I have structured the project into 5 key sections. Below, I will share with you the steps I took together with insights for each section.

PART A

1. Data Understanding of Input Dataset
2. Data Preparation/EDA of Input Dataset
3. Modelling/Prediction/Evaluation of Input Dataset 3.1 Building base model as stated in project guide requirements 3.2.Tuning Hyperparameters of CNN 3.3 Tuning Hyperparameters in .compile function 3.4 Adding Data Augmentation 3.5 Overall Model Comparison 3.6 Conclusion

PART B

1. Data Understanding of Cifar-10 colored Dataset
2. Data Preparation/EDA of Cifar-10 colored Dataset
3. Modelling/Prediction/Evaluation of Cifar-10 colored Dataset **Note: Click on the links to go to the respective section**

PART A - Input Dataset

1. Data Understanding of Input Dataset

```
import pandas as pd
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from six.moves import cPickle
import tensorflow as tf
from keras.datasets import cifar10
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
import visualkeras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten,
Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D,
MaxPooling2D,MaxPool2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
```

```
import matplotlib.pyplot as plt
import visualkeras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.image as mpimg
from PIL import ImageFont, Image
#font = ImageFont.truetype("arial.ttf", 12)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import tensorflow as tf
import kerastuner as kt
import cv2
import tensorflow.keras as keras

import warnings
warnings.filterwarnings('ignore')

print("Tensorflow version:",tf.__version__)
print("Keras version:",tf.keras.__version__)
```

```
Tensorflow version: 2.8.0
Keras version: 2.6.0
```

```
C:\Users\JiaYi\AppData\Local\Temp\ipykernel_27920\4204285841.py:32:
DeprecationWarning: `import kerastuner` is deprecated, please use
`import keras_tuner`.
  import kerastuner as kt
```

```
test_batch1=pd.read_pickle("IT3312/test_batch1.pkl")

train_batch1=pd.read_pickle("IT3312/train_batch1.pkl")
train_batch2=pd.read_pickle("IT3312/train_batch2.pkl")
train_batch3=pd.read_pickle("IT3312/train_batch3.pkl")
train_batch4=pd.read_pickle("IT3312/train_batch4.pkl")
train_batch5=pd.read_pickle("IT3312/train_batch5.pkl")

train_data=pd.concat([train_batch1,train_batch2,train_batch3,train_bat
ch4,train_batch5])

test_data=test_batch1

data=pd.concat([train_data,test_data])
```

2. Data Preparation/EDA of Input Dataset

2.2 Split Input Features and Label

```
X_train=train_data.iloc[:,:-1]
y_train=train_data['label']
```

```
X_test=test_data.iloc[:,:-1]
y_test=test_data['label']

X=data.iloc[:,:-1]
y=data['label']
```

2.3 Data Normalization - X_train/X_test

```
X_train/=255
X_test/=255
X/=255

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

(50000, 1024) (50000,)
(10000, 1024) (10000,)

print(X.shape,y.shape)

(60000, 1024) (60000,)
```

2.4 Data Reshaping - X_train/X_test

```
#building the input vector from the 32x32 pixels
X_train = X_train.values.reshape(50000, 32, 32, 1)
X_test = X_test.values.reshape(10000, 32, 32, 1)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

(50000, 32, 32, 1) (50000,)
(10000, 32, 32, 1) (10000,)
```

2.5 Data Encoding - y_train/y_test

Encode target labels with value between 0 and n_classes-1.

```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
y_train=le.fit_transform(y_train)
y_test=le.fit_transform(y_test)
```

3. Modelling, Evaluation and Prediction of Input Dataset

For modelling stage, for each CNN model i will be tuning the hyper parameters, tuning the data(data augmentation) and also tuning the model layers.This will be the overview of this section:

1. Modelling/Prediction/Evaluation 3.1 Building base model as stated in project guide requirements 3.2.Tuning Hyperparameters of CNN 3.2.1 Tuning Conv2D layer 3.2.2 Tuning Dropout layer 3.2.3 Tuning Batch Normalization layer 3.2.4 Tuning Dense layer 3.2.5 Tuning Activation Function 3.3 Tuning Hyperparameters in .compile function 3.3.1 Tuning Optimizer 3.3.2 Tuning Learning Rate 3.3.3 Tuning Batch Size 3.3.4 Tuning Loss Function 3.3.5 Best combinatin of hyperparameters
3.4 Adding Data Augmentation 3.5 Overall Model Comparison

Metrics used: F1-score as for this use case, both recall and precision are important

3.1 CNN-1: CNN base model

A: Building Base Model - CNN-1

The base model will be built based on the project guide requirements as stated in the project guide. I will first be building 3 Convolutional 2D layers followed by Max Pooling 2D layer as it is a great way to reduce the size of parameters with out loosing much information.

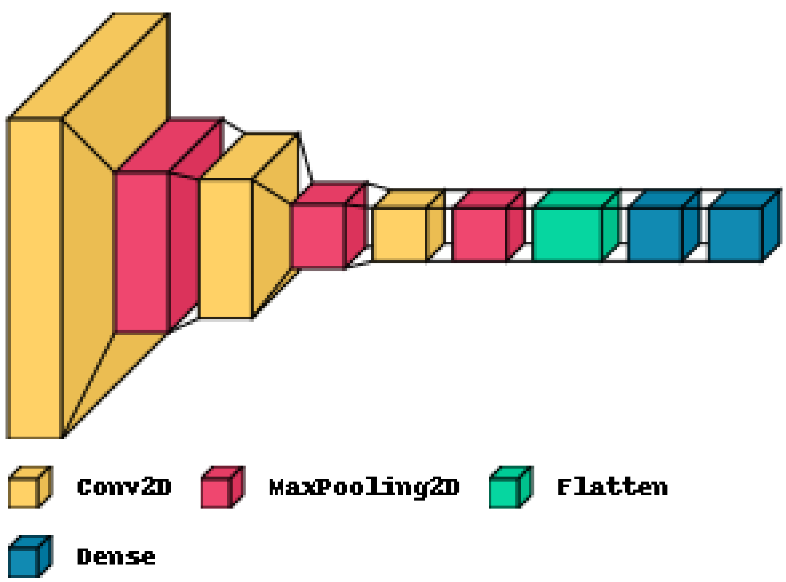
After building 3 consecutive layer, I will then flatten the intermediate layers results and pass them to a Dense network. Then the dense network result will be passes to a final output layer where the number of units represent the number of categories in the data which is 10 in our case. Softmax is chosen as final activation because we need the highest probable class out of 10.

```
from keras.callbacks import EarlyStopping
es_callback = EarlyStopping(monitor='val_loss', mode='min',
patience=5)

def build_CNN_1():
    CNN_1 = Sequential()
    CNN_1.add(Conv2D(32, (3, 3), activation='relu',  input_shape=(32,
32,1)))
    CNN_1.add(MaxPooling2D((2, 2)))
    CNN_1.add(Conv2D(64, (3, 3),  activation='relu'))
    CNN_1.add(MaxPooling2D((2, 2)))
    CNN_1.add(Conv2D(64, (3, 3), activation='relu'))
    CNN_1.add(MaxPooling2D((2, 2)))
    CNN_1.add(Flatten())
    CNN_1.add(Dense(64, activation='relu'))
    CNN_1.add(Dense(10, activation='softmax'))
    return CNN_1
CNN_1=build_CNN_1()
CNN_1.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
```

```
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 72,842
Trainable params: 72,842
Non-trainable params: 0
_________________________________________________________________
```

Next, I have used the visualkeras function to better visualize and view the model.

```
visualkeras.layered_view(CNN_1, legend=True)
```



Now, we will start training the model. Take note that I have used EarlyStopping to train till the epoch that will give the optimum accuracy.

```
CNN_1=build_CNN_1()
CNN_1.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
CNN_1_history = CNN_1.fit(X_train, y_train, epochs=500, batch_size=64,
```

```
verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 11s 4ms/step - loss: 1.8252
- accuracy: 0.3368 - val_loss: 1.5524 - val_accuracy: 0.4539
Epoch 2/500
625/625 [==============================] - 2s 3ms/step - loss: 1.4684
- accuracy: 0.4853 - val_loss: 1.4014 - val_accuracy: 0.5016
Epoch 3/500
625/625 [==============================] - 2s 3ms/step - loss: 1.3188
- accuracy: 0.5443 - val_loss: 1.2558 - val_accuracy: 0.5648
Epoch 4/500
625/625 [==============================] - 2s 3ms/step - loss: 1.2058
- accuracy: 0.5809 - val_loss: 1.2342 - val_accuracy: 0.5725
Epoch 5/500
625/625 [==============================] - 2s 3ms/step - loss: 1.1286
- accuracy: 0.6094 - val_loss: 1.1840 - val_accuracy: 0.5884
Epoch 6/500
625/625 [==============================] - 2s 3ms/step - loss: 1.0761
- accuracy: 0.6277 - val_loss: 1.0928 - val_accuracy: 0.6175
Epoch 7/500
625/625 [==============================] - 2s 3ms/step - loss: 1.0151
- accuracy: 0.6474 - val_loss: 1.0584 - val_accuracy: 0.6356
Epoch 8/500
625/625 [==============================] - 2s 3ms/step - loss: 0.9714
- accuracy: 0.6651 - val_loss: 1.0422 - val_accuracy: 0.6419
Epoch 9/500
625/625 [==============================] - 2s 3ms/step - loss: 0.9348
- accuracy: 0.6755 - val_loss: 1.0336 - val_accuracy: 0.6466
Epoch 10/500
625/625 [==============================] - 2s 3ms/step - loss: 0.8974
- accuracy: 0.6889 - val_loss: 1.0187 - val_accuracy: 0.6496
Epoch 11/500
625/625 [==============================] - 2s 3ms/step - loss: 0.8659
- accuracy: 0.7009 - val_loss: 0.9916 - val_accuracy: 0.6584
Epoch 12/500
625/625 [==============================] - 2s 3ms/step - loss: 0.8330
- accuracy: 0.7137 - val_loss: 1.0326 - val_accuracy: 0.6458
Epoch 13/500
625/625 [==============================] - 2s 3ms/step - loss: 0.8077
- accuracy: 0.7211 - val_loss: 1.0106 - val_accuracy: 0.6598
Epoch 14/500
625/625 [==============================] - 2s 3ms/step - loss: 0.7786
- accuracy: 0.7304 - val_loss: 0.9605 - val_accuracy: 0.6739
Epoch 15/500
625/625 [==============================] - 2s 3ms/step - loss: 0.7529
- accuracy: 0.7389 - val_loss: 0.9701 - val_accuracy: 0.6769
Epoch 16/500
```

```
625/625 [==============================] - 2s 3ms/step - loss: 0.7311
- accuracy: 0.7460 - val_loss: 0.9814 - val_accuracy: 0.6754
Epoch 17/500
625/625 [==============================] - 2s 3ms/step - loss: 0.7122
- accuracy: 0.7537 - val_loss: 1.0160 - val_accuracy: 0.6623
Epoch 18/500
625/625 [==============================] - 2s 3ms/step - loss: 0.6895
- accuracy: 0.7611 - val_loss: 1.0046 - val_accuracy: 0.6733
Epoch 19/500
625/625 [==============================] - 2s 3ms/step - loss: 0.6740
- accuracy: 0.7659 - val_loss: 1.0152 - val_accuracy: 0.6647
```

B: Model Evaluation

```
preds = CNN_1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.69      0.68      0.69      1000
           1       0.81      0.80      0.80      1000
           2       0.57      0.50      0.53      1000
           3       0.45      0.52      0.48      1000
           4       0.53      0.71      0.61      1000
           5       0.73      0.41      0.53      1000
           6       0.69      0.78      0.73      1000
           7       0.74      0.69      0.71      1000
           8       0.75      0.80      0.77      1000
           9       0.81      0.76      0.79      1000

    accuracy                           0.67     10000
   macro avg       0.68      0.67      0.67     10000
weighted avg       0.68      0.67      0.67     10000

Macro F1-score: 0.6652326744791927
```

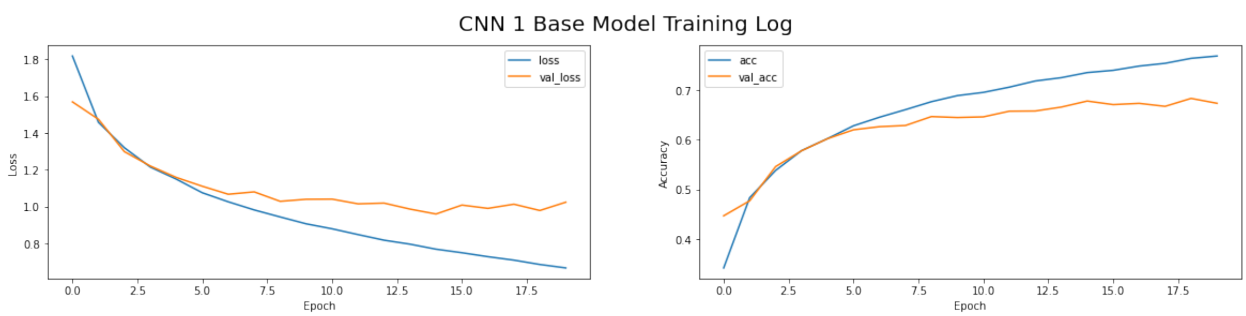
```
accuracy = CNN_1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
```

```
313/313 - 0s - loss: 1.0218 - accuracy: 0.6666
Accuracy: 66.6599988937378
```

As shown in the classification report, the f1-score is 66%, which is not that great but still as we are using a very simple model without any fine tuning. If we perform more tweeks, we can still acheive pretty good accuracy.

```
loss = CNN_1_history.history['loss']
val_loss = CNN_1_history.history['val_loss']
```

```python
acc = CNN_1_history.history['accuracy']
val_acc = CNN_1_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As shown in the model training log, the base model has major underfitting as the validation loss increases as more training is done while the accuracy decreases as there is more training. The validation accuracy decreases and deviates from the training accuracy as the epoch increases. Even though the overfitting is not big and the difference between the val_acc/val_loss and acc/loss is not big, we will still need to fine tune the CNN model, hyperparameters in the neural network and also explore some data augmentation in the later stages.

Conclusion: CNN_1 Base Model F1-Score is 65%

3.2 Tuning Hyperparameters in CNN

In this section, I will be tuning the layers of the base model.It will include these few sections: 3.2.1. Tuning Conv2D layer 3.2.1.1 Adding Conv2D layers 3.2.1.2. Tuning Number of neurons in Conv2d layer

3.2.2.1. Adding Dropout layers 3.2.2.2. Tuning Dropout Rate

3.2.3.1. Adding Batch Normalization layers

3.2.4.1.Tuning Number of neurons in dense layer

3.2.5.1.Tuning Activation Function

3.2.1.1 Adding Conv2d Layer

Now I will be increasing the depth of the model to increase its capacity as those with many hidden layers can be computationally more efficient than training a model that has lesser number of layers with vast number of nodes

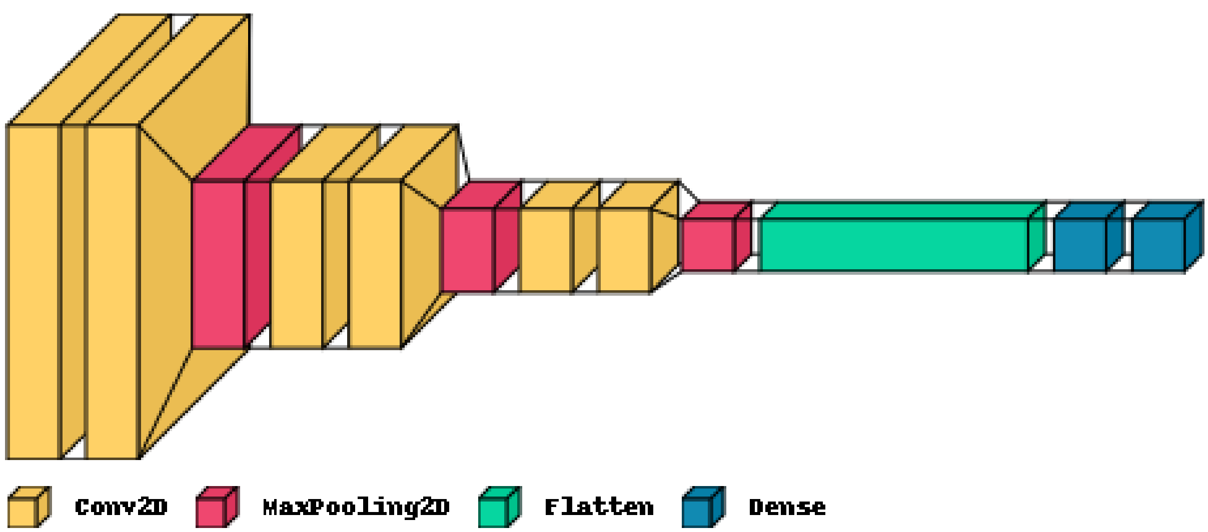
I will be adding one Conv2d layer after each Conv2d layer

3.2.1.1.1 CNN 2: Doubling Conv2d Layer

```
def build_CNN_2():
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
    model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))


    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))


    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))

    model.add(MaxPooling2D(2,2))

    model.add(Flatten())
    model.add(Dense(64,activation='relu'))

    model.add(Dense(10,activation='softmax'))
    return model
CNN_2=build_CNN_2()
CNN_2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
```

```
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 205,098
Trainable params: 205,098
Non-trainable params: 0
_________________________________________________________________

visualkeras.layered_view(CNN_2, legend=True)
```




```
CNN_2=build_CNN_2()
CNN_2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
CNN_2_history = CNN_2.fit(X_train, y_train, epochs=500, batch_size=64,
verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 1.7480
- accuracy: 0.3638 - val_loss: 1.3421 - val_accuracy: 0.5237
Epoch 2/500
625/625 [==============================] - 3s 5ms/step - loss: 1.2018
- accuracy: 0.5798 - val_loss: 1.0556 - val_accuracy: 0.6315
Epoch 3/500
625/625 [==============================] - 3s 5ms/step - loss: 0.9745
- accuracy: 0.6611 - val_loss: 0.9505 - val_accuracy: 0.6720
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8317
- accuracy: 0.7123 - val_loss: 0.8595 - val_accuracy: 0.7018
Epoch 5/500
625/625 [==============================] - 3s 5ms/step - loss: 0.7269
- accuracy: 0.7470 - val_loss: 0.8469 - val_accuracy: 0.7065
Epoch 6/500
625/625 [==============================] - 3s 5ms/step - loss: 0.6459
- accuracy: 0.7767 - val_loss: 0.8018 - val_accuracy: 0.7318
Epoch 7/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5705
- accuracy: 0.8012 - val_loss: 0.8247 - val_accuracy: 0.7291
Epoch 8/500
625/625 [==============================] - 3s 6ms/step - loss: 0.5072
- accuracy: 0.8249 - val_loss: 0.7986 - val_accuracy: 0.7380
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 0.4383
- accuracy: 0.8475 - val_loss: 0.8852 - val_accuracy: 0.7286
Epoch 10/500
625/625 [==============================] - 3s 5ms/step - loss: 0.3874
- accuracy: 0.8649 - val_loss: 0.8952 - val_accuracy: 0.7313
Epoch 11/500
625/625 [==============================] - 4s 6ms/step - loss: 0.3329
- accuracy: 0.8852 - val_loss: 0.9275 - val_accuracy: 0.7292
Epoch 12/500
625/625 [==============================] - 3s 5ms/step - loss: 0.2921
- accuracy: 0.8962 - val_loss: 0.9647 - val_accuracy: 0.7335
Epoch 13/500
625/625 [==============================] - 3s 5ms/step - loss: 0.2506
- accuracy: 0.9112 - val_loss: 1.0102 - val_accuracy: 0.7353
```

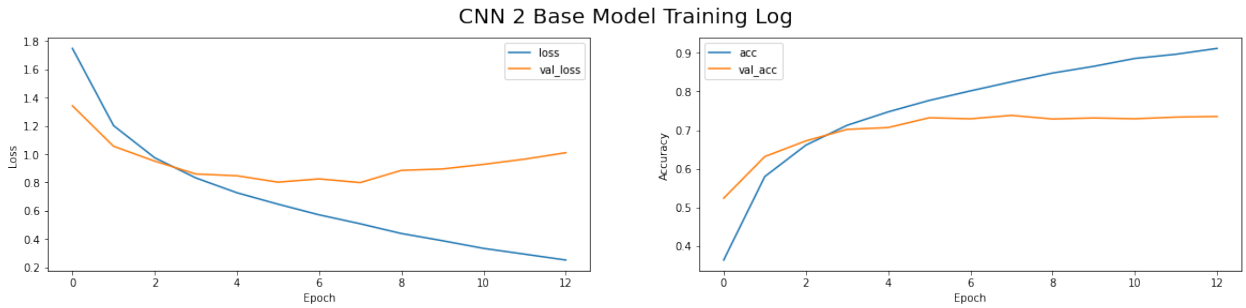
```python
preds = CNN_2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.75      0.74      0.74      1000
```

```
           1       0.85      0.86      0.85      1000
           2       0.54      0.64      0.59      1000
           3       0.54      0.55      0.54      1000
           4       0.67      0.71      0.69      1000
           5       0.72      0.56      0.63      1000
           6       0.75      0.78      0.76      1000
           7       0.81      0.72      0.76      1000
           8       0.81      0.86      0.83      1000
           9       0.84      0.81      0.83      1000

    accuracy                           0.72     10000
   macro avg       0.73      0.72      0.72     10000
weighted avg       0.73      0.72      0.72     10000

313/313 - 1s - loss: 1.0749 - accuracy: 0.7228
Accuracy: 72.28000164031982
Macro F1-score: 0.7231424691248046
```

We can see that by adding double layer of Conv2d, the accuracy has increased to 72% from the original 65% which is a good sign

```
loss = CNN_2_history.history['loss']
val_loss = CNN_2_history.history['val_loss']
acc = CNN_2_history.history['accuracy']
val_acc = CNN_2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 2 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



However, as shown in the model training log, the CNN 2 Version 2 model has increased underfitting as the validation loss increases as more training is done while the accuracy decreases as there is more training. The validation accuracy decreases and deviates from the training accuracy as the epoch increases. We will still need to fine tune the CNN model in the the later section (Dropout section)

Let's compare the summary of all model results:

3.2.1.2 Tuning Number of Neurons for Conv2d Layer

Now after I have added Conv2d layers, I will be tuning the neurons combination for each double layer. Currently the combination of the number of neurons is 32, 64, 64. I will be trying all combinations of neurons number from 32,64,128,256,512 for each pair of layer and see which combinations will be best optimum f1-score.

I will first be building a CNN_3 model building function as it will be easier to call out the function for each neuron combinations and fit in the values

```
def build_CNN_3(n1,n2,n3):
    model=Sequential()
    model.add(Conv2D(n1,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
    model.add(Conv2D(n1,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))


    model.add(Conv2D(n2,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(n2,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))


    model.add(Conv2D(n3,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(n3,(3,3),activation="relu", padding='same'))

    model.add(MaxPooling2D(2,2))

    model.add(Flatten())
    model.add(Dense(64,activation='relu'))
```

```
    model.add(Dense(10,activation='softmax'))
    return model
```

Neurons Combination :

1. 16 32 64
2. 32 64 128
3. 64 128 256
4. 128 256 512

3.2.1.2.1 CNN 3 v0: Neurons Combination 1: 16,32,64

```
CNN_3_v0=build_CNN_3(16,32,64)
CNN_3_v0.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 32, 32, 16)        160
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 16)        2320
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 32)        4640
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 16, 16, 32)        9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650
=================================================================
Total params: 138,042
Trainable params: 138,042
Non-trainable params: 0
_________________________________________________________________
```

```
CNN_3_v0.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_3_v0_history = CNN_3_v0.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 7s 4ms/step - loss: 1.7392
- accuracy: 0.3632 - val_loss: 1.4069 - val_accuracy: 0.4968
Epoch 2/500
625/625 [==============================] - 2s 4ms/step - loss: 1.3169
- accuracy: 0.5323 - val_loss: 1.1722 - val_accuracy: 0.5860
Epoch 3/500
625/625 [==============================] - 2s 4ms/step - loss: 1.1238
- accuracy: 0.6067 - val_loss: 1.0549 - val_accuracy: 0.6335
Epoch 4/500
625/625 [==============================] - 2s 4ms/step - loss: 0.9800
- accuracy: 0.6568 - val_loss: 0.9601 - val_accuracy: 0.6636
Epoch 5/500
625/625 [==============================] - 2s 4ms/step - loss: 0.8749
- accuracy: 0.6952 - val_loss: 0.9135 - val_accuracy: 0.6829
Epoch 6/500
625/625 [==============================] - 2s 4ms/step - loss: 0.7869
- accuracy: 0.7244 - val_loss: 0.9177 - val_accuracy: 0.6846
Epoch 7/500
625/625 [==============================] - 2s 4ms/step - loss: 0.7179
- accuracy: 0.7494 - val_loss: 0.8370 - val_accuracy: 0.7114
Epoch 8/500
625/625 [==============================] - 2s 4ms/step - loss: 0.6509
- accuracy: 0.7736 - val_loss: 0.8184 - val_accuracy: 0.7224
Epoch 9/500
625/625 [==============================] - 2s 4ms/step - loss: 0.6001
- accuracy: 0.7907 - val_loss: 0.8374 - val_accuracy: 0.7189
Epoch 10/500
625/625 [==============================] - 2s 4ms/step - loss: 0.5377
- accuracy: 0.8110 - val_loss: 0.8267 - val_accuracy: 0.7274
Epoch 11/500
625/625 [==============================] - 3s 4ms/step - loss: 0.4902
- accuracy: 0.8260 - val_loss: 0.8441 - val_accuracy: 0.7291
Epoch 12/500
625/625 [==============================] - 3s 5ms/step - loss: 0.4374
- accuracy: 0.8467 - val_loss: 0.9176 - val_accuracy: 0.7197
Epoch 13/500
625/625 [==============================] - 3s 5ms/step - loss: 0.3998
- accuracy: 0.8575 - val_loss: 0.9382 - val_accuracy: 0.7175
```

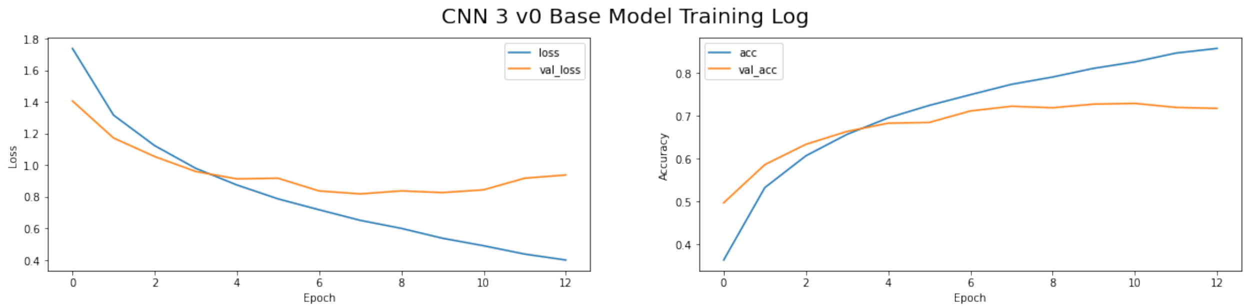
```
preds = CNN_3_v0.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_3_v0.evaluate(X_test, y_test, verbose=2)
```

```
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.73      0.80      0.76      1000
           1       0.79      0.89      0.83      1000
           2       0.75      0.45      0.56      1000
           3       0.53      0.50      0.51      1000
           4       0.69      0.59      0.64      1000
           5       0.52      0.72      0.61      1000
           6       0.70      0.81      0.75      1000
           7       0.79      0.74      0.77      1000
           8       0.84      0.79      0.82      1000
           9       0.81      0.81      0.81      1000

    accuracy                           0.71     10000
   macro avg       0.72      0.71      0.71     10000
weighted avg       0.72      0.71      0.71     10000

313/313 - 1s - loss: 0.9874 - accuracy: 0.7089
Accuracy: 70.8899974822998
Macro F1-score: 0.7057414029372829
```

This neurons combination caused a decreased in perfromance as the f1- score dropped from the original 72% to 70% as lesser neurons is used from the previous CNN2 model

```
loss = CNN_3_v0_history.history['loss']
val_loss = CNN_3_v0_history.history['val_loss']
acc = CNN_3_v0_history.history['accuracy']
val_acc = CNN_3_v0_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 3 v0 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As shown in the model training log, the CNN 3 Version 0 model has soem overfitting at the start followed by major undefitting as the validation loss increases as more training is done while the accuracy decreases as there is more training. The validation accuracy decreases and deviates from the training accuracy as the epoch increases. We will still need to fine tune the CNN model in the the later section (Dropout section). Hence, this model is not good performing. Now I will be increasing the neurons in the next section.

3.2.1.2.2 CNN 3 v1 Neurons Combination 1: 32, 64, 128

```
CNN_3_v1=build_CNN_3(32,64,128)
CNN_3_v1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_24 (Conv2D)           (None, 32, 32, 32)        320       
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 64)                131136    
```

```
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0
_________________________________________________________________
```

```
CNN_3_v1.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_3_v1_history = CNN_3_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 1.7361
- accuracy: 0.3618 - val_loss: 1.3122 - val_accuracy: 0.5379
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1809
- accuracy: 0.5839 - val_loss: 1.0435 - val_accuracy: 0.6373
Epoch 3/500
625/625 [==============================] - 3s 5ms/step - loss: 0.9325
- accuracy: 0.6755 - val_loss: 0.9018 - val_accuracy: 0.6924
Epoch 4/500
625/625 [==============================] - 3s 6ms/step - loss: 0.7682
- accuracy: 0.7352 - val_loss: 0.8239 - val_accuracy: 0.7141
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6458
- accuracy: 0.7766 - val_loss: 0.7952 - val_accuracy: 0.7296
Epoch 6/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5433
- accuracy: 0.8106 - val_loss: 0.7761 - val_accuracy: 0.7482
Epoch 7/500
625/625 [==============================] - 3s 6ms/step - loss: 0.4491
- accuracy: 0.8439 - val_loss: 0.7929 - val_accuracy: 0.7480
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 0.3641
- accuracy: 0.8726 - val_loss: 0.8670 - val_accuracy: 0.7440
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 0.2949
- accuracy: 0.8964 - val_loss: 0.9149 - val_accuracy: 0.7427
Epoch 10/500
625/625 [==============================] - 4s 6ms/step - loss: 0.2324
- accuracy: 0.9180 - val_loss: 1.0043 - val_accuracy: 0.7413
Epoch 11/500
625/625 [==============================] - 3s 6ms/step - loss: 0.1897
- accuracy: 0.9322 - val_loss: 1.0662 - val_accuracy: 0.7485
```

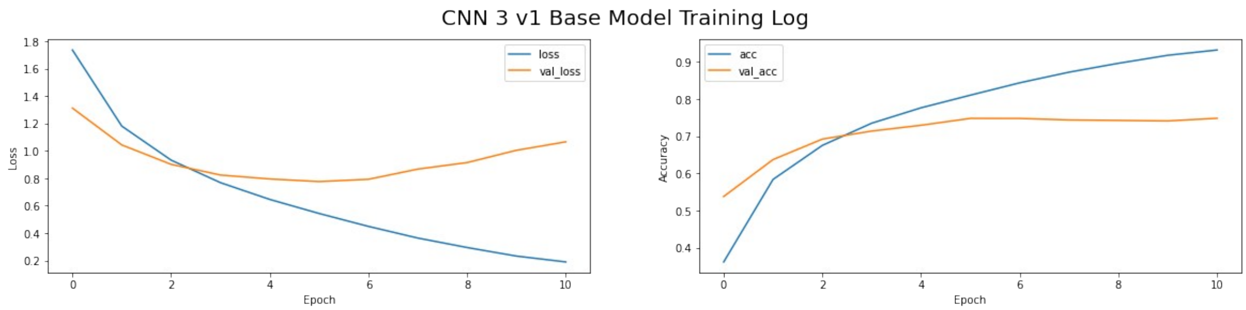
```
preds = CNN_3_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_3_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      1000
           1       0.84      0.89      0.86      1000
           2       0.61      0.64      0.62      1000
           3       0.55      0.51      0.53      1000
           4       0.69      0.71      0.70      1000
           5       0.68      0.62      0.65      1000
           6       0.77      0.78      0.78      1000
           7       0.75      0.80      0.77      1000
           8       0.82      0.84      0.83      1000
           9       0.84      0.81      0.83      1000

    accuracy                           0.73     10000
   macro avg       0.73      0.73      0.73     10000
weighted avg       0.73      0.73      0.73     10000

313/313 - 1s - loss: 1.1322 - accuracy: 0.7348
Accuracy: 73.47999811172485
Macro F1-score: 0.7335395749600165
```

As the neurons increased to 32, 64, 128 from the original combination of 32, 64,64, the doubled neurons for the last layer has increased the f1-sscore from 72% to 73%. It is a good sign, hence, we will keep on adding the neurons in the next stage.

```
loss = CNN_3_v1_history.history['loss']
val_loss = CNN_3_v1_history.history['val_loss']
acc = CNN_3_v1_history.history['accuracy']
val_acc = CNN_3_v1_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 3 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As the neurons combination increased, it also caused underfitting to occur at an earlier stage when epoch was only 2. Hence we will need to fin tune the CNN model in the later section(Dropout layer). For now, we will focus on improving model performance first.

