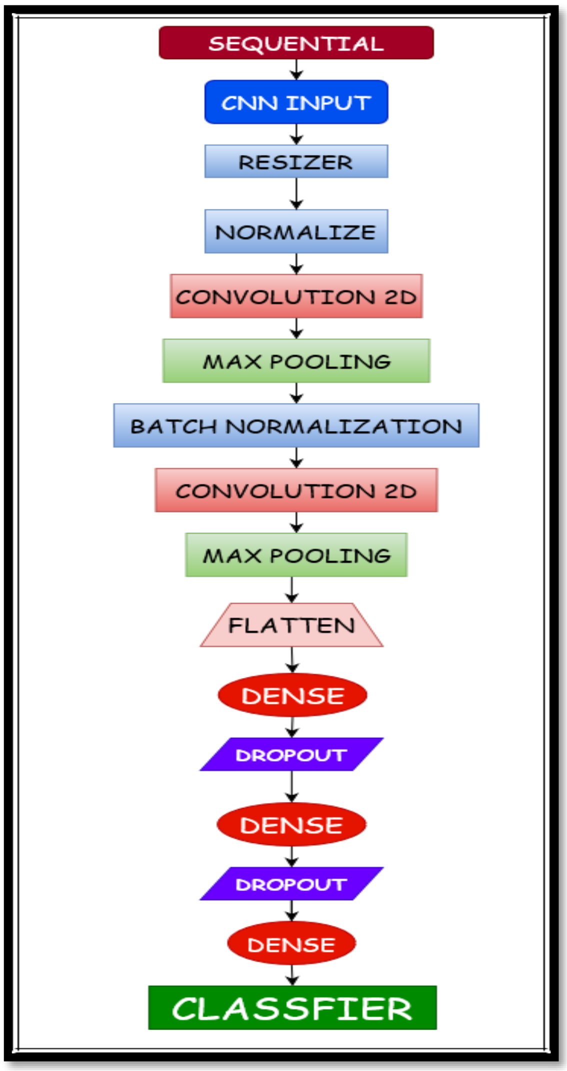


Figure 3. Architecture of the Neural Networks

The model consists of two 2-D Convolution layers with 64 filters and kernel size of 7 with padding as 'same' and 'relu' activation function. After Convolution the data is fed into two Maximum Pooling layers with pooling size as 4 with padding as 'same'. In between, there is a Batch Normalization layer. The output from CNN is then flattened and fed to three dense layers with 192, 84 and 16 units respectively with 'relu' activation. There is two dropout layers after the first dense layer and second dense layer. Finally, there is the final dense layer with the number of classes i.e., 4 with 'softmax' activation.

The loss function used is Categorical Crossentropy function. The optimizer used is Adam optimizer with learning rate of 0.001. The metrics used to measure the accuracy is selected as Categorical Accuracy. The model is finally compiled and trained.

4. Analysis

4.1 Image Generation

The images are generated as per the procedure mentioned in Section 3.1. Images are generated for all the three cutters. The first 75% of data in each cut event file is taken as Training set and the next 25% is taken as the Validation set. A total of more than 200,000 images were obtained. As mentioned, the time series of Forces in XYZ Direction are combined into one image file with 3 channels.

