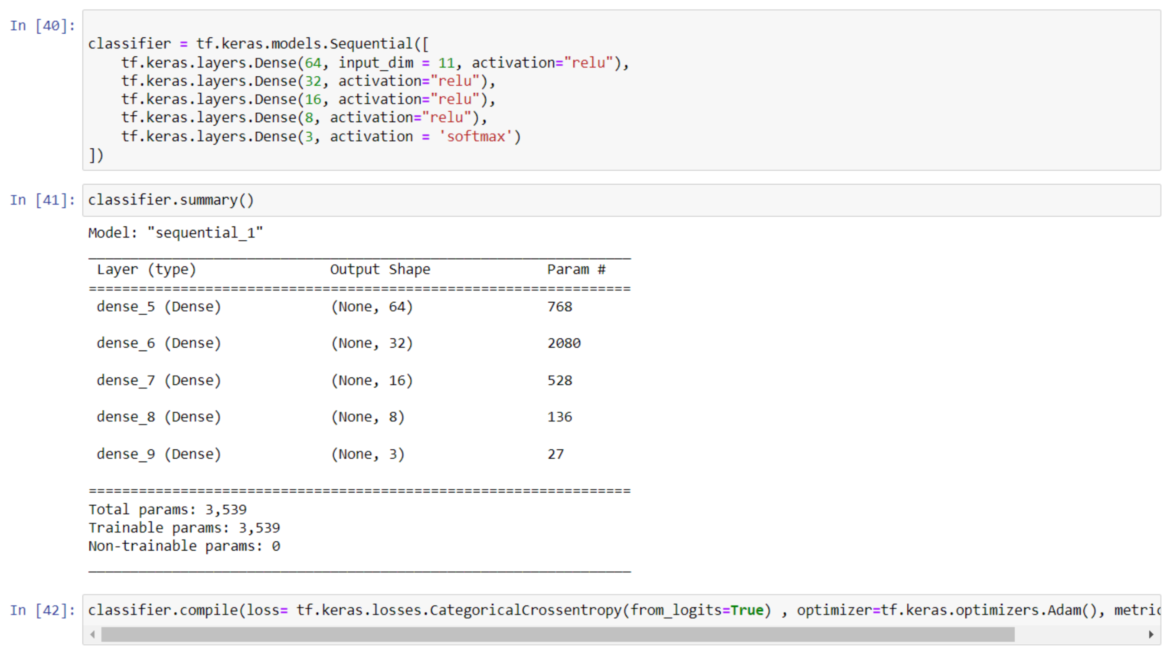


```
In [40]:
classifier = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, input_dim = 11, activation="relu"),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(8, activation="relu"),
    tf.keras.layers.Dense(3, activation = 'softmax')
])
```

```
In [41]: classifier.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 64)                768

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (Dense)             (None, 16)                528

 dense_8 (Dense)             (None, 8)                 136

 dense_9 (Dense)             (None, 3)                 27

=================================================================
Total params: 3,539
Trainable params: 3,539
Non-trainable params: 0
_________________________________________________________________
```

```
In [42]: classifier.compile(loss= tf.keras.losses.CategoricalCrossentropy(from_logits=True) , optimizer=tf.keras.optimizers.Adam(), metric
```

Fig. 9 Python code showing the architecture of the ANN

The input layer consists of 11 input neurons corresponding to 5 process parameters pulse-on time, pulse–off time, wire feed rate, servo voltage, input current and 6 pulse data parameters OSR, NSR, ASR, SSR, Discharge energy and Spark frequency. It has 4 hidden layers with ReLU (Rectified Linear Unit) as an activation function because it avoids the vanishing gradient problem. The output layer has 3 neurons in which the softmax activation function is used due to the multi-categorical nature of the classification problem. Softmax function is generally used as the activation function of the last layer in most of the ANN architecture as it converts the result into a probability distribution. Finally, the ANN predicts the probability of each class and depending upon the highest probability the class is assigned. The output is categorized into three classes corresponding to each mode of failure :- SA(Spark Absence), NM(Normal Machining) and WB(Wire Break). Spark absence is when the spark gets quenched resulting no machining. This results is loss of productivity and time. Normal machining as the name suggests refers to the normal state of the wire without any rupture. Wire break refers to failure or rupture of wire resulting in interruption of machining process. During training the loss function used is categorical cross-entropy. This loss function is used when the data is one-hot encoded and the classification problem is multi-categorical or multi-class in nature. In this study Adam optimizer is used which is an extension of stochastic gradient descent. “This optimizer is study this value of ‘k’ has been specified to be 5. kNN has several algorithms of itself for computing the distance between neighbors like Brute force, kDTree and Ball Tree, which have their own merits and demerits. The time complexity for Brute force, kDTree and Ball Tree is parabolic, logarithmic and linear respectively, thus demonstrating the fact that time complexity of Ball Tree is best. But for space complexity the order is reversed and Ball Tree performs the worst. So we need to find a balance between them as per the needs of our problem. Fortunately Scikit learn comes to our aid here by providing an “auto” algorithm option which adjusts the algorithm automatically as per the dataset.