3.2.1.2.2 Neurons Combination 2: 64, 128, 256

```
CNN_3_v2=build_CNN_3(64,128,256)
CNN_3_v2.summary()

Model: "sequential_5"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_30 (Conv2D)          (None, 32, 32, 64)        640
_________________________________________________________________
conv2d_31 (Conv2D)          (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 16, 64)       0
_________________________________________________________________
conv2d_32 (Conv2D)          (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_33 (Conv2D)          (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 8, 8, 128)        0
_________________________________________________________________
conv2d_34 (Conv2D)          (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_35 (Conv2D)          (None, 8, 8, 256)         590080
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 4, 4, 256)        0
_________________________________________________________________
flatten_5 (Flatten)         (None, 4096)              0
```

```
_________________________________________________________________
dense_10 (Dense)             (None, 64)                262208
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 1,407,114
Trainable params: 1,407,114
Non-trainable params: 0
_________________________________________________________________
```

```
CNN_3_v2.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_3_v2_history = CNN_3_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 9s 12ms/step - loss: 1.8604
- accuracy: 0.3094 - val_loss: 1.5402 - val_accuracy: 0.4461
Epoch 2/500
625/625 [==============================] - 8s 12ms/step - loss: 1.3052
- accuracy: 0.5364 - val_loss: 1.1063 - val_accuracy: 0.6070
Epoch 3/500
625/625 [==============================] - 7s 12ms/step - loss: 1.0060
- accuracy: 0.6499 - val_loss: 0.9400 - val_accuracy: 0.6787
Epoch 4/500
625/625 [==============================] - 7s 12ms/step - loss: 0.8299
- accuracy: 0.7117 - val_loss: 0.8446 - val_accuracy: 0.7107
Epoch 5/500
625/625 [==============================] - 7s 12ms/step - loss: 0.6920
- accuracy: 0.7592 - val_loss: 0.7716 - val_accuracy: 0.7433
Epoch 6/500
625/625 [==============================] - 7s 12ms/step - loss: 0.5864
- accuracy: 0.7977 - val_loss: 0.7676 - val_accuracy: 0.7398
Epoch 7/500
625/625 [==============================] - 7s 12ms/step - loss: 0.4791
- accuracy: 0.8329 - val_loss: 0.7830 - val_accuracy: 0.7419
Epoch 8/500
625/625 [==============================] - 7s 12ms/step - loss: 0.3841
- accuracy: 0.8662 - val_loss: 0.8339 - val_accuracy: 0.7435
Epoch 9/500
625/625 [==============================] - 7s 12ms/step - loss: 0.3058
- accuracy: 0.8927 - val_loss: 0.8979 - val_accuracy: 0.7453
Epoch 10/500
625/625 [==============================] - 7s 12ms/step - loss: 0.2386
- accuracy: 0.9153 - val_loss: 1.0776 - val_accuracy: 0.7259
Epoch 11/500
625/625 [==============================] - 7s 12ms/step - loss: 0.1970
- accuracy: 0.9311 - val_loss: 1.0744 - val_accuracy: 0.7411
```

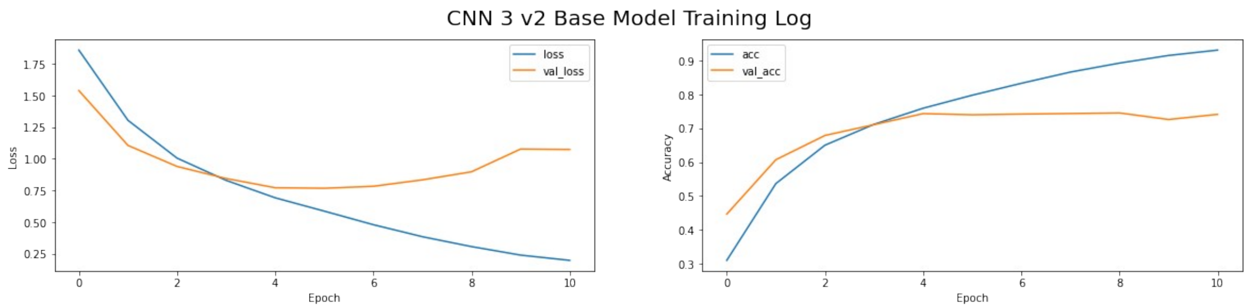
```
preds = CNN_3_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_3_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.77      0.74      0.75      1000
           1       0.86      0.86      0.86      1000
           2       0.62      0.64      0.63      1000
           3       0.56      0.50      0.53      1000
           4       0.71      0.64      0.67      1000
           5       0.66      0.66      0.66      1000
           6       0.76      0.81      0.78      1000
           7       0.77      0.78      0.78      1000
           8       0.86      0.81      0.84      1000
           9       0.74      0.88      0.80      1000

    accuracy                           0.73     10000
   macro avg       0.73      0.73      0.73     10000
weighted avg       0.73      0.73      0.73     10000

313/313 - 1s - loss: 1.1151 - accuracy: 0.7318
Accuracy: 73.18000197410583
Macro F1-score: 0.7300428602646897
```

We can see that as the neurons increased to 64, 128, 256 the model perfromance did not increase at all. It remained stagnant at 73%. Hence, we will be using the previous model as it gives the same performance with lesser parameters

```
loss = CNN_3_v2_history.history['loss']
val_loss = CNN_3_v2_history.history['val_loss']
acc = CNN_3_v2_history.history['accuracy']
val_acc = CNN_3_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 3 v2 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As the neurons combination increased, it also caused underfitting to occur at an earlier stage when epoch was only 2. Hence we will need to fin tune the CNN model in the later section(Dropout layer). For now, we will focus on improving model performance first.

3.2.1.2.3 Neurons Combination 3: 128, 256, 512

```
CNN_3_v3=build_CNN_3(128,256,512)
CNN_3_v3.summary()

Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 32, 32, 128)       1280
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 16, 16, 256)       295168
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 16, 16, 256)       590080
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 8, 8, 512)         1180160
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 4, 4, 512)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                524352
```

```
_________________________________________________________________
dense_13 (Dense)             (None, 10)                650
=================================================================
Total params: 5,099,082
Trainable params: 5,099,082
Non-trainable params: 0
_________________________________________________________________
```

```
CNN_3_v3.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_3_v3_history = CNN_3_v3.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 19s 28ms/step - loss:
1.8680 - accuracy: 0.3047 - val_loss: 1.4509 - val_accuracy: 0.4719
Epoch 2/500
625/625 [==============================] - 17s 27ms/step - loss:
1.2655 - accuracy: 0.5493 - val_loss: 1.0410 - val_accuracy: 0.6349
Epoch 3/500
625/625 [==============================] - 17s 27ms/step - loss:
0.9382 - accuracy: 0.6729 - val_loss: 0.8611 - val_accuracy: 0.7006
Epoch 4/500
625/625 [==============================] - 17s 27ms/step - loss:
0.7537 - accuracy: 0.7377 - val_loss: 0.8015 - val_accuracy: 0.7225
Epoch 5/500
625/625 [==============================] - 17s 28ms/step - loss:
0.6188 - accuracy: 0.7857 - val_loss: 0.7768 - val_accuracy: 0.7412
Epoch 6/500
625/625 [==============================] - 17s 28ms/step - loss:
0.4998 - accuracy: 0.8250 - val_loss: 0.7234 - val_accuracy: 0.7622
Epoch 7/500
625/625 [==============================] - 18s 29ms/step - loss:
0.3857 - accuracy: 0.8653 - val_loss: 0.8323 - val_accuracy: 0.7456
Epoch 8/500
625/625 [==============================] - 18s 29ms/step - loss:
0.2891 - accuracy: 0.8992 - val_loss: 0.8812 - val_accuracy: 0.7457
Epoch 9/500
625/625 [==============================] - 18s 29ms/step - loss:
0.2165 - accuracy: 0.9241 - val_loss: 0.9270 - val_accuracy: 0.7467
Epoch 10/500
625/625 [==============================] - 18s 29ms/step - loss:
0.1543 - accuracy: 0.9458 - val_loss: 1.0772 - val_accuracy: 0.7473
Epoch 11/500
625/625 [==============================] - 18s 29ms/step - loss:
0.1303 - accuracy: 0.9546 - val_loss: 1.1687 - val_accuracy: 0.7473
```

```
preds = CNN_3_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_3_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

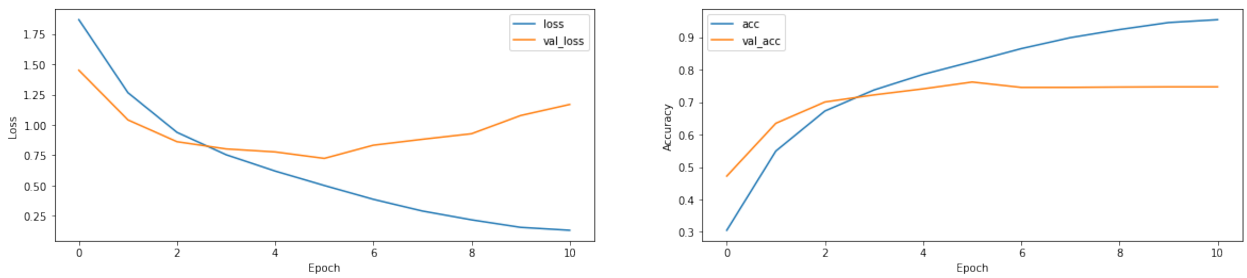
```
              precision    recall  f1-score   support

           0       0.73      0.80      0.76      1000
           1       0.84      0.86      0.85      1000
           2       0.61      0.65      0.63      1000
           3       0.57      0.52      0.54      1000
           4       0.67      0.70      0.68      1000
           5       0.63      0.67      0.65      1000
           6       0.81      0.77      0.79      1000
           7       0.88      0.73      0.80      1000
           8       0.86      0.83      0.85      1000
           9       0.84      0.84      0.84      1000

    accuracy                           0.74     10000
   macro avg       0.74      0.74      0.74     10000
weighted avg       0.74      0.74      0.74     10000

313/313 - 2s - loss: 1.2324 - accuracy: 0.7389
Accuracy: 73.89000058174133
Macro F1-score: 0.7394277856365191
```

We can see that with a higher number of neurons, the model f1-score did increased a little by 0.9%. As the parameters is reaching 5 million, I would be choosing the CNN 3 v1 model as it gives roughly the same result and uses less paramaters to train

```
loss = CNN_3_v3_history.history['loss']
val_loss = CNN_3_v3_history.history['val_loss']
acc = CNN_3_v3_history.history['accuracy']
val_acc = CNN_3_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 3 v3 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

CNN 3 v3 Base Model Training Log



As the neurons combination increased, it also caused underfitting to occur at an earlier stage when epoch was only 2. Hence we will need to fin tune the CNN model in the later section(Dropout layer). For now, we will focus on improving model performance first.

Let's compare the summary of all model results:

I will be choosing CNN 3 v1 as it gives a increased model performance with 400,000 paramaters as compared to CNN 3v2 which has 5 million paramaters. The trade off for results to paramaters is not worth it. Hence I will be looking for other ways to improve the model performance in the later stages

Conclusion: CNN 3 v1 F1-Score is 73%

To allow for better model training(no over or underfitting), I will be adding some regularization techniques which in this case, adding dropout layer. Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

3.2.2.1 CNN 4 v1: Adding Dropout Layer

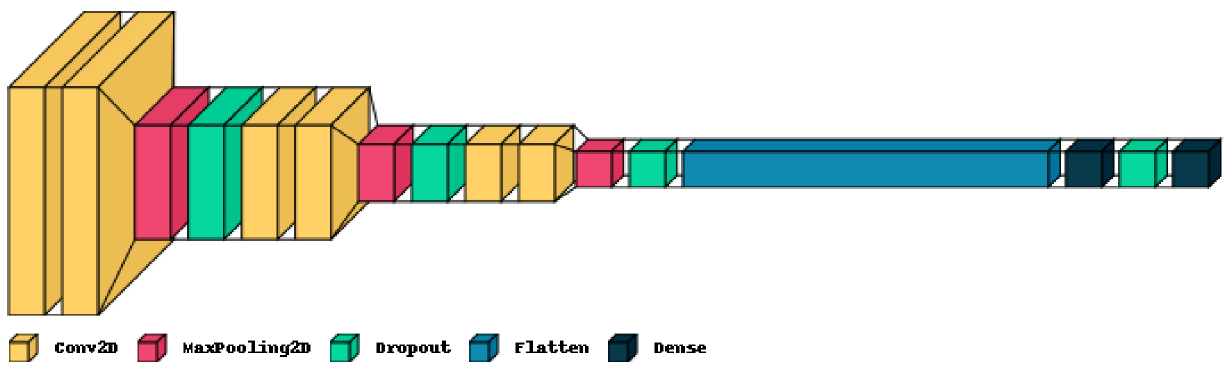
As Dropout rate is between 0 to 1, o have no bias, I will setting the first dropout rate to 0.5 which is a middle point between 0 and 1. I will be adding the Dropout layer to under every max pooling layer to create a significant difference. If I add the dropout layer before the max pooling layer, the number of dead neurons would be greater than or equal to that if I add the dropout layer after the max pooling layer.

```
CNN_4_v1=Sequential()
CNN_4_v1.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
CNN_4_v1.add(Conv2D(32,(3,3),activation="relu", padding='same'))
```

```
CNN_4_v1.add(MaxPooling2D(2,2))
CNN_4_v1.add(Dropout(0.5))

CNN_4_v1.add(Conv2D(64,(3,3),activation="relu", padding='same'))
CNN_4_v1.add(Conv2D(64,(3,3),activation="relu", padding='same'))
CNN_4_v1.add(MaxPooling2D(2,2))
CNN_4_v1.add(Dropout(0.5))


CNN_4_v1.add(Conv2D(128,(3,3),activation="relu", padding='same'))
CNN_4_v1.add(Conv2D(128,(3,3),activation="relu", padding='same'))
CNN_4_v1.add(MaxPooling2D(2,2))
CNN_4_v1.add(Dropout(0.5))

CNN_4_v1.add(Flatten())
CNN_4_v1.add(Dense(64,activation='relu'))
CNN_4_v1.add(Dropout(0.5))
CNN_4_v1.add(Dense(10,activation='softmax'))
CNN_4_v1.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 2048)              0
_________________________________________________________________
```

```
dense_12 (Dense)             (None, 64)                131136

_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0

_________________________________________________________________
dense_13 (Dense)             (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0

_________________________________________________________________

visualkeras.layered_view(CNN_4_v1, legend=True)
```




```
CNN_4_v1.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_4_v1_history = CNN_4_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 5s 6ms/step - loss: 2.0593
- accuracy: 0.2255 - val_loss: 1.7895 - val_accuracy: 0.3412
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.7311
- accuracy: 0.3619 - val_loss: 1.6034 - val_accuracy: 0.4261
Epoch 3/500
625/625 [==============================] - 4s 6ms/step - loss: 1.5640
- accuracy: 0.4316 - val_loss: 1.4059 - val_accuracy: 0.5012
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 1.4587
- accuracy: 0.4832 - val_loss: 1.3700 - val_accuracy: 0.5094
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 1.3734
- accuracy: 0.5158 - val_loss: 1.2542 - val_accuracy: 0.5620
Epoch 6/500
625/625 [==============================] - 4s 6ms/step - loss: 1.3032
- accuracy: 0.5429 - val_loss: 1.0441 - val_accuracy: 0.6301
```

```
Epoch 7/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2418
- accuracy: 0.5679 - val_loss: 1.0851 - val_accuracy: 0.6176
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2021
- accuracy: 0.5835 - val_loss: 1.0148 - val_accuracy: 0.6462
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1590
- accuracy: 0.6023 - val_loss: 1.0461 - val_accuracy: 0.6349
Epoch 10/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1282
- accuracy: 0.6136 - val_loss: 0.9509 - val_accuracy: 0.6653
Epoch 11/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1014
- accuracy: 0.6200 - val_loss: 0.8856 - val_accuracy: 0.6890
Epoch 12/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0752
- accuracy: 0.6334 - val_loss: 0.9327 - val_accuracy: 0.6725
Epoch 13/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0592
- accuracy: 0.6356 - val_loss: 0.8823 - val_accuracy: 0.6894
Epoch 14/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0496
- accuracy: 0.6403 - val_loss: 0.8235 - val_accuracy: 0.7152
Epoch 15/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0262
- accuracy: 0.6507 - val_loss: 0.8294 - val_accuracy: 0.7102
Epoch 16/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0090
- accuracy: 0.6554 - val_loss: 0.8348 - val_accuracy: 0.7097
Epoch 17/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0098
- accuracy: 0.6557 - val_loss: 0.7956 - val_accuracy: 0.7268
Epoch 18/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9874
- accuracy: 0.6636 - val_loss: 0.7943 - val_accuracy: 0.7237
Epoch 19/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9813
- accuracy: 0.6656 - val_loss: 0.8573 - val_accuracy: 0.7072
Epoch 20/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9654
- accuracy: 0.6696 - val_loss: 0.7940 - val_accuracy: 0.7308
Epoch 21/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9569
- accuracy: 0.6734 - val_loss: 0.8043 - val_accuracy: 0.7211
Epoch 22/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9448
- accuracy: 0.6795 - val_loss: 0.7572 - val_accuracy: 0.7389
Epoch 23/500
```

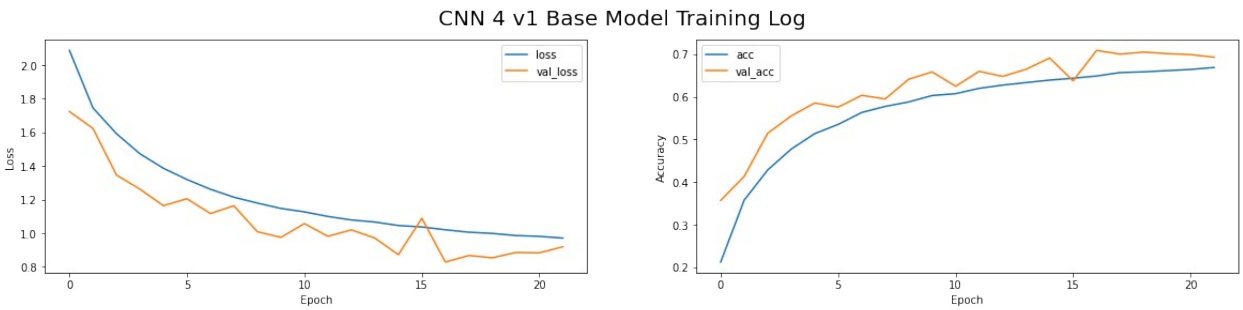
```
625/625 [==============================] - 4s 6ms/step - loss: 0.9377
- accuracy: 0.6796 - val_loss: 0.7490 - val_accuracy: 0.7436
Epoch 24/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9259
- accuracy: 0.6886 - val_loss: 0.7871 - val_accuracy: 0.7303
Epoch 25/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9223
- accuracy: 0.6847 - val_loss: 0.7684 - val_accuracy: 0.7376
Epoch 26/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9165
- accuracy: 0.6885 - val_loss: 0.7563 - val_accuracy: 0.7402
Epoch 27/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9023
- accuracy: 0.6908 - val_loss: 0.7270 - val_accuracy: 0.7526
Epoch 28/500
625/625 [==============================] - 4s 7ms/step - loss: 0.9035
- accuracy: 0.6924 - val_loss: 0.7383 - val_accuracy: 0.7496
Epoch 29/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8933
- accuracy: 0.6962 - val_loss: 0.7643 - val_accuracy: 0.7361
Epoch 30/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8857
- accuracy: 0.6975 - val_loss: 0.7210 - val_accuracy: 0.7518
Epoch 31/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8832
- accuracy: 0.7013 - val_loss: 0.7195 - val_accuracy: 0.7549
Epoch 32/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8728
- accuracy: 0.7035 - val_loss: 0.7171 - val_accuracy: 0.7520
Epoch 33/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8753
- accuracy: 0.7053 - val_loss: 0.7570 - val_accuracy: 0.7406
Epoch 34/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8582
- accuracy: 0.7107 - val_loss: 0.7189 - val_accuracy: 0.7507
Epoch 35/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8528
- accuracy: 0.7092 - val_loss: 0.7729 - val_accuracy: 0.7389
Epoch 36/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8587
- accuracy: 0.7082 - val_loss: 0.6875 - val_accuracy: 0.7667
Epoch 37/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8510
- accuracy: 0.7103 - val_loss: 0.7497 - val_accuracy: 0.7447
Epoch 38/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8459
- accuracy: 0.7109 - val_loss: 0.7612 - val_accuracy: 0.7439
Epoch 39/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8388
```

```
- accuracy: 0.7149 - val_loss: 0.7171 - val_accuracy: 0.7528
Epoch 40/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8346
- accuracy: 0.7192 - val_loss: 0.7106 - val_accuracy: 0.7557
Epoch 41/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8386
- accuracy: 0.7167 - val_loss: 0.7055 - val_accuracy: 0.7645

preds = CNN_4_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_4_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))

              precision    recall  f1-score   support

           0       0.84      0.64      0.72      1000
           1       0.91      0.79      0.85      1000
           2       0.70      0.39      0.50      1000
           3       0.44      0.60      0.51      1000
           4       0.61      0.66      0.63      1000
           5       0.55      0.64      0.59      1000
           6       0.55      0.89      0.68      1000
           7       0.86      0.62      0.72      1000
           8       0.91      0.74      0.82      1000
           9       0.79      0.84      0.82      1000

    accuracy                           0.68     10000
   macro avg       0.72      0.68      0.68     10000
weighted avg       0.72      0.68      0.68     10000

313/313 - 1s - loss: 0.9503 - accuracy: 0.6805
Accuracy: 68.04999709129333
Macro F1-score: 0.683772790779732
```

We can see that the Dropout layer have decreased the f1 score to 68%. Hence, I will need to tune the dropout layer value in the later section

```
loss = CNN_4_v1_history.history['loss']
val_loss = CNN_4_v1_history.history['val_loss']
acc = CNN_4_v1_history.history['accuracy']
val_acc = CNN_4_v1_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 4 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



We can see that even though there is still overfitting, the deviation of the validation loss and the train loss have closed up! Even though the model perfromance has dropped slightly, the problem of overfitting/underfitting has been reduced greatly as the val_loss and val_acc are deviating towards the training accuracy and loss. This is a good sign. Hence, we will keep the dropout layers.In the next section, to close up the deviation we will be tuning the dropout layer value

Let's compare the summary of all model results:

3.2.2.3 Tuning Dropout Layer

As the model performance has dropped in the previous section, we will need to find the optimum dropout rate that gives the highest f1-score. I have done a function where it will be easier to call the function and fit in different Dropout rates

```
def build_CNN_4(d1,d2,d3,d4):
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
    model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(d1))

    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(d1))
```

```
    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(d1))

    model.add(Flatten())
    model.add(Dense(64,activation='relu'))
    model.add(Dropout(d1))
    model.add(Dense(10,activation='softmax'))
    return model
```

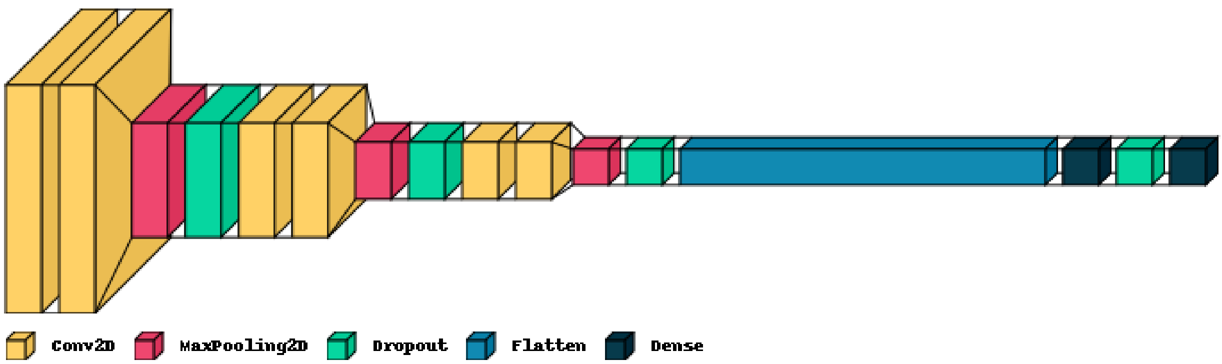
3.2.2.3.1 CNN 4 v2: Ascending Dropout Layer Combination - [0.2, 0.3, 0.4, 0.5]

First, I will be trying out ascending dropout layer combiantion that starts from 0.2 has we do not want too heavy dropout that misses out on the data

```
CNN_4_v2=build_CNN_4(0.2, 0.3, 0.4, 0.5)
CNN_4_v2.summary()

Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_90 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_91 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_30 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_92 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_93 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_31 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
```

```
dropout_32 (Dropout)         (None, 4, 4, 128)         0

_________________________________________________________________
flatten_14 (Flatten)         (None, 2048)              0

_________________________________________________________________
dense_28 (Dense)             (None, 64)                131136

_________________________________________________________________
dropout_33 (Dropout)         (None, 64)                0

_________________________________________________________________
dense_29 (Dense)             (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0

_________________________________________________________________
```

```
visualkeras.layered_view(CNN_4_v2, legend=True)
```




```
CNN_4_v2.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_4_v2_history = CNN_4_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 2.0909
- accuracy: 0.2130 - val_loss: 1.6676 - val_accuracy: 0.4028
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.6157
- accuracy: 0.4137 - val_loss: 1.3243 - val_accuracy: 0.5163
Epoch 3/500
625/625 [==============================] - 4s 6ms/step - loss: 1.3982
- accuracy: 0.5046 - val_loss: 1.1625 - val_accuracy: 0.5842
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2647
- accuracy: 0.5584 - val_loss: 1.0230 - val_accuracy: 0.6400
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1704
```

```
- accuracy: 0.5932 - val_loss: 0.9450 - val_accuracy: 0.6674
Epoch 6/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1035
- accuracy: 0.6206 - val_loss: 0.9031 - val_accuracy: 0.6817
Epoch 7/500
625/625 [==============================] - 4s 7ms/step - loss: 1.0385
- accuracy: 0.6386 - val_loss: 0.8490 - val_accuracy: 0.7052
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9947
- accuracy: 0.6558 - val_loss: 0.8188 - val_accuracy: 0.7126
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9464
- accuracy: 0.6705 - val_loss: 0.7920 - val_accuracy: 0.7195
Epoch 10/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9192
- accuracy: 0.6880 - val_loss: 0.8074 - val_accuracy: 0.7204
Epoch 11/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8909
- accuracy: 0.6968 - val_loss: 0.7682 - val_accuracy: 0.7352
Epoch 12/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8674
- accuracy: 0.7028 - val_loss: 0.7443 - val_accuracy: 0.7404
Epoch 13/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8437
- accuracy: 0.7132 - val_loss: 0.7436 - val_accuracy: 0.7463
Epoch 14/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8267
- accuracy: 0.7200 - val_loss: 0.7658 - val_accuracy: 0.7383
Epoch 15/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7994
- accuracy: 0.7282 - val_loss: 0.6934 - val_accuracy: 0.7616
Epoch 16/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7850
- accuracy: 0.7318 - val_loss: 0.7360 - val_accuracy: 0.7493
Epoch 17/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7673
- accuracy: 0.7369 - val_loss: 0.6943 - val_accuracy: 0.7586
Epoch 18/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7496
- accuracy: 0.7436 - val_loss: 0.6949 - val_accuracy: 0.7608
Epoch 19/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7440
- accuracy: 0.7490 - val_loss: 0.6979 - val_accuracy: 0.7584
Epoch 20/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7312
- accuracy: 0.7494 - val_loss: 0.6440 - val_accuracy: 0.7806
Epoch 21/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7187
- accuracy: 0.7522 - val_loss: 0.6792 - val_accuracy: 0.7639
```

```
Epoch 22/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6944
- accuracy: 0.7618 - val_loss: 0.6405 - val_accuracy: 0.7821
Epoch 23/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6949
- accuracy: 0.7618 - val_loss: 0.6495 - val_accuracy: 0.7805
Epoch 24/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6869
- accuracy: 0.7643 - val_loss: 0.6577 - val_accuracy: 0.7796
Epoch 25/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6788
- accuracy: 0.7677 - val_loss: 0.6295 - val_accuracy: 0.7837
Epoch 26/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6693
- accuracy: 0.7704 - val_loss: 0.6337 - val_accuracy: 0.7833
Epoch 27/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6516
- accuracy: 0.7786 - val_loss: 0.6431 - val_accuracy: 0.7862
Epoch 28/500
625/625 [==============================] - 4s 7ms/step - loss: 0.6517
- accuracy: 0.7779 - val_loss: 0.6471 - val_accuracy: 0.7811
Epoch 29/500
625/625 [==============================] - 4s 7ms/step - loss: 0.6449
- accuracy: 0.7830 - val_loss: 0.6543 - val_accuracy: 0.7808
Epoch 30/500
625/625 [==============================] - 4s 7ms/step - loss: 0.6369
- accuracy: 0.7822 - val_loss: 0.6485 - val_accuracy: 0.7866
```

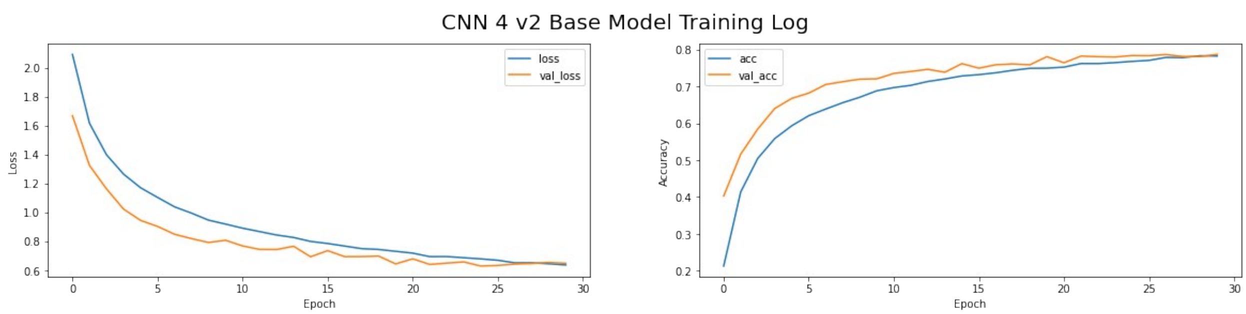
```
preds = CNN_4_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_4_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.85      0.77      0.81      1000
           1       0.94      0.86      0.90      1000
           2       0.71      0.63      0.67      1000
           3       0.61      0.63      0.62      1000
           4       0.69      0.81      0.74      1000
           5       0.74      0.69      0.71      1000
           6       0.74      0.88      0.80      1000
           7       0.84      0.83      0.83      1000
           8       0.90      0.88      0.89      1000
           9       0.88      0.87      0.87      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
```

```
weighted avg       0.79      0.79      0.79     10000

313/313 - 1s - loss: 0.6781 - accuracy: 0.7856
Accuracy: 78.56000065803528
Macro F1-score: 0.7858889558584797
```

With the ascending dropout layer combination, the model performance shot up from 68% to 78% which is 10% improvememt!. This is a good sign, so lets continuing tuning the dropout layer values in the later stage

```
loss = CNN_4_v2_history.history['loss']
val_loss = CNN_4_v2_history.history['val_loss']
acc = CNN_4_v2_history.history['accuracy']
val_acc = CNN_4_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 4 v2 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



From the model training log, we can see that there is slight overfitting but the deviation from the validation loss/acc to the training loss/acc has closed up alot.

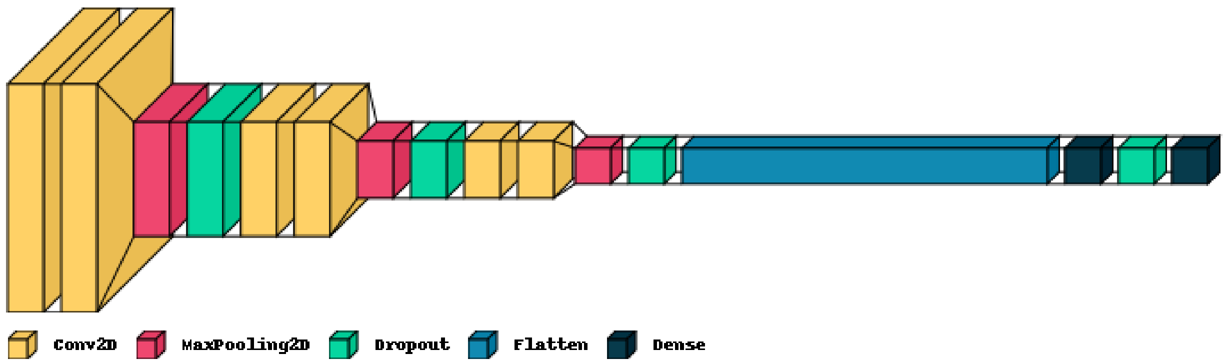
3.2.2.3.2 CNN 4 v3: Ascending Dropout Layer Combination - [0.3, 0.4, 0.5, 0.6]

Now, I will be trying the ascending dropout layer combination but with a much higher value as shown where the starting value is 0.3

```
CNN_4_v3=build_CNN_4(0.3, 0.4, 0.5, 0.6)
CNN_4_v3.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_96 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_97 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_48 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_34 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_98 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_99 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_35 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_100 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_101 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_50 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_36 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_15 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_30 (Dense)             (None, 64)                131136
_________________________________________________________________
dropout_37 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_4_v2, legend=True)
```




```
CNN_4_v3.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_4_v3_history = CNN_4_v3.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 2.1320
- accuracy: 0.1900 - val_loss: 1.7693 - val_accuracy: 0.3476
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.7961
- accuracy: 0.3318 - val_loss: 1.5863 - val_accuracy: 0.4193
Epoch 3/500
625/625 [==============================] - 4s 6ms/step - loss: 1.6425
- accuracy: 0.4003 - val_loss: 1.3855 - val_accuracy: 0.5017
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 1.5072
- accuracy: 0.4623 - val_loss: 1.2710 - val_accuracy: 0.5498
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 1.4076
- accuracy: 0.5064 - val_loss: 1.1675 - val_accuracy: 0.5940
Epoch 6/500
625/625 [==============================] - 4s 6ms/step - loss: 1.3168
- accuracy: 0.5396 - val_loss: 1.0562 - val_accuracy: 0.6241
Epoch 7/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2470
- accuracy: 0.5655 - val_loss: 1.0027 - val_accuracy: 0.6504
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2024
- accuracy: 0.5841 - val_loss: 0.9822 - val_accuracy: 0.6530
Epoch 9/500
625/625 [==============================] - 4s 7ms/step - loss: 1.1567
- accuracy: 0.6010 - val_loss: 0.9318 - val_accuracy: 0.6740
Epoch 10/500
```

```
625/625 [==============================] - 4s 7ms/step - loss: 1.1267
- accuracy: 0.6100 - val_loss: 0.8877 - val_accuracy: 0.6853
Epoch 11/500
625/625 [==============================] - 4s 7ms/step - loss: 1.0997
- accuracy: 0.6201 - val_loss: 0.9481 - val_accuracy: 0.6644
Epoch 12/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0717
- accuracy: 0.6339 - val_loss: 0.8551 - val_accuracy: 0.7027
Epoch 13/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0546
- accuracy: 0.6400 - val_loss: 0.8573 - val_accuracy: 0.7044
Epoch 14/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0266
- accuracy: 0.6481 - val_loss: 0.8631 - val_accuracy: 0.6990
Epoch 15/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0120
- accuracy: 0.6544 - val_loss: 0.8408 - val_accuracy: 0.7060
Epoch 16/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9945
- accuracy: 0.6597 - val_loss: 0.7863 - val_accuracy: 0.7257
Epoch 17/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9768
- accuracy: 0.6687 - val_loss: 0.7801 - val_accuracy: 0.7333
Epoch 18/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9594
- accuracy: 0.6743 - val_loss: 0.8216 - val_accuracy: 0.7166
Epoch 19/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9490
- accuracy: 0.6786 - val_loss: 0.7791 - val_accuracy: 0.7322
Epoch 20/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9360
- accuracy: 0.6849 - val_loss: 0.8071 - val_accuracy: 0.7222
Epoch 21/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9256
- accuracy: 0.6896 - val_loss: 0.7628 - val_accuracy: 0.7385
Epoch 22/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9057
- accuracy: 0.6908 - val_loss: 0.7176 - val_accuracy: 0.7536
Epoch 23/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8969
- accuracy: 0.6963 - val_loss: 0.7454 - val_accuracy: 0.7453
Epoch 24/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8871
- accuracy: 0.7000 - val_loss: 0.6978 - val_accuracy: 0.7607
Epoch 25/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8762
- accuracy: 0.7038 - val_loss: 0.7114 - val_accuracy: 0.7567
Epoch 26/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8849
```

```
- accuracy: 0.7007 - val_loss: 0.7029 - val_accuracy: 0.7617
Epoch 27/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8656
- accuracy: 0.7103 - val_loss: 0.7298 - val_accuracy: 0.7507
Epoch 28/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8696
- accuracy: 0.7057 - val_loss: 0.6999 - val_accuracy: 0.7614
Epoch 29/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8531
- accuracy: 0.7137 - val_loss: 0.6889 - val_accuracy: 0.7592
Epoch 30/500
625/625 [==============================] - 4s 7ms/step - loss: 0.8385
- accuracy: 0.7186 - val_loss: 0.6949 - val_accuracy: 0.7608
Epoch 31/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8426
- accuracy: 0.7174 - val_loss: 0.7151 - val_accuracy: 0.7575
Epoch 32/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8346
- accuracy: 0.7149 - val_loss: 0.6987 - val_accuracy: 0.7657
Epoch 33/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8326
- accuracy: 0.7205 - val_loss: 0.7048 - val_accuracy: 0.7587
Epoch 34/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8254
- accuracy: 0.7245 - val_loss: 0.6600 - val_accuracy: 0.7744
Epoch 35/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8142
- accuracy: 0.7264 - val_loss: 0.6906 - val_accuracy: 0.7685
Epoch 36/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8078
- accuracy: 0.7295 - val_loss: 0.6720 - val_accuracy: 0.7698
Epoch 37/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8049
- accuracy: 0.7288 - val_loss: 0.6651 - val_accuracy: 0.7781
Epoch 38/500
625/625 [==============================] - 4s 7ms/step - loss: 0.7975
- accuracy: 0.7325 - val_loss: 0.6738 - val_accuracy: 0.7703
Epoch 39/500
625/625 [==============================] - 4s 7ms/step - loss: 0.7883
- accuracy: 0.7352 - val_loss: 0.6945 - val_accuracy: 0.7680
```

```
preds = CNN_4_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_4_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

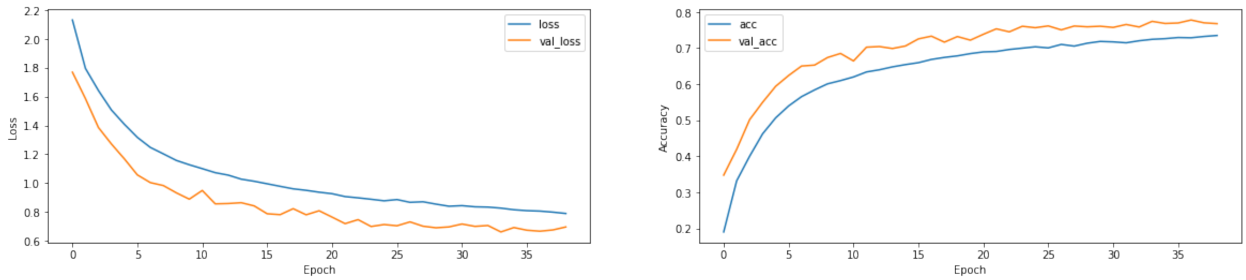
```
              precision    recall  f1-score   support
```

```
              0       0.85      0.70      0.77      1000
              1       0.91      0.88      0.90      1000
              2       0.84      0.46      0.60      1000
              3       0.55      0.66      0.60      1000
              4       0.64      0.78      0.71      1000
              5       0.70      0.66      0.68      1000
              6       0.71      0.89      0.79      1000
              7       0.87      0.75      0.81      1000
              8       0.84      0.88      0.86      1000
              9       0.82      0.91      0.86      1000

    accuracy                          0.76     10000
   macro avg       0.77      0.76      0.76     10000
weighted avg       0.77      0.76      0.76     10000

313/313 - 1s - loss: 0.7381 - accuracy: 0.7586
Accuracy: 75.85999965667725
Macro F1-score: 0.7563097086809136
```

We can see that by increasing the dropout value, the model performance has decreased form 78% to 75%.

```
loss = CNN_4_v3_history.history['loss']
val_loss = CNN_4_v3_history.history['val_loss']
acc = CNN_4_v3_history.history['accuracy']
val_acc = CNN_4_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 4 v3 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

CNN 4 v3 Base Model Training Log



Also, the deviation from the validation and training loss and accuracy has increased. Hence, I will not be using this dropout layer combination

3.2.2.3.3 CNN 4 v4: Dropout Layer Combination - [0.2, 0.3, 0.4, 0.3]

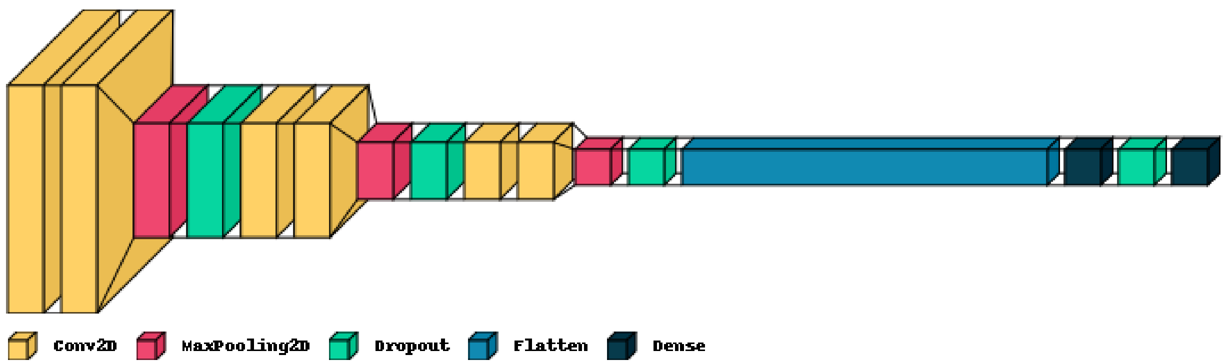
Now, lets try reducing the values and try out the ascending then decreasing dropout.