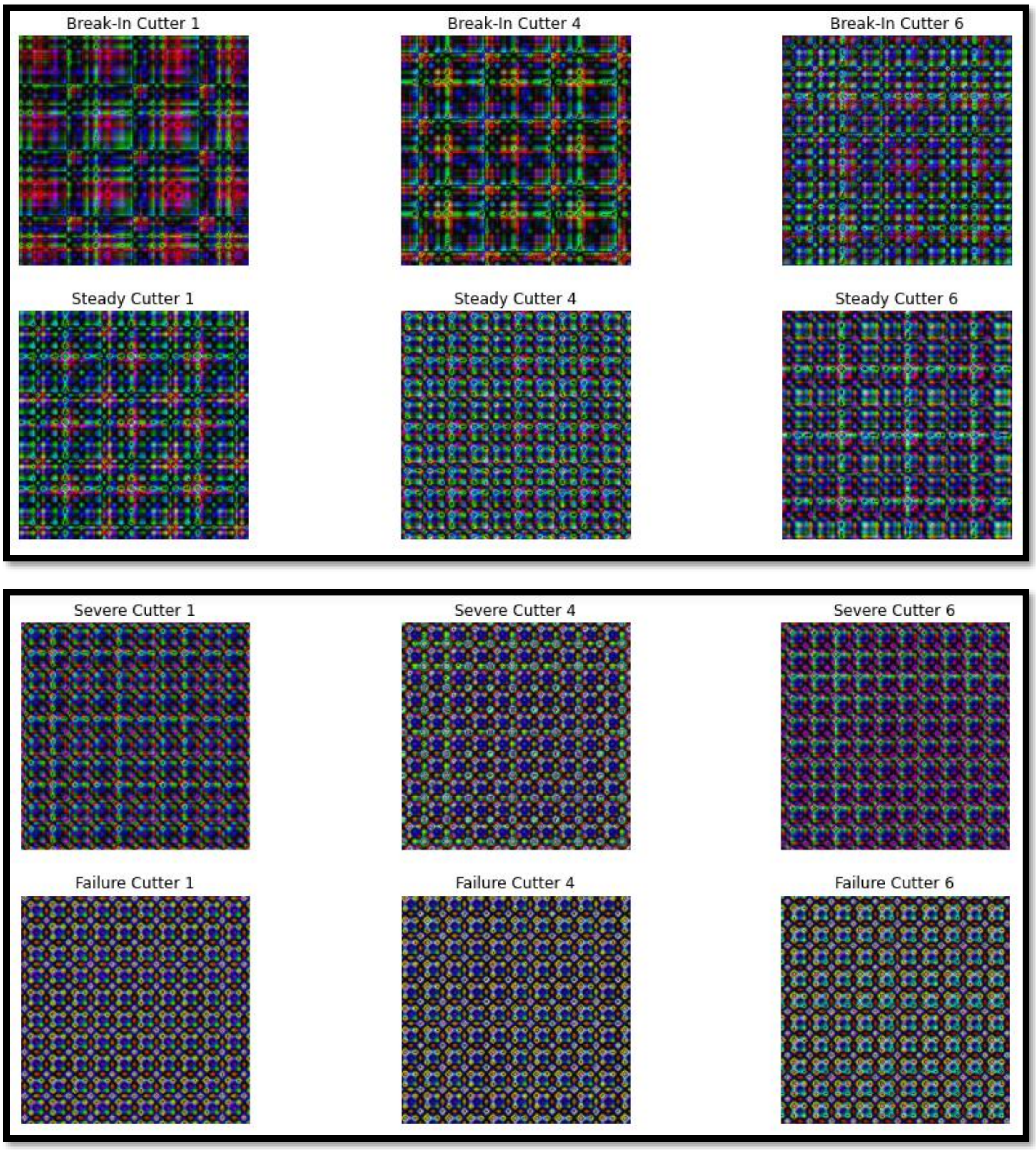


Figure 4. GASF Images.

The tool wear classification is according to the wear generation with respect to time. There are four classes of wear when a tool is used in machined. They are given below:

1. Break-in Wear: Here the tool is new. During machining wear increases linearly and sharply. Since the tool is new, initial wear is high.

2. Steady Wear: During machining there is not much change in tool wear and machining quality is the best in this region.

3. Severe Wear: In this region, the wear starts to increase slightly. Machining quality stars to worsen and chattering of spindle starts to takes place.

4. Failure Wear: In this region wear is heavy and chattering is also high. The tool needs to be discarded as machining quality is low and material removal worse.