The model accuracy was found to be 0.8636 or 86.36 % with a mean cross-validation score of 0.90833. The confusion matrix and classification report are shown in the Fig. 16 and Fig. 17 respectively.

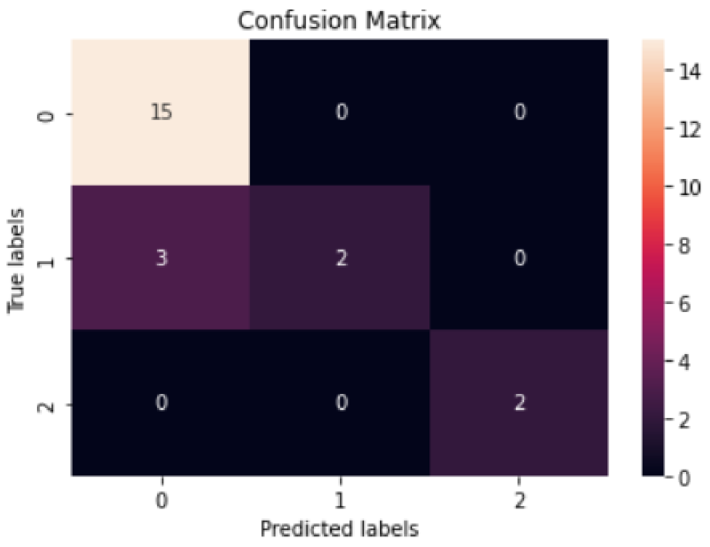
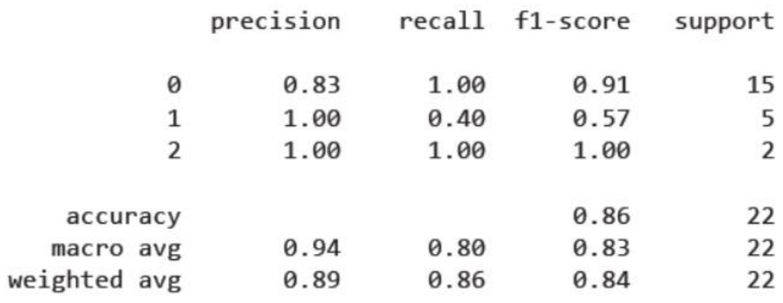


Fig. 16 Confusion matrix for kNN model



```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91        15
           1       1.00      0.40      0.57         5
           2       1.00      1.00      1.00         2

    accuracy                           0.86        22
   macro avg       0.94      0.80      0.83        22
weighted avg       0.89      0.86      0.84        22
```

Fig. 17 Classification report for kNN model

Hyper-parameter optimization was done using GridSearchCV with 3 parameters, whose ranges as specified in the function input are provided in Table 1. It led to the best model score or accuracy of 0.94379 with the corresponding best parameters:-

- Algorithm :-auto
- Number of neighbors :-1
- Leaf size :-1

Table 1 GridSearchCV parameters and their respective ranges for kNN

Parameter	**Parameter syntax**	**Range**
Number of neighbors	n_neighbors	1-49
Algorithm	Algorithm	Auto, ball_tree, kd_tree
Leaf Size	leaf_size	1-49

- **Random Forest**

  It is a powerful ML algorithm which can be used for multi-class classification problem. This approach combines numerous decision trees to create a so-called "forest". For this study the number of estimators was set to 100 and the rest of the parameters were set at a default setting.

  The model accuracy obtained was 100 %. The confusion matrix and classification report are shown in Fig. 18 and Fig. 19 respectively.

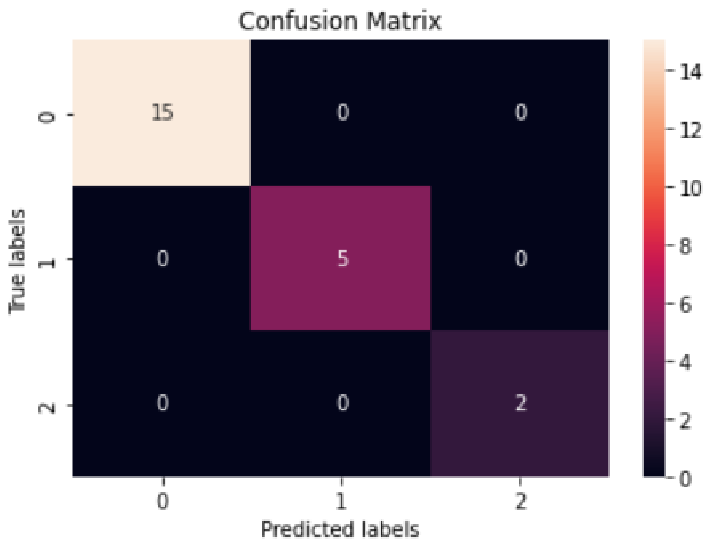


Fig. 18 Confusion matrix for Random Forest model