```
CNN_4_v4=build_CNN_4(0.2, 0.3, 0.4, 0.3)
CNN_4_v4.summary()
```

```
Model: "sequential_82"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_486 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_487 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_243 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_294 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_488 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_489 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_244 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_295 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_490 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_491 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_245 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_296 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
```

```
flatten_80 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_164 (Dense)            (None, 64)                131136
_________________________________________________________________
dropout_297 (Dropout)        (None, 64)                0
_________________________________________________________________
dense_165 (Dense)            (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_4_v4, legend=True)
```




```
CNN_4_v4.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_4_v4_history = CNN_4_v4.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 1.9742
- accuracy: 0.2611 - val_loss: 1.5952 - val_accuracy: 0.4090
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.5675
- accuracy: 0.4335 - val_loss: 1.3392 - val_accuracy: 0.5237
Epoch 3/500
625/625 [==============================] - 4s 6ms/step - loss: 1.3721
- accuracy: 0.5124 - val_loss: 1.2120 - val_accuracy: 0.5721
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2421
- accuracy: 0.5645 - val_loss: 1.0978 - val_accuracy: 0.6129
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1433
- accuracy: 0.6011 - val_loss: 0.9323 - val_accuracy: 0.6732
Epoch 6/500
```

```
625/625 [==============================] - 4s 6ms/step - loss: 1.0732
- accuracy: 0.6280 - val_loss: 0.9121 - val_accuracy: 0.6756
Epoch 7/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0219
- accuracy: 0.6435 - val_loss: 0.8552 - val_accuracy: 0.6967
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9721
- accuracy: 0.6626 - val_loss: 0.8200 - val_accuracy: 0.7096
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9471
- accuracy: 0.6745 - val_loss: 0.8081 - val_accuracy: 0.7188
Epoch 10/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9068
- accuracy: 0.6850 - val_loss: 0.7671 - val_accuracy: 0.7346
Epoch 11/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8917
- accuracy: 0.6937 - val_loss: 0.7972 - val_accuracy: 0.7236
Epoch 12/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8610
- accuracy: 0.7006 - val_loss: 0.7274 - val_accuracy: 0.7472
Epoch 13/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8422
- accuracy: 0.7107 - val_loss: 0.6985 - val_accuracy: 0.7564
Epoch 14/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8329
- accuracy: 0.7142 - val_loss: 0.7399 - val_accuracy: 0.7452
Epoch 15/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8083
- accuracy: 0.7238 - val_loss: 0.7484 - val_accuracy: 0.7451
Epoch 16/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7973
- accuracy: 0.7291 - val_loss: 0.6906 - val_accuracy: 0.7637
Epoch 17/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7774
- accuracy: 0.7333 - val_loss: 0.7079 - val_accuracy: 0.7587
Epoch 18/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7707
- accuracy: 0.7346 - val_loss: 0.7640 - val_accuracy: 0.7431
Epoch 19/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7566
- accuracy: 0.7379 - val_loss: 0.6761 - val_accuracy: 0.7695
Epoch 20/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7471
- accuracy: 0.7417 - val_loss: 0.6640 - val_accuracy: 0.7724
Epoch 21/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7409
- accuracy: 0.7443 - val_loss: 0.6382 - val_accuracy: 0.7786
Epoch 22/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7267
```

```
- accuracy: 0.7489 - val_loss: 0.6610 - val_accuracy: 0.7673
Epoch 23/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7178
- accuracy: 0.7534 - val_loss: 0.6802 - val_accuracy: 0.7639
Epoch 24/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7121
- accuracy: 0.7569 - val_loss: 0.6809 - val_accuracy: 0.7650
Epoch 25/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7032
- accuracy: 0.7581 - val_loss: 0.6556 - val_accuracy: 0.7739
Epoch 26/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6976
- accuracy: 0.7600 - val_loss: 0.6797 - val_accuracy: 0.7692
```

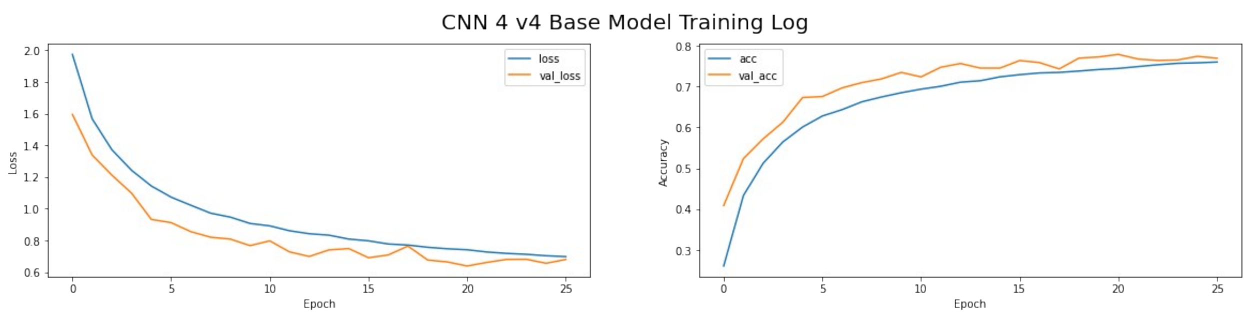
```
preds = CNN_4_v4.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_4_v4.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.82      0.76      0.79      1000
           1       0.90      0.92      0.91      1000
           2       0.60      0.71      0.65      1000
           3       0.71      0.43      0.54      1000
           4       0.67      0.78      0.72      1000
           5       0.74      0.65      0.70      1000
           6       0.66      0.91      0.77      1000
           7       0.84      0.79      0.81      1000
           8       0.87      0.89      0.88      1000
           9       0.92      0.81      0.86      1000

    accuracy                           0.77     10000
   macro avg       0.77      0.77      0.76     10000
weighted avg       0.77      0.77      0.76     10000

313/313 - 1s - loss: 0.7059 - accuracy: 0.7656
Accuracy: 76.56000256538391
Macro F1-score: 0.7626442117827071
```

We can see that this dropout layer value combination has increased the performance from 75% to 76%.

```
loss = CNN_4_v4_history.history['loss']
val_loss = CNN_4_v4_history.history['val_loss']
acc = CNN_4_v4_history.history['accuracy']
val_acc = CNN_4_v4_history.history['val_accuracy']
epoch = range(len(loss))
```

```
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 4 v4 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



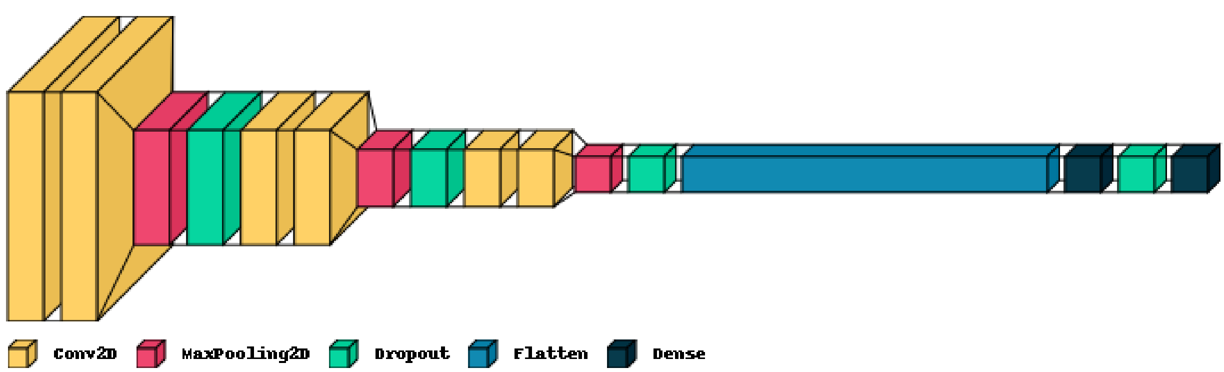
From the model training log, there is still slight overfitting and the model training is not as good as the CNN v2 training log.

3.2.2.3.4 CNN 4 v5: Dropout Layer Combination - [0.3, 0.4, 0.5, 0.3]

Now, lets try an increased dropout layer value combination by increasing a 0.1 from CNN v4 model

```
CNN_4_v5=build_CNN_4(0.3, 0.4, 0.5, 0.3)
CNN_4_v5.summary()

Model: "sequential_83"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_492 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
conv2d_493 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_246 (MaxPoolin (None, 16, 16, 32)        0
```

```
_________________________________________________________________
dropout_298 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_494 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_495 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_247 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_299 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_496 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_497 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_248 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_300 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_81 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_166 (Dense)            (None, 64)                131136
_________________________________________________________________
dropout_301 (Dropout)        (None, 64)                0
_________________________________________________________________
dense_167 (Dense)            (None, 10)                650
=================================================================
Total params: 418,218
Trainable params: 418,218
Non-trainable params: 0
_________________________________________________________________

visualkeras.layered_view(CNN_4_v5, legend=True)
```




```
CNN_4_v5.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_4_v5_history = CNN_4_v5.fit(X_train, y_train, epochs=500,
```

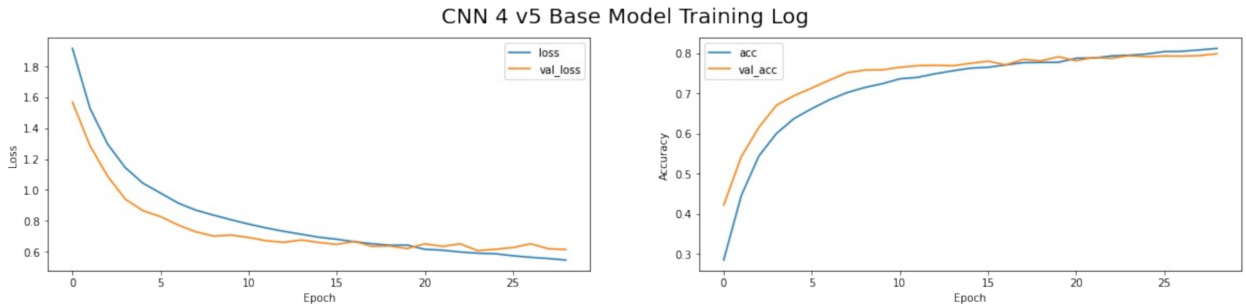
```
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 4s 6ms/step - loss: 1.9162
- accuracy: 0.2851 - val_loss: 1.5678 - val_accuracy: 0.4217
Epoch 2/500
625/625 [==============================] - 4s 6ms/step - loss: 1.5267
- accuracy: 0.4457 - val_loss: 1.2848 - val_accuracy: 0.5422
Epoch 3/500
625/625 [==============================] - 4s 6ms/step - loss: 1.2955
- accuracy: 0.5441 - val_loss: 1.0882 - val_accuracy: 0.6155
Epoch 4/500
625/625 [==============================] - 4s 6ms/step - loss: 1.1438
- accuracy: 0.6003 - val_loss: 0.9398 - val_accuracy: 0.6704
Epoch 5/500
625/625 [==============================] - 4s 6ms/step - loss: 1.0429
- accuracy: 0.6374 - val_loss: 0.8649 - val_accuracy: 0.6941
Epoch 6/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9793
- accuracy: 0.6618 - val_loss: 0.8266 - val_accuracy: 0.7131
Epoch 7/500
625/625 [==============================] - 4s 6ms/step - loss: 0.9142
- accuracy: 0.6838 - val_loss: 0.7719 - val_accuracy: 0.7327
Epoch 8/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8687
- accuracy: 0.7017 - val_loss: 0.7284 - val_accuracy: 0.7512
Epoch 9/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8365
- accuracy: 0.7144 - val_loss: 0.7007 - val_accuracy: 0.7576
Epoch 10/500
625/625 [==============================] - 4s 6ms/step - loss: 0.8063
- accuracy: 0.7239 - val_loss: 0.7071 - val_accuracy: 0.7584
Epoch 11/500
625/625 [==============================] - 4s 7ms/step - loss: 0.7781
- accuracy: 0.7361 - val_loss: 0.6912 - val_accuracy: 0.7647
Epoch 12/500
625/625 [==============================] - 4s 7ms/step - loss: 0.7530
- accuracy: 0.7396 - val_loss: 0.6701 - val_accuracy: 0.7687
Epoch 13/500
625/625 [==============================] - 4s 6ms/step - loss: 0.7311
- accuracy: 0.7486 - val_loss: 0.6600 - val_accuracy: 0.7695
Epoch 14/500
625/625 [==============================] - 4s 7ms/step - loss: 0.7123
- accuracy: 0.7559 - val_loss: 0.6753 - val_accuracy: 0.7685
Epoch 15/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6928
- accuracy: 0.7625 - val_loss: 0.6590 - val_accuracy: 0.7745
Epoch 16/500
```

```
625/625 [==============================] - 4s 6ms/step - loss: 0.6798
- accuracy: 0.7647 - val_loss: 0.6467 - val_accuracy: 0.7799
Epoch 17/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6642
- accuracy: 0.7706 - val_loss: 0.6666 - val_accuracy: 0.7706
Epoch 18/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6506
- accuracy: 0.7763 - val_loss: 0.6337 - val_accuracy: 0.7843
Epoch 19/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6413
- accuracy: 0.7768 - val_loss: 0.6364 - val_accuracy: 0.7808
Epoch 20/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6421
- accuracy: 0.7772 - val_loss: 0.6196 - val_accuracy: 0.7908
Epoch 21/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6147
- accuracy: 0.7873 - val_loss: 0.6503 - val_accuracy: 0.7812
Epoch 22/500
625/625 [==============================] - 4s 6ms/step - loss: 0.6092
- accuracy: 0.7878 - val_loss: 0.6336 - val_accuracy: 0.7891
Epoch 23/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5973
- accuracy: 0.7929 - val_loss: 0.6506 - val_accuracy: 0.7871
Epoch 24/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5897
- accuracy: 0.7946 - val_loss: 0.6063 - val_accuracy: 0.7936
Epoch 25/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5858
- accuracy: 0.7977 - val_loss: 0.6151 - val_accuracy: 0.7912
Epoch 26/500
625/625 [==============================] - 4s 7ms/step - loss: 0.5729
- accuracy: 0.8037 - val_loss: 0.6260 - val_accuracy: 0.7929
Epoch 27/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5630
- accuracy: 0.8044 - val_loss: 0.6507 - val_accuracy: 0.7926
Epoch 28/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5551
- accuracy: 0.8078 - val_loss: 0.6186 - val_accuracy: 0.7937
Epoch 29/500
625/625 [==============================] - 4s 6ms/step - loss: 0.5454
- accuracy: 0.8116 - val_loss: 0.6140 - val_accuracy: 0.7986

preds = CNN_4_v5.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_4_v5.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
                precision    recall  f1-score   support

           0       0.86      0.77      0.81      1000
           1       0.91      0.90      0.90      1000
           2       0.71      0.65      0.68      1000
           3       0.65      0.60      0.62      1000
           4       0.68      0.82      0.74      1000
           5       0.71      0.73      0.72      1000
           6       0.80      0.86      0.83      1000
           7       0.88      0.80      0.84      1000
           8       0.87      0.89      0.88      1000
           9       0.86      0.90      0.88      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
weighted avg       0.79      0.79      0.79     10000

313/313 - 1s - loss: 0.6395 - accuracy: 0.7912
Accuracy: 79.11999821662903
Macro F1-score: 0.790620777578482
```

We can see that we have achived the highest model perfromance of 79% which is higher than the CNN 3 v2 model.

```
loss = CNN_4_v5_history.history['loss']
val_loss = CNN_4_v5_history.history['val_loss']
acc = CNN_4_v5_history.history['accuracy']
val_acc = CNN_4_v5_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 4 v5 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Looking at the CNN 4 version 5 model training log, the val loss and val accuracy are mostly aligned with the training loss and accuracy. This means that the problem of overfitting is solved! Hence, we will keep this model for the next section. Also this model has the highest model performance out of every model.

Let's compare the summary of all model results:

Conclusion: CNN 4 v5 F1-Score is 79%

3.2.3.1 Adding Batch Normalization layer

Now, I will be adding Batch normalization layers.It does this scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer. It works just the same way as we normalize the input data where we divided the x_train/255.

What we are trying to do there is we are arranging all the features in same scale so that model converges easily and we can reduce the distrotions.Whenever we passs the CNN throuh a batch normalization layer we are normalizing the weights so that our model will be stable and we can train model longer and also use larger learning rate.

I will be adding it after every conv2d layer and the dense layer of 64

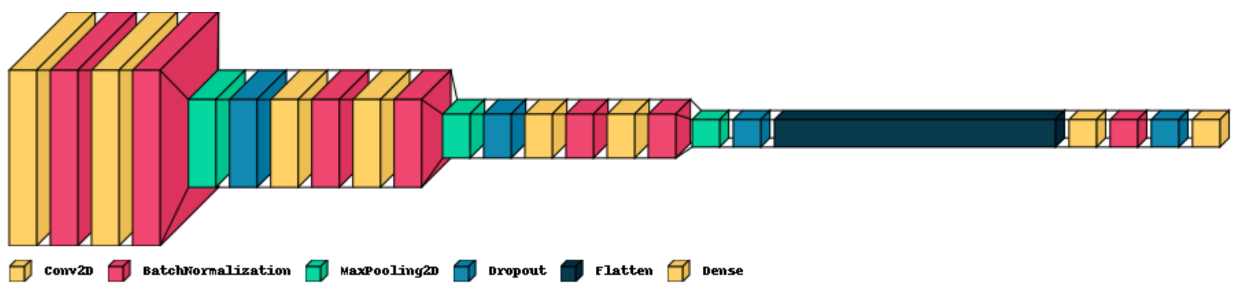
```
CNN_5_v1=Sequential()
CNN_5_v1.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(Conv2D(32,(3,3),activation="relu", padding='same'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(MaxPooling2D(2,2))
CNN_5_v1.add(Dropout(0.3))

CNN_5_v1.add(Conv2D(64,(3,3),activation="relu", padding='same'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(Conv2D(64,(3,3),activation="relu", padding='same'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(MaxPooling2D(2,2))
CNN_5_v1.add(Dropout(0.4))
```

```
CNN_5_v1.add(Conv2D(128,(3,3),activation="relu", padding='same'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(Conv2D(128,(3,3),activation="relu", padding='same'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(MaxPooling2D(2,2))
CNN_5_v1.add(Dropout(0.5))

CNN_5_v1.add(Flatten())
CNN_5_v1.add(Dense(64,activation='relu'))
CNN_5_v1.add(BatchNormalization())
CNN_5_v1.add(Dropout(0.3))
CNN_5_v1.add(Dense(10,activation='softmax'))
CNN_5_v1.summary()
```

```
Model: "sequential_84"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_498 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_389 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_499 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_390 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_249 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_302 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_500 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_391 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_501 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_392 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_250 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_303 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_502 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_393 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_503 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_394 (Bat (None, 8, 8, 128)         512
```

```
max_pooling2d_251 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_304 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_82 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_168 (Dense)            (None, 64)                131136
_________________________________________________________________
batch_normalization_395 (Bat (None, 64)                256
_________________________________________________________________
dropout_305 (Dropout)        (None, 64)                0
_________________________________________________________________
dense_169 (Dense)            (None, 10)                650
=================================================================
Total params: 420,266
Trainable params: 419,242
Non-trainable params: 1,024
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_5_v1, legend=True)
```




```
CNN_5_v1.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_5_v1_history = CNN_5_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 8s 11ms/step - loss: 1.7221
- accuracy: 0.4027 - val_loss: 1.7279 - val_accuracy: 0.3832
Epoch 2/500
625/625 [==============================] - 6s 9ms/step - loss: 1.1951
- accuracy: 0.5786 - val_loss: 1.3308 - val_accuracy: 0.5575
Epoch 3/500
625/625 [==============================] - 7s 11ms/step - loss: 1.0093
- accuracy: 0.6486 - val_loss: 0.9651 - val_accuracy: 0.6692
Epoch 4/500
625/625 [==============================] - 7s 11ms/step - loss: 0.9048
```

```
- accuracy: 0.6852 - val_loss: 0.7615 - val_accuracy: 0.7335
Epoch 5/500
625/625 [==============================] - 7s 11ms/step - loss: 0.8373
- accuracy: 0.7102 - val_loss: 0.7078 - val_accuracy: 0.7562
Epoch 6/500
625/625 [==============================] - 7s 11ms/step - loss: 0.7799
- accuracy: 0.7316 - val_loss: 0.7681 - val_accuracy: 0.7362
Epoch 7/500
625/625 [==============================] - 7s 11ms/step - loss: 0.7397
- accuracy: 0.7447 - val_loss: 0.6965 - val_accuracy: 0.7630
Epoch 8/500
625/625 [==============================] - 7s 11ms/step - loss: 0.6983
- accuracy: 0.7613 - val_loss: 0.6507 - val_accuracy: 0.7748
Epoch 9/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6626
- accuracy: 0.7742 - val_loss: 0.7828 - val_accuracy: 0.7218
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6447
- accuracy: 0.7790 - val_loss: 0.6522 - val_accuracy: 0.7780
Epoch 11/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6165
- accuracy: 0.7879 - val_loss: 0.5840 - val_accuracy: 0.8015
Epoch 12/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5923
- accuracy: 0.7966 - val_loss: 0.5818 - val_accuracy: 0.8008
Epoch 13/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5796
- accuracy: 0.8001 - val_loss: 0.5621 - val_accuracy: 0.8103
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5557
- accuracy: 0.8104 - val_loss: 0.5741 - val_accuracy: 0.8046
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5383
- accuracy: 0.8158 - val_loss: 0.5720 - val_accuracy: 0.8067
Epoch 16/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5238
- accuracy: 0.8169 - val_loss: 0.5281 - val_accuracy: 0.8220
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5086
- accuracy: 0.8264 - val_loss: 0.5324 - val_accuracy: 0.8221
Epoch 18/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5041
- accuracy: 0.8261 - val_loss: 0.5534 - val_accuracy: 0.8109
Epoch 19/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4873
- accuracy: 0.8310 - val_loss: 0.5226 - val_accuracy: 0.8258
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4791
- accuracy: 0.8322 - val_loss: 0.4981 - val_accuracy: 0.8323
```

```
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4658
- accuracy: 0.8393 - val_loss: 0.5076 - val_accuracy: 0.8272
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4565
- accuracy: 0.8411 - val_loss: 0.5720 - val_accuracy: 0.8106
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4537
- accuracy: 0.8420 - val_loss: 0.5345 - val_accuracy: 0.8216
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4364
- accuracy: 0.8498 - val_loss: 0.5875 - val_accuracy: 0.8119
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4362
- accuracy: 0.8503 - val_loss: 0.6154 - val_accuracy: 0.8021
```

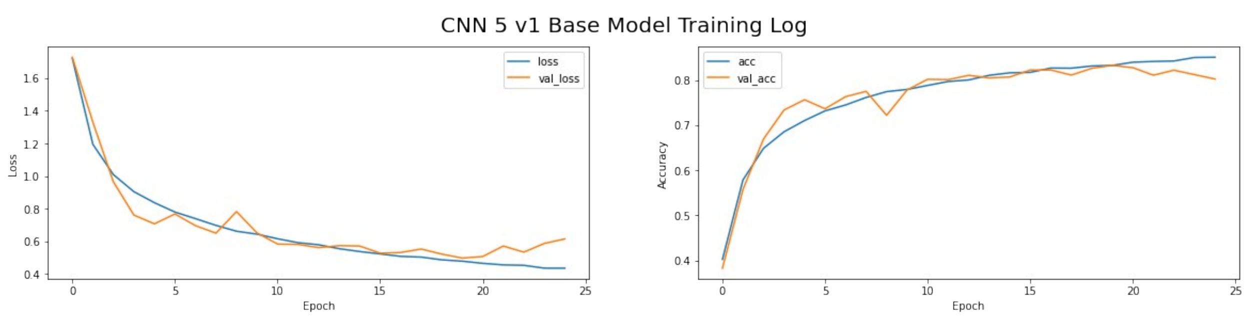
```
preds = CNN_5_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_5_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.89      0.63      0.74      1000
           1       0.94      0.90      0.92      1000
           2       0.81      0.67      0.74      1000
           3       0.69      0.63      0.66      1000
           4       0.69      0.85      0.76      1000
           5       0.81      0.66      0.73      1000
           6       0.81      0.88      0.84      1000
           7       0.84      0.89      0.86      1000
           8       0.73      0.96      0.83      1000
           9       0.85      0.91      0.88      1000

    accuracy                           0.80     10000
   macro avg       0.81      0.80      0.80     10000
weighted avg       0.81      0.80      0.80     10000

313/313 - 1s - loss: 0.6444 - accuracy: 0.7983
Accuracy: 79.830002784729
Macro F1-score: 0.7952089568619523
```

Looks like model perfromance has increased by 0.01% by adding batch normalization

```
loss = CNN_5_v1_history.history['loss']
val_loss = CNN_5_v1_history.history['val_loss']
acc = CNN_5_v1_history.history['accuracy']
val_acc = CNN_5_v1_history.history['val_accuracy']
```

```
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 5 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Looking at the CNN 5 version 1 model training log, even though batch normzalization layers has allowed the model to converge faster, however, underfitting has increased by quite abit but not too much as compared to the previous training log with only dropout layers and no batch normalization layers. Hence given that the model perfromance has only increase by 0.01% with the slight overfitting, I would be keeping this model with batch normalization as I can continue reducing the overfitting in the later stage.

Let's compare the summary of all model results:

Conclusion: CNN 5 v1 F1-Score is 80%

Now I will be tuning the number of neurons in Dense layer.

3.2.4.1 Tuning number of neurons in Dense layer

```
def build_CNN_6(d1):
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
```

```
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.3))

    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.4))


    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(d1,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(Dense(10,activation='softmax'))
    return model
```

Note that I will be trying out values - 32, 64, 128, 256, 512

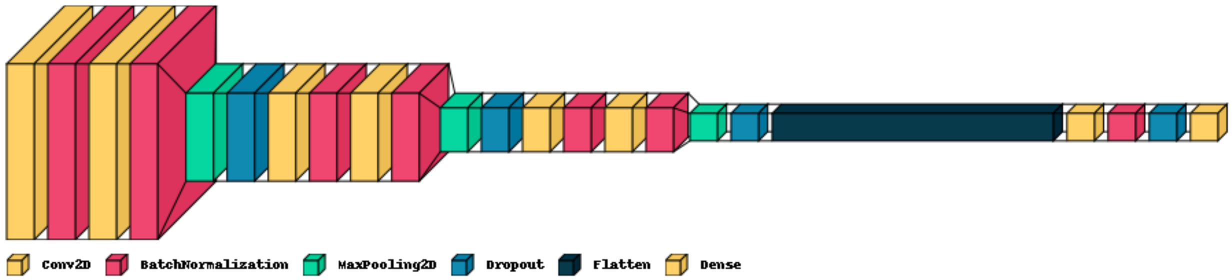
3.2.4.1.1 CNN 6 v1: Dense layer Value - 32

```
CNN_6_v1=build_CNN_6(32)
CNN_6_v1.summary()
```

```
Model: "sequential_85"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_504 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_396 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_505 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_397 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_252 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_306 (Dropout)        (None, 16, 16, 32)        0
```

```
_________________________________________________________________
conv2d_506 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_398 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_507 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_399 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_253 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_307 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_508 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_400 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_509 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_401 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_254 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_308 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_83 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_170 (Dense)            (None, 32)                65568
_________________________________________________________________
batch_normalization_402 (Bat (None, 32)                128
_________________________________________________________________
dropout_309 (Dropout)        (None, 32)                0
_________________________________________________________________
dense_171 (Dense)            (None, 10)                330
=================================================================
Total params: 354,250
Trainable params: 353,290
Non-trainable params: 960
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_6_v1, legend=True)
```




```
CNN_6_v1.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_6_v1_history = CNN_6_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.6969
- accuracy: 0.3950 - val_loss: 1.4024 - val_accuracy: 0.4663
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.2126
- accuracy: 0.5751 - val_loss: 0.9787 - val_accuracy: 0.6592
Epoch 3/500
625/625 [==============================] - 5s 9ms/step - loss: 1.0322
- accuracy: 0.6432 - val_loss: 0.8728 - val_accuracy: 0.6981
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9319
- accuracy: 0.6784 - val_loss: 0.7750 - val_accuracy: 0.7297
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8642
- accuracy: 0.7042 - val_loss: 0.7721 - val_accuracy: 0.7291
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8085
- accuracy: 0.7224 - val_loss: 0.7384 - val_accuracy: 0.7472
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7637
- accuracy: 0.7397 - val_loss: 0.6884 - val_accuracy: 0.7614
Epoch 8/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7229
- accuracy: 0.7547 - val_loss: 0.9854 - val_accuracy: 0.6762
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6993
- accuracy: 0.7620 - val_loss: 0.6278 - val_accuracy: 0.7819
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6653
- accuracy: 0.7771 - val_loss: 0.6431 - val_accuracy: 0.7776
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6492
- accuracy: 0.7786 - val_loss: 0.5503 - val_accuracy: 0.8131
Epoch 12/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.6158
- accuracy: 0.7916 - val_loss: 0.6736 - val_accuracy: 0.7743
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6030
- accuracy: 0.7950 - val_loss: 0.5648 - val_accuracy: 0.8069
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5871
- accuracy: 0.8008 - val_loss: 0.5613 - val_accuracy: 0.8090
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5667
- accuracy: 0.8074 - val_loss: 0.5571 - val_accuracy: 0.8109
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5534
- accuracy: 0.8107 - val_loss: 0.5916 - val_accuracy: 0.7975
```

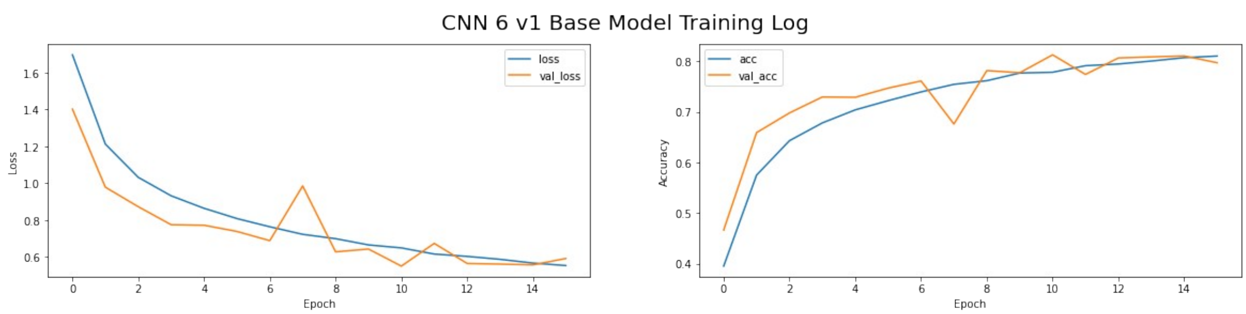
```
preds = CNN_6_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_6_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.83      0.79      0.81      1000
           1       0.94      0.90      0.92      1000
           2       0.79      0.65      0.71      1000
           3       0.61      0.68      0.64      1000
           4       0.78      0.72      0.75      1000
           5       0.58      0.84      0.69      1000
           6       0.86      0.81      0.83      1000
           7       0.90      0.81      0.85      1000
           8       0.89      0.89      0.89      1000
           9       0.90      0.89      0.90      1000

    accuracy                           0.80     10000
   macro avg       0.81      0.80      0.80     10000
weighted avg       0.81      0.80      0.80     10000

313/313 - 1s - loss: 0.6160 - accuracy: 0.7965
Accuracy: 79.6500027179718
Macro F1-score: 0.7994981397584691
```

By tuning the dense layer value, it makes not much difference to the model performance as the f1-score is still stagnant 80%

```
loss = CNN_6_v1_history.history['loss']
val_loss = CNN_6_v1_history.history['val_loss']
acc = CNN_6_v1_history.history['accuracy']
val_acc = CNN_6_v1_history.history['val_accuracy']
```

```
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 6 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



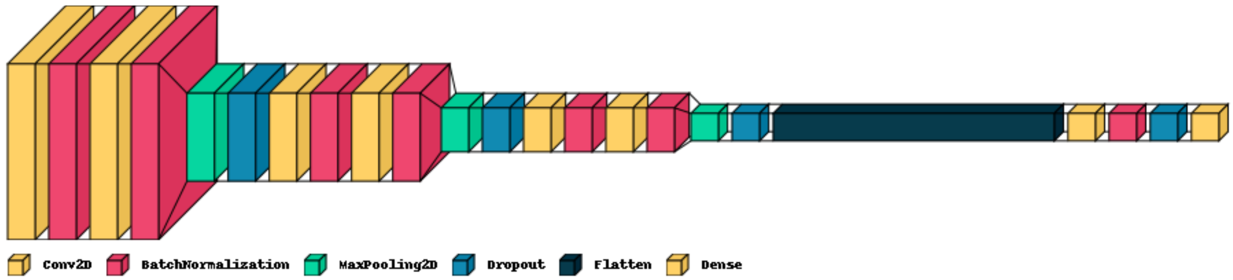
We can see that the model performance has remained at 80% and the model trainign has gotten worse from the previous CNN 5 v1 where there is major fluctuations in deviation from the training and validation loss and accuracy. Hence, in the next stage, I will be increasing the dense layer value to 64.

3.2.4.1.2 Dense layer Value - 64

```
CNN_6_v2=build_CNN_6(64)
CNN_6_v2.summary()

Model: "sequential_86"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_510 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_403 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_511 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_404 (Bat (None, 32, 32, 32)        128
```

```
_________________________________________________________________
max_pooling2d_255 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_310 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_512 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_405 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_513 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_406 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_256 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_311 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_514 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_407 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_515 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_408 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_257 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_312 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_84 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_172 (Dense)            (None, 64)                131136
_________________________________________________________________
batch_normalization_409 (Bat (None, 64)                256
_________________________________________________________________
dropout_313 (Dropout)        (None, 64)                0
_________________________________________________________________
dense_173 (Dense)            (None, 10)                650
=================================================================
Total params: 420,266
Trainable params: 419,242
Non-trainable params: 1,024
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_6_v2, legend=True)
```




```
CNN_6_v2.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_6_v2_history = CNN_6_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.7911
- accuracy: 0.3793 - val_loss: 1.9215 - val_accuracy: 0.3472
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.2256
- accuracy: 0.5689 - val_loss: 1.4650 - val_accuracy: 0.5137
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0261
- accuracy: 0.6428 - val_loss: 0.9228 - val_accuracy: 0.6795
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9278
- accuracy: 0.6787 - val_loss: 0.8442 - val_accuracy: 0.7056
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8531
- accuracy: 0.7041 - val_loss: 0.8363 - val_accuracy: 0.7053
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8012
- accuracy: 0.7239 - val_loss: 0.7108 - val_accuracy: 0.7530
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7552
- accuracy: 0.7403 - val_loss: 0.6988 - val_accuracy: 0.7546
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7201
- accuracy: 0.7516 - val_loss: 0.8246 - val_accuracy: 0.7170
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6847
- accuracy: 0.7670 - val_loss: 0.6770 - val_accuracy: 0.7625
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6616
- accuracy: 0.7712 - val_loss: 0.6335 - val_accuracy: 0.7811
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6320
- accuracy: 0.7826 - val_loss: 0.6711 - val_accuracy: 0.7728
Epoch 12/500
```

```
625/625 [==============================] - 6s 9ms/step - loss: 0.6084
- accuracy: 0.7924 - val_loss: 0.6690 - val_accuracy: 0.7747
Epoch 13/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5908
- accuracy: 0.7979 - val_loss: 0.5911 - val_accuracy: 0.7941
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5678
- accuracy: 0.8063 - val_loss: 0.5997 - val_accuracy: 0.7961
Epoch 15/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5591
- accuracy: 0.8088 - val_loss: 0.6515 - val_accuracy: 0.7795
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5368
- accuracy: 0.8159 - val_loss: 0.5683 - val_accuracy: 0.8061
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5267
- accuracy: 0.8198 - val_loss: 0.5175 - val_accuracy: 0.8239
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5065
- accuracy: 0.8273 - val_loss: 0.5386 - val_accuracy: 0.8184
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5023
- accuracy: 0.8279 - val_loss: 0.6297 - val_accuracy: 0.7934
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4885
- accuracy: 0.8320 - val_loss: 0.5033 - val_accuracy: 0.8295
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4854
- accuracy: 0.8311 - val_loss: 0.5134 - val_accuracy: 0.8262
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4733
- accuracy: 0.8349 - val_loss: 0.4923 - val_accuracy: 0.8315
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4633
- accuracy: 0.8411 - val_loss: 0.4838 - val_accuracy: 0.8340
Epoch 24/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4510
- accuracy: 0.8441 - val_loss: 0.5141 - val_accuracy: 0.8267
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4487
- accuracy: 0.8442 - val_loss: 0.5368 - val_accuracy: 0.8197
Epoch 26/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4262
- accuracy: 0.8525 - val_loss: 0.5130 - val_accuracy: 0.8278
Epoch 27/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4239
- accuracy: 0.8533 - val_loss: 0.5070 - val_accuracy: 0.8290
Epoch 28/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.4259
- accuracy: 0.8523 - val_loss: 0.7273 - val_accuracy: 0.7691
```

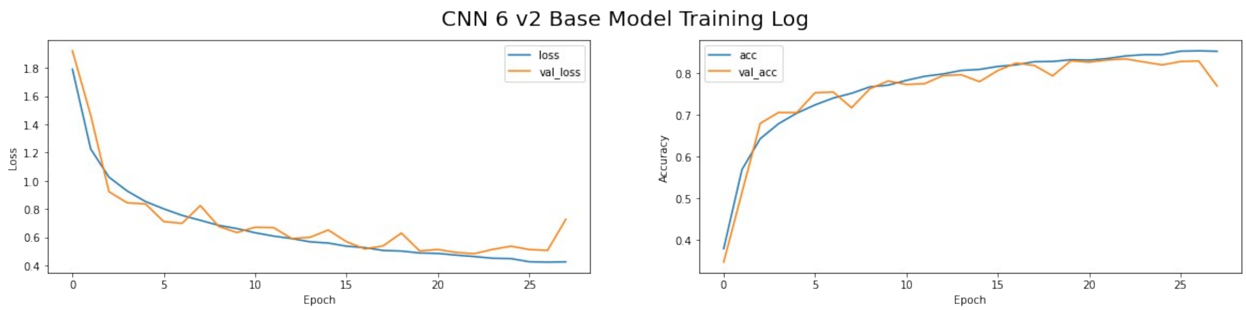
```
preds = CNN_6_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_6_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.93      0.55      0.69      1000
           1       0.95      0.84      0.90      1000
           2       0.73      0.63      0.67      1000
           3       0.69      0.60      0.64      1000
           4       0.52      0.91      0.66      1000
           5       0.76      0.62      0.69      1000
           6       0.81      0.85      0.83      1000
           7       0.89      0.78      0.83      1000
           8       0.89      0.87      0.88      1000
           9       0.72      0.96      0.83      1000

    accuracy                           0.76     10000
   macro avg       0.79      0.76      0.76     10000
weighted avg       0.79      0.76      0.76     10000

313/313 - 1s - loss: 0.7383 - accuracy: 0.7618
Accuracy: 76.17999911308289
Macro F1-score: 0.7622597094878987
```

We can see that by increasing the dense layer value to 64, the model performance has decreased instead. Hence, I will be ommitting this dense layer value.