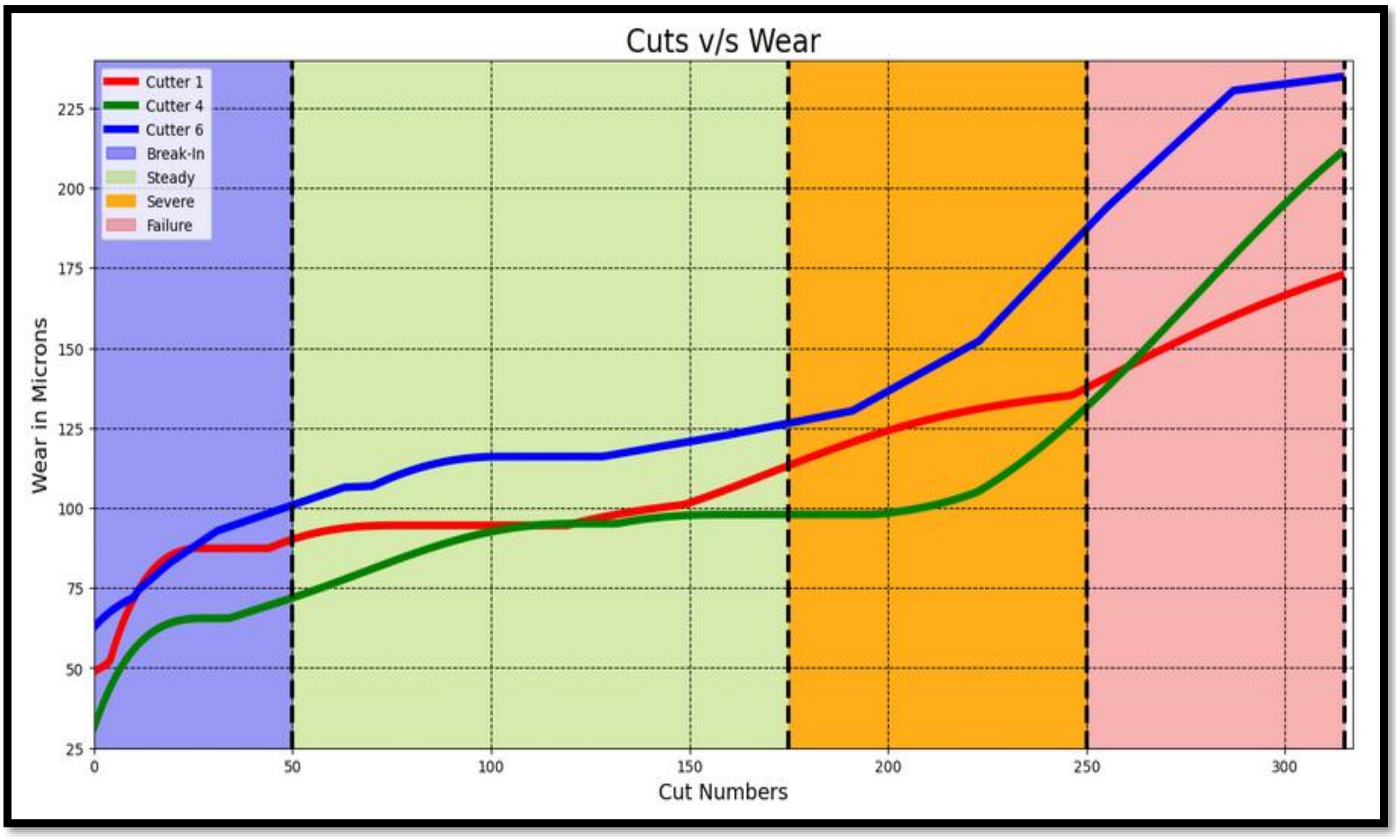


Figure 5. Wear propagation as function of Cutting time/Cutting Event.

Here:

a. From Cut 001 ==> Cut 050 is defined as ==> Break-In Wear.

b. From Cut 051 ==> Cut 175 is defined as ==> Steady Wear.

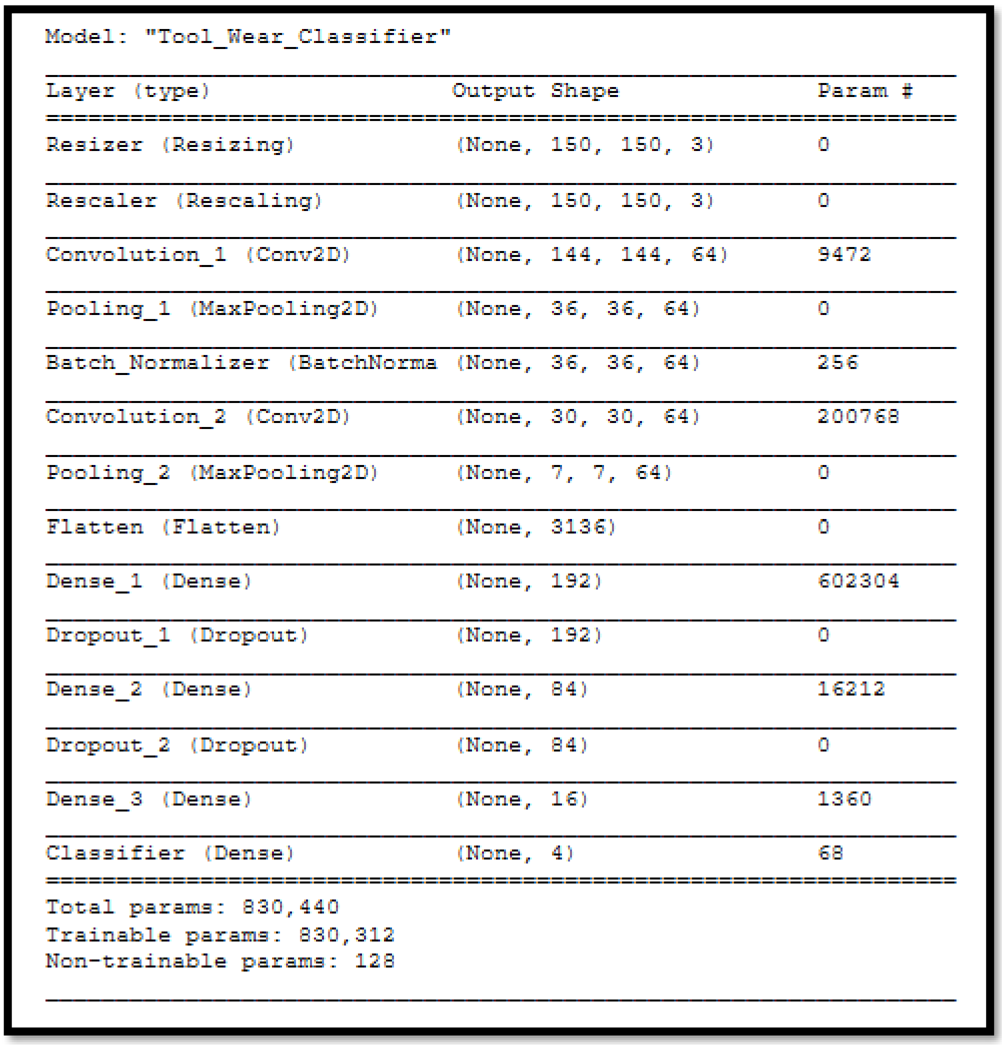
c. From Cut 176 ==> Cut 250 is defined as ==> Severe Wear.

d. From Cut 251 ==> Cut 315 is defined as ==> Failure Wear.