```
loss = CNN_6_v2_history.history['loss']
val_loss = CNN_6_v2_history.history['val_loss']
acc = CNN_6_v2_history.history['accuracy']
val_acc = CNN_6_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 6 v2 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
```

```
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

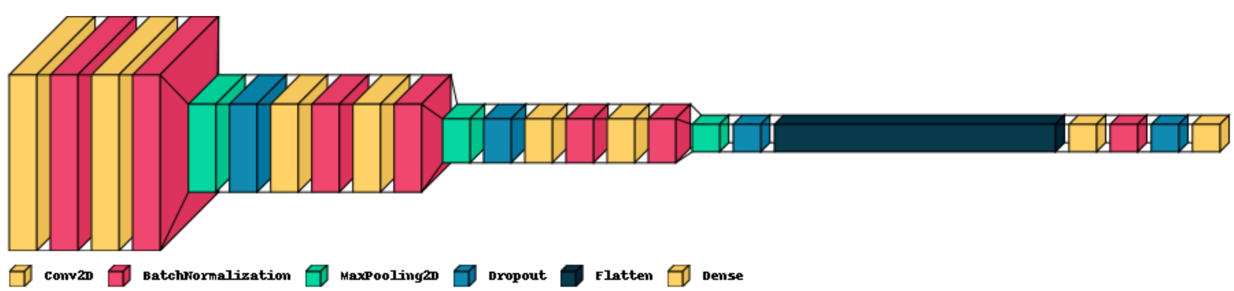


Evem though the model performance has decreased, the model training is better than the previous model. However, we can see that at the end of the epoch there is a tendency of underfitting. Hence, let increasr the dense layer value

3.2.4.1.3 Dense layer Value - 128

```
CNN_6_v3=build_CNN_6(128)
CNN_6_v3.summary()

Model: "sequential_87"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_516 (Conv2D)          (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_410 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_517 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_411 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_258 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_314 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_518 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_412 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_519 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_413 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
```

```
max_pooling2d_259 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_315 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_520 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_414 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_521 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_415 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_260 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_316 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_85 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_174 (Dense)            (None, 128)               262272
_________________________________________________________________
batch_normalization_416 (Bat (None, 128)               512
_________________________________________________________________
dropout_317 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_175 (Dense)            (None, 10)                1290
=================================================================
Total params: 552,298
Trainable params: 551,146
Non-trainable params: 1,152
_________________________________________________________________

visualkeras.layered_view(CNN_6_v3, legend=True)
```




```
CNN_6_v3.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_6_v3_history = CNN_6_v3.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.7640
- accuracy: 0.3952 - val_loss: 1.5410 - val_accuracy: 0.4498
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1800
- accuracy: 0.5823 - val_loss: 1.1390 - val_accuracy: 0.6044
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9952
- accuracy: 0.6493 - val_loss: 0.8697 - val_accuracy: 0.6959
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8974
- accuracy: 0.6848 - val_loss: 0.8881 - val_accuracy: 0.6904
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8347
- accuracy: 0.7080 - val_loss: 0.7187 - val_accuracy: 0.7500
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7711
- accuracy: 0.7323 - val_loss: 0.6867 - val_accuracy: 0.7619
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7311
- accuracy: 0.7470 - val_loss: 0.8020 - val_accuracy: 0.7178
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6908
- accuracy: 0.7622 - val_loss: 0.6126 - val_accuracy: 0.7887
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6655
- accuracy: 0.7685 - val_loss: 0.6077 - val_accuracy: 0.7881
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6325
- accuracy: 0.7794 - val_loss: 0.6217 - val_accuracy: 0.7877
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6065
- accuracy: 0.7917 - val_loss: 0.6256 - val_accuracy: 0.7861
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5885
- accuracy: 0.7963 - val_loss: 0.6030 - val_accuracy: 0.7931
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5595
- accuracy: 0.8059 - val_loss: 0.5554 - val_accuracy: 0.8066
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5471
- accuracy: 0.8113 - val_loss: 0.5690 - val_accuracy: 0.8059
Epoch 15/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5302
- accuracy: 0.8149 - val_loss: 0.5360 - val_accuracy: 0.8170
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5110
- accuracy: 0.8221 - val_loss: 0.5691 - val_accuracy: 0.8035
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4997
```

```
- accuracy: 0.8267 - val_loss: 0.5254 - val_accuracy: 0.8190
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4820
- accuracy: 0.8312 - val_loss: 0.5211 - val_accuracy: 0.8224
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4709
- accuracy: 0.8354 - val_loss: 0.5184 - val_accuracy: 0.8192
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4557
- accuracy: 0.8408 - val_loss: 0.5077 - val_accuracy: 0.8255
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4532
- accuracy: 0.8401 - val_loss: 0.5390 - val_accuracy: 0.8175
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4448
- accuracy: 0.8446 - val_loss: 0.5323 - val_accuracy: 0.8225
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4325
- accuracy: 0.8480 - val_loss: 0.5033 - val_accuracy: 0.8287
Epoch 24/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4199
- accuracy: 0.8543 - val_loss: 0.5617 - val_accuracy: 0.8111
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4122
- accuracy: 0.8563 - val_loss: 0.5117 - val_accuracy: 0.8285
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4045
- accuracy: 0.8584 - val_loss: 0.6122 - val_accuracy: 0.7999
Epoch 27/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3958
- accuracy: 0.8618 - val_loss: 0.5663 - val_accuracy: 0.8147
Epoch 28/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3954
- accuracy: 0.8619 - val_loss: 0.5063 - val_accuracy: 0.8329
```

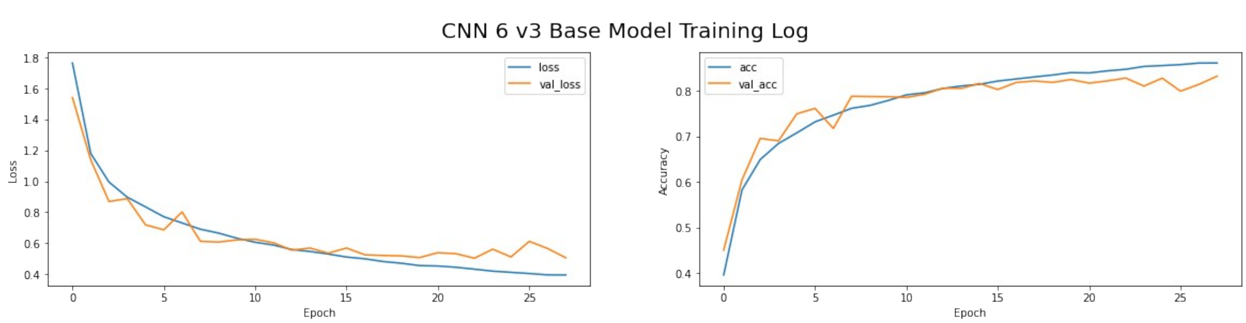
```
preds = CNN_6_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_6_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      1000
           1       0.92      0.93      0.92      1000
           2       0.80      0.73      0.76      1000
           3       0.76      0.65      0.70      1000
           4       0.72      0.87      0.79      1000
           5       0.78      0.75      0.76      1000
```

```
           6       0.77      0.92      0.84      1000
           7       0.89      0.86      0.88      1000
           8       0.92      0.88      0.90      1000
           9       0.92      0.89      0.90      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

313/313 - 1s - loss: 0.5340 - accuracy: 0.8313
Accuracy: 83.13000202178955
Macro F1-score: 0.8305555884405095
```

By increasing the dense layer value to 83%m we can see that there is a huge increase in accuracy and f1 score 83%

```python
loss = CNN_6_v3_history.history['loss']
val_loss = CNN_6_v3_history.history['val_loss']
acc = CNN_6_v3_history.history['accuracy']
val_acc = CNN_6_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 6 v3 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



From the model training log, we can see that the model training has slight underfitting. However, it is better than the previous model as the deviation from the validation and training loss and accuracy is very less. Hence, I will be keeping this model and tryin out a higher dense layer value

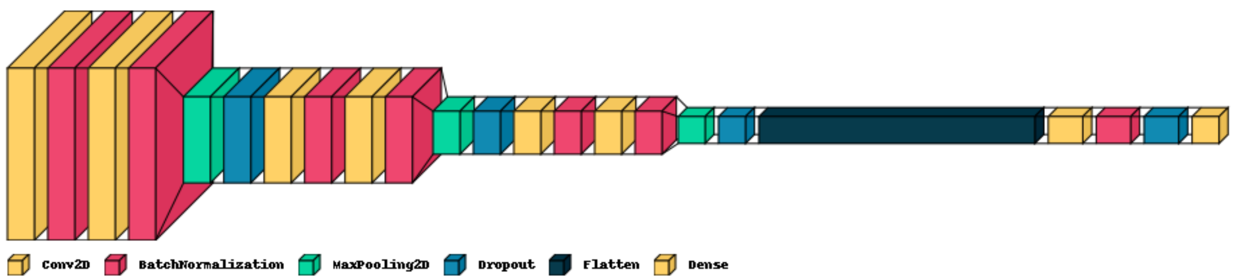
3.2.4.1.3 Dense layer Value - 256

```
CNN_6_v4=build_CNN_6(256)
CNN_6_v4.summary()
```

```
Model: "sequential_88"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_522 (Conv2D)          (None, 32, 32, 32)        320       
_________________________________________________________________
batch_normalization_417 (Bat (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_523 (Conv2D)          (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_418 (Bat (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_261 (MaxPoolin (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_318 (Dropout)        (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_524 (Conv2D)          (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_419 (Bat (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_525 (Conv2D)          (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_420 (Bat (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_262 (MaxPoolin (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_319 (Dropout)        (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_526 (Conv2D)          (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_421 (Bat (None, 8, 8, 128)         512       
_________________________________________________________________
conv2d_527 (Conv2D)          (None, 8, 8, 128)         147584    
_________________________________________________________________
batch_normalization_422 (Bat (None, 8, 8, 128)         512       
_________________________________________________________________
max_pooling2d_263 (MaxPoolin (None, 4, 4, 128)         0         
_________________________________________________________________
dropout_320 (Dropout)        (None, 4, 4, 128)         0         
_________________________________________________________________
```

```
flatten_86 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_176 (Dense)            (None, 256)               524544
_________________________________________________________________
batch_normalization_423 (Bat (None, 256)               1024
_________________________________________________________________
dropout_321 (Dropout)        (None, 256)               0
_________________________________________________________________
dense_177 (Dense)            (None, 10)                2570
=================================================================
Total params: 816,362
Trainable params: 814,954
Non-trainable params: 1,408
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_6_v4, legend=True)
```




```
CNN_6_v4.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_6_v4_history = CNN_6_v4.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.7290
- accuracy: 0.4094 - val_loss: 1.7631 - val_accuracy: 0.3955
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1596
- accuracy: 0.5915 - val_loss: 1.1528 - val_accuracy: 0.5970
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9825
- accuracy: 0.6552 - val_loss: 0.9324 - val_accuracy: 0.6790
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8827
- accuracy: 0.6908 - val_loss: 0.8680 - val_accuracy: 0.6987
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8227
- accuracy: 0.7135 - val_loss: 0.9776 - val_accuracy: 0.6661
Epoch 6/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.7715
- accuracy: 0.7288 - val_loss: 0.6664 - val_accuracy: 0.7682
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7210
- accuracy: 0.7478 - val_loss: 0.6762 - val_accuracy: 0.7690
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6890
- accuracy: 0.7603 - val_loss: 0.6297 - val_accuracy: 0.7840
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6537
- accuracy: 0.7724 - val_loss: 0.6646 - val_accuracy: 0.7721
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6195
- accuracy: 0.7818 - val_loss: 0.6073 - val_accuracy: 0.7941
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5980
- accuracy: 0.7907 - val_loss: 0.6798 - val_accuracy: 0.7692
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5681
- accuracy: 0.8006 - val_loss: 0.5636 - val_accuracy: 0.8058
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5460
- accuracy: 0.8100 - val_loss: 0.5621 - val_accuracy: 0.8068
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5291
- accuracy: 0.8148 - val_loss: 0.5579 - val_accuracy: 0.8122
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5148
- accuracy: 0.8185 - val_loss: 0.6467 - val_accuracy: 0.7843
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5000
- accuracy: 0.8249 - val_loss: 0.5430 - val_accuracy: 0.8122
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4804
- accuracy: 0.8327 - val_loss: 0.5490 - val_accuracy: 0.8143
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4638
- accuracy: 0.8382 - val_loss: 0.5234 - val_accuracy: 0.8243
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4514
- accuracy: 0.8408 - val_loss: 0.5281 - val_accuracy: 0.8220
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4401
- accuracy: 0.8454 - val_loss: 0.5194 - val_accuracy: 0.8262
Epoch 21/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4259
- accuracy: 0.8499 - val_loss: 0.5576 - val_accuracy: 0.8146
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4183
```

```
- accuracy: 0.8527 - val_loss: 0.5554 - val_accuracy: 0.8155
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4102
- accuracy: 0.8541 - val_loss: 0.4936 - val_accuracy: 0.8352
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3956
- accuracy: 0.8605 - val_loss: 0.5433 - val_accuracy: 0.8242
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3934
- accuracy: 0.8606 - val_loss: 0.5234 - val_accuracy: 0.8250
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3794
- accuracy: 0.8658 - val_loss: 0.5100 - val_accuracy: 0.8316
Epoch 27/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3653
- accuracy: 0.8704 - val_loss: 0.5117 - val_accuracy: 0.8299
Epoch 28/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3681
- accuracy: 0.8705 - val_loss: 0.4933 - val_accuracy: 0.8384
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3603
- accuracy: 0.8717 - val_loss: 0.4924 - val_accuracy: 0.8383
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3501
- accuracy: 0.8766 - val_loss: 0.4873 - val_accuracy: 0.8410
Epoch 31/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3502
- accuracy: 0.8742 - val_loss: 0.5678 - val_accuracy: 0.8174
Epoch 32/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3436
- accuracy: 0.8786 - val_loss: 0.4920 - val_accuracy: 0.8402
Epoch 33/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3320
- accuracy: 0.8833 - val_loss: 0.5108 - val_accuracy: 0.8355
Epoch 34/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3264
- accuracy: 0.8824 - val_loss: 0.5351 - val_accuracy: 0.8307
Epoch 35/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3235
- accuracy: 0.8848 - val_loss: 0.5375 - val_accuracy: 0.8286
```

```
preds = CNN_6_v4.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_6_v4.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support
```

```
              0       0.92      0.76      0.83      1000
              1       0.95      0.89      0.92      1000
              2       0.83      0.71      0.76      1000
              3       0.67      0.71      0.69      1000
              4       0.76      0.85      0.80      1000
              5       0.73      0.80      0.76      1000
              6       0.84      0.91      0.87      1000
              7       0.93      0.83      0.87      1000
              8       0.93      0.88      0.91      1000
              9       0.81      0.95      0.87      1000

    accuracy                          0.83     10000
   macro avg       0.84      0.83      0.83     10000
weighted avg       0.84      0.83      0.83     10000

313/313 - 1s - loss: 0.5532 - accuracy: 0.8282
Accuracy: 82.81999826431274
Macro F1-score: 0.8287833937717825
```

We can see that by increasing the dense layer value to 256 the model perfromance has decreased slightly.

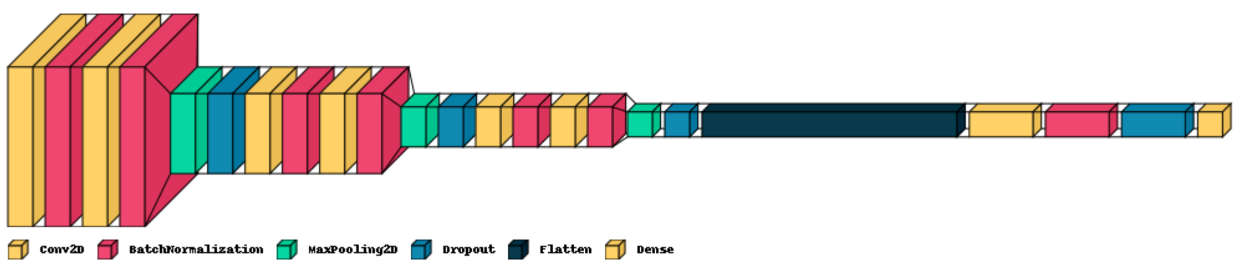
```
loss = CNN_6_v4_history.history['loss']
val_loss = CNN_6_v4_history.history['val_loss']
acc = CNN_6_v4_history.history['accuracy']
val_acc = CNN_6_v4_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 6 v4 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

However, having a higher dense layer value has resulted in a higher deviation from the training and validation loss/acc as compared to the previous model

3.2.4.1.4 Dense layer Value - 512

```
CNN_6_v5=build_CNN_6(512)
CNN_6_v5.summary()

Model: "sequential_89"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_528 (Conv2D)          (None, 32, 32, 32)        320       
_________________________________________________________________
batch_normalization_424 (Bat (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_529 (Conv2D)          (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_425 (Bat (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_264 (MaxPoolin (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_322 (Dropout)        (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_530 (Conv2D)          (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_426 (Bat (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_531 (Conv2D)          (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_427 (Bat (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_265 (MaxPoolin (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_323 (Dropout)        (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_532 (Conv2D)          (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_428 (Bat (None, 8, 8, 128)         512       
_________________________________________________________________
```

```
conv2d_533 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_429 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_266 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_324 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_87 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_178 (Dense)            (None, 512)               1049088
_________________________________________________________________
batch_normalization_430 (Bat (None, 512)               2048
_________________________________________________________________
dropout_325 (Dropout)        (None, 512)               0
_________________________________________________________________
dense_179 (Dense)            (None, 10)                5130
=================================================================
Total params: 1,344,490
Trainable params: 1,342,570
Non-trainable params: 1,920
_________________________________________________________________

visualkeras.layered_view(CNN_6_v5, legend=True)
```




```
CNN_6_v5.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_6_v5_history = CNN_6_v5.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.8113
- accuracy: 0.3970 - val_loss: 2.1160 - val_accuracy: 0.3196
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1893
- accuracy: 0.5842 - val_loss: 1.1796 - val_accuracy: 0.5982
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9837
- accuracy: 0.6561 - val_loss: 1.0892 - val_accuracy: 0.6239
```

```
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8823
- accuracy: 0.6899 - val_loss: 0.8136 - val_accuracy: 0.7159
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8139
- accuracy: 0.7144 - val_loss: 0.7222 - val_accuracy: 0.7537
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7623
- accuracy: 0.7337 - val_loss: 0.7885 - val_accuracy: 0.7318
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7137
- accuracy: 0.7506 - val_loss: 0.7397 - val_accuracy: 0.7419
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6703
- accuracy: 0.7668 - val_loss: 0.8373 - val_accuracy: 0.7192
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6425
- accuracy: 0.7764 - val_loss: 0.6755 - val_accuracy: 0.7685
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6052
- accuracy: 0.7881 - val_loss: 0.6151 - val_accuracy: 0.7890
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5831
- accuracy: 0.7958 - val_loss: 0.6052 - val_accuracy: 0.7919
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5465
- accuracy: 0.8090 - val_loss: 0.7149 - val_accuracy: 0.7629
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5322
- accuracy: 0.8122 - val_loss: 0.8152 - val_accuracy: 0.7251
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5098
- accuracy: 0.8204 - val_loss: 0.6628 - val_accuracy: 0.7793
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5005
- accuracy: 0.8256 - val_loss: 0.7398 - val_accuracy: 0.7550
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4690
- accuracy: 0.8361 - val_loss: 0.5523 - val_accuracy: 0.8109
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4601
- accuracy: 0.8393 - val_loss: 0.5564 - val_accuracy: 0.8108
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4428
- accuracy: 0.8418 - val_loss: 0.5669 - val_accuracy: 0.8152
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4298
- accuracy: 0.8489 - val_loss: 0.5558 - val_accuracy: 0.8110
Epoch 20/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 0.4116
- accuracy: 0.8560 - val_loss: 0.5074 - val_accuracy: 0.8273
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4066
- accuracy: 0.8548 - val_loss: 0.5962 - val_accuracy: 0.8029
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3915
- accuracy: 0.8608 - val_loss: 0.5466 - val_accuracy: 0.8186
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3818
- accuracy: 0.8666 - val_loss: 0.5198 - val_accuracy: 0.8295
Epoch 24/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3709
- accuracy: 0.8687 - val_loss: 0.5282 - val_accuracy: 0.8253
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3591
- accuracy: 0.8719 - val_loss: 0.4985 - val_accuracy: 0.8356
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3570
- accuracy: 0.8740 - val_loss: 0.5202 - val_accuracy: 0.8300
Epoch 27/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3408
- accuracy: 0.8783 - val_loss: 0.4884 - val_accuracy: 0.8409
Epoch 28/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3327
- accuracy: 0.8823 - val_loss: 0.5186 - val_accuracy: 0.8347
Epoch 29/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3299
- accuracy: 0.8804 - val_loss: 0.5592 - val_accuracy: 0.8210
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3258
- accuracy: 0.8835 - val_loss: 0.5658 - val_accuracy: 0.8199
Epoch 31/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3135
- accuracy: 0.8885 - val_loss: 0.5186 - val_accuracy: 0.8311
Epoch 32/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3075
- accuracy: 0.8898 - val_loss: 0.4974 - val_accuracy: 0.8391
```

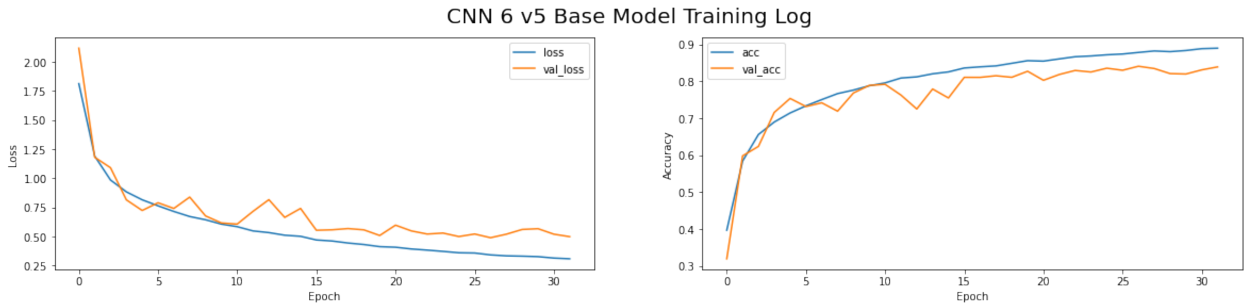
```
preds = CNN_6_v5.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_6_v5.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1000
           1       0.95      0.91      0.93      1000
```

```
           2       0.82      0.71      0.76      1000
           3       0.68      0.72      0.70      1000
           4       0.76      0.85      0.81      1000
           5       0.82      0.71      0.76      1000
           6       0.83      0.88      0.86      1000
           7       0.89      0.87      0.88      1000
           8       0.86      0.94      0.90      1000
           9       0.89      0.92      0.91      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000

313/313 - 1s - loss: 0.5087 - accuracy: 0.8363
Accuracy: 83.63000154495239
Macro F1-score: 0.8357599331433068
```

We can see that having a dense layer value of 512, has resulted in the highest model performance of 84% for f1 score.

```
loss = CNN_6_v5_history.history['loss']
val_loss = CNN_6_v5_history.history['val_loss']
acc = CNN_6_v5_history.history['accuracy']
val_acc = CNN_6_v5_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 6 v5 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



However, having a higher dense layer value has resulted in a higher deviation from the training and validation loss/acc as compared to the CNN 6 v5

Let's compare the summary of all model results:

Conclusion: CNN 6 v3 F1-Score is 83%

For activation function, I will be tuning in a way where by I have created a function with 2 parameters, the first one is activation function for all Conv2d layers while the second acivtation function for second last Dense layer

3.2.5.1 Tuning Activation Function

```
def build_CNN_7(f1,f2):
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation=f1,
padding='same',input_shape=(32,32,1)))
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation=f1, padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.3))

    model.add(Conv2D(64,(3,3),activation=f1, padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation=f1, padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.4))


    model.add(Conv2D(128,(3,3),activation=f1, padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation=f1, padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.5))

    model.add(Flatten())
```

```
    model.add(Dense(128,activation=f2))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(Dense(10,activation='softmax'))
    return model
```

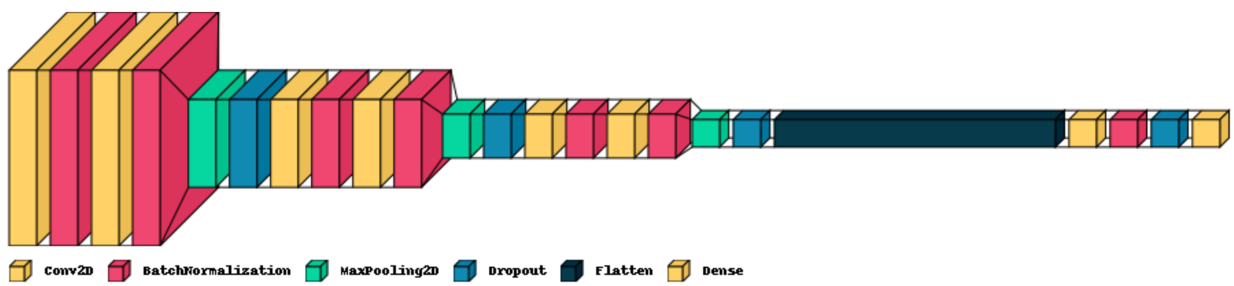
3.2.5.1.1 Relu for Conv2d , Relu for Dense

```
CNN_7_v1=build_CNN_7('relu','relu')
CNN_7_v1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_14 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_15 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_16 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_17 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_18 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_19 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
```

```
dropout_10 (Dropout)         (None, 4, 4, 128)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)             0
_________________________________________________________________
dense_4 (Dense)              (None, 128)              262272
_________________________________________________________________
batch_normalization_20 (Batc (None, 128)              512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)               1290
=================================================================
Total params: 552,298
Trainable params: 551,146
Non-trainable params: 1,152
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_7_v1, legend=True)
```




```
CNN_7_v1.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_7_v1_history = CNN_7_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.7623
- accuracy: 0.3968 - val_loss: 1.8305 - val_accuracy: 0.3748
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1960
- accuracy: 0.5756 - val_loss: 1.2228 - val_accuracy: 0.5792
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0028
- accuracy: 0.6492 - val_loss: 0.9488 - val_accuracy: 0.6654
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9035
- accuracy: 0.6861 - val_loss: 0.8404 - val_accuracy: 0.7066
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8256
```

```
- accuracy: 0.7119 - val_loss: 0.7568 - val_accuracy: 0.7403
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7759
- accuracy: 0.7321 - val_loss: 0.7217 - val_accuracy: 0.7517
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7329
- accuracy: 0.7458 - val_loss: 0.6337 - val_accuracy: 0.7829
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6855
- accuracy: 0.7630 - val_loss: 0.7005 - val_accuracy: 0.7578
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6596
- accuracy: 0.7717 - val_loss: 0.6533 - val_accuracy: 0.7689
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6295
- accuracy: 0.7810 - val_loss: 0.5910 - val_accuracy: 0.7949
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6075
- accuracy: 0.7885 - val_loss: 0.5966 - val_accuracy: 0.7947
Epoch 12/500
625/625 [==============================] - 5s 7ms/step - loss: 0.5814
- accuracy: 0.7987 - val_loss: 0.5976 - val_accuracy: 0.7977
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5578
- accuracy: 0.8058 - val_loss: 0.5414 - val_accuracy: 0.8115
Epoch 14/500
625/625 [==============================] - 5s 7ms/step - loss: 0.5408
- accuracy: 0.8119 - val_loss: 0.5412 - val_accuracy: 0.8125
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5236
- accuracy: 0.8170 - val_loss: 0.5597 - val_accuracy: 0.8129
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5130
- accuracy: 0.8223 - val_loss: 0.6057 - val_accuracy: 0.7944
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4986
- accuracy: 0.8274 - val_loss: 0.5051 - val_accuracy: 0.8277
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4777
- accuracy: 0.8337 - val_loss: 0.5632 - val_accuracy: 0.8098
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4712
- accuracy: 0.8352 - val_loss: 0.5606 - val_accuracy: 0.8156
Epoch 20/500
625/625 [==============================] - 5s 7ms/step - loss: 0.4654
- accuracy: 0.8371 - val_loss: 0.5429 - val_accuracy: 0.8195
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4501
- accuracy: 0.8425 - val_loss: 0.5781 - val_accuracy: 0.8112
```

```
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4360
- accuracy: 0.8476 - val_loss: 0.5311 - val_accuracy: 0.8216
```

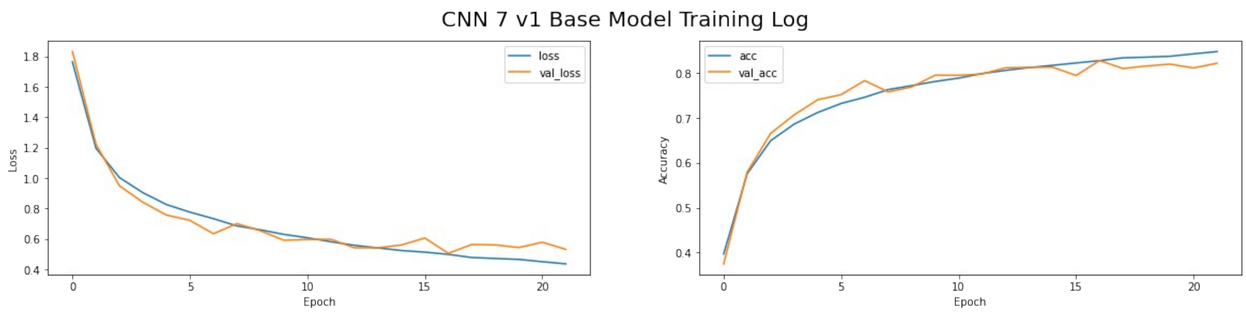
```
preds = CNN_7_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_7_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      1000
           1       0.94      0.90      0.92      1000
           2       0.80      0.69      0.74      1000
           3       0.73      0.60      0.66      1000
           4       0.76      0.83      0.79      1000
           5       0.74      0.77      0.75      1000
           6       0.74      0.92      0.82      1000
           7       0.91      0.85      0.88      1000
           8       0.84      0.94      0.89      1000
           9       0.85      0.94      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000

313/313 - 1s - loss: 0.5608 - accuracy: 0.8188
Accuracy: 81.87999725341797
Macro F1-score: 0.8166377155527705
```

We can see that accuracy is decent at 81% when both dense andc conv2d layer activation function is relu

```
loss = CNN_7_v1_history.history['loss']
val_loss = CNN_7_v1_history.history['val_loss']
acc = CNN_7_v1_history.history['accuracy']
val_acc = CNN_7_v1_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 7 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
```

```
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



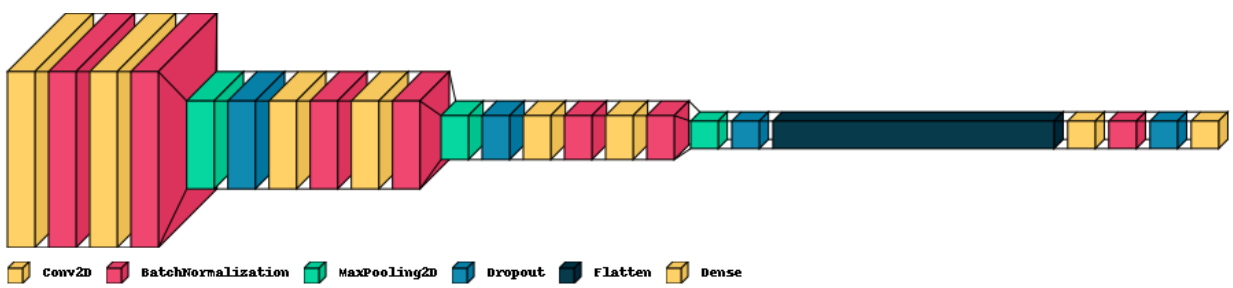
We can also see that the model training log is good as there is no underfitting and overfitting observed

3.2.5.1.2 Tanh for Conv2d , Tanh for Dense

```
CNN_7_v2=build_CNN_7('tanh','tanh')
CNN_7_v2.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_18 (Conv2D)           (None, 32, 32, 32)        320       
_________________________________________________________________
batch_normalization_21 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_22 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_23 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_24 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
```

```
max_pooling2d_10 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_25 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_26 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_14 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               262272
_________________________________________________________________
batch_normalization_27 (Batc (None, 128)               512
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 552,298
Trainable params: 551,146
Non-trainable params: 1,152
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_7_v2, legend=True)
```




```
CNN_7_v2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
CNN_7_v2_history = CNN_7_v2.fit(X_train, y_train, epochs=500, batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test, y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 8ms/step - loss: 1.9765
- accuracy: 0.3030 - val_loss: 2.4714 - val_accuracy: 0.2601
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.5315
- accuracy: 0.4464 - val_loss: 1.4323 - val_accuracy: 0.4954
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.3612
- accuracy: 0.5214 - val_loss: 1.2568 - val_accuracy: 0.5657
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.2828
- accuracy: 0.5482 - val_loss: 1.2221 - val_accuracy: 0.5703
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.2184
- accuracy: 0.5764 - val_loss: 1.1085 - val_accuracy: 0.6078
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1786
- accuracy: 0.5885 - val_loss: 1.2480 - val_accuracy: 0.5625
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1504
- accuracy: 0.6008 - val_loss: 1.2154 - val_accuracy: 0.5837
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1178
- accuracy: 0.6109 - val_loss: 1.2055 - val_accuracy: 0.5988
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0837
- accuracy: 0.6198 - val_loss: 1.3697 - val_accuracy: 0.5384
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 1.0754
- accuracy: 0.6256 - val_loss: 1.0483 - val_accuracy: 0.6325
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0509
- accuracy: 0.6324 - val_loss: 1.3408 - val_accuracy: 0.5642
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0505
- accuracy: 0.6338 - val_loss: 1.2424 - val_accuracy: 0.5899
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 1.0232
- accuracy: 0.6426 - val_loss: 1.0698 - val_accuracy: 0.6289
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0143
- accuracy: 0.6478 - val_loss: 0.9983 - val_accuracy: 0.6482
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0084
- accuracy: 0.6468 - val_loss: 1.1897 - val_accuracy: 0.5923
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9894
- accuracy: 0.6554 - val_loss: 1.2305 - val_accuracy: 0.5932
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9855
```

```
- accuracy: 0.6565 - val_loss: 0.9029 - val_accuracy: 0.6855
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9739
- accuracy: 0.6613 - val_loss: 1.0783 - val_accuracy: 0.6274
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9693
- accuracy: 0.6627 - val_loss: 0.9270 - val_accuracy: 0.6789
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9607
- accuracy: 0.6645 - val_loss: 0.9865 - val_accuracy: 0.6568
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9564
- accuracy: 0.6676 - val_loss: 1.0454 - val_accuracy: 0.6348
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9501
- accuracy: 0.6704 - val_loss: 0.9670 - val_accuracy: 0.6695
```

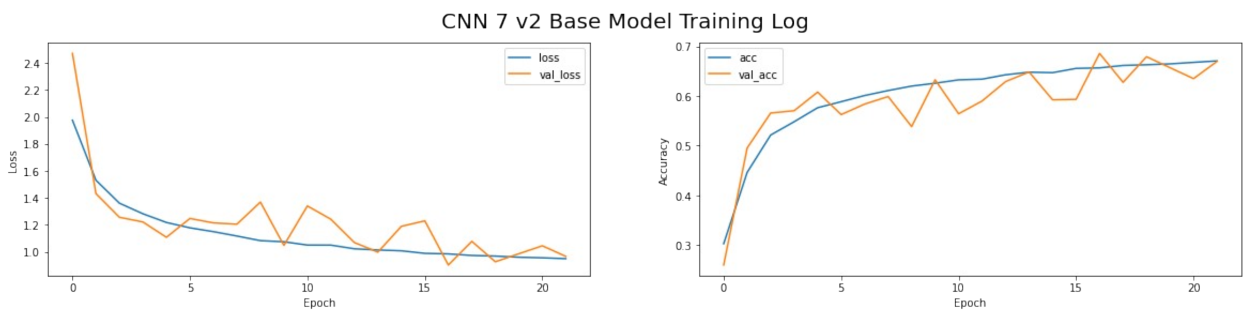
```
preds = CNN_7_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_7_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.71      0.67      0.69      1000
           1       0.79      0.82      0.80      1000
           2       0.60      0.46      0.52      1000
           3       0.59      0.35      0.44      1000
           4       0.62      0.59      0.60      1000
           5       0.64      0.52      0.57      1000
           6       0.68      0.80      0.73      1000
           7       0.61      0.78      0.68      1000
           8       0.76      0.79      0.77      1000
           9       0.65      0.91      0.76      1000

    accuracy                           0.67     10000
   macro avg       0.67      0.67      0.66     10000
weighted avg       0.67      0.67      0.66     10000

313/313 - 1s - loss: 0.9790 - accuracy: 0.6689
Accuracy: 66.89000129699707
Macro F1-score: 0.6586422145661863
```

When both Conv2d and dense activation is set as 'tanh', we can see that the model performance decreased alot to 65%. Hence, We will not be using this activation function combination

```
loss = CNN_7_v2_history.history['loss']
val_loss = CNN_7_v2_history.history['val_loss']
```

```
acc = CNN_7_v2_history.history['accuracy']
val_acc = CNN_7_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 7 v2 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



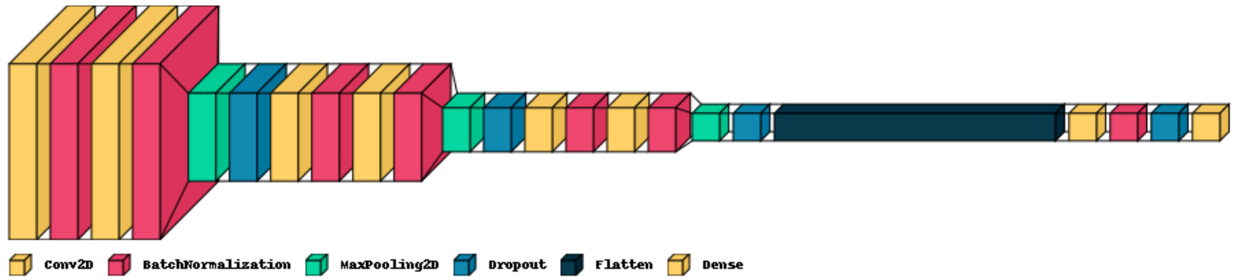
For model training log, we can see that there is are major fluctuations between test and train acc/loss coupled with a high loss, and low accuracy

3.2.5.1.3 Sigmoid for Conv2d , Sigmoid for Dense

```
CNN_7_v3=build_CNN_7('sigmoid','sigmoid')
CNN_7_v3.summary()

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization_28 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_29 (Batc (None, 32, 32, 32)        128
```

```
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_30 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_31 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_17 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_32 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_33 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_18 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               262272
_________________________________________________________________
batch_normalization_34 (Batc (None, 128)               512
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 552,298
Trainable params: 551,146
Non-trainable params: 1,152
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_7_v3, legend=True)
```




```
CNN_7_v3.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_7_v3_history = CNN_7_v3.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 8ms/step - loss: 1.8309
- accuracy: 0.3619 - val_loss: 1.8156 - val_accuracy: 0.4047
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.3094
- accuracy: 0.5401 - val_loss: 1.0830 - val_accuracy: 0.6149
Epoch 3/500
625/625 [==============================] - 5s 7ms/step - loss: 1.1507
- accuracy: 0.5988 - val_loss: 1.3705 - val_accuracy: 0.5243
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0678
- accuracy: 0.6292 - val_loss: 1.1446 - val_accuracy: 0.6048
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0027
- accuracy: 0.6532 - val_loss: 1.0672 - val_accuracy: 0.6360
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9506
- accuracy: 0.6715 - val_loss: 0.9627 - val_accuracy: 0.6767
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9123
- accuracy: 0.6841 - val_loss: 1.0899 - val_accuracy: 0.6280
Epoch 8/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8749
- accuracy: 0.6985 - val_loss: 0.9019 - val_accuracy: 0.6905
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8502
- accuracy: 0.7048 - val_loss: 0.8849 - val_accuracy: 0.6931
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8235
- accuracy: 0.7158 - val_loss: 0.9167 - val_accuracy: 0.7000
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8036
- accuracy: 0.7259 - val_loss: 1.0764 - val_accuracy: 0.6622
Epoch 12/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.7854
- accuracy: 0.7296 - val_loss: 0.8225 - val_accuracy: 0.7143
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7689
- accuracy: 0.7326 - val_loss: 0.7782 - val_accuracy: 0.7349
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7573
- accuracy: 0.7403 - val_loss: 0.8123 - val_accuracy: 0.7289
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7447
- accuracy: 0.7421 - val_loss: 0.8548 - val_accuracy: 0.7092
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7251
- accuracy: 0.7504 - val_loss: 0.8203 - val_accuracy: 0.7343
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7181
- accuracy: 0.7539 - val_loss: 0.6995 - val_accuracy: 0.7650
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7074
- accuracy: 0.7570 - val_loss: 0.7399 - val_accuracy: 0.7469
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6932
- accuracy: 0.7592 - val_loss: 0.6831 - val_accuracy: 0.7668
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6847
- accuracy: 0.7618 - val_loss: 0.7584 - val_accuracy: 0.7453
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6825
- accuracy: 0.7662 - val_loss: 0.7623 - val_accuracy: 0.7472
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6698
- accuracy: 0.7714 - val_loss: 0.7344 - val_accuracy: 0.7579
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6655
- accuracy: 0.7704 - val_loss: 0.7478 - val_accuracy: 0.7538
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6559
- accuracy: 0.7712 - val_loss: 0.8076 - val_accuracy: 0.7379
```

```
preds = CNN_7_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_7_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

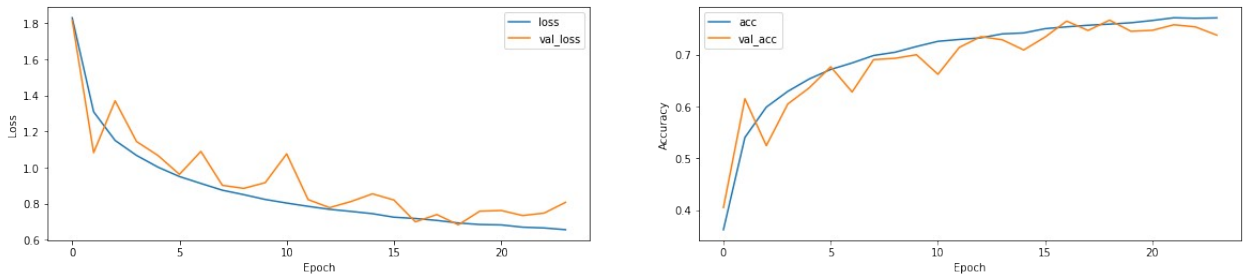
```
              precision    recall  f1-score   support

           0       0.86      0.66      0.74      1000
           1       0.92      0.80      0.86      1000
```

```
           2       0.76      0.52      0.62      1000
           3       0.58      0.56      0.57      1000
           4       0.61      0.84      0.71      1000
           5       0.79      0.51      0.62      1000
           6       0.86      0.75      0.80      1000
           7       0.77      0.82      0.80      1000
           8       0.72      0.92      0.80      1000
           9       0.64      0.95      0.76      1000

    accuracy                           0.73     10000
   macro avg       0.75      0.73      0.73     10000
weighted avg       0.75      0.73      0.73     10000

313/313 - 1s - loss: 0.8193 - accuracy: 0.7321
Accuracy: 73.21000099182129
Macro F1-score: 0.7278821093685516
```

When both Conv2d and dense activation is set as 'sigmoid', we can see that the model performance decreased alot to 73%. Hence, We will not be using this activation function combination

```
loss = CNN_7_v3_history.history['loss']
val_loss = CNN_7_v3_history.history['val_loss']
acc = CNN_7_v3_history.history['accuracy']
val_acc = CNN_7_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 7 v3 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

CNN 7 v3 Base Model Training Log



For model training log, we can see that there is are major fluctuations between test and train acc/loss coupled with a high loss, and low accuracy

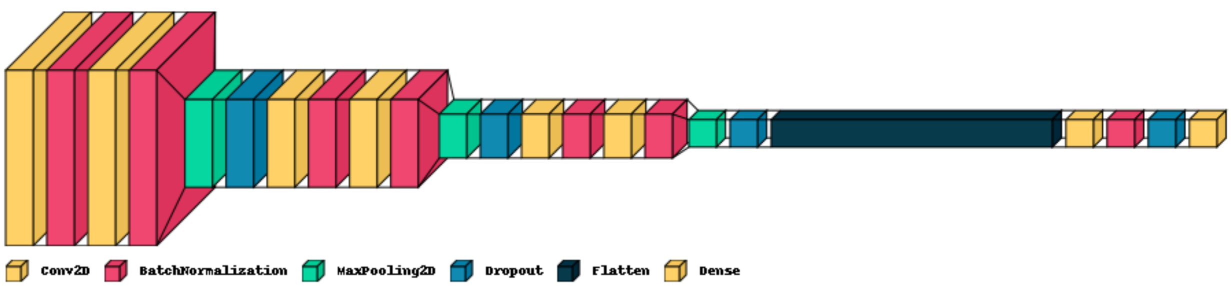
3.2.5.1.4 Relu for Conv2d , Tanh for Dense

```
CNN_7_v4=build_CNN_7('relu','tanh')
CNN_7_v4.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_30 (Conv2D)           (None, 32, 32, 32)        320       
_________________________________________________________________
batch_normalization_35 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_36 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_20 (Dropout)         (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_37 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_38 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_21 (Dropout)         (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_39 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
```

```
conv2d_35 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_40 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_22 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               262272
_________________________________________________________________
batch_normalization_41 (Batc (None, 128)               512
_________________________________________________________________
dropout_23 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
Total params: 552,298
Trainable params: 551,146
Non-trainable params: 1,152
_________________________________________________________________
```

```
visualkeras.layered_view(CNN_7_v4, legend=True)
```




```
CNN_7_v4.compile(optimizer='adam',loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_7_v4_history = CNN_7_v4.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 8ms/step - loss: 1.3655
- accuracy: 0.5168 - val_loss: 1.3036 - val_accuracy: 0.5483
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1321
- accuracy: 0.6037 - val_loss: 1.0564 - val_accuracy: 0.6322
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0251
```

```
- accuracy: 0.6432 - val_loss: 0.9883 - val_accuracy: 0.6603
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9651
- accuracy: 0.6664 - val_loss: 0.9058 - val_accuracy: 0.6796
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9194
- accuracy: 0.6823 - val_loss: 0.9449 - val_accuracy: 0.6769
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8822
- accuracy: 0.6973 - val_loss: 0.8897 - val_accuracy: 0.6950
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8400
- accuracy: 0.7105 - val_loss: 0.7615 - val_accuracy: 0.7364
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8153
- accuracy: 0.7214 - val_loss: 0.7439 - val_accuracy: 0.7440
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7910
- accuracy: 0.7274 - val_loss: 0.7283 - val_accuracy: 0.7498
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7677
- accuracy: 0.7376 - val_loss: 0.6951 - val_accuracy: 0.7667
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7498
- accuracy: 0.7424 - val_loss: 0.8254 - val_accuracy: 0.7087
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7320
- accuracy: 0.7498 - val_loss: 0.7476 - val_accuracy: 0.7516
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7200
- accuracy: 0.7553 - val_loss: 0.7183 - val_accuracy: 0.7566
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7063
- accuracy: 0.7577 - val_loss: 0.7570 - val_accuracy: 0.7501
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6934
- accuracy: 0.7645 - val_loss: 0.6687 - val_accuracy: 0.7743
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6767
- accuracy: 0.7697 - val_loss: 0.6223 - val_accuracy: 0.7849
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6695
- accuracy: 0.7733 - val_loss: 0.6245 - val_accuracy: 0.7923
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6479
- accuracy: 0.7800 - val_loss: 0.6395 - val_accuracy: 0.7826
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6430
- accuracy: 0.7806 - val_loss: 0.6767 - val_accuracy: 0.7718
```

```
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6345
- accuracy: 0.7854 - val_loss: 0.5937 - val_accuracy: 0.8014
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6269
- accuracy: 0.7865 - val_loss: 0.6084 - val_accuracy: 0.7973
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6241
- accuracy: 0.7886 - val_loss: 0.6239 - val_accuracy: 0.7932
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6123
- accuracy: 0.7920 - val_loss: 0.5893 - val_accuracy: 0.8026
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6016
- accuracy: 0.7940 - val_loss: 0.6292 - val_accuracy: 0.7875
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6001
- accuracy: 0.7958 - val_loss: 0.7042 - val_accuracy: 0.7595
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5884
- accuracy: 0.7991 - val_loss: 0.5917 - val_accuracy: 0.8025
Epoch 27/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5837
- accuracy: 0.8012 - val_loss: 0.5965 - val_accuracy: 0.8004
Epoch 28/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5737
- accuracy: 0.8030 - val_loss: 0.6030 - val_accuracy: 0.7987
```

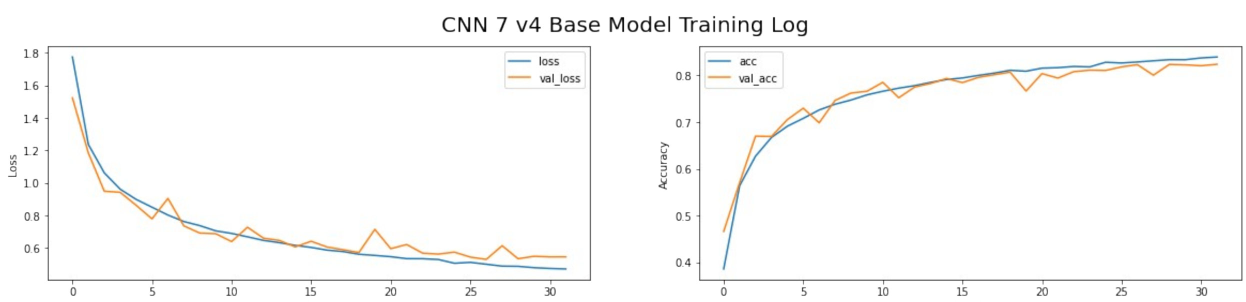
```
preds = CNN_7_v4.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_7_v4.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.89      0.75      0.81      1000
           1       0.91      0.92      0.92      1000
           2       0.75      0.73      0.74      1000
           3       0.73      0.64      0.68      1000
           4       0.72      0.83      0.77      1000
           5       0.77      0.75      0.76      1000
           6       0.83      0.87      0.85      1000
           7       0.84      0.90      0.87      1000
           8       0.89      0.89      0.89      1000
           9       0.86      0.91      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
```

```
weighted avg       0.82      0.82      0.82     10000

313/313 - 1s - loss: 0.5622 - accuracy: 0.8194
Accuracy: 81.94000124931335
Macro F1-score: 0.8181551904943317
```

For this combination of activation function, it gives a decent model perfromance where the f1-score is at 82%.

```
loss = CNN_7_v4_history.history['loss']
val_loss = CNN_7_v4_history.history['val_loss']
acc = CNN_7_v4_history.history['accuracy']
val_acc = CNN_7_v4_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 7 v4 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As for the model training log, we can see that this combination has allowed a good and effective training while mantianing the model performance. We can also see that the model training log is good as there is no underfitting and overfitting observed

Let's compare the summary of all model results:

Even though CNN 7v1 and CNN 7 v3 has almost similar model performance, I will be using CNN 7v3 as it has slightly higher model performance of 0.002

Conclusion: CNN 7 v3 F1-Score is 82%

3.3 Tuning Hyperparameters in compile function

As mentioned in the previous section, after much tuning of layers and the corresponding value, now I will be using CNN 7 v3 with the highest model performance with vert sligh underftting. In this section, I would be tuning the hyperparameters in the neural network, specifically parameters in the compile function.

An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy. You can use different optimizers to make changes in your weights and learning rate. However, choosing the best optimizer depends upon the application.

```
def build_CNN_8():
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.3))

    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.4))


    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(128,activation='tanh'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(Dense(10,activation='softmax'))
    return model

def tuning_optimizers(params,results_dict):
    for i in range(len(params)):
        CNN_8_v1=build_CNN_8()
```

```
CNN_8_v1.compile(optimizer=params[i],loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
        result=CNN_8_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
        preds = CNN_8_v1.predict(X_test)

results_dict[params[i]]=round(f1_score(y_test,preds.argmax(axis=1),ave
rage="macro"),3)
    return results_dict
results_dict={}
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam',
'Adamax', 'Nadam']
print(tuning_optimizers(optimizer,results_dict))
```

```
Epoch 1/500
625/625 [==============================] - 6s 8ms/step - loss: 2.1966
- accuracy: 0.2740 - val_loss: 1.8890 - val_accuracy: 0.3255
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.6805
- accuracy: 0.3974 - val_loss: 1.7497 - val_accuracy: 0.3659
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.5054
- accuracy: 0.4579 - val_loss: 1.9929 - val_accuracy: 0.3172
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.3902
- accuracy: 0.5036 - val_loss: 1.6902 - val_accuracy: 0.4087
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.3040
- accuracy: 0.5404 - val_loss: 1.1617 - val_accuracy: 0.5901
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 1.2279
- accuracy: 0.5658 - val_loss: 1.3321 - val_accuracy: 0.5278
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1799
- accuracy: 0.5832 - val_loss: 1.1335 - val_accuracy: 0.6033
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1338
- accuracy: 0.5982 - val_loss: 1.3569 - val_accuracy: 0.5279
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0931
- accuracy: 0.6140 - val_loss: 1.1080 - val_accuracy: 0.6139
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0636
- accuracy: 0.6260 - val_loss: 1.2373 - val_accuracy: 0.5659
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0282
- accuracy: 0.6373 - val_loss: 0.9488 - val_accuracy: 0.6644
```

```
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9991
- accuracy: 0.6493 - val_loss: 0.9637 - val_accuracy: 0.6596
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9749
- accuracy: 0.6579 - val_loss: 1.0817 - val_accuracy: 0.6266
Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9612
- accuracy: 0.6598 - val_loss: 0.9733 - val_accuracy: 0.6591
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9276
- accuracy: 0.6742 - val_loss: 0.9478 - val_accuracy: 0.6687
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9120
- accuracy: 0.6796 - val_loss: 0.9432 - val_accuracy: 0.6732
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8931
- accuracy: 0.6866 - val_loss: 0.8541 - val_accuracy: 0.6999
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8724
- accuracy: 0.6948 - val_loss: 0.8165 - val_accuracy: 0.7185
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8682
- accuracy: 0.6964 - val_loss: 0.9251 - val_accuracy: 0.6766
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8532
- accuracy: 0.7020 - val_loss: 0.7507 - val_accuracy: 0.7397
Epoch 21/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8375
- accuracy: 0.7049 - val_loss: 0.8970 - val_accuracy: 0.6897
Epoch 22/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8265
- accuracy: 0.7083 - val_loss: 0.7596 - val_accuracy: 0.7350
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8133
- accuracy: 0.7161 - val_loss: 0.8078 - val_accuracy: 0.7219
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8073
- accuracy: 0.7155 - val_loss: 0.7829 - val_accuracy: 0.7276
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7893
- accuracy: 0.7229 - val_loss: 0.6857 - val_accuracy: 0.7618
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7785
- accuracy: 0.7276 - val_loss: 0.7559 - val_accuracy: 0.7358
Epoch 27/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7677
- accuracy: 0.7311 - val_loss: 0.7251 - val_accuracy: 0.7472
Epoch 28/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.7630
- accuracy: 0.7290 - val_loss: 0.7014 - val_accuracy: 0.7562
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7470
- accuracy: 0.7389 - val_loss: 0.6592 - val_accuracy: 0.7716
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7401
- accuracy: 0.7390 - val_loss: 0.6996 - val_accuracy: 0.7592
Epoch 31/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7252
- accuracy: 0.7464 - val_loss: 0.6441 - val_accuracy: 0.7765
Epoch 32/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7245
- accuracy: 0.7458 - val_loss: 0.7516 - val_accuracy: 0.7427
Epoch 33/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7163
- accuracy: 0.7479 - val_loss: 0.6452 - val_accuracy: 0.7763
Epoch 34/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7034
- accuracy: 0.7562 - val_loss: 0.6964 - val_accuracy: 0.7632
Epoch 35/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6988
- accuracy: 0.7549 - val_loss: 0.6876 - val_accuracy: 0.7567
Epoch 36/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6916
- accuracy: 0.7585 - val_loss: 0.6504 - val_accuracy: 0.7728
Epoch 1/500
625/625 [==============================] - 7s 10ms/step - loss: 1.6335
- accuracy: 0.4430 - val_loss: 1.3171 - val_accuracy: 0.5504
Epoch 2/500
625/625 [==============================] - 6s 10ms/step - loss: 1.0737
- accuracy: 0.6265 - val_loss: 1.0006 - val_accuracy: 0.6433
Epoch 3/500
625/625 [==============================] - 6s 10ms/step - loss: 0.9174
- accuracy: 0.6802 - val_loss: 0.8644 - val_accuracy: 0.6973
Epoch 4/500
625/625 [==============================] - 6s 10ms/step - loss: 0.8230
- accuracy: 0.7142 - val_loss: 0.7736 - val_accuracy: 0.7284
Epoch 5/500
625/625 [==============================] - 6s 10ms/step - loss: 0.7591
- accuracy: 0.7369 - val_loss: 0.6954 - val_accuracy: 0.7592
Epoch 6/500
625/625 [==============================] - 6s 10ms/step - loss: 0.7177
- accuracy: 0.7523 - val_loss: 0.7200 - val_accuracy: 0.7560
Epoch 7/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6758
- accuracy: 0.7666 - val_loss: 0.6506 - val_accuracy: 0.7746
Epoch 8/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6410
```

```
- accuracy: 0.7796 - val_loss: 0.5863 - val_accuracy: 0.7970
Epoch 9/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6154
- accuracy: 0.7879 - val_loss: 0.5929 - val_accuracy: 0.7988
Epoch 10/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5896
- accuracy: 0.7955 - val_loss: 0.6484 - val_accuracy: 0.7794
Epoch 11/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5672
- accuracy: 0.8040 - val_loss: 0.6134 - val_accuracy: 0.7906
Epoch 12/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5530
- accuracy: 0.8093 - val_loss: 0.6182 - val_accuracy: 0.7973
Epoch 13/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5382
- accuracy: 0.8147 - val_loss: 0.6208 - val_accuracy: 0.7917
Epoch 1/500
625/625 [==============================] - 6s 8ms/step - loss: 2.6473
- accuracy: 0.2015 - val_loss: 2.6325 - val_accuracy: 0.1752
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 2.2421
- accuracy: 0.2773 - val_loss: 1.9344 - val_accuracy: 0.3078
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 2.0848
- accuracy: 0.3128 - val_loss: 1.9193 - val_accuracy: 0.3138
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.9904
- accuracy: 0.3295 - val_loss: 1.9494 - val_accuracy: 0.3075
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.9311
- accuracy: 0.3463 - val_loss: 1.9547 - val_accuracy: 0.3125
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 1.8783
- accuracy: 0.3578 - val_loss: 1.9748 - val_accuracy: 0.3116
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 1.8425
- accuracy: 0.3686 - val_loss: 1.9688 - val_accuracy: 0.3186
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 1.8124
- accuracy: 0.3772 - val_loss: 1.9928 - val_accuracy: 0.3189

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 3.2623
- accuracy: 0.1037 - val_loss: 2.5404 - val_accuracy: 0.1105
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 3.1949
- accuracy: 0.1084 - val_loss: 2.5178 - val_accuracy: 0.1254
Epoch 3/500
625/625 [==============================] - 5s 9ms/step - loss: 3.1404
- accuracy: 0.1149 - val_loss: 2.4874 - val_accuracy: 0.1347
```

```
Epoch 4/500
625/625 [==============================] - 5s 9ms/step - loss: 3.1091
- accuracy: 0.1181 - val_loss: 2.4605 - val_accuracy: 0.1370
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 3.0633
- accuracy: 0.1252 - val_loss: 2.4337 - val_accuracy: 0.1420
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 3.0257
- accuracy: 0.1292 - val_loss: 2.4179 - val_accuracy: 0.1425
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 2.9880
- accuracy: 0.1335 - val_loss: 2.4034 - val_accuracy: 0.1439
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 2.9570
- accuracy: 0.1422 - val_loss: 2.3919 - val_accuracy: 0.1474
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 2.9019
- accuracy: 0.1462 - val_loss: 2.3718 - val_accuracy: 0.1528
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 2.8896
- accuracy: 0.1524 - val_loss: 2.3624 - val_accuracy: 0.1551
Epoch 11/500
625/625 [==============================] - 5s 9ms/step - loss: 2.8553
- accuracy: 0.1574 - val_loss: 2.3493 - val_accuracy: 0.1611
Epoch 12/500
625/625 [==============================] - 5s 9ms/step - loss: 2.8150
- accuracy: 0.1630 - val_loss: 2.3449 - val_accuracy: 0.1636
Epoch 13/500
625/625 [==============================] - 5s 9ms/step - loss: 2.8022
- accuracy: 0.1669 - val_loss: 2.3323 - val_accuracy: 0.1669
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 2.7708
- accuracy: 0.1696 - val_loss: 2.3206 - val_accuracy: 0.1715
Epoch 15/500
625/625 [==============================] - 5s 9ms/step - loss: 2.7468
- accuracy: 0.1750 - val_loss: 2.3149 - val_accuracy: 0.1726
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 2.7467
- accuracy: 0.1738 - val_loss: 2.3002 - val_accuracy: 0.1762
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 2.7122
- accuracy: 0.1848 - val_loss: 2.2950 - val_accuracy: 0.1783
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 2.6794
- accuracy: 0.1865 - val_loss: 2.2796 - val_accuracy: 0.1837
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 2.6667
- accuracy: 0.1902 - val_loss: 2.2761 - val_accuracy: 0.1848
Epoch 20/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 2.6586
- accuracy: 0.1946 - val_loss: 2.2726 - val_accuracy: 0.1863
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 2.6478
- accuracy: 0.1957 - val_loss: 2.2689 - val_accuracy: 0.1898
Epoch 22/500
625/625 [==============================] - 5s 9ms/step - loss: 2.6180
- accuracy: 0.2032 - val_loss: 2.2618 - val_accuracy: 0.1927
Epoch 23/500
625/625 [==============================] - 5s 9ms/step - loss: 2.6114
- accuracy: 0.2012 - val_loss: 2.2478 - val_accuracy: 0.1976
Epoch 24/500
625/625 [==============================] - 6s 9ms/step - loss: 2.5823
- accuracy: 0.2061 - val_loss: 2.2404 - val_accuracy: 0.2004
Epoch 25/500
625/625 [==============================] - 6s 9ms/step - loss: 2.5702
- accuracy: 0.2092 - val_loss: 2.2353 - val_accuracy: 0.2021
Epoch 26/500
625/625 [==============================] - 6s 9ms/step - loss: 2.5503
- accuracy: 0.2140 - val_loss: 2.2314 - val_accuracy: 0.2048
Epoch 27/500
625/625 [==============================] - 6s 9ms/step - loss: 2.5342
- accuracy: 0.2153 - val_loss: 2.2280 - val_accuracy: 0.2080
Epoch 28/500
625/625 [==============================] - 5s 9ms/step - loss: 2.5355
- accuracy: 0.2164 - val_loss: 2.2185 - val_accuracy: 0.2099
Epoch 29/500
625/625 [==============================] - 6s 9ms/step - loss: 2.5230
- accuracy: 0.2207 - val_loss: 2.2139 - val_accuracy: 0.2130
Epoch 30/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4850
- accuracy: 0.2262 - val_loss: 2.2064 - val_accuracy: 0.2131
Epoch 31/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4912
- accuracy: 0.2269 - val_loss: 2.2102 - val_accuracy: 0.2127
Epoch 32/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4779
- accuracy: 0.2327 - val_loss: 2.1995 - val_accuracy: 0.2166
Epoch 33/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4611
- accuracy: 0.2328 - val_loss: 2.1910 - val_accuracy: 0.2181
Epoch 34/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4431
- accuracy: 0.2350 - val_loss: 2.1834 - val_accuracy: 0.2215
Epoch 35/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4283
- accuracy: 0.2398 - val_loss: 2.1858 - val_accuracy: 0.2199
Epoch 36/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4198
```

```
- accuracy: 0.2397 - val_loss: 2.1806 - val_accuracy: 0.2215
Epoch 37/500
625/625 [==============================] - 5s 9ms/step - loss: 2.4139
- accuracy: 0.2410 - val_loss: 2.1711 - val_accuracy: 0.2256
Epoch 38/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3954
- accuracy: 0.2465 - val_loss: 2.1765 - val_accuracy: 0.2245
Epoch 39/500
625/625 [==============================] - 6s 9ms/step - loss: 2.3854
- accuracy: 0.2444 - val_loss: 2.1707 - val_accuracy: 0.2277
Epoch 40/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3778
- accuracy: 0.2462 - val_loss: 2.1703 - val_accuracy: 0.2286
Epoch 41/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3770
- accuracy: 0.2497 - val_loss: 2.1590 - val_accuracy: 0.2317
Epoch 42/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3749
- accuracy: 0.2521 - val_loss: 2.1555 - val_accuracy: 0.2321
Epoch 43/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3532
- accuracy: 0.2546 - val_loss: 2.1468 - val_accuracy: 0.2347
Epoch 44/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3450
- accuracy: 0.2563 - val_loss: 2.1407 - val_accuracy: 0.2369
Epoch 45/500
625/625 [==============================] - 5s 9ms/step - loss: 2.3267
- accuracy: 0.2601 - val_loss: 2.1455 - val_accuracy: 0.2350
Epoch 46/500
625/625 [==============================] - 6s 9ms/step - loss: 2.3234
- accuracy: 0.2628 - val_loss: 2.1429 - val_accuracy: 0.2367
Epoch 47/500
625/625 [==============================] - 6s 9ms/step - loss: 2.3208
- accuracy: 0.2619 - val_loss: 2.1352 - val_accuracy: 0.2396
Epoch 48/500
625/625 [==============================] - 6s 9ms/step - loss: 2.3077
- accuracy: 0.2619 - val_loss: 2.1305 - val_accuracy: 0.2415
Epoch 49/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2928
- accuracy: 0.2682 - val_loss: 2.1355 - val_accuracy: 0.2408
Epoch 50/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2805
- accuracy: 0.2707 - val_loss: 2.1139 - val_accuracy: 0.2477
Epoch 51/500
625/625 [==============================] - 6s 9ms/step - loss: 2.2775
- accuracy: 0.2704 - val_loss: 2.1257 - val_accuracy: 0.2455
Epoch 52/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2656
- accuracy: 0.2686 - val_loss: 2.1226 - val_accuracy: 0.2448
```

```
Epoch 53/500
625/625 [==============================] - 6s 9ms/step - loss: 2.2538
- accuracy: 0.2762 - val_loss: 2.1207 - val_accuracy: 0.2466
Epoch 54/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2485
- accuracy: 0.2736 - val_loss: 2.1111 - val_accuracy: 0.2491
Epoch 55/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2413
- accuracy: 0.2750 - val_loss: 2.1154 - val_accuracy: 0.2486
Epoch 56/500
625/625 [==============================] - 5s 9ms/step - loss: 2.2297
- accuracy: 0.2772 - val_loss: 2.1036 - val_accuracy: 0.2513
Epoch 57/500
625/625 [==============================] - 6s 9ms/step - loss: 2.2210
- accuracy: 0.2811 - val_loss: 2.1089 - val_accuracy: 0.2514

Epoch 58/500
625/625 [==============================] - 6s 9ms/step - loss: 2.1998
- accuracy: 0.2837 - val_loss: 2.1038 - val_accuracy: 0.2528
Epoch 59/500
625/625 [==============================] - 6s 9ms/step - loss: 2.2039
- accuracy: 0.2819 - val_loss: 2.1056 - val_accuracy: 0.2538
Epoch 60/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1941
- accuracy: 0.2900 - val_loss: 2.1024 - val_accuracy: 0.2550
Epoch 61/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1916
- accuracy: 0.2885 - val_loss: 2.0974 - val_accuracy: 0.2561
Epoch 62/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1887
- accuracy: 0.2841 - val_loss: 2.0993 - val_accuracy: 0.2548
Epoch 63/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1723
- accuracy: 0.2925 - val_loss: 2.0823 - val_accuracy: 0.2597
Epoch 64/500
625/625 [==============================] - 6s 9ms/step - loss: 2.1648
- accuracy: 0.2947 - val_loss: 2.0853 - val_accuracy: 0.2597
Epoch 65/500
625/625 [==============================] - 6s 9ms/step - loss: 2.1572
- accuracy: 0.2942 - val_loss: 2.0902 - val_accuracy: 0.2582
Epoch 66/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1601
- accuracy: 0.2952 - val_loss: 2.0853 - val_accuracy: 0.2595
Epoch 67/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1428
- accuracy: 0.2959 - val_loss: 2.0814 - val_accuracy: 0.2621
Epoch 68/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1321
- accuracy: 0.2999 - val_loss: 2.0758 - val_accuracy: 0.2635
Epoch 69/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 2.1294
- accuracy: 0.3017 - val_loss: 2.0773 - val_accuracy: 0.2634
Epoch 70/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1261
- accuracy: 0.3033 - val_loss: 2.0836 - val_accuracy: 0.2622
Epoch 71/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1140
- accuracy: 0.3028 - val_loss: 2.0740 - val_accuracy: 0.2645
Epoch 72/500
625/625 [==============================] - 5s 9ms/step - loss: 2.1041
- accuracy: 0.3073 - val_loss: 2.0643 - val_accuracy: 0.2677
Epoch 73/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0959
- accuracy: 0.3064 - val_loss: 2.0696 - val_accuracy: 0.2666
Epoch 74/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0870
- accuracy: 0.3104 - val_loss: 2.0688 - val_accuracy: 0.2671
Epoch 75/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0834
- accuracy: 0.3120 - val_loss: 2.0661 - val_accuracy: 0.2680
Epoch 76/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0835
- accuracy: 0.3113 - val_loss: 2.0667 - val_accuracy: 0.2669
Epoch 77/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0653
- accuracy: 0.3182 - val_loss: 2.0570 - val_accuracy: 0.2705
Epoch 78/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0620
- accuracy: 0.3171 - val_loss: 2.0534 - val_accuracy: 0.2712
Epoch 79/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0582
- accuracy: 0.3172 - val_loss: 2.0470 - val_accuracy: 0.2743
Epoch 80/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0546
- accuracy: 0.3185 - val_loss: 2.0524 - val_accuracy: 0.2713
Epoch 81/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0566
- accuracy: 0.3183 - val_loss: 2.0531 - val_accuracy: 0.2714
Epoch 82/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0434
- accuracy: 0.3202 - val_loss: 2.0492 - val_accuracy: 0.2722
Epoch 83/500
625/625 [==============================] - 6s 9ms/step - loss: 2.0336
- accuracy: 0.3229 - val_loss: 2.0487 - val_accuracy: 0.2733
Epoch 84/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0371
- accuracy: 0.3221 - val_loss: 2.0461 - val_accuracy: 0.2745
Epoch 85/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0273
```

```
- accuracy: 0.3229 - val_loss: 2.0345 - val_accuracy: 0.2784
Epoch 86/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0190
- accuracy: 0.3225 - val_loss: 2.0333 - val_accuracy: 0.2796
Epoch 87/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0164
- accuracy: 0.3241 - val_loss: 2.0383 - val_accuracy: 0.2773
Epoch 88/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0177
- accuracy: 0.3279 - val_loss: 2.0295 - val_accuracy: 0.2812
Epoch 89/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0089
- accuracy: 0.3282 - val_loss: 2.0285 - val_accuracy: 0.2815
Epoch 90/500
625/625 [==============================] - 5s 9ms/step - loss: 2.0049
- accuracy: 0.3247 - val_loss: 2.0339 - val_accuracy: 0.2805
Epoch 91/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9985
- accuracy: 0.3308 - val_loss: 2.0315 - val_accuracy: 0.2803
Epoch 92/500
625/625 [==============================] - 6s 10ms/step - loss: 1.9934
- accuracy: 0.3289 - val_loss: 2.0187 - val_accuracy: 0.2839
Epoch 93/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9915
- accuracy: 0.3334 - val_loss: 2.0199 - val_accuracy: 0.2855
Epoch 94/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9836
- accuracy: 0.3329 - val_loss: 2.0129 - val_accuracy: 0.2868
Epoch 95/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9778
- accuracy: 0.3333 - val_loss: 2.0135 - val_accuracy: 0.2877
Epoch 96/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9718
- accuracy: 0.3375 - val_loss: 2.0035 - val_accuracy: 0.2885
Epoch 97/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9678
- accuracy: 0.3386 - val_loss: 2.0093 - val_accuracy: 0.2884
Epoch 98/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9598
- accuracy: 0.3405 - val_loss: 2.0082 - val_accuracy: 0.2892
Epoch 99/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9569
- accuracy: 0.3375 - val_loss: 2.0127 - val_accuracy: 0.2889
Epoch 100/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9540
- accuracy: 0.3423 - val_loss: 2.0098 - val_accuracy: 0.2891
Epoch 101/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9523
- accuracy: 0.3421 - val_loss: 2.0147 - val_accuracy: 0.2888
```

```
Epoch 1/500
625/625 [==============================] - 9s 9ms/step - loss: 1.7577
- accuracy: 0.3940 - val_loss: 1.6815 - val_accuracy: 0.4079
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1889
- accuracy: 0.5792 - val_loss: 1.1282 - val_accuracy: 0.6053
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9887
- accuracy: 0.6551 - val_loss: 0.8771 - val_accuracy: 0.6956
Epoch 4/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8929
- accuracy: 0.6878 - val_loss: 0.7612 - val_accuracy: 0.7339
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8317
- accuracy: 0.7103 - val_loss: 0.7347 - val_accuracy: 0.7474
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7732
- accuracy: 0.7314 - val_loss: 0.7606 - val_accuracy: 0.7419
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7300
- accuracy: 0.7488 - val_loss: 0.7056 - val_accuracy: 0.7575
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6950
- accuracy: 0.7595 - val_loss: 0.6576 - val_accuracy: 0.7722
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6585
- accuracy: 0.7707 - val_loss: 0.6565 - val_accuracy: 0.7742
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6370
- accuracy: 0.7788 - val_loss: 0.7272 - val_accuracy: 0.7564
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6080
- accuracy: 0.7894 - val_loss: 0.7105 - val_accuracy: 0.7656
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5800
- accuracy: 0.7996 - val_loss: 0.5887 - val_accuracy: 0.7930
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5583
- accuracy: 0.8062 - val_loss: 0.5764 - val_accuracy: 0.8032

Epoch 14/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5424
- accuracy: 0.8117 - val_loss: 0.5665 - val_accuracy: 0.8086
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5274
- accuracy: 0.8152 - val_loss: 0.5944 - val_accuracy: 0.7991
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5135
- accuracy: 0.8213 - val_loss: 0.5298 - val_accuracy: 0.8167
Epoch 17/500
```

```
625/625 [==============================] - 5s 8ms/step - loss: 0.4986
- accuracy: 0.8253 - val_loss: 0.5538 - val_accuracy: 0.8135
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4946
- accuracy: 0.8299 - val_loss: 0.5926 - val_accuracy: 0.8022
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4705
- accuracy: 0.8342 - val_loss: 0.5194 - val_accuracy: 0.8243
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4605
- accuracy: 0.8413 - val_loss: 0.5545 - val_accuracy: 0.8065
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4532
- accuracy: 0.8414 - val_loss: 0.5189 - val_accuracy: 0.8229
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4420
- accuracy: 0.8476 - val_loss: 0.5089 - val_accuracy: 0.8320
Epoch 23/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4300
- accuracy: 0.8499 - val_loss: 0.5057 - val_accuracy: 0.8330
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4270
- accuracy: 0.8499 - val_loss: 0.5027 - val_accuracy: 0.8335
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4129
- accuracy: 0.8546 - val_loss: 0.5045 - val_accuracy: 0.8327
Epoch 26/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4083
- accuracy: 0.8576 - val_loss: 0.6127 - val_accuracy: 0.8008
Epoch 27/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3985
- accuracy: 0.8603 - val_loss: 0.5134 - val_accuracy: 0.8280
Epoch 28/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3928
- accuracy: 0.8610 - val_loss: 0.5590 - val_accuracy: 0.8165
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3906
- accuracy: 0.8609 - val_loss: 0.4989 - val_accuracy: 0.8323
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3843
- accuracy: 0.8648 - val_loss: 0.5023 - val_accuracy: 0.8326
Epoch 31/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3729
- accuracy: 0.8678 - val_loss: 0.5062 - val_accuracy: 0.8359
Epoch 32/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3756
- accuracy: 0.8686 - val_loss: 0.5014 - val_accuracy: 0.8368
Epoch 33/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3612
```

```
- accuracy: 0.8749 - val_loss: 0.5537 - val_accuracy: 0.8223
Epoch 34/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3626
- accuracy: 0.8723 - val_loss: 0.4903 - val_accuracy: 0.8383
Epoch 35/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3526
- accuracy: 0.8772 - val_loss: 0.5163 - val_accuracy: 0.8314
Epoch 36/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3486
- accuracy: 0.8755 - val_loss: 0.5342 - val_accuracy: 0.8281
Epoch 37/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3406
- accuracy: 0.8812 - val_loss: 0.4898 - val_accuracy: 0.8432
Epoch 38/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3344
- accuracy: 0.8819 - val_loss: 0.5277 - val_accuracy: 0.8342
Epoch 39/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3340
- accuracy: 0.8826 - val_loss: 0.5213 - val_accuracy: 0.8380
Epoch 40/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3334
- accuracy: 0.8820 - val_loss: 0.5201 - val_accuracy: 0.8333
Epoch 41/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3283
- accuracy: 0.8835 - val_loss: 0.5487 - val_accuracy: 0.8256
Epoch 42/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3209
- accuracy: 0.8863 - val_loss: 0.5058 - val_accuracy: 0.8406
Epoch 1/500
625/625 [==============================] - 7s 10ms/step - loss: 2.0404
- accuracy: 0.3191 - val_loss: 2.4112 - val_accuracy: 0.2417
Epoch 2/500
625/625 [==============================] - 6s 10ms/step - loss: 1.5186
- accuracy: 0.4624 - val_loss: 1.4867 - val_accuracy: 0.4942
Epoch 3/500
625/625 [==============================] - 6s 10ms/step - loss: 1.2710
- accuracy: 0.5519 - val_loss: 1.4392 - val_accuracy: 0.5012
Epoch 4/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1093
- accuracy: 0.6098 - val_loss: 1.0905 - val_accuracy: 0.6211
Epoch 5/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9979
- accuracy: 0.6514 - val_loss: 0.8660 - val_accuracy: 0.6963
Epoch 6/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9231
- accuracy: 0.6758 - val_loss: 1.0111 - val_accuracy: 0.6445
Epoch 7/500
625/625 [==============================] - 6s 9ms/step - loss: 0.8668
- accuracy: 0.6996 - val_loss: 0.7844 - val_accuracy: 0.7232
```

```
Epoch 8/500
625/625 [==============================] - 6s 9ms/step - loss: 0.8186
- accuracy: 0.7133 - val_loss: 0.7635 - val_accuracy: 0.7349
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7763
- accuracy: 0.7300 - val_loss: 0.7352 - val_accuracy: 0.7454
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7335
- accuracy: 0.7450 - val_loss: 0.6731 - val_accuracy: 0.7658
Epoch 11/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7034
- accuracy: 0.7544 - val_loss: 0.6405 - val_accuracy: 0.7750
Epoch 12/500
625/625 [==============================] - 7s 11ms/step - loss: 0.6759
- accuracy: 0.7643 - val_loss: 0.6210 - val_accuracy: 0.7856
Epoch 13/500
625/625 [==============================] - 7s 11ms/step - loss: 0.6510
- accuracy: 0.7727 - val_loss: 0.6081 - val_accuracy: 0.7873
Epoch 14/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6265
- accuracy: 0.7833 - val_loss: 0.6069 - val_accuracy: 0.7889
Epoch 15/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6064
- accuracy: 0.7894 - val_loss: 0.6008 - val_accuracy: 0.7897
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5895
- accuracy: 0.7952 - val_loss: 0.6262 - val_accuracy: 0.7870
Epoch 17/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5722
- accuracy: 0.8027 - val_loss: 0.6282 - val_accuracy: 0.7876
Epoch 18/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5603
- accuracy: 0.8064 - val_loss: 0.5841 - val_accuracy: 0.7984
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5420
- accuracy: 0.8127 - val_loss: 0.5975 - val_accuracy: 0.7976
Epoch 20/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5326
- accuracy: 0.8166 - val_loss: 0.5789 - val_accuracy: 0.8038
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5125
- accuracy: 0.8221 - val_loss: 0.5258 - val_accuracy: 0.8178
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5035
- accuracy: 0.8260 - val_loss: 0.5601 - val_accuracy: 0.8116
Epoch 23/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4958
- accuracy: 0.8289 - val_loss: 0.6258 - val_accuracy: 0.7888
Epoch 24/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 0.4874
- accuracy: 0.8311 - val_loss: 0.5380 - val_accuracy: 0.8187
Epoch 25/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4679
- accuracy: 0.8350 - val_loss: 0.5241 - val_accuracy: 0.8186
Epoch 26/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4631
- accuracy: 0.8400 - val_loss: 0.5211 - val_accuracy: 0.8264
Epoch 27/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4576
- accuracy: 0.8406 - val_loss: 0.5537 - val_accuracy: 0.8184
Epoch 28/500
625/625 [==============================] - 7s 11ms/step - loss: 0.4448
- accuracy: 0.8451 - val_loss: 0.5516 - val_accuracy: 0.8182

Epoch 29/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4400
- accuracy: 0.8456 - val_loss: 0.5745 - val_accuracy: 0.8102
Epoch 30/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4288
- accuracy: 0.8498 - val_loss: 0.5200 - val_accuracy: 0.8279
Epoch 31/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4213
- accuracy: 0.8529 - val_loss: 0.5130 - val_accuracy: 0.8251
Epoch 32/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4133
- accuracy: 0.8536 - val_loss: 0.5304 - val_accuracy: 0.8259
Epoch 33/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4169
- accuracy: 0.8530 - val_loss: 0.5148 - val_accuracy: 0.8244
Epoch 34/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3998
- accuracy: 0.8605 - val_loss: 0.5189 - val_accuracy: 0.8282
Epoch 35/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3915
- accuracy: 0.8630 - val_loss: 0.5072 - val_accuracy: 0.8312
Epoch 36/500
625/625 [==============================] - 6s 10ms/step - loss: 0.3963
- accuracy: 0.8612 - val_loss: 0.5506 - val_accuracy: 0.8257
Epoch 37/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3832
- accuracy: 0.8651 - val_loss: 0.5142 - val_accuracy: 0.8296
Epoch 38/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3785
- accuracy: 0.8683 - val_loss: 0.5072 - val_accuracy: 0.8319
Epoch 39/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3772
- accuracy: 0.8687 - val_loss: 0.4976 - val_accuracy: 0.8397
Epoch 40/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3670
```

```
- accuracy: 0.8717 - val_loss: 0.4971 - val_accuracy: 0.8381
Epoch 41/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3724
- accuracy: 0.8693 - val_loss: 0.5158 - val_accuracy: 0.8347
Epoch 42/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3604
- accuracy: 0.8717 - val_loss: 0.5306 - val_accuracy: 0.8277
Epoch 43/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3546
- accuracy: 0.8756 - val_loss: 0.5071 - val_accuracy: 0.8365
Epoch 44/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3503
- accuracy: 0.8773 - val_loss: 0.4958 - val_accuracy: 0.8402
Epoch 45/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3457
- accuracy: 0.8780 - val_loss: 0.5077 - val_accuracy: 0.8348
Epoch 46/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3471
- accuracy: 0.8774 - val_loss: 0.4979 - val_accuracy: 0.8378
Epoch 47/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3345
- accuracy: 0.8815 - val_loss: 0.5232 - val_accuracy: 0.8293
Epoch 48/500
625/625 [==============================] - 6s 10ms/step - loss: 0.3329
- accuracy: 0.8845 - val_loss: 0.5344 - val_accuracy: 0.8325
Epoch 49/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3324
- accuracy: 0.8833 - val_loss: 0.5418 - val_accuracy: 0.8303
Epoch 1/500
625/625 [==============================] - 11s 14ms/step - loss:
1.7609 - accuracy: 0.3951 - val_loss: 1.4438 - val_accuracy: 0.4951
Epoch 2/500
625/625 [==============================] - 9s 14ms/step - loss: 1.1936
- accuracy: 0.5795 - val_loss: 1.0836 - val_accuracy: 0.6297
Epoch 3/500
625/625 [==============================] - 9s 14ms/step - loss: 1.0028
- accuracy: 0.6484 - val_loss: 1.0068 - val_accuracy: 0.6539
Epoch 4/500
625/625 [==============================] - 9s 14ms/step - loss: 0.8957
- accuracy: 0.6893 - val_loss: 0.8677 - val_accuracy: 0.6986
Epoch 5/500
625/625 [==============================] - 8s 13ms/step - loss: 0.8189
- accuracy: 0.7157 - val_loss: 1.0198 - val_accuracy: 0.6582
Epoch 6/500
625/625 [==============================] - 8s 13ms/step - loss: 0.7721
- accuracy: 0.7299 - val_loss: 0.6861 - val_accuracy: 0.7596
Epoch 7/500
625/625 [==============================] - 8s 13ms/step - loss: 0.7244
- accuracy: 0.7476 - val_loss: 0.8439 - val_accuracy: 0.7103
```

```
Epoch 8/500
625/625 [==============================] - 8s 13ms/step - loss: 0.6916
- accuracy: 0.7613 - val_loss: 0.6413 - val_accuracy: 0.7793
Epoch 9/500
625/625 [==============================] - 9s 14ms/step - loss: 0.6521
- accuracy: 0.7733 - val_loss: 0.6135 - val_accuracy: 0.7877
Epoch 10/500
625/625 [==============================] - 9s 14ms/step - loss: 0.6233
- accuracy: 0.7864 - val_loss: 0.6518 - val_accuracy: 0.7788
Epoch 11/500
625/625 [==============================] - 9s 14ms/step - loss: 0.6034
- accuracy: 0.7920 - val_loss: 0.5898 - val_accuracy: 0.7998
Epoch 12/500
625/625 [==============================] - 9s 14ms/step - loss: 0.5738
- accuracy: 0.8028 - val_loss: 0.6280 - val_accuracy: 0.7872
Epoch 13/500
625/625 [==============================] - 8s 14ms/step - loss: 0.5606
- accuracy: 0.8060 - val_loss: 0.5901 - val_accuracy: 0.7968
Epoch 14/500
625/625 [==============================] - 9s 14ms/step - loss: 0.5349
- accuracy: 0.8156 - val_loss: 0.5620 - val_accuracy: 0.8080
Epoch 15/500
625/625 [==============================] - 9s 14ms/step - loss: 0.5270
- accuracy: 0.8155 - val_loss: 0.6790 - val_accuracy: 0.7770
Epoch 16/500
625/625 [==============================] - 8s 13ms/step - loss: 0.5125
- accuracy: 0.8203 - val_loss: 0.5465 - val_accuracy: 0.8135
Epoch 17/500
625/625 [==============================] - 8s 14ms/step - loss: 0.4905
- accuracy: 0.8298 - val_loss: 0.5595 - val_accuracy: 0.8093
Epoch 18/500
625/625 [==============================] - 8s 13ms/step - loss: 0.4855
- accuracy: 0.8320 - val_loss: 0.5101 - val_accuracy: 0.8250
Epoch 19/500
625/625 [==============================] - 9s 14ms/step - loss: 0.4675
- accuracy: 0.8379 - val_loss: 0.5051 - val_accuracy: 0.8289
Epoch 20/500
625/625 [==============================] - 10s 16ms/step - loss:
0.4571 - accuracy: 0.8421 - val_loss: 0.5216 - val_accuracy: 0.8236
Epoch 21/500
625/625 [==============================] - 8s 13ms/step - loss: 0.4455
- accuracy: 0.8439 - val_loss: 0.5135 - val_accuracy: 0.8263
Epoch 22/500
625/625 [==============================] - 9s 14ms/step - loss: 0.4347
- accuracy: 0.8487 - val_loss: 0.5654 - val_accuracy: 0.8070
Epoch 23/500
625/625 [==============================] - 9s 15ms/step - loss: 0.4291
- accuracy: 0.8506 - val_loss: 0.5353 - val_accuracy: 0.8255
Epoch 24/500
```

```
625/625 [==============================] - 8s 14ms/step - loss: 0.4207
- accuracy: 0.8539 - val_loss: 0.5128 - val_accuracy: 0.8301
{'SGD': 0.766, 'RMSprop': 0.793, 'Adagrad': 0.308, 'Adadelta': 0.249,
'Adam': 0.832, 'Adamax': 0.828, 'Nadam': 0.825}
```

```
print("F1-score of different optimizers:",{'SGD': 0.766, 'RMSprop':
0.793, 'Adagrad': 0.308, 'Adadelta': 0.249, 'Adam': 0.832, 'Adamax':
0.828, 'Nadam': 0.825})
```

```
F1-score of different optimizers: {'SGD': 0.766, 'RMSprop': 0.793,
'Adagrad': 0.308, 'Adadelta': 0.249, 'Adam': 0.832, 'Adamax': 0.828,
'Nadam': 0.825}
```