4.2 Neural Network Model

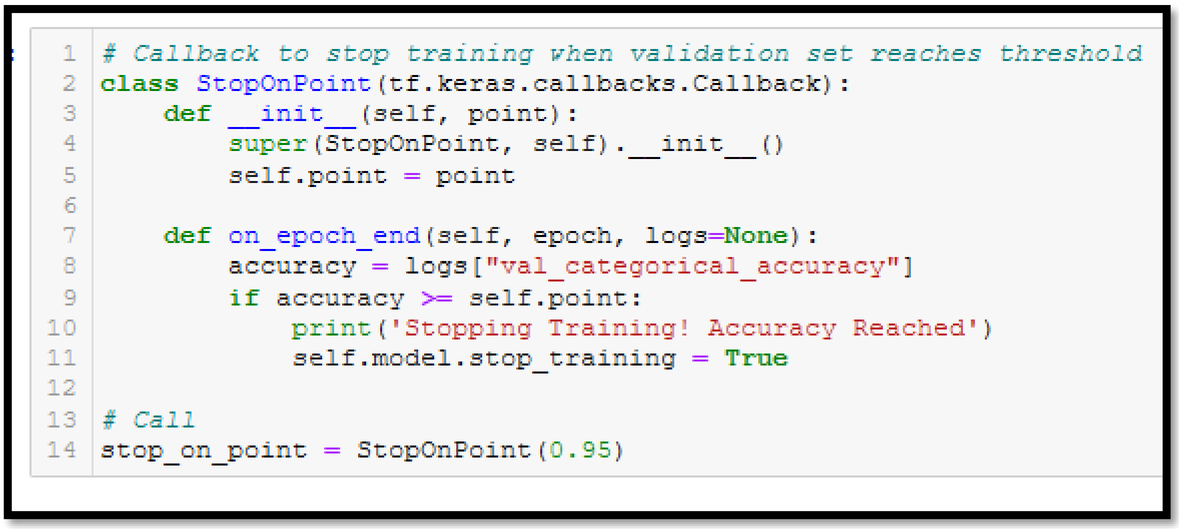
Using TensorFlow's Keras Library the model is defined as per the architecture mentioned in Section 3.2. The training and validation set is fed to the model. A sequential model is used. The model is compiled with Adam Optimizer, Categorical Crossentropy Loss Function and Categorical Accuracy Metrics.

Two callback's are defined consisting of a CSV Logger to record the losses and accuracies over training epochs. Another custom callback is defined to stop training when validation accuracy reaches a certain threshold. The threshold is set to 0.95 in the project.



```
Model: "Tool_Wear_Classifier"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Resizer (Resizing)           (None, 150, 150, 3)       0
_________________________________________________________________
Rescaler (Rescaling)         (None, 150, 150, 3)       0
_________________________________________________________________
Convolution_1 (Conv2D)       (None, 144, 144, 64)      9472
_________________________________________________________________
Pooling_1 (MaxPooling2D)     (None, 36, 36, 64)        0
_________________________________________________________________
Batch_Normalizer (BatchNorma (None, 36, 36, 64)        256
_________________________________________________________________
Convolution_2 (Conv2D)       (None, 30, 30, 64)        200768
_________________________________________________________________
Pooling_2 (MaxPooling2D)     (None, 7, 7, 64)          0
_________________________________________________________________
Flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
Dense_1 (Dense)              (None, 192)               602304
_________________________________________________________________
Dropout_1 (Dropout)          (None, 192)               0
_________________________________________________________________
Dense_2 (Dense)              (None, 84)                16212
_________________________________________________________________
Dropout_2 (Dropout)          (None, 84)                0
_________________________________________________________________
Dense_3 (Dense)              (None, 16)                1360
_________________________________________________________________
Classifier (Dense)           (None, 4)                 68
=================================================================
Total params: 830,440
Trainable params: 830,312
Non-trainable params: 128
_________________________________________________________________
```

Figure 6. Model Summary.



```python
# Callback to stop training when validation set reaches threshold
class StopOnPoint(tf.keras.callbacks.Callback):
    def __init__(self, point):
        super(StopOnPoint, self).__init__()
        self.point = point

    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs["val_categorical_accuracy"]
        if accuracy >= self.point:
            print('Stopping Training! Accuracy Reached')
            self.model.stop_training = True

# Call
stop_on_point = StopOnPoint(0.95)
```

Figure 7. Custom Callback.

The model was trained for 10 epochs and the following table describes the change in losses and accuracy.

epoch	categorical_accuracy	loss	val_categorical_accuracy	val_loss
1	0.894051	0.259612	0.81846	0.533846
2	0.954077	0.11971	0.766969	0.706282
3	0.960554	0.100405	0.895604	0.255363
4	0.964896	0.091553	0.877658	0.383292
5	0.96853	0.081732	0.858175	0.45545
6	0.969487	0.079129	0.867036	0.504218
7	0.972617	0.071479	0.918911	0.253069
8	0.974192	0.068471	0.799907	1.346509
9	0.976468	0.062258	0.910191	0.535839
10	0.978891	0.056943	0.908249	0.510947

The training accuracy and validation accuracy at the end of epoch was 97.88 % and 90.82 %. The peak validation accuracy was 91.89% at epoch 7. The model was finally saved for future usage.