After having tried a variety of optimizers, Adam optimizer gives the highest f1-score of 83.2%. Hence, I will be using Adam for the rest of the sections

Now I will be using the Adam Optimizer to compile the built model to record this version of the model

```
CNN_8_v1=build_CNN_8()
CNN_8_v1.compile(optimizer="Adam",loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
CNN_8_v1_history=CNN_8_v1.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.7739
- accuracy: 0.3887 - val_loss: 1.3657 - val_accuracy: 0.5158
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1923
- accuracy: 0.5783 - val_loss: 1.4811 - val_accuracy: 0.5219
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0062
- accuracy: 0.6494 - val_loss: 0.9047 - val_accuracy: 0.6852
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9030
- accuracy: 0.6866 - val_loss: 0.8336 - val_accuracy: 0.7108
Epoch 5/500
625/625 [==============================] - 6s 9ms/step - loss: 0.8381
- accuracy: 0.7096 - val_loss: 0.8846 - val_accuracy: 0.6950
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7774
- accuracy: 0.7316 - val_loss: 0.7397 - val_accuracy: 0.7432
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7352
- accuracy: 0.7441 - val_loss: 0.8908 - val_accuracy: 0.6960
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6959
```

```
- accuracy: 0.7597 - val_loss: 0.6849 - val_accuracy: 0.7597
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6648
- accuracy: 0.7687 - val_loss: 0.6461 - val_accuracy: 0.7745
Epoch 10/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6308
- accuracy: 0.7812 - val_loss: 0.5948 - val_accuracy: 0.7918
Epoch 11/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6066
- accuracy: 0.7894 - val_loss: 0.6247 - val_accuracy: 0.7863
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5831
- accuracy: 0.7980 - val_loss: 0.5773 - val_accuracy: 0.8034
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5597
- accuracy: 0.8037 - val_loss: 0.5687 - val_accuracy: 0.8038
Epoch 14/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5423
- accuracy: 0.8125 - val_loss: 0.5594 - val_accuracy: 0.8097
Epoch 15/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5184
- accuracy: 0.8188 - val_loss: 0.6635 - val_accuracy: 0.7774
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5157
- accuracy: 0.8207 - val_loss: 0.5345 - val_accuracy: 0.8188
Epoch 17/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4940
- accuracy: 0.8262 - val_loss: 0.5522 - val_accuracy: 0.8143
Epoch 18/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4806
- accuracy: 0.8317 - val_loss: 0.5973 - val_accuracy: 0.8023
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4690
- accuracy: 0.8343 - val_loss: 0.5410 - val_accuracy: 0.8140
Epoch 20/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4638
- accuracy: 0.8373 - val_loss: 0.5365 - val_accuracy: 0.8191
Epoch 21/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4458
- accuracy: 0.8438 - val_loss: 0.5224 - val_accuracy: 0.8231
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4379
- accuracy: 0.8470 - val_loss: 0.5100 - val_accuracy: 0.8303
Epoch 23/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4268
- accuracy: 0.8503 - val_loss: 0.5235 - val_accuracy: 0.8266
Epoch 24/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4230
- accuracy: 0.8517 - val_loss: 0.5748 - val_accuracy: 0.8092
```

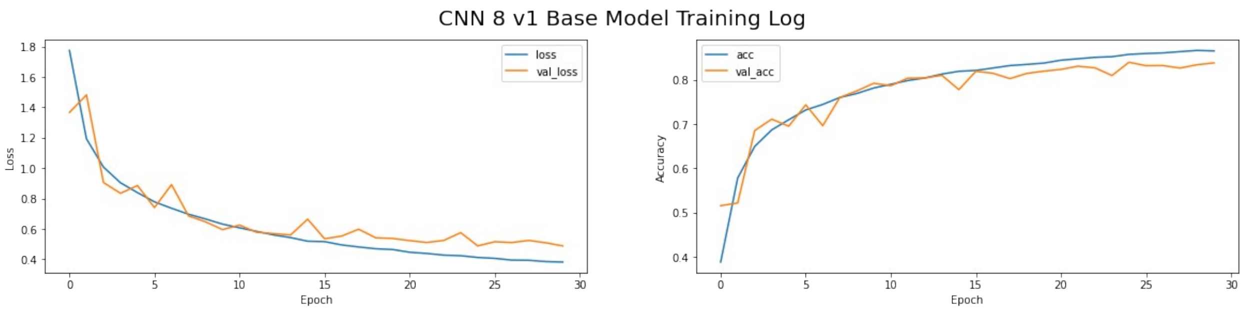
```
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4114
- accuracy: 0.8569 - val_loss: 0.4874 - val_accuracy: 0.8392
Epoch 26/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4056
- accuracy: 0.8589 - val_loss: 0.5149 - val_accuracy: 0.8314
Epoch 27/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3942
- accuracy: 0.8602 - val_loss: 0.5097 - val_accuracy: 0.8317
Epoch 28/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3929
- accuracy: 0.8632 - val_loss: 0.5236 - val_accuracy: 0.8261
Epoch 29/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3845
- accuracy: 0.8658 - val_loss: 0.5080 - val_accuracy: 0.8336
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3815
- accuracy: 0.8648 - val_loss: 0.4875 - val_accuracy: 0.8377
```

```
preds = CNN_8_v1.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v1.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1000
           1       0.93      0.92      0.93      1000
           2       0.85      0.67      0.75      1000
           3       0.78      0.62      0.69      1000
           4       0.75      0.85      0.80      1000
           5       0.73      0.81      0.77      1000
           6       0.88      0.86      0.87      1000
           7       0.82      0.92      0.87      1000
           8       0.91      0.91      0.91      1000
           9       0.88      0.93      0.90      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.83     10000
weighted avg       0.84      0.84      0.83     10000

313/313 - 1s - loss: 0.5231 - accuracy: 0.8354
Accuracy: 83.53999853134155
Macro F1-score: 0.8333226412165653
```

```
loss = CNN_8_v1_history.history['loss']
val_loss = CNN_8_v1_history.history['val_loss']
acc = CNN_8_v1_history.history['accuracy']
```

```
val_acc = CNN_8_v1_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v1 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Next, I will be tuning the learning rate parameter, where I will be trying learning rate ranging from 0.1 to 0.00001 for the Adamx optimizer to see which will increase the current f1-score

3.3.2.1 Learning Rate=0.1

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.1)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 2.1246
- accuracy: 0.2587 - val_loss: 5.2166 - val_accuracy: 0.1919
Epoch 2/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 1.7956
- accuracy: 0.3737 - val_loss: 1.3790 - val_accuracy: 0.5251
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.6197
- accuracy: 0.4536 - val_loss: 1.6773 - val_accuracy: 0.4954
Epoch 4/500
625/625 [==============================] - 5s 9ms/step - loss: 1.4628
- accuracy: 0.5099 - val_loss: 1.7416 - val_accuracy: 0.4663
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.5256
- accuracy: 0.4888 - val_loss: 1.8363 - val_accuracy: 0.4367
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 1.4027
- accuracy: 0.5301 - val_loss: 1.1645 - val_accuracy: 0.5966
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 1.3129
- accuracy: 0.5691 - val_loss: 1.4521 - val_accuracy: 0.5398
Epoch 8/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2626
- accuracy: 0.5948 - val_loss: 1.3993 - val_accuracy: 0.5760
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 1.2228
- accuracy: 0.6096 - val_loss: 1.2087 - val_accuracy: 0.6239
Epoch 10/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2085
- accuracy: 0.6191 - val_loss: 1.0464 - val_accuracy: 0.6550
Epoch 11/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1031
- accuracy: 0.6402 - val_loss: 1.2309 - val_accuracy: 0.6314
Epoch 12/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1073
- accuracy: 0.6480 - val_loss: 1.3849 - val_accuracy: 0.6201
Epoch 13/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1218
- accuracy: 0.6526 - val_loss: 0.8837 - val_accuracy: 0.7076
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 1.0270
- accuracy: 0.6697 - val_loss: 0.9349 - val_accuracy: 0.6973
Epoch 15/500
625/625 [==============================] - 6s 9ms/step - loss: 1.0770
- accuracy: 0.6690 - val_loss: 1.5111 - val_accuracy: 0.5932
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9889
- accuracy: 0.6883 - val_loss: 1.1442 - val_accuracy: 0.6379
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9849
- accuracy: 0.6902 - val_loss: 0.7127 - val_accuracy: 0.7635
Epoch 18/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9550
```

```
- accuracy: 0.7030 - val_loss: 0.9346 - val_accuracy: 0.7185
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 1.0205
- accuracy: 0.6927 - val_loss: 0.8013 - val_accuracy: 0.7558
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8752
- accuracy: 0.7205 - val_loss: 0.8109 - val_accuracy: 0.7406
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9015
- accuracy: 0.7220 - val_loss: 0.7575 - val_accuracy: 0.7667
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9626
- accuracy: 0.7089 - val_loss: 1.1251 - val_accuracy: 0.6822
```

```
preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.68      0.66      0.67      1000
           1       0.90      0.86      0.88      1000
           2       0.60      0.59      0.60      1000
           3       0.72      0.27      0.40      1000
           4       0.55      0.74      0.63      1000
           5       0.84      0.51      0.64      1000
           6       0.96      0.48      0.64      1000
           7       0.68      0.86      0.76      1000
           8       0.48      0.97      0.64      1000
           9       0.85      0.81      0.83      1000

    accuracy                           0.68     10000
   macro avg       0.73      0.68      0.67     10000
weighted avg       0.73      0.68      0.67     10000

313/313 - 1s - loss: 1.1557 - accuracy: 0.6767
Accuracy: 67.66999959945679
Macro F1-score: 0.6684897418173119
```

When learning rate is 0.1, the model f1-score dropped to 67%. Hence, the learning rate may be too high for the model to learn any useful things, hence I will be increasing the learning rate in the next section.

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
```

```
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log [LR=0.1]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

From the model training log, we can see that there are frequent fluctuation from the validation and training loss/acc. There is alos high loss and low accuracy. Hence, we will be lowering the learning rate in the next section for the model to learn more effectively.

3.3.2.2 Learning Rate=0.01

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.01)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.6211
- accuracy: 0.4224 - val_loss: 1.5663 - val_accuracy: 0.5030
Epoch 2/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1373
- accuracy: 0.5996 - val_loss: 1.2593 - val_accuracy: 0.5665
```

```
Epoch 3/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9714
- accuracy: 0.6606 - val_loss: 1.4261 - val_accuracy: 0.5053
Epoch 4/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8897
- accuracy: 0.6931 - val_loss: 0.9513 - val_accuracy: 0.6939
Epoch 5/500
625/625 [==============================] - 6s 9ms/step - loss: 0.8191
- accuracy: 0.7171 - val_loss: 0.7718 - val_accuracy: 0.7362
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7777
- accuracy: 0.7317 - val_loss: 0.7168 - val_accuracy: 0.7556
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7443
- accuracy: 0.7451 - val_loss: 0.8636 - val_accuracy: 0.7186
Epoch 8/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7129
- accuracy: 0.7544 - val_loss: 0.7705 - val_accuracy: 0.7406
Epoch 9/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6833
- accuracy: 0.7667 - val_loss: 0.6383 - val_accuracy: 0.7804
Epoch 10/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6540
- accuracy: 0.7742 - val_loss: 0.7947 - val_accuracy: 0.7490
Epoch 11/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6379
- accuracy: 0.7818 - val_loss: 0.5793 - val_accuracy: 0.7986
Epoch 12/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6171
- accuracy: 0.7896 - val_loss: 0.6049 - val_accuracy: 0.7941
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5970
- accuracy: 0.7952 - val_loss: 0.5816 - val_accuracy: 0.7990
Epoch 14/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5779
- accuracy: 0.8010 - val_loss: 0.5593 - val_accuracy: 0.8121
Epoch 15/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5618
- accuracy: 0.8066 - val_loss: 0.6002 - val_accuracy: 0.7976
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5526
- accuracy: 0.8102 - val_loss: 0.5530 - val_accuracy: 0.8090
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5346
- accuracy: 0.8144 - val_loss: 0.5854 - val_accuracy: 0.8025
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5270
- accuracy: 0.8174 - val_loss: 0.5646 - val_accuracy: 0.8067
Epoch 19/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 0.5123
- accuracy: 0.8236 - val_loss: 0.9153 - val_accuracy: 0.7234
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4989
- accuracy: 0.8298 - val_loss: 0.5309 - val_accuracy: 0.8192
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4903
- accuracy: 0.8310 - val_loss: 0.5905 - val_accuracy: 0.7999
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4859
- accuracy: 0.8344 - val_loss: 0.5675 - val_accuracy: 0.8140
Epoch 23/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4670
- accuracy: 0.8398 - val_loss: 0.7349 - val_accuracy: 0.7698
Epoch 24/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4619
- accuracy: 0.8403 - val_loss: 0.5424 - val_accuracy: 0.8201
Epoch 25/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4560
- accuracy: 0.8441 - val_loss: 0.6326 - val_accuracy: 0.7918
```

```
preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.88      0.73      0.80      1000
           1       0.94      0.88      0.91      1000
           2       0.84      0.60      0.70      1000
           3       0.56      0.74      0.63      1000
           4       0.71      0.81      0.75      1000
           5       0.68      0.78      0.73      1000
           6       0.80      0.89      0.84      1000
           7       0.96      0.69      0.81      1000
           8       0.93      0.82      0.87      1000
           9       0.80      0.94      0.87      1000

    accuracy                           0.79     10000
   macro avg       0.81      0.79      0.79     10000
weighted avg       0.81      0.79      0.79     10000

313/313 - 1s - loss: 0.6515 - accuracy: 0.7882
Accuracy: 78.82000207901001
Macro F1-score: 0.7904081714872032
```

When learning rate was increased to 0.01, the accuracy increased to 78% however, it was still lower than the previous CNN 7v3. Hence, I will still be increasing the learning rate.

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log [LR=0.01]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

From the model training log, we can see that the learning curve is not good as there are frequent deviation from the training loss to the validation loss. Hence, We will still be lowering the learning rate for it learning more effectively

3.3.2.3 Learning Rate=0.001

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.001)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 7s 9ms/step - loss: 1.7560
- accuracy: 0.3928 - val_loss: 1.7854 - val_accuracy: 0.3920
Epoch 2/500
625/625 [==============================] - 6s 9ms/step - loss: 1.1847
- accuracy: 0.5814 - val_loss: 1.0157 - val_accuracy: 0.6448
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0035
- accuracy: 0.6493 - val_loss: 1.0962 - val_accuracy: 0.6131
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9045
- accuracy: 0.6842 - val_loss: 0.8271 - val_accuracy: 0.7104
Epoch 5/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8461
- accuracy: 0.7056 - val_loss: 0.7868 - val_accuracy: 0.7271
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7860
- accuracy: 0.7270 - val_loss: 0.7347 - val_accuracy: 0.7445
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7439
- accuracy: 0.7419 - val_loss: 0.7056 - val_accuracy: 0.7537
Epoch 8/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7040
- accuracy: 0.7562 - val_loss: 0.7497 - val_accuracy: 0.7458
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6757
- accuracy: 0.7671 - val_loss: 0.6040 - val_accuracy: 0.7907
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6480
- accuracy: 0.7752 - val_loss: 0.6115 - val_accuracy: 0.7918
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6202
- accuracy: 0.7871 - val_loss: 0.6417 - val_accuracy: 0.7830
Epoch 12/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5916
- accuracy: 0.7939 - val_loss: 0.5626 - val_accuracy: 0.8066
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5735
- accuracy: 0.8007 - val_loss: 0.6009 - val_accuracy: 0.7964
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5576
- accuracy: 0.8052 - val_loss: 0.6007 - val_accuracy: 0.7963
Epoch 15/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5404
- accuracy: 0.8104 - val_loss: 0.6213 - val_accuracy: 0.7978
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5166
- accuracy: 0.8199 - val_loss: 0.5618 - val_accuracy: 0.8130
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5072
```

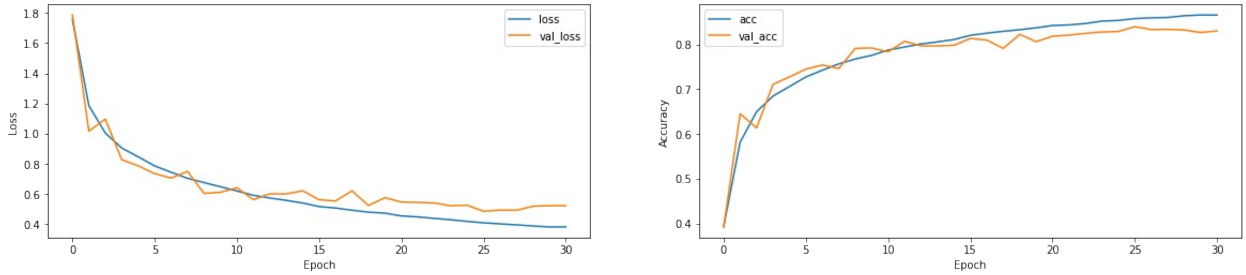
```
- accuracy: 0.8248 - val_loss: 0.5538 - val_accuracy: 0.8095
Epoch 18/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4926
- accuracy: 0.8289 - val_loss: 0.6211 - val_accuracy: 0.7906
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4796
- accuracy: 0.8325 - val_loss: 0.5238 - val_accuracy: 0.8222
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4733
- accuracy: 0.8366 - val_loss: 0.5752 - val_accuracy: 0.8058
Epoch 21/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4546
- accuracy: 0.8419 - val_loss: 0.5467 - val_accuracy: 0.8180
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4483
- accuracy: 0.8432 - val_loss: 0.5442 - val_accuracy: 0.8204
Epoch 23/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4385
- accuracy: 0.8463 - val_loss: 0.5405 - val_accuracy: 0.8244
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4296
- accuracy: 0.8516 - val_loss: 0.5222 - val_accuracy: 0.8273
Epoch 25/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4185
- accuracy: 0.8532 - val_loss: 0.5257 - val_accuracy: 0.8286
Epoch 26/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4092
- accuracy: 0.8573 - val_loss: 0.4854 - val_accuracy: 0.8393
Epoch 27/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4022
- accuracy: 0.8590 - val_loss: 0.4933 - val_accuracy: 0.8329
Epoch 28/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3957
- accuracy: 0.8598 - val_loss: 0.4922 - val_accuracy: 0.8334
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3881
- accuracy: 0.8637 - val_loss: 0.5189 - val_accuracy: 0.8319
Epoch 30/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3818
- accuracy: 0.8653 - val_loss: 0.5222 - val_accuracy: 0.8263
Epoch 31/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3820
- accuracy: 0.8652 - val_loss: 0.5226 - val_accuracy: 0.8298

preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.87      0.79      0.83      1000
           1       0.92      0.93      0.93      1000
           2       0.78      0.72      0.75      1000
           3       0.75      0.61      0.68      1000
           4       0.78      0.82      0.80      1000
           5       0.72      0.80      0.76      1000
           6       0.90      0.83      0.86      1000
           7       0.78      0.93      0.85      1000
           8       0.88      0.92      0.90      1000
           9       0.90      0.91      0.91      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

313/313 - 1s - loss: 0.5384 - accuracy: 0.8270
Accuracy: 82.70000219345093
Macro F1-score: 0.8254859540046618
```

Now that the learning rate is lowered to 0.001, the model performance has inceased back to 82 to 83% range.

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.001]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

CNN 8 v2 Base Model Training Log [LR=0.001]



We can also see from the model training log that the graph is more stable with a smaller gap between the training and validation accuracy/loss. However, I would still be lowering the learning rate to close the gap more.