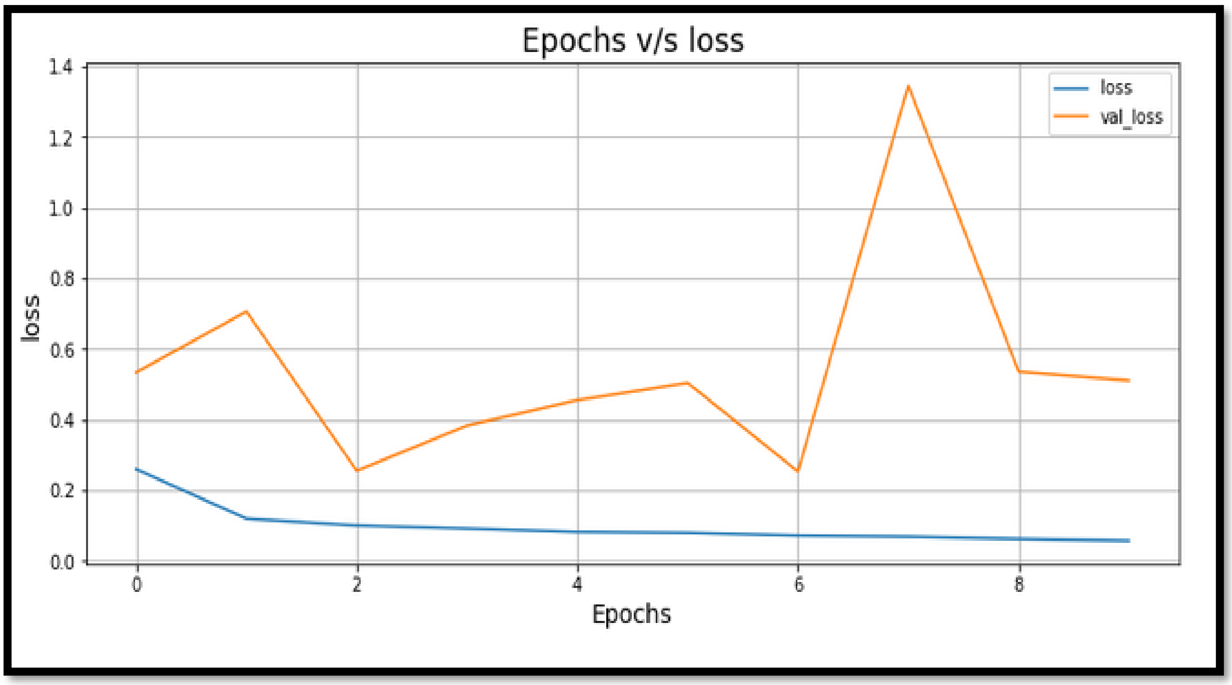


Figure 8. Loss variation in different Epochs

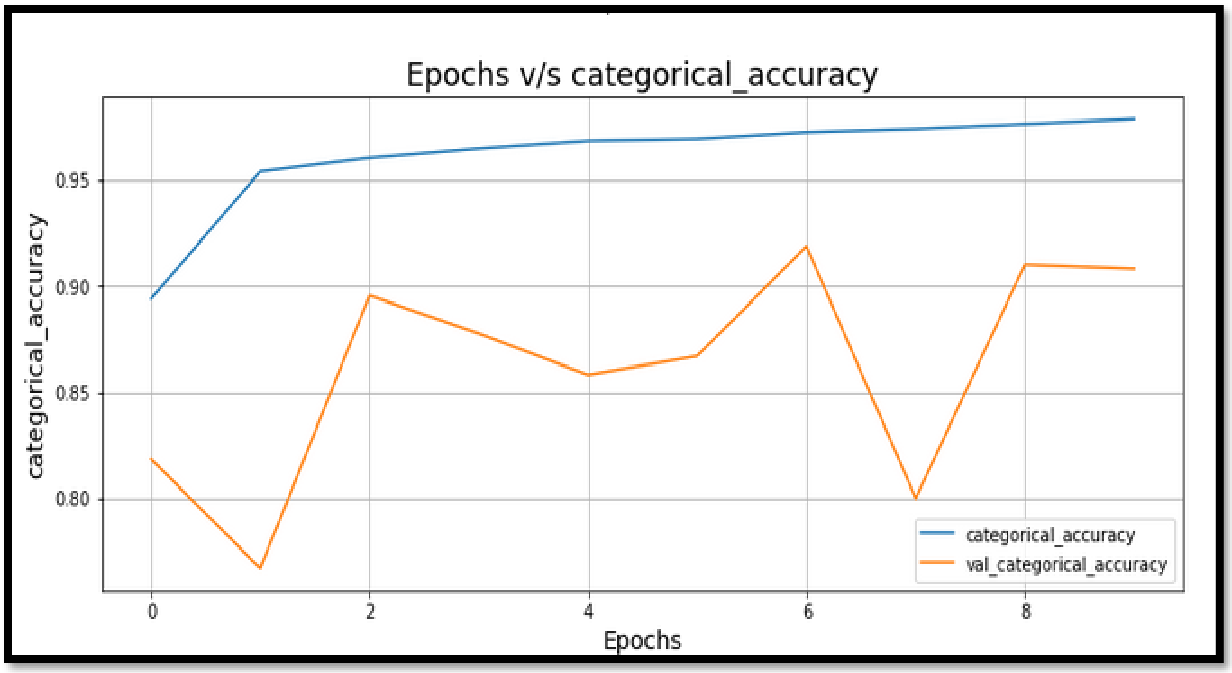
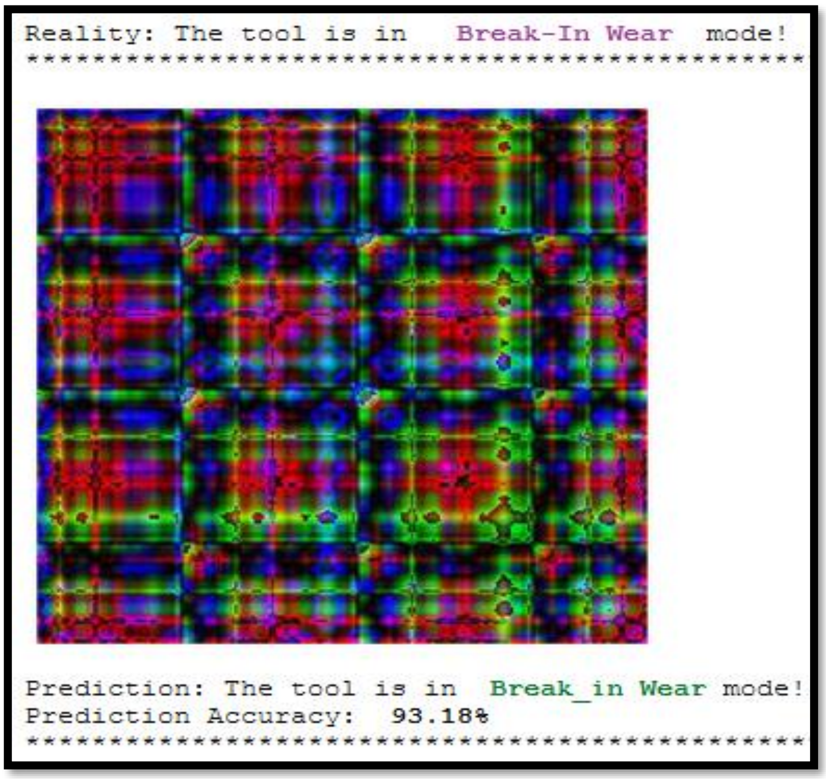