3.3.2.4 Learning Rate=0.0001

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0001)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 2.4466
- accuracy: 0.2306 - val_loss: 2.3999 - val_accuracy: 0.1716
Epoch 2/500
625/625 [==============================] - 5s 9ms/step - loss: 1.9214
- accuracy: 0.3459 - val_loss: 2.1018 - val_accuracy: 0.2725
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.7221
- accuracy: 0.4011 - val_loss: 2.0953 - val_accuracy: 0.2984
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.5755
- accuracy: 0.4477 - val_loss: 1.6765 - val_accuracy: 0.4199
Epoch 5/500
625/625 [==============================] - 6s 9ms/step - loss: 1.4444
- accuracy: 0.4932 - val_loss: 1.6654 - val_accuracy: 0.4355
Epoch 6/500
625/625 [==============================] - 6s 9ms/step - loss: 1.3379
- accuracy: 0.5329 - val_loss: 1.3682 - val_accuracy: 0.5244
Epoch 7/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2514
- accuracy: 0.5602 - val_loss: 1.3745 - val_accuracy: 0.5286
Epoch 8/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1886
- accuracy: 0.5831 - val_loss: 1.1242 - val_accuracy: 0.6070
Epoch 9/500
```

```
625/625 [==============================] - 5s 9ms/step - loss: 1.1125
- accuracy: 0.6123 - val_loss: 1.1522 - val_accuracy: 0.5995
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0640
- accuracy: 0.6275 - val_loss: 0.9788 - val_accuracy: 0.6651
Epoch 11/500
625/625 [==============================] - 6s 9ms/step - loss: 1.0180
- accuracy: 0.6456 - val_loss: 1.1389 - val_accuracy: 0.6108
Epoch 12/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9844
- accuracy: 0.6546 - val_loss: 0.9764 - val_accuracy: 0.6672
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9432
- accuracy: 0.6720 - val_loss: 0.9088 - val_accuracy: 0.6894
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9152
- accuracy: 0.6822 - val_loss: 0.8193 - val_accuracy: 0.7185
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8834
- accuracy: 0.6930 - val_loss: 0.8911 - val_accuracy: 0.6978
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8576
- accuracy: 0.7019 - val_loss: 0.7968 - val_accuracy: 0.7258
Epoch 17/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8379
- accuracy: 0.7099 - val_loss: 0.8085 - val_accuracy: 0.7227
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.8170
- accuracy: 0.7149 - val_loss: 0.7460 - val_accuracy: 0.7455
Epoch 19/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7988
- accuracy: 0.7205 - val_loss: 0.7850 - val_accuracy: 0.7336
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7788
- accuracy: 0.7293 - val_loss: 0.7601 - val_accuracy: 0.7369
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7605
- accuracy: 0.7351 - val_loss: 0.7436 - val_accuracy: 0.7453
Epoch 22/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7456
- accuracy: 0.7408 - val_loss: 0.6640 - val_accuracy: 0.7693
Epoch 23/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7341
- accuracy: 0.7449 - val_loss: 0.7219 - val_accuracy: 0.7539
Epoch 24/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7128
- accuracy: 0.7538 - val_loss: 0.6988 - val_accuracy: 0.7572
Epoch 25/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6991
```

```
- accuracy: 0.7559 - val_loss: 0.6763 - val_accuracy: 0.7667
Epoch 26/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6821
- accuracy: 0.7638 - val_loss: 0.6573 - val_accuracy: 0.7731
Epoch 27/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6717
- accuracy: 0.7659 - val_loss: 0.6565 - val_accuracy: 0.7698
Epoch 28/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6581
- accuracy: 0.7705 - val_loss: 0.6171 - val_accuracy: 0.7862
Epoch 29/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6477
- accuracy: 0.7749 - val_loss: 0.6331 - val_accuracy: 0.7789
Epoch 30/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6361
- accuracy: 0.7775 - val_loss: 0.6673 - val_accuracy: 0.7716
Epoch 31/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6253
- accuracy: 0.7844 - val_loss: 0.6081 - val_accuracy: 0.7914
Epoch 32/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6128
- accuracy: 0.7862 - val_loss: 0.5930 - val_accuracy: 0.7945
Epoch 33/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6060
- accuracy: 0.7891 - val_loss: 0.6184 - val_accuracy: 0.7863
Epoch 34/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5986
- accuracy: 0.7897 - val_loss: 0.5748 - val_accuracy: 0.8013
Epoch 35/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5895
- accuracy: 0.7942 - val_loss: 0.5879 - val_accuracy: 0.7981
Epoch 36/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5772
- accuracy: 0.7990 - val_loss: 0.6009 - val_accuracy: 0.7915
Epoch 37/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5691
- accuracy: 0.8001 - val_loss: 0.5733 - val_accuracy: 0.8014
Epoch 38/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5645
- accuracy: 0.8016 - val_loss: 0.5654 - val_accuracy: 0.8028
Epoch 39/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5530
- accuracy: 0.8073 - val_loss: 0.5861 - val_accuracy: 0.7981
Epoch 40/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5428
- accuracy: 0.8106 - val_loss: 0.5521 - val_accuracy: 0.8079
Epoch 41/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5427
- accuracy: 0.8111 - val_loss: 0.5482 - val_accuracy: 0.8106
```

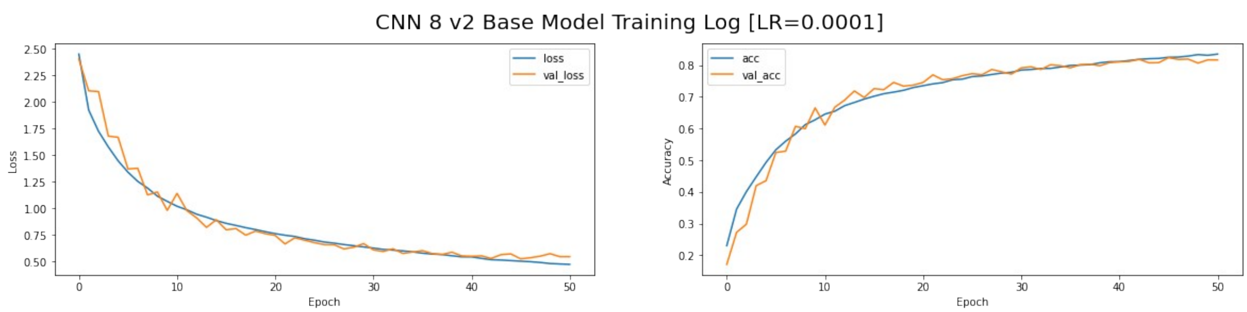
```
Epoch 42/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5285
- accuracy: 0.8146 - val_loss: 0.5524 - val_accuracy: 0.8118
Epoch 43/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5163
- accuracy: 0.8183 - val_loss: 0.5280 - val_accuracy: 0.8184
Epoch 44/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5130
- accuracy: 0.8203 - val_loss: 0.5633 - val_accuracy: 0.8073
Epoch 45/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5076
- accuracy: 0.8212 - val_loss: 0.5704 - val_accuracy: 0.8078
Epoch 46/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5024
- accuracy: 0.8247 - val_loss: 0.5245 - val_accuracy: 0.8234
Epoch 47/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4973
- accuracy: 0.8255 - val_loss: 0.5345 - val_accuracy: 0.8180
Epoch 48/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4901
- accuracy: 0.8284 - val_loss: 0.5491 - val_accuracy: 0.8191
Epoch 49/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4801
- accuracy: 0.8332 - val_loss: 0.5725 - val_accuracy: 0.8064
Epoch 50/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4760
- accuracy: 0.8310 - val_loss: 0.5445 - val_accuracy: 0.8167
Epoch 51/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4723
- accuracy: 0.8348 - val_loss: 0.5446 - val_accuracy: 0.8165
```

```python
preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      1000
           1       0.96      0.89      0.92      1000
           2       0.78      0.69      0.73      1000
           3       0.70      0.60      0.65      1000
           4       0.72      0.84      0.78      1000
           5       0.73      0.77      0.75      1000
           6       0.76      0.91      0.83      1000
           7       0.88      0.84      0.86      1000
           8       0.87      0.92      0.89      1000
           9       0.86      0.92      0.89      1000
```

```
    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000

313/313 - 1s - loss: 0.5648 - accuracy: 0.8114
Accuracy: 81.13999962806702
Macro F1-score: 0.8100927103686075
```

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.0001]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



When the learning rate decreased to 0.0001, the model training log loks very good as the validation and training loss and accuracy looks very aligned. However the model performance is compromised by 0.01. Hence, we may need to increase the learning rate by abit to increase the model performance.

I will be trying learning rate from 0.0002 to 0.0005. Note that I will be writing all my analysis after the trying out all the learning rate for easier comparison

3.3.2.5 Learning Rate=0.0002

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0002)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 2.2337
- accuracy: 0.2813 - val_loss: 1.9861 - val_accuracy: 0.3018
Epoch 2/500
625/625 [==============================] - 6s 9ms/step - loss: 1.6515
- accuracy: 0.4285 - val_loss: 1.6601 - val_accuracy: 0.4424
Epoch 3/500
625/625 [==============================] - 5s 9ms/step - loss: 1.4030
- accuracy: 0.5072 - val_loss: 1.4512 - val_accuracy: 0.5081
Epoch 4/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2516
- accuracy: 0.5600 - val_loss: 1.1921 - val_accuracy: 0.5904
Epoch 5/500
625/625 [==============================] - 5s 9ms/step - loss: 1.1454
- accuracy: 0.5993 - val_loss: 1.1588 - val_accuracy: 0.6021
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 1.0531
- accuracy: 0.6314 - val_loss: 1.0919 - val_accuracy: 0.6235
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9925
- accuracy: 0.6532 - val_loss: 0.8531 - val_accuracy: 0.7051
Epoch 8/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9351
- accuracy: 0.6732 - val_loss: 0.8225 - val_accuracy: 0.7117
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8895
- accuracy: 0.6883 - val_loss: 0.8513 - val_accuracy: 0.6994
Epoch 10/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8501
- accuracy: 0.7020 - val_loss: 0.8484 - val_accuracy: 0.7102
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8167
- accuracy: 0.7164 - val_loss: 0.7155 - val_accuracy: 0.7497
Epoch 12/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7786
- accuracy: 0.7281 - val_loss: 0.7170 - val_accuracy: 0.7504
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7517
- accuracy: 0.7396 - val_loss: 0.6545 - val_accuracy: 0.7705
```

```
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7257
- accuracy: 0.7502 - val_loss: 0.6839 - val_accuracy: 0.7619
Epoch 15/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7037
- accuracy: 0.7570 - val_loss: 0.6465 - val_accuracy: 0.7780
Epoch 16/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6790
- accuracy: 0.7644 - val_loss: 0.6389 - val_accuracy: 0.7781
Epoch 17/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6625
- accuracy: 0.7716 - val_loss: 0.6792 - val_accuracy: 0.7672
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6377
- accuracy: 0.7789 - val_loss: 0.6465 - val_accuracy: 0.7790
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6275
- accuracy: 0.7829 - val_loss: 0.6135 - val_accuracy: 0.7858
Epoch 20/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6068
- accuracy: 0.7903 - val_loss: 0.5836 - val_accuracy: 0.8035
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5914
- accuracy: 0.7959 - val_loss: 0.6382 - val_accuracy: 0.7825
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5737
- accuracy: 0.8017 - val_loss: 0.5826 - val_accuracy: 0.8009
Epoch 23/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5627
- accuracy: 0.8060 - val_loss: 0.5825 - val_accuracy: 0.7973
Epoch 24/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5444
- accuracy: 0.8101 - val_loss: 0.5898 - val_accuracy: 0.8029
Epoch 25/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5404
- accuracy: 0.8112 - val_loss: 0.5696 - val_accuracy: 0.8051
Epoch 26/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5252
- accuracy: 0.8189 - val_loss: 0.5794 - val_accuracy: 0.8014
Epoch 27/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5111
- accuracy: 0.8231 - val_loss: 0.5812 - val_accuracy: 0.8062
Epoch 28/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4997
- accuracy: 0.8263 - val_loss: 0.5554 - val_accuracy: 0.8133
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4942
- accuracy: 0.8278 - val_loss: 0.5630 - val_accuracy: 0.8087
Epoch 30/500
```

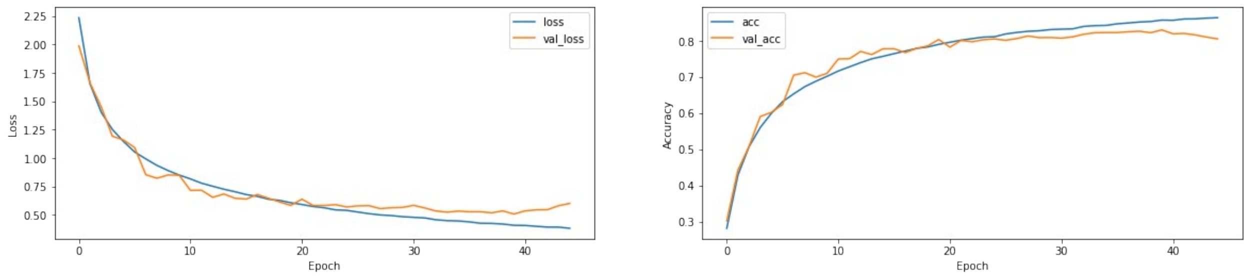
```
625/625 [==============================] - 5s 8ms/step - loss: 0.4842
- accuracy: 0.8310 - val_loss: 0.5658 - val_accuracy: 0.8091
Epoch 31/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4787
- accuracy: 0.8322 - val_loss: 0.5841 - val_accuracy: 0.8074
Epoch 32/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4734
- accuracy: 0.8331 - val_loss: 0.5614 - val_accuracy: 0.8108
Epoch 33/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4574
- accuracy: 0.8398 - val_loss: 0.5348 - val_accuracy: 0.8181
Epoch 34/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4494
- accuracy: 0.8419 - val_loss: 0.5240 - val_accuracy: 0.8223
Epoch 35/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4468
- accuracy: 0.8426 - val_loss: 0.5334 - val_accuracy: 0.8229
Epoch 36/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4386
- accuracy: 0.8470 - val_loss: 0.5276 - val_accuracy: 0.8228
Epoch 37/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4266
- accuracy: 0.8493 - val_loss: 0.5278 - val_accuracy: 0.8251
Epoch 38/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4253
- accuracy: 0.8519 - val_loss: 0.5183 - val_accuracy: 0.8265
Epoch 39/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4198
- accuracy: 0.8533 - val_loss: 0.5351 - val_accuracy: 0.8223
Epoch 40/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4087
- accuracy: 0.8573 - val_loss: 0.5076 - val_accuracy: 0.8302
Epoch 41/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4072
- accuracy: 0.8567 - val_loss: 0.5358 - val_accuracy: 0.8193
Epoch 42/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3996
- accuracy: 0.8604 - val_loss: 0.5444 - val_accuracy: 0.8204
Epoch 43/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3930
- accuracy: 0.8608 - val_loss: 0.5458 - val_accuracy: 0.8162
Epoch 44/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3923
- accuracy: 0.8627 - val_loss: 0.5807 - val_accuracy: 0.8103
Epoch 45/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3824
- accuracy: 0.8639 - val_loss: 0.6010 - val_accuracy: 0.8053

preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
```

```python
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.88      0.74      0.80      1000
           1       0.95      0.86      0.90      1000
           2       0.84      0.60      0.70      1000
           3       0.71      0.61      0.66      1000
           4       0.64      0.89      0.75      1000
           5       0.81      0.67      0.73      1000
           6       0.79      0.89      0.84      1000
           7       0.88      0.86      0.87      1000
           8       0.81      0.94      0.87      1000
           9       0.80      0.95      0.86      1000

    accuracy                           0.80     10000
   macro avg       0.81      0.80      0.80     10000
weighted avg       0.81      0.80      0.80     10000

313/313 - 1s - loss: 0.6294 - accuracy: 0.8012
Accuracy: 80.11999726295471
Macro F1-score: 0.7984919900421775
```

```python
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.0002]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

CNN 8 v2 Base Model Training Log [LR=0.0002]



3.3.2.6 Learning Rate=0.0003

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0003)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 2.0603
- accuracy: 0.3167 - val_loss: 1.9737 - val_accuracy: 0.3068
Epoch 2/500
625/625 [==============================] - 6s 9ms/step - loss: 1.5104
- accuracy: 0.4718 - val_loss: 1.4775 - val_accuracy: 0.4832
Epoch 3/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2765
- accuracy: 0.5530 - val_loss: 1.1260 - val_accuracy: 0.6053
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1306
- accuracy: 0.6023 - val_loss: 1.1446 - val_accuracy: 0.6082
Epoch 5/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0297
- accuracy: 0.6395 - val_loss: 0.9358 - val_accuracy: 0.6739
Epoch 6/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9565
- accuracy: 0.6658 - val_loss: 0.9789 - val_accuracy: 0.6639
Epoch 7/500
625/625 [==============================] - 5s 8ms/step - loss: 0.9016
- accuracy: 0.6865 - val_loss: 0.8006 - val_accuracy: 0.7224
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8544
- accuracy: 0.7027 - val_loss: 0.7976 - val_accuracy: 0.7228
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8101
- accuracy: 0.7195 - val_loss: 0.7200 - val_accuracy: 0.7495
Epoch 10/500
625/625 [==============================] - 6s 10ms/step - loss: 0.7715
```

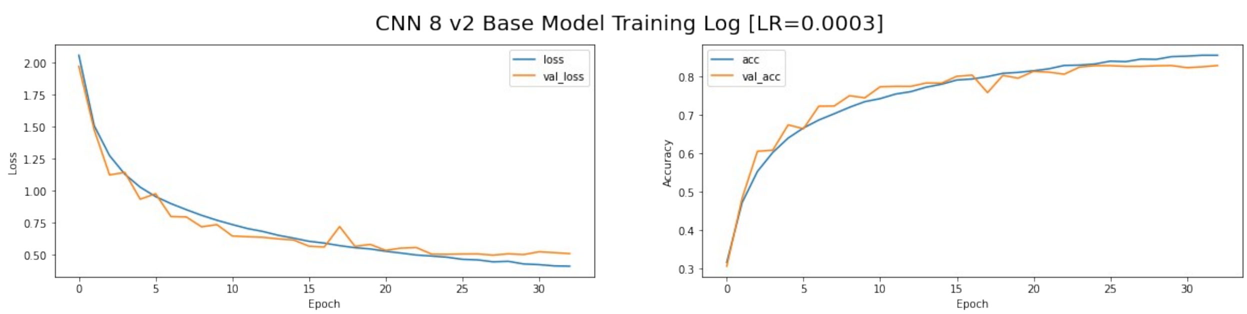
```
- accuracy: 0.7342 - val_loss: 0.7376 - val_accuracy: 0.7441
Epoch 11/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7379
- accuracy: 0.7420 - val_loss: 0.6486 - val_accuracy: 0.7729
Epoch 12/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7065
- accuracy: 0.7539 - val_loss: 0.6429 - val_accuracy: 0.7740
Epoch 13/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6837
- accuracy: 0.7602 - val_loss: 0.6384 - val_accuracy: 0.7739
Epoch 14/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6540
- accuracy: 0.7718 - val_loss: 0.6260 - val_accuracy: 0.7826
Epoch 15/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6315
- accuracy: 0.7794 - val_loss: 0.6171 - val_accuracy: 0.7822
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6071
- accuracy: 0.7904 - val_loss: 0.5690 - val_accuracy: 0.8000
Epoch 17/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5931
- accuracy: 0.7930 - val_loss: 0.5625 - val_accuracy: 0.8031
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5731
- accuracy: 0.7993 - val_loss: 0.7222 - val_accuracy: 0.7576
Epoch 19/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5571
- accuracy: 0.8077 - val_loss: 0.5679 - val_accuracy: 0.8024
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5472
- accuracy: 0.8102 - val_loss: 0.5825 - val_accuracy: 0.7952
Epoch 21/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5293
- accuracy: 0.8145 - val_loss: 0.5362 - val_accuracy: 0.8129
Epoch 22/500
625/625 [==============================] - 5s 8ms/step - loss: 0.5154
- accuracy: 0.8195 - val_loss: 0.5537 - val_accuracy: 0.8108
Epoch 23/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4999
- accuracy: 0.8282 - val_loss: 0.5582 - val_accuracy: 0.8055
Epoch 24/500
625/625 [==============================] - 7s 10ms/step - loss: 0.4919
- accuracy: 0.8291 - val_loss: 0.5069 - val_accuracy: 0.8240
Epoch 25/500
625/625 [==============================] - 7s 11ms/step - loss: 0.4829
- accuracy: 0.8319 - val_loss: 0.5057 - val_accuracy: 0.8277
Epoch 26/500
625/625 [==============================] - 7s 10ms/step - loss: 0.4664
- accuracy: 0.8393 - val_loss: 0.5077 - val_accuracy: 0.8277
```

```
Epoch 27/500
625/625 [==============================] - 7s 11ms/step - loss: 0.4619
- accuracy: 0.8381 - val_loss: 0.5083 - val_accuracy: 0.8258
Epoch 28/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4468
- accuracy: 0.8447 - val_loss: 0.4990 - val_accuracy: 0.8258
Epoch 29/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4506
- accuracy: 0.8440 - val_loss: 0.5091 - val_accuracy: 0.8273
Epoch 30/500
625/625 [==============================] - 7s 11ms/step - loss: 0.4301
- accuracy: 0.8511 - val_loss: 0.5034 - val_accuracy: 0.8279
Epoch 31/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4248
- accuracy: 0.8523 - val_loss: 0.5252 - val_accuracy: 0.8225
Epoch 32/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4148
- accuracy: 0.8547 - val_loss: 0.5185 - val_accuracy: 0.8246
Epoch 33/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4126
- accuracy: 0.8545 - val_loss: 0.5109 - val_accuracy: 0.8280
```

```
preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.82      0.84      0.83      1000
           1       0.93      0.92      0.92      1000
           2       0.83      0.67      0.74      1000
           3       0.70      0.69      0.69      1000
           4       0.80      0.78      0.79      1000
           5       0.76      0.76      0.76      1000
           6       0.89      0.84      0.86      1000
           7       0.77      0.92      0.84      1000
           8       0.89      0.89      0.89      1000
           9       0.85      0.92      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000

313/313 - 1s - loss: 0.5359 - accuracy: 0.8234
Accuracy: 82.34000205993652
Macro F1-score: 0.8222587933595203
```

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.0003]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



3.3.2.7 Learning Rate=0.0004

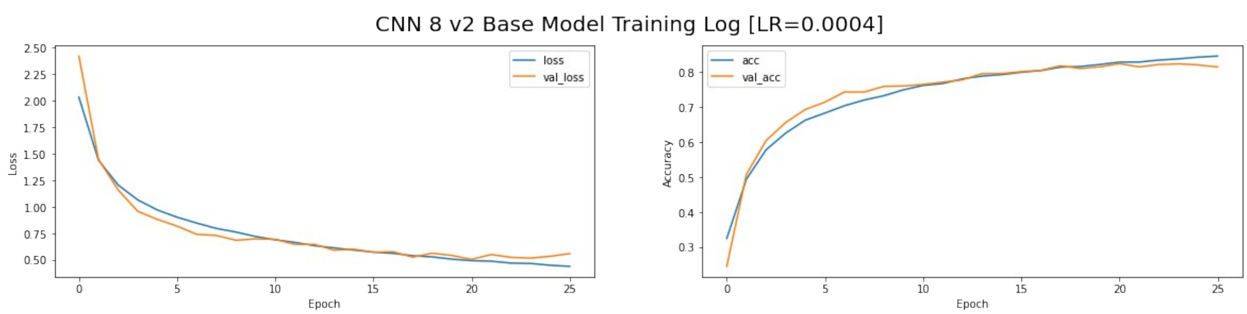
```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 9s 10ms/step - loss: 2.0329
- accuracy: 0.3246 - val_loss: 2.4200 - val_accuracy: 0.2448
Epoch 2/500
625/625 [==============================] - 6s 9ms/step - loss: 1.4398
```

```
- accuracy: 0.4941 - val_loss: 1.4471 - val_accuracy: 0.5075
Epoch 3/500
625/625 [==============================] - 6s 9ms/step - loss: 1.2051
- accuracy: 0.5778 - val_loss: 1.1593 - val_accuracy: 0.6043
Epoch 4/500
625/625 [==============================] - 6s 9ms/step - loss: 1.0649
- accuracy: 0.6257 - val_loss: 0.9577 - val_accuracy: 0.6559
Epoch 5/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9700
- accuracy: 0.6627 - val_loss: 0.8813 - val_accuracy: 0.6933
Epoch 6/500
625/625 [==============================] - 6s 9ms/step - loss: 0.9022
- accuracy: 0.6830 - val_loss: 0.8180 - val_accuracy: 0.7142
Epoch 7/500
625/625 [==============================] - 6s 10ms/step - loss: 0.8467
- accuracy: 0.7040 - val_loss: 0.7409 - val_accuracy: 0.7433
Epoch 8/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7965
- accuracy: 0.7202 - val_loss: 0.7300 - val_accuracy: 0.7431
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7622
- accuracy: 0.7326 - val_loss: 0.6850 - val_accuracy: 0.7591
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.7200
- accuracy: 0.7493 - val_loss: 0.6976 - val_accuracy: 0.7603
Epoch 11/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6889
- accuracy: 0.7619 - val_loss: 0.6937 - val_accuracy: 0.7648
Epoch 12/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6649
- accuracy: 0.7673 - val_loss: 0.6458 - val_accuracy: 0.7713
Epoch 13/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6336
- accuracy: 0.7807 - val_loss: 0.6465 - val_accuracy: 0.7779
Epoch 14/500
625/625 [==============================] - 5s 9ms/step - loss: 0.6128
- accuracy: 0.7886 - val_loss: 0.5910 - val_accuracy: 0.7956
Epoch 15/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5922
- accuracy: 0.7929 - val_loss: 0.6006 - val_accuracy: 0.7958
Epoch 16/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5731
- accuracy: 0.7994 - val_loss: 0.5722 - val_accuracy: 0.8013
Epoch 17/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5615
- accuracy: 0.8044 - val_loss: 0.5767 - val_accuracy: 0.8049
Epoch 18/500
625/625 [==============================] - 6s 10ms/step - loss: 0.5388
- accuracy: 0.8142 - val_loss: 0.5257 - val_accuracy: 0.8181
```

```
Epoch 19/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5282
- accuracy: 0.8160 - val_loss: 0.5622 - val_accuracy: 0.8105
Epoch 20/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5069
- accuracy: 0.8217 - val_loss: 0.5413 - val_accuracy: 0.8150
Epoch 21/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4929
- accuracy: 0.8286 - val_loss: 0.5048 - val_accuracy: 0.8243
Epoch 22/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4878
- accuracy: 0.8288 - val_loss: 0.5503 - val_accuracy: 0.8150
Epoch 23/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4697
- accuracy: 0.8343 - val_loss: 0.5239 - val_accuracy: 0.8217
Epoch 24/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4664
- accuracy: 0.8378 - val_loss: 0.5170 - val_accuracy: 0.8237
Epoch 25/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4493
- accuracy: 0.8427 - val_loss: 0.5338 - val_accuracy: 0.8206
Epoch 26/500
625/625 [==============================] - 6s 10ms/step - loss: 0.4389
- accuracy: 0.8457 - val_loss: 0.5586 - val_accuracy: 0.8145

preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))

              precision    recall  f1-score   support

           0       0.89      0.76      0.82      1000
           1       0.96      0.88      0.92      1000
           2       0.75      0.75      0.75      1000
           3       0.73      0.60      0.66      1000
           4       0.71      0.86      0.78      1000
           5       0.83      0.66      0.74      1000
           6       0.72      0.93      0.81      1000
           7       0.90      0.83      0.86      1000
           8       0.83      0.94      0.88      1000
           9       0.86      0.93      0.89      1000

    accuracy                           0.81     10000
   macro avg       0.82      0.81      0.81     10000
weighted avg       0.82      0.81      0.81     10000

313/313 - 1s - loss: 0.5755 - accuracy: 0.8131
```

```
Accuracy: 81.30999803543091
Macro F1-score: 0.8112449783515515
```

```
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.0004]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



3.3.2.8 Learning Rate=0.0005

```
CNN_8_v2=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0005)
CNN_8_v2.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v2_history=CNN_8_v2.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 1.9438
```

```
- accuracy: 0.3471 - val_loss: 2.2221 - val_accuracy: 0.2946
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.3546
- accuracy: 0.5203 - val_loss: 1.6773 - val_accuracy: 0.4403
Epoch 3/500
625/625 [==============================] - 5s 8ms/step - loss: 1.1282
- accuracy: 0.6021 - val_loss: 0.9135 - val_accuracy: 0.6838
Epoch 4/500
625/625 [==============================] - 6s 10ms/step - loss: 0.9943
- accuracy: 0.6529 - val_loss: 0.9025 - val_accuracy: 0.6855
Epoch 5/500
625/625 [==============================] - 6s 10ms/step - loss: 0.9092
- accuracy: 0.6840 - val_loss: 0.8024 - val_accuracy: 0.7186
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8426
- accuracy: 0.7078 - val_loss: 0.7621 - val_accuracy: 0.7297
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7976
- accuracy: 0.7215 - val_loss: 0.9249 - val_accuracy: 0.6872
Epoch 8/500
625/625 [==============================] - 6s 10ms/step - loss: 0.7513
- accuracy: 0.7376 - val_loss: 0.7271 - val_accuracy: 0.7488
Epoch 9/500
625/625 [==============================] - 5s 9ms/step - loss: 0.7136
- accuracy: 0.7513 - val_loss: 0.6252 - val_accuracy: 0.7811
Epoch 10/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6749
- accuracy: 0.7638 - val_loss: 0.6463 - val_accuracy: 0.7778
Epoch 11/500
625/625 [==============================] - 5s 8ms/step - loss: 0.6503
- accuracy: 0.7741 - val_loss: 0.5967 - val_accuracy: 0.7961
Epoch 12/500
625/625 [==============================] - 6s 10ms/step - loss: 0.6225
- accuracy: 0.7825 - val_loss: 0.5862 - val_accuracy: 0.7973
Epoch 13/500
625/625 [==============================] - 7s 11ms/step - loss: 0.6001
- accuracy: 0.7912 - val_loss: 0.6454 - val_accuracy: 0.7781
Epoch 14/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5793
- accuracy: 0.7987 - val_loss: 0.5922 - val_accuracy: 0.7975
Epoch 15/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5619
- accuracy: 0.8044 - val_loss: 0.5735 - val_accuracy: 0.8031
Epoch 16/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5451
- accuracy: 0.8092 - val_loss: 0.5823 - val_accuracy: 0.8067
Epoch 17/500
625/625 [==============================] - 7s 11ms/step - loss: 0.5245
- accuracy: 0.8168 - val_loss: 0.6018 - val_accuracy: 0.7969
```

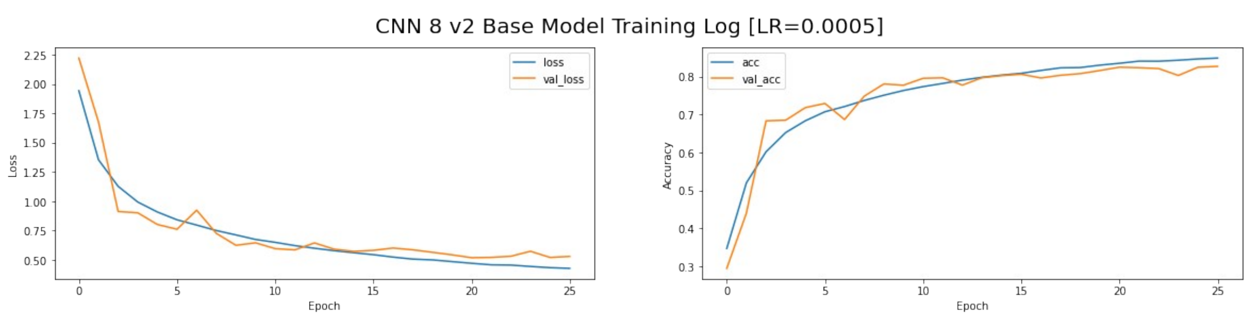
```
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5075
- accuracy: 0.8237 - val_loss: 0.5869 - val_accuracy: 0.8040
Epoch 19/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5007
- accuracy: 0.8244 - val_loss: 0.5655 - val_accuracy: 0.8083
Epoch 20/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4863
- accuracy: 0.8307 - val_loss: 0.5438 - val_accuracy: 0.8165
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4718
- accuracy: 0.8357 - val_loss: 0.5192 - val_accuracy: 0.8254
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4587
- accuracy: 0.8413 - val_loss: 0.5219 - val_accuracy: 0.8240
Epoch 23/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4565
- accuracy: 0.8411 - val_loss: 0.5315 - val_accuracy: 0.8217
Epoch 24/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4448
- accuracy: 0.8437 - val_loss: 0.5743 - val_accuracy: 0.8035
Epoch 25/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4344
- accuracy: 0.8471 - val_loss: 0.5214 - val_accuracy: 0.8253
Epoch 26/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4285
- accuracy: 0.8493 - val_loss: 0.5302 - val_accuracy: 0.8276
```

```
preds = CNN_8_v2.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v2.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.86      0.81      0.84      1000
           1       0.93      0.90      0.91      1000
           2       0.75      0.74      0.75      1000
           3       0.71      0.63      0.67      1000
           4       0.80      0.73      0.76      1000
           5       0.76      0.77      0.76      1000
           6       0.70      0.93      0.80      1000
           7       0.91      0.82      0.86      1000
           8       0.88      0.91      0.90      1000
           9       0.88      0.90      0.89      1000

    accuracy                           0.81     10000
   macro avg       0.82      0.81      0.81     10000
```

```
weighted avg       0.82      0.81      0.81     10000

313/313 - 1s - loss: 0.5501 - accuracy: 0.8146
Accuracy: 81.45999908447266
Macro F1-score: 0.8142266023523238
```

```python
loss = CNN_8_v2_history.history['loss']
val_loss = CNN_8_v2_history.history['val_loss']
acc = CNN_8_v2_history.history['accuracy']
val_acc = CNN_8_v2_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v2 Base Model Training Log
[LR=0.0005]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Insights:

Learning rate of 0.0004 is the best learning rate as the allows for a good model performance of 81% for f1 score while ensuring that the validation acc/loss is closely aligned to training acc/loss.