Figure 9. Accuracy variation in different Epochs

5. Results

5.1 Prediction

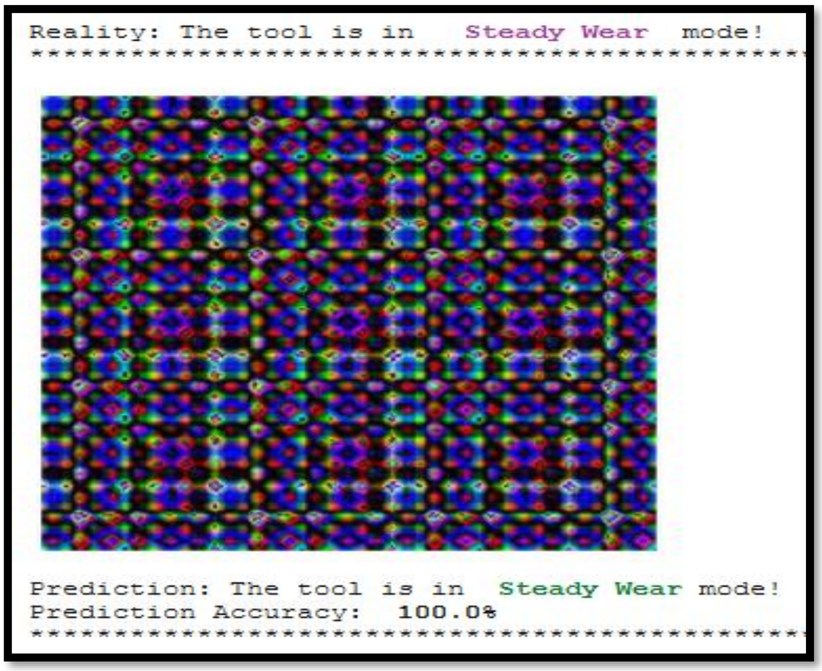
The saved model was loaded and used to predict tool wear. A function was written, where it prints the prediction class and its respective accuracy of prediction.



```
Reality: The tool is in   Break-In Wear  mode!
***********************************************
```

```
Prediction: The tool is in  Break_in Wear mode!
Prediction Accuracy:  93.18%
***********************************************
```

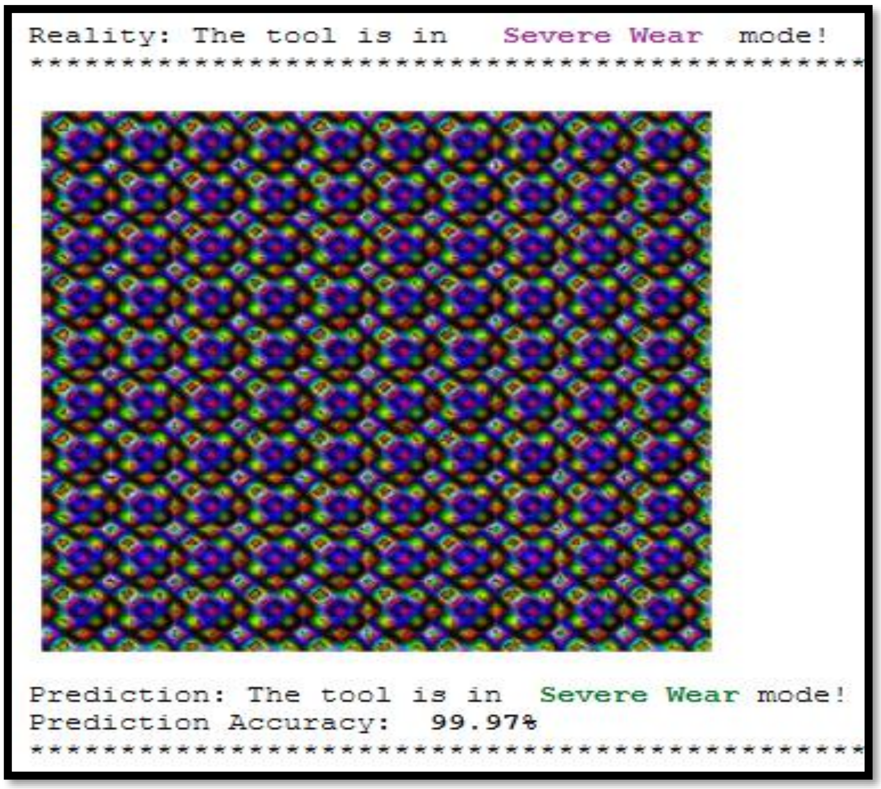
Figure 10. Break-In Wear Prediction.



```
Reality: The tool is in   Steady Wear  mode!
***********************************************
```

```
Prediction: The tool is in  Steady Wear mode!
Prediction Accuracy:  100.0%
***********************************************
```

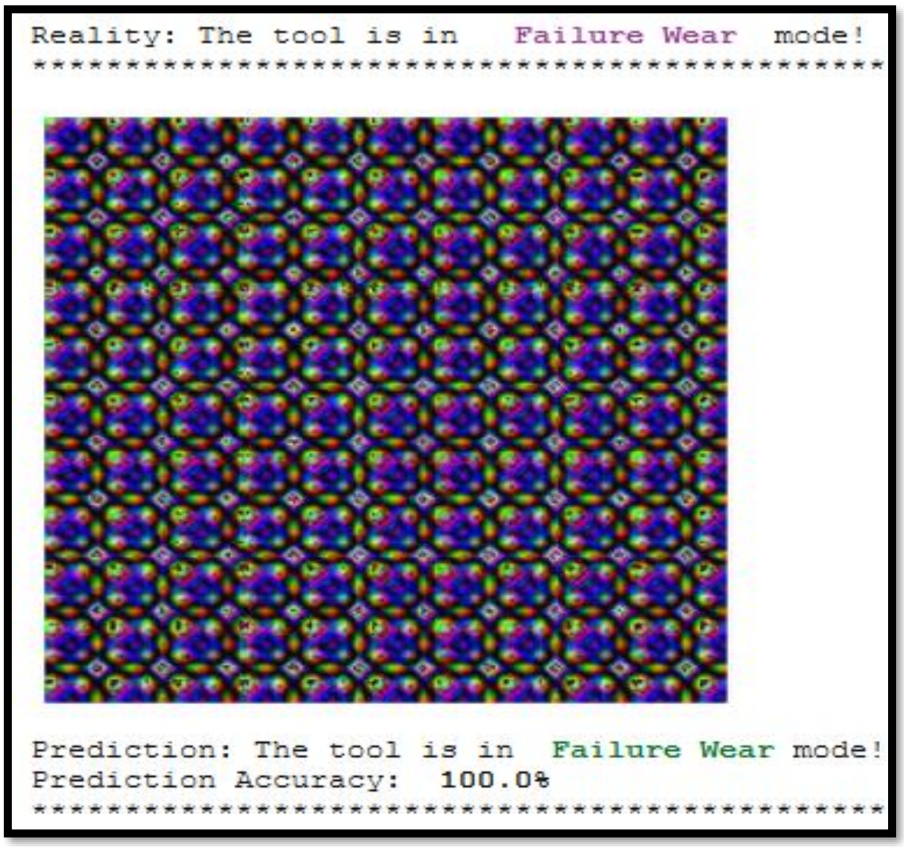
Figure 11. Steady Wear Prediction.



```
Reality: The tool is in   Severe Wear  mode!
*********************************************
```

```
Prediction: The tool is in  Severe Wear mode!
Prediction Accuracy:  99.97%
*********************************************
```

Figure 12. Severe Wear Prediction.



```
Reality: The tool is in   Failure Wear  mode!
*********************************************
```

```
Prediction: The tool is in  Failure Wear mode!
Prediction Accuracy:  100.0%
*********************************************
```

Figure 13. Failure Wear Prediction.

5.2 Confusion Matrix and Classification Report

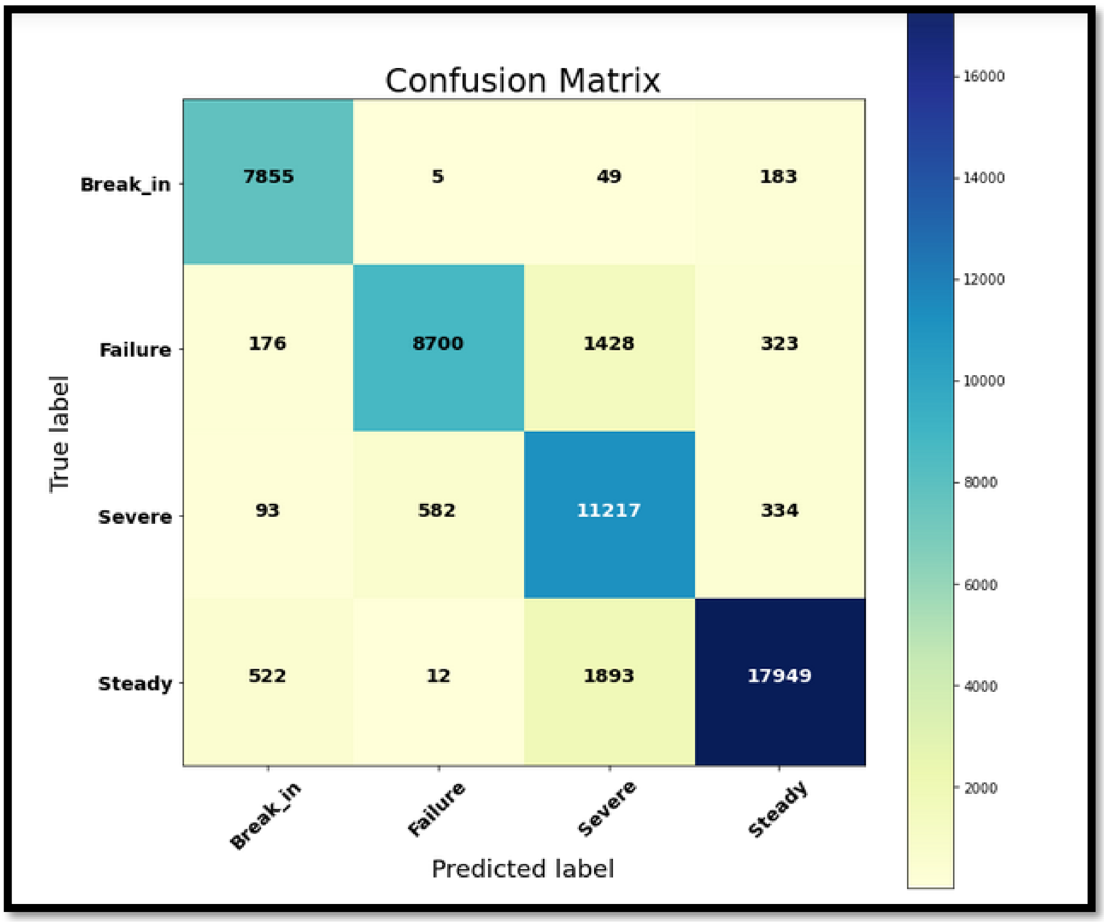
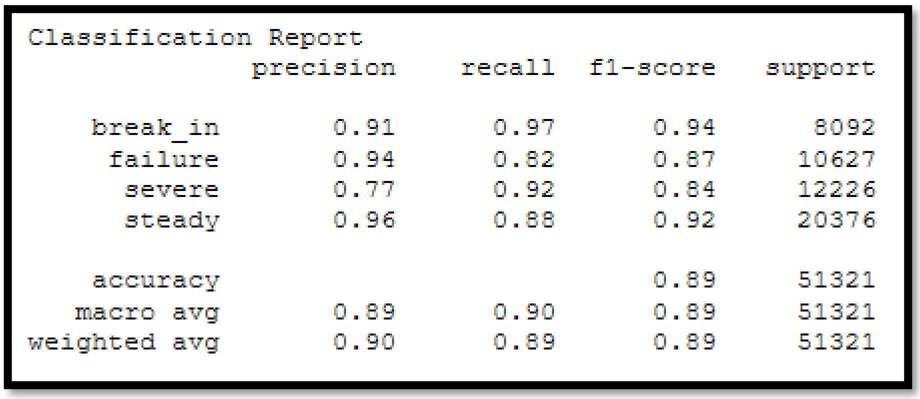


Figure 14. Confusion Matrix for Validation Data Prediction.



```
Classification Report
              precision    recall  f1-score   support

    break_in       0.91      0.97      0.94      8092
     failure       0.94      0.82      0.87     10627
      severe       0.77      0.92      0.84     12226
      steady       0.96      0.88      0.92     20376

    accuracy                           0.89     51321
   macro avg       0.89      0.90      0.89     51321
weighted avg       0.90      0.89      0.89     51321
```

Figure 15. Classification Report.

6. Conclusion

Gramian Angular Field Encoding can be successfully used to classify tool wear regions. Convolution Neural Networks can be easily used to classify GASF images.

In this project, the training accuracy was about 97% and validation accuracy was about 90%. Further, different architecture can be used to increase the validation accuracy. The Confusion Matrix shows how many images were classified accurately and how many were falsely classified. The Classification report gives us the precision-recall- f1-score details.

Only the force data of cutter 1, 4, 6 were used. Further vibration and acoustic emission data can be added to increase the model accuracy and can be generalised better. If the data of Cutter 2, 3, 5 are available, then the model can be further generalised.

7. Reference

1. Research Paper:

Tool wear classification using time series imaging and deep learning by Giovanna Martínez-Arellano, German Terrazas & Svetan Ratchev, The International Journal of Advanced Manufacturing Technology.

2. Data: https://www.kaggle.com/tobbyrui/phm2010