For other learning rate, it either comprises on the model performance or the model training process. Hence when lr=0.0004 , it gives the best balance between the both/

I will be trying tuning with batch size of 16,32,64,128. Note that I will be writing all my analysis after the trying out all the learning rate for easier comparison

3.3.3.1 Batch Size=16

```
CNN_8_v3=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
CNN_8_v3.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v3_history=CNN_8_v3.fit(X_train, y_train, epochs=500,
batch_size=16, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
2500/2500 [==============================] - 17s 7ms/step - loss:
1.8880 - accuracy: 0.3559 - val_loss: 1.3933 - val_accuracy: 0.5132
Epoch 2/500
2500/2500 [==============================] - 16s 6ms/step - loss:
1.3139 - accuracy: 0.5409 - val_loss: 1.1937 - val_accuracy: 0.5808
Epoch 3/500
2500/2500 [==============================] - 16s 6ms/step - loss:
1.1014 - accuracy: 0.6195 - val_loss: 0.8326 - val_accuracy: 0.7067
Epoch 4/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.9861 - accuracy: 0.6591 - val_loss: 0.7741 - val_accuracy: 0.7288
Epoch 5/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.9094 - accuracy: 0.6864 - val_loss: 0.7436 - val_accuracy: 0.7427
Epoch 6/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.8487 - accuracy: 0.7092 - val_loss: 0.6972 - val_accuracy: 0.7613
Epoch 7/500
2500/2500 [==============================] - 19s 7ms/step - loss:
0.8047 - accuracy: 0.7250 - val_loss: 0.6473 - val_accuracy: 0.7779
Epoch 8/500
2500/2500 [==============================] - 19s 8ms/step - loss:
0.7625 - accuracy: 0.7365 - val_loss: 1.0776 - val_accuracy: 0.6600
Epoch 9/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.7289 - accuracy: 0.7494 - val_loss: 0.5929 - val_accuracy: 0.7953
Epoch 10/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.7123 - accuracy: 0.7566 - val_loss: 0.5664 - val_accuracy: 0.8034
Epoch 11/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.6785 - accuracy: 0.7680 - val_loss: 0.5641 - val_accuracy: 0.8035
Epoch 12/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.6591 - accuracy: 0.7757 - val_loss: 0.5455 - val_accuracy: 0.8128
```

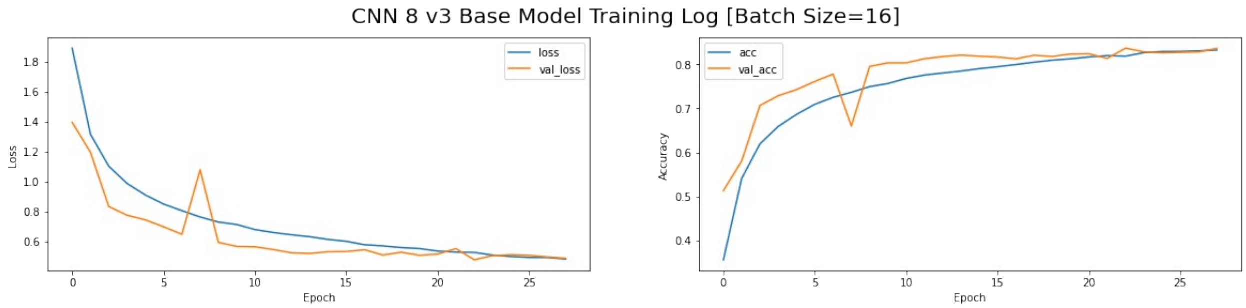
```
Epoch 13/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.6440 - accuracy: 0.7804 - val_loss: 0.5238 - val_accuracy: 0.8178
Epoch 14/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.6311 - accuracy: 0.7848 - val_loss: 0.5198 - val_accuracy: 0.8211
Epoch 15/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.6129 - accuracy: 0.7904 - val_loss: 0.5310 - val_accuracy: 0.8187
Epoch 16/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.6001 - accuracy: 0.7947 - val_loss: 0.5325 - val_accuracy: 0.8167
Epoch 17/500
2500/2500 [==============================] - 16s 7ms/step - loss:
0.5772 - accuracy: 0.7996 - val_loss: 0.5445 - val_accuracy: 0.8126
Epoch 18/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.5698 - accuracy: 0.8048 - val_loss: 0.5087 - val_accuracy: 0.8207
Epoch 19/500
2500/2500 [==============================] - 19s 7ms/step - loss:
0.5586 - accuracy: 0.8095 - val_loss: 0.5277 - val_accuracy: 0.8180
Epoch 20/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.5521 - accuracy: 0.8126 - val_loss: 0.5072 - val_accuracy: 0.8237
Epoch 21/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.5356 - accuracy: 0.8169 - val_loss: 0.5152 - val_accuracy: 0.8242
Epoch 22/500
2500/2500 [==============================] - 17s 7ms/step - loss:
0.5287 - accuracy: 0.8200 - val_loss: 0.5519 - val_accuracy: 0.8136
Epoch 23/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.5266 - accuracy: 0.8187 - val_loss: 0.4772 - val_accuracy: 0.8368
Epoch 24/500
2500/2500 [==============================] - 16s 7ms/step - loss:
0.5087 - accuracy: 0.8267 - val_loss: 0.5044 - val_accuracy: 0.8287
Epoch 25/500
2500/2500 [==============================] - 18s 7ms/step - loss:
0.4988 - accuracy: 0.8293 - val_loss: 0.5113 - val_accuracy: 0.8263
Epoch 26/500
2500/2500 [==============================] - 16s 6ms/step - loss:
0.4927 - accuracy: 0.8296 - val_loss: 0.5073 - val_accuracy: 0.8274
Epoch 27/500
2500/2500 [==============================] - 14s 6ms/step - loss:
0.4921 - accuracy: 0.8307 - val_loss: 0.4961 - val_accuracy: 0.8285
Epoch 28/500
2500/2500 [==============================] - 14s 6ms/step - loss:
0.4824 - accuracy: 0.8327 - val_loss: 0.4871 - val_accuracy: 0.8363
```

```python
preds = CNN_8_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.85      1000
           1       0.90      0.94      0.92      1000
           2       0.85      0.68      0.75      1000
           3       0.77      0.63      0.69      1000
           4       0.77      0.85      0.81      1000
           5       0.74      0.80      0.77      1000
           6       0.87      0.87      0.87      1000
           7       0.84      0.90      0.87      1000
           8       0.87      0.91      0.89      1000
           9       0.91      0.90      0.91      1000

    accuracy                           0.83     10000
   macro avg       0.84      0.83      0.83     10000
weighted avg       0.84      0.83      0.83     10000

313/313 - 1s - loss: 0.5013 - accuracy: 0.8350
Accuracy: 83.49999785423279
Macro F1-score: 0.8328463791656908
```

```python
loss = CNN_8_v3_history.history['loss']
val_loss = CNN_8_v3_history.history['val_loss']
acc = CNN_8_v3_history.history['accuracy']
val_acc = CNN_8_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v3 Base Model Training Log [Batch
Size=16]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()
```



3.3.3.2 Batch Size=32

```
CNN_8_v3=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
CNN_8_v3.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v3_history=CNN_8_v3.fit(X_train, y_train, epochs=500,
batch_size=32, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

Epoch 1/500
1250/1250 [==============================] - 9s 7ms/step - loss:
1.8911 - accuracy: 0.3573 - val_loss: 1.7674 - val_accuracy: 0.4124
Epoch 2/500
1250/1250 [==============================] - 8s 6ms/step - loss:
1.3134 - accuracy: 0.5364 - val_loss: 1.0244 - val_accuracy: 0.6411
Epoch 3/500
1250/1250 [==============================] - 10s 8ms/step - loss:
1.0937 - accuracy: 0.6159 - val_loss: 0.9149 - val_accuracy: 0.6817
Epoch 4/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.9888 - accuracy: 0.6574 - val_loss: 0.8343 - val_accuracy: 0.7071
Epoch 5/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.9081 - accuracy: 0.6833 - val_loss: 0.7532 - val_accuracy: 0.7390
Epoch 6/500
1250/1250 [==============================] - 11s 9ms/step - loss:
0.8387 - accuracy: 0.7102 - val_loss: 0.8203 - val_accuracy: 0.7146
Epoch 7/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7948 - accuracy: 0.7249 - val_loss: 0.6844 - val_accuracy: 0.7587
Epoch 8/500
1250/1250 [==============================] - 11s 8ms/step - loss:
0.7530 - accuracy: 0.7400 - val_loss: 0.6564 - val_accuracy: 0.7743
Epoch 9/500
```

```
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7168 - accuracy: 0.7558 - val_loss: 0.6290 - val_accuracy: 0.7792
Epoch 10/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.6833 - accuracy: 0.7632 - val_loss: 0.7049 - val_accuracy: 0.7534
Epoch 11/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.6563 - accuracy: 0.7717 - val_loss: 0.5850 - val_accuracy: 0.7987
Epoch 12/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.6353 - accuracy: 0.7804 - val_loss: 0.5656 - val_accuracy: 0.8041
Epoch 13/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.6189 - accuracy: 0.7872 - val_loss: 0.5669 - val_accuracy: 0.8018
Epoch 14/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5933 - accuracy: 0.7940 - val_loss: 0.5793 - val_accuracy: 0.8039
Epoch 15/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5755 - accuracy: 0.8005 - val_loss: 0.5353 - val_accuracy: 0.8163
Epoch 16/500
1250/1250 [==============================] - 10s 8ms/step - loss:
0.5576 - accuracy: 0.8092 - val_loss: 0.5522 - val_accuracy: 0.8113
Epoch 17/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5407 - accuracy: 0.8144 - val_loss: 0.5924 - val_accuracy: 0.7973
Epoch 18/500
1250/1250 [==============================] - 9s 8ms/step - loss:
0.5274 - accuracy: 0.8181 - val_loss: 0.5322 - val_accuracy: 0.8165
Epoch 19/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5227 - accuracy: 0.8185 - val_loss: 0.5299 - val_accuracy: 0.8219
Epoch 20/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5042 - accuracy: 0.8268 - val_loss: 0.5952 - val_accuracy: 0.8047
Epoch 21/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4930 - accuracy: 0.8296 - val_loss: 0.5408 - val_accuracy: 0.8188
Epoch 22/500
1250/1250 [==============================] - 10s 8ms/step - loss:
0.4865 - accuracy: 0.8303 - val_loss: 0.5113 - val_accuracy: 0.8245
Epoch 23/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4766 - accuracy: 0.8353 - val_loss: 0.5146 - val_accuracy: 0.8225
Epoch 24/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.4622 - accuracy: 0.8402 - val_loss: 0.5153 - val_accuracy: 0.8242
Epoch 25/500
1250/1250 [==============================] - 9s 7ms/step - loss:
```

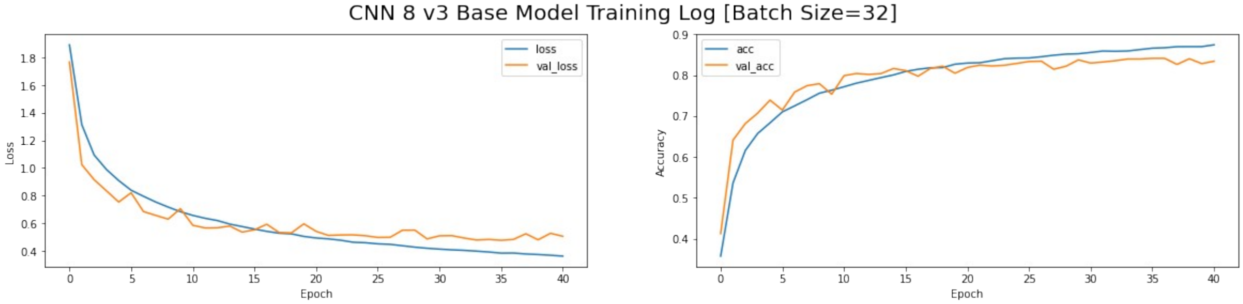
```
0.4587 - accuracy: 0.8417 - val_loss: 0.5090 - val_accuracy: 0.8287
Epoch 26/500
1250/1250 [==============================] - 10s 8ms/step - loss:
0.4505 - accuracy: 0.8421 - val_loss: 0.4966 - val_accuracy: 0.8334
Epoch 27/500
1250/1250 [==============================] - 10s 8ms/step - loss:
0.4469 - accuracy: 0.8451 - val_loss: 0.4978 - val_accuracy: 0.8342
Epoch 28/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4367 - accuracy: 0.8488 - val_loss: 0.5491 - val_accuracy: 0.8145
Epoch 29/500
1250/1250 [==============================] - 10s 8ms/step - loss:
0.4263 - accuracy: 0.8515 - val_loss: 0.5498 - val_accuracy: 0.8219
Epoch 30/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4183 - accuracy: 0.8525 - val_loss: 0.4862 - val_accuracy: 0.8374
Epoch 31/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.4122 - accuracy: 0.8557 - val_loss: 0.5084 - val_accuracy: 0.8296
Epoch 32/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4070 - accuracy: 0.8590 - val_loss: 0.5097 - val_accuracy: 0.8322
Epoch 33/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.4033 - accuracy: 0.8583 - val_loss: 0.4926 - val_accuracy: 0.8353
Epoch 34/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3975 - accuracy: 0.8592 - val_loss: 0.4782 - val_accuracy: 0.8396
Epoch 35/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3914 - accuracy: 0.8626 - val_loss: 0.4824 - val_accuracy: 0.8396
Epoch 36/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3832 - accuracy: 0.8659 - val_loss: 0.4764 - val_accuracy: 0.8412
Epoch 37/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3839 - accuracy: 0.8670 - val_loss: 0.4825 - val_accuracy: 0.8411
Epoch 38/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3771 - accuracy: 0.8696 - val_loss: 0.5234 - val_accuracy: 0.8263
Epoch 39/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.3732 - accuracy: 0.8698 - val_loss: 0.4798 - val_accuracy: 0.8402
Epoch 40/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3680 - accuracy: 0.8698 - val_loss: 0.5268 - val_accuracy: 0.8282
Epoch 41/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.3616 - accuracy: 0.8742 - val_loss: 0.5051 - val_accuracy: 0.8341
```

```
preds = CNN_8_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.89      0.82      0.85      1000
           1       0.94      0.91      0.93      1000
           2       0.78      0.75      0.77      1000
           3       0.74      0.66      0.70      1000
           4       0.73      0.88      0.80      1000
           5       0.79      0.74      0.76      1000
           6       0.79      0.91      0.84      1000
           7       0.90      0.86      0.88      1000
           8       0.89      0.93      0.91      1000
           9       0.93      0.89      0.91      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.83     10000
weighted avg       0.84      0.84      0.83     10000

313/313 - 1s - loss: 0.5133 - accuracy: 0.8351
Accuracy: 83.5099995136261
Macro F1-score: 0.8345594456176757
```

```
loss = CNN_8_v3_history.history['loss']
val_loss = CNN_8_v3_history.history['val_loss']
acc = CNN_8_v3_history.history['accuracy']
val_acc = CNN_8_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v3 Base Model Training Log [Batch
Size=32]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()
```



3.3.3.3 Batch Size=64

```
CNN_8_v3=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
CNN_8_v3.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v3_history=CNN_8_v3.fit(X_train, y_train, epochs=500,
batch_size=64, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

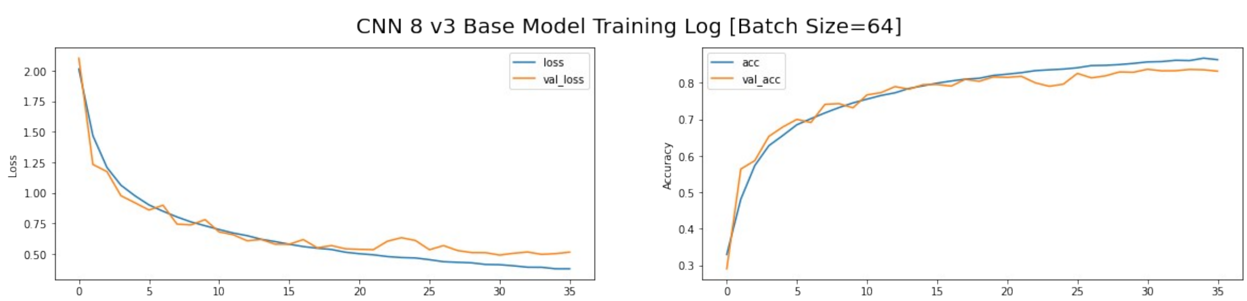
```
Epoch 1/500
625/625 [==============================] - 6s 9ms/step - loss: 2.0138
- accuracy: 0.3297 - val_loss: 2.1024 - val_accuracy: 0.2905
Epoch 2/500
625/625 [==============================] - 5s 8ms/step - loss: 1.4688
- accuracy: 0.4812 - val_loss: 1.2335 - val_accuracy: 0.5637
Epoch 3/500
625/625 [==============================] - 5s 9ms/step - loss: 1.2094
- accuracy: 0.5734 - val_loss: 1.1751 - val_accuracy: 0.5868
Epoch 4/500
625/625 [==============================] - 5s 8ms/step - loss: 1.0637
- accuracy: 0.6277 - val_loss: 0.9778 - val_accuracy: 0.6531
Epoch 5/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9771
- accuracy: 0.6557 - val_loss: 0.9198 - val_accuracy: 0.6791
Epoch 6/500
625/625 [==============================] - 5s 9ms/step - loss: 0.9026
- accuracy: 0.6851 - val_loss: 0.8608 - val_accuracy: 0.6998
Epoch 7/500
625/625 [==============================] - 5s 9ms/step - loss: 0.8509
- accuracy: 0.7017 - val_loss: 0.9001 - val_accuracy: 0.6912
Epoch 8/500
625/625 [==============================] - 5s 8ms/step - loss: 0.8042
- accuracy: 0.7175 - val_loss: 0.7454 - val_accuracy: 0.7410
Epoch 9/500
```

```
625/625 [==============================] - 6s 9ms/step - loss: 0.7632
- accuracy: 0.7321 - val_loss: 0.7391 - val_accuracy: 0.7429
Epoch 10/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7322
- accuracy: 0.7449 - val_loss: 0.7824 - val_accuracy: 0.7315
Epoch 11/500
625/625 [==============================] - 6s 9ms/step - loss: 0.7014
- accuracy: 0.7553 - val_loss: 0.6817 - val_accuracy: 0.7669
Epoch 12/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6728
- accuracy: 0.7653 - val_loss: 0.6589 - val_accuracy: 0.7731
Epoch 13/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6511
- accuracy: 0.7726 - val_loss: 0.6096 - val_accuracy: 0.7896
Epoch 14/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6216
- accuracy: 0.7847 - val_loss: 0.6199 - val_accuracy: 0.7824
Epoch 15/500
625/625 [==============================] - 6s 9ms/step - loss: 0.6026
- accuracy: 0.7917 - val_loss: 0.5815 - val_accuracy: 0.7949
Epoch 16/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5813
- accuracy: 0.7991 - val_loss: 0.5808 - val_accuracy: 0.7954
Epoch 17/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5620
- accuracy: 0.8049 - val_loss: 0.6191 - val_accuracy: 0.7910
Epoch 18/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5485
- accuracy: 0.8100 - val_loss: 0.5520 - val_accuracy: 0.8094
Epoch 19/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5383
- accuracy: 0.8123 - val_loss: 0.5702 - val_accuracy: 0.8042
Epoch 20/500
625/625 [==============================] - 5s 9ms/step - loss: 0.5159
- accuracy: 0.8202 - val_loss: 0.5436 - val_accuracy: 0.8159
Epoch 21/500
625/625 [==============================] - 6s 9ms/step - loss: 0.5036
- accuracy: 0.8237 - val_loss: 0.5393 - val_accuracy: 0.8149
Epoch 22/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4946
- accuracy: 0.8277 - val_loss: 0.5368 - val_accuracy: 0.8178
Epoch 23/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4804
- accuracy: 0.8333 - val_loss: 0.6057 - val_accuracy: 0.8000
Epoch 24/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4722
- accuracy: 0.8356 - val_loss: 0.6344 - val_accuracy: 0.7904
Epoch 25/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4687
```

```
- accuracy: 0.8376 - val_loss: 0.6116 - val_accuracy: 0.7963
Epoch 26/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4548
- accuracy: 0.8412 - val_loss: 0.5361 - val_accuracy: 0.8257
Epoch 27/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4383
- accuracy: 0.8472 - val_loss: 0.5700 - val_accuracy: 0.8136
Epoch 28/500
625/625 [==============================] - 6s 9ms/step - loss: 0.4335
- accuracy: 0.8479 - val_loss: 0.5297 - val_accuracy: 0.8190
Epoch 29/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4291
- accuracy: 0.8500 - val_loss: 0.5128 - val_accuracy: 0.8297
Epoch 30/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4159
- accuracy: 0.8532 - val_loss: 0.5120 - val_accuracy: 0.8289
Epoch 31/500
625/625 [==============================] - 5s 8ms/step - loss: 0.4138
- accuracy: 0.8572 - val_loss: 0.4925 - val_accuracy: 0.8371
Epoch 32/500
625/625 [==============================] - 5s 9ms/step - loss: 0.4047
- accuracy: 0.8581 - val_loss: 0.5071 - val_accuracy: 0.8328
Epoch 33/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3932
- accuracy: 0.8616 - val_loss: 0.5175 - val_accuracy: 0.8330
Epoch 34/500
625/625 [==============================] - 5s 8ms/step - loss: 0.3926
- accuracy: 0.8609 - val_loss: 0.4984 - val_accuracy: 0.8366
Epoch 35/500
625/625 [==============================] - 5s 9ms/step - loss: 0.3804
- accuracy: 0.8674 - val_loss: 0.5043 - val_accuracy: 0.8357
Epoch 36/500
625/625 [==============================] - 6s 9ms/step - loss: 0.3804
- accuracy: 0.8633 - val_loss: 0.5165 - val_accuracy: 0.8318

preds = CNN_8_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))

              precision    recall  f1-score   support

           0       0.86      0.81      0.84      1000
           1       0.89      0.94      0.92      1000
           2       0.82      0.68      0.74      1000
           3       0.70      0.62      0.66      1000
           4       0.79      0.80      0.79      1000
           5       0.72      0.80      0.76      1000
```

```
           6       0.75      0.92      0.83      1000
           7       0.89      0.86      0.87      1000
           8       0.92      0.89      0.90      1000
           9       0.89      0.90      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000

313/313 - 1s - loss: 0.5427 - accuracy: 0.8215
Accuracy: 82.15000033378601
Macro F1-score: 0.8202548959781236
```

```
loss = CNN_8_v3_history.history['loss']
val_loss = CNN_8_v3_history.history['val_loss']
acc = CNN_8_v3_history.history['accuracy']
val_acc = CNN_8_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v3 Base Model Training Log [Batch 
Size=64]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



3.3.3.4 Batch Size=128

```
CNN_8_v3=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
CNN_8_v3.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v3_history=CNN_8_v3.fit(X_train, y_train, epochs=500,
batch_size=128, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

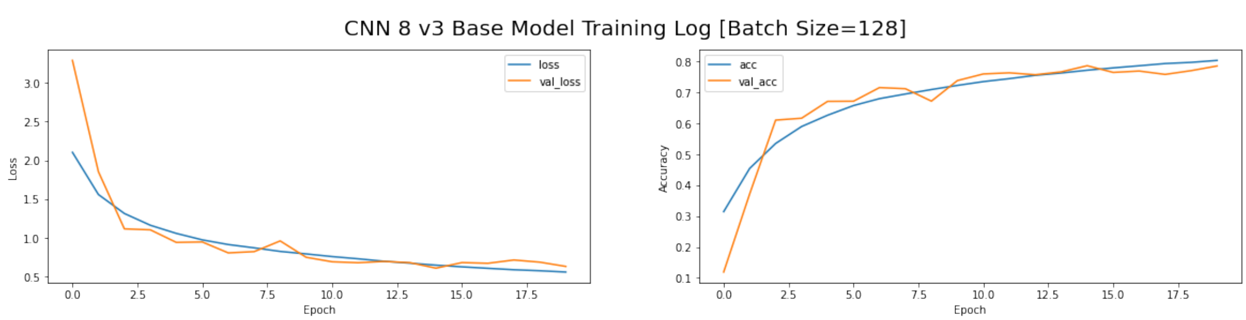
```
Epoch 1/500
313/313 [==============================] - 10s 16ms/step - loss:
2.1770 - accuracy: 0.2878 - val_loss: 3.9801 - val_accuracy: 0.1008
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss: 1.6460
- accuracy: 0.4177 - val_loss: 2.2256 - val_accuracy: 0.3109
Epoch 3/500
313/313 [==============================] - 4s 14ms/step - loss: 1.4107
- accuracy: 0.4973 - val_loss: 1.2811 - val_accuracy: 0.5520
Epoch 4/500
313/313 [==============================] - 4s 14ms/step - loss: 1.2762
- accuracy: 0.5489 - val_loss: 1.2783 - val_accuracy: 0.5571
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss: 1.1857
- accuracy: 0.5814 - val_loss: 1.4520 - val_accuracy: 0.4995
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss: 1.0979
- accuracy: 0.6169 - val_loss: 1.1311 - val_accuracy: 0.6099
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss: 1.0352
- accuracy: 0.6386 - val_loss: 1.0858 - val_accuracy: 0.6223
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss: 0.9832
- accuracy: 0.6568 - val_loss: 1.0073 - val_accuracy: 0.6511
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss: 0.9349
- accuracy: 0.6730 - val_loss: 0.9620 - val_accuracy: 0.6680
Epoch 10/500
313/313 [==============================] - 4s 14ms/step - loss: 0.8965
- accuracy: 0.6868 - val_loss: 0.8317 - val_accuracy: 0.7129
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss: 0.8659
- accuracy: 0.6972 - val_loss: 0.8406 - val_accuracy: 0.7126
Epoch 12/500
313/313 [==============================] - 5s 15ms/step - loss: 0.8336
- accuracy: 0.7117 - val_loss: 0.8456 - val_accuracy: 0.7091
Epoch 13/500
313/313 [==============================] - 5s 15ms/step - loss: 0.8104
- accuracy: 0.7179 - val_loss: 0.7620 - val_accuracy: 0.7393
```

```
Epoch 14/500
313/313 [==============================] - 4s 14ms/step - loss: 0.7885
- accuracy: 0.7273 - val_loss: 0.7806 - val_accuracy: 0.7309
Epoch 15/500
313/313 [==============================] - 4s 14ms/step - loss: 0.7637
- accuracy: 0.7352 - val_loss: 0.7915 - val_accuracy: 0.7324
Epoch 16/500
313/313 [==============================] - 4s 14ms/step - loss: 0.7454
- accuracy: 0.7426 - val_loss: 0.7038 - val_accuracy: 0.7600
Epoch 17/500
313/313 [==============================] - 4s 14ms/step - loss: 0.7190
- accuracy: 0.7510 - val_loss: 0.6717 - val_accuracy: 0.7713
Epoch 18/500
313/313 [==============================] - 4s 14ms/step - loss: 0.7011
- accuracy: 0.7566 - val_loss: 0.7988 - val_accuracy: 0.7305
Epoch 19/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6853
- accuracy: 0.7630 - val_loss: 0.6220 - val_accuracy: 0.7873
Epoch 20/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6714
- accuracy: 0.7708 - val_loss: 0.6388 - val_accuracy: 0.7809
Epoch 21/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6559
- accuracy: 0.7732 - val_loss: 0.7080 - val_accuracy: 0.7631
Epoch 22/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6401
- accuracy: 0.7796 - val_loss: 0.6479 - val_accuracy: 0.7788
Epoch 23/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6281
- accuracy: 0.7836 - val_loss: 0.6754 - val_accuracy: 0.7727
Epoch 24/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6152
- accuracy: 0.7878 - val_loss: 0.6518 - val_accuracy: 0.7791
```

```
preds = CNN_8_v3.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v3.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.87      0.71      0.78      1000
           1       0.94      0.87      0.90      1000
           2       0.77      0.62      0.69      1000
           3       0.65      0.59      0.62      1000
           4       0.68      0.84      0.75      1000
           5       0.79      0.61      0.69      1000
           6       0.71      0.91      0.80      1000
```

```
              7       0.87      0.81      0.84      1000
              8       0.82      0.90      0.86      1000
              9       0.77      0.93      0.85      1000

       accuracy                           0.78     10000
      macro avg       0.79      0.78      0.78     10000
   weighted avg       0.79      0.78      0.78     10000

313/313 - 1s - loss: 0.6602 - accuracy: 0.7804
Accuracy: 78.03999781608582
Macro F1-score: 0.7774112270954031

loss = CNN_8_v3_history.history['loss']
val_loss = CNN_8_v3_history.history['val_loss']
acc = CNN_8_v3_history.history['accuracy']
val_acc = CNN_8_v3_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v3 Base Model Training Log [Batch
Size=128]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Insights:

Batch size of 32 is the best learning rate as the allows for the BEST model performance of 83.5% which is currently the best model perfromance achieved. It has comprised just a little on the model training. However, I will be solving it data auugmentation

```
def tuning_loss_function(params,results_dict):
    for i in range(len(params)):
        CNN_8_v4=build_CNN_8()
        optimizer = tf.keras.optimizers.Adam(lr=0.0004)

CNN_8_v4.compile(optimizer=optimizer,loss=params[i],metrics=['accuracy
'])
        result=CNN_8_v4.fit(X_train, y_train, epochs=500,
batch_size=32, verbose=1, validation_split=0.2,
                      callbacks=[es_callback],validation_data=(X_test,
y_test))
        preds = CNN_8_v4.predict(X_test)

results_dict[params[i]]=round(f1_score(y_test,preds.argmax(axis=1),ave
rage="macro"),3)
    return results_dict
results_dict={}
loss_function=['sparse_categorical_crossentropy','mean_squared_logarit
hmic_error','mean_squared_error','hinge','kullback_leibler_divergence'
]
print(tuning_loss_function(loss_function,results_dict))
```

```
Epoch 1/500
313/313 [==============================] - 5s 15ms/step - loss: 2.2327
- accuracy: 0.2944 - val_loss: 3.5216 - val_accuracy: 0.1240
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss: 1.6791
- accuracy: 0.4292 - val_loss: 1.8480 - val_accuracy: 0.3458
Epoch 3/500
313/313 [==============================] - 4s 13ms/step - loss: 1.4412
- accuracy: 0.5010 - val_loss: 1.2119 - val_accuracy: 0.5796
Epoch 4/500
313/313 [==============================] - 4s 14ms/step - loss: 1.2714
- accuracy: 0.5578 - val_loss: 1.0369 - val_accuracy: 0.6337
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss: 1.1426
- accuracy: 0.6018 - val_loss: 1.0177 - val_accuracy: 0.6508
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss: 1.0651
- accuracy: 0.6290 - val_loss: 0.9709 - val_accuracy: 0.6663
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss: 0.9928
```

```
- accuracy: 0.6532 - val_loss: 1.0027 - val_accuracy: 0.6603
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss: 0.9313
- accuracy: 0.6755 - val_loss: 0.8568 - val_accuracy: 0.7051
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss: 0.8815
- accuracy: 0.6906 - val_loss: 0.8027 - val_accuracy: 0.7213
Epoch 10/500
313/313 [==============================] - 4s 14ms/step - loss: 0.8386
- accuracy: 0.7074 - val_loss: 0.7865 - val_accuracy: 0.7263
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss: 0.8066
- accuracy: 0.7194 - val_loss: 0.7889 - val_accuracy: 0.7332
Epoch 12/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7689
- accuracy: 0.7315 - val_loss: 0.7404 - val_accuracy: 0.7441
Epoch 13/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7418
- accuracy: 0.7376 - val_loss: 0.7219 - val_accuracy: 0.7503
Epoch 14/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7065
- accuracy: 0.7549 - val_loss: 0.7460 - val_accuracy: 0.7427
Epoch 15/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6852
- accuracy: 0.7631 - val_loss: 0.6676 - val_accuracy: 0.7669
Epoch 16/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6576
- accuracy: 0.7704 - val_loss: 0.6657 - val_accuracy: 0.7693
Epoch 17/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6310
- accuracy: 0.7800 - val_loss: 0.6363 - val_accuracy: 0.7802
Epoch 18/500
313/313 [==============================] - 4s 14ms/step - loss: 0.6131
- accuracy: 0.7855 - val_loss: 0.6283 - val_accuracy: 0.7843
Epoch 19/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5916
- accuracy: 0.7915 - val_loss: 0.6586 - val_accuracy: 0.7763
Epoch 20/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5716
- accuracy: 0.8002 - val_loss: 0.6080 - val_accuracy: 0.7938
Epoch 21/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5475
- accuracy: 0.8091 - val_loss: 0.6196 - val_accuracy: 0.7895
Epoch 22/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5359
- accuracy: 0.8105 - val_loss: 0.5794 - val_accuracy: 0.8040
Epoch 23/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5127
- accuracy: 0.8222 - val_loss: 0.6076 - val_accuracy: 0.7942
```

```
Epoch 24/500
313/313 [==============================] - 4s 14ms/step - loss: 0.4953
- accuracy: 0.8243 - val_loss: 0.6226 - val_accuracy: 0.7902
Epoch 25/500
313/313 [==============================] - 4s 14ms/step - loss: 0.4835
- accuracy: 0.8301 - val_loss: 0.6212 - val_accuracy: 0.7891
Epoch 26/500
313/313 [==============================] - 4s 13ms/step - loss: 0.4693
- accuracy: 0.8356 - val_loss: 0.5693 - val_accuracy: 0.8085
Epoch 27/500
313/313 [==============================] - 4s 13ms/step - loss: 0.4549
- accuracy: 0.8385 - val_loss: 0.5600 - val_accuracy: 0.8162
Epoch 28/500
313/313 [==============================] - 4s 13ms/step - loss: 0.4373
- accuracy: 0.8476 - val_loss: 0.5739 - val_accuracy: 0.8062
Epoch 29/500
313/313 [==============================] - 4s 13ms/step - loss: 0.4252
- accuracy: 0.8497 - val_loss: 0.5547 - val_accuracy: 0.8111
Epoch 30/500
313/313 [==============================] - 4s 14ms/step - loss: 0.4136
- accuracy: 0.8557 - val_loss: 0.5646 - val_accuracy: 0.8125
Epoch 31/500
313/313 [==============================] - 4s 13ms/step - loss: 0.4021
- accuracy: 0.8572 - val_loss: 0.5521 - val_accuracy: 0.8189
Epoch 32/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3845
- accuracy: 0.8632 - val_loss: 0.5722 - val_accuracy: 0.8102
Epoch 33/500
313/313 [==============================] - 4s 13ms/step - loss: 0.3807
- accuracy: 0.8658 - val_loss: 0.5573 - val_accuracy: 0.8181
Epoch 34/500
313/313 [==============================] - 4s 13ms/step - loss: 0.3621
- accuracy: 0.8716 - val_loss: 0.6136 - val_accuracy: 0.8000
Epoch 35/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3552
- accuracy: 0.8751 - val_loss: 0.5535 - val_accuracy: 0.8153
Epoch 36/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3445
- accuracy: 0.8775 - val_loss: 0.5398 - val_accuracy: 0.8238
Epoch 37/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3347
- accuracy: 0.8826 - val_loss: 0.5706 - val_accuracy: 0.8160
Epoch 38/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3243
- accuracy: 0.8850 - val_loss: 0.5621 - val_accuracy: 0.8211
Epoch 39/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3185
- accuracy: 0.8880 - val_loss: 0.5571 - val_accuracy: 0.8179
Epoch 40/500
```

```
313/313 [==============================] - 4s 14ms/step - loss: 0.3061
- accuracy: 0.8922 - val_loss: 0.5409 - val_accuracy: 0.8248
Epoch 41/500
313/313 [==============================] - 4s 14ms/step - loss: 0.3000
- accuracy: 0.8936 - val_loss: 0.5646 - val_accuracy: 0.8228
Epoch 1/500
313/313 [==============================] - 5s 14ms/step - loss: 2.5041
- accuracy: 0.0986 - val_loss: 2.4873 - val_accuracy: 0.1046
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4927
- accuracy: 0.1011 - val_loss: 2.4874 - val_accuracy: 0.1157
Epoch 3/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4906
- accuracy: 0.1021 - val_loss: 2.4875 - val_accuracy: 0.0861
Epoch 4/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4889
- accuracy: 0.1030 - val_loss: 2.4876 - val_accuracy: 0.0920
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4881
- accuracy: 0.1031 - val_loss: 2.4889 - val_accuracy: 0.0904
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4876
- accuracy: 0.0983 - val_loss: 2.4873 - val_accuracy: 0.0965
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4875
- accuracy: 0.1013 - val_loss: 2.4872 - val_accuracy: 0.0994
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4873
- accuracy: 0.1020 - val_loss: 2.4876 - val_accuracy: 0.0893
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4873
- accuracy: 0.1029 - val_loss: 2.4872 - val_accuracy: 0.0986
Epoch 10/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4872
- accuracy: 0.1015 - val_loss: 2.4878 - val_accuracy: 0.1051
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4870
- accuracy: 0.0997 - val_loss: 2.4878 - val_accuracy: 0.1026
Epoch 12/500
313/313 [==============================] - 4s 14ms/step - loss: 2.4872
- accuracy: 0.0992 - val_loss: 2.4880 - val_accuracy: 0.1014
Epoch 1/500
313/313 [==============================] - 5s 14ms/step - loss:
27.6048 - accuracy: 0.1000 - val_loss: 27.6773 - val_accuracy: 0.0952
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5982 - accuracy: 0.1008 - val_loss: 27.6767 - val_accuracy: 0.0997
Epoch 3/500
313/313 [==============================] - 4s 14ms/step - loss:
```

```
27.5969 - accuracy: 0.0981 - val_loss: 27.6769 - val_accuracy: 0.0950
Epoch 4/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5961 - accuracy: 0.1018 - val_loss: 27.6766 - val_accuracy: 0.1134
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5953 - accuracy: 0.1025 - val_loss: 27.6766 - val_accuracy: 0.1130
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5948 - accuracy: 0.1007 - val_loss: 27.6823 - val_accuracy: 0.1058
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5945 - accuracy: 0.0998 - val_loss: 27.6766 - val_accuracy: 0.1120
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5945 - accuracy: 0.0998 - val_loss: 27.6766 - val_accuracy: 0.1167
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5944 - accuracy: 0.0993 - val_loss: 27.6768 - val_accuracy: 0.0907
Epoch 10/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5943 - accuracy: 0.0979 - val_loss: 27.6776 - val_accuracy: 0.0946
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5943 - accuracy: 0.0998 - val_loss: 27.6767 - val_accuracy: 0.0938
Epoch 12/500
313/313 [==============================] - 4s 14ms/step - loss:
27.5946 - accuracy: 0.1005 - val_loss: 27.6884 - val_accuracy: 0.1199
Epoch 1/500
313/313 [==============================] - 5s 14ms/step - loss: 0.6426
- accuracy: 0.0982 - val_loss: 0.5546 - val_accuracy: 0.0816
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5854
- accuracy: 0.0988 - val_loss: 0.5506 - val_accuracy: 0.1161
Epoch 3/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5703
- accuracy: 0.1021 - val_loss: 0.5505 - val_accuracy: 0.0945
Epoch 4/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5611
- accuracy: 0.1021 - val_loss: 0.5499 - val_accuracy: 0.1143
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5560
- accuracy: 0.0990 - val_loss: 0.5501 - val_accuracy: 0.1090
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5542
- accuracy: 0.1020 - val_loss: 0.5498 - val_accuracy: 0.1047
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5534
- accuracy: 0.1003 - val_loss: 0.5527 - val_accuracy: 0.1004
```

```
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5534
- accuracy: 0.1014 - val_loss: 0.5500 - val_accuracy: 0.0973
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5524
- accuracy: 0.1041 - val_loss: 0.5498 - val_accuracy: 0.1424
Epoch 10/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5523
- accuracy: 0.1022 - val_loss: 0.5509 - val_accuracy: 0.1027
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5528
- accuracy: 0.0966 - val_loss: 0.5511 - val_accuracy: 0.0806
Epoch 12/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5527
- accuracy: 0.0996 - val_loss: 0.5498 - val_accuracy: 0.1074
Epoch 13/500
313/313 [==============================] - 5s 15ms/step - loss: 0.5518
- accuracy: 0.1051 - val_loss: 0.5577 - val_accuracy: 0.0901
Epoch 14/500
313/313 [==============================] - 4s 14ms/step - loss: 0.5524
- accuracy: 0.0980 - val_loss: 0.5541 - val_accuracy: 0.0698
Epoch 1/500
313/313 [==============================] - 6s 15ms/step - loss:
24.7195 - accuracy: 0.1008 - val_loss: 20.7745 - val_accuracy: 0.1005
Epoch 2/500
313/313 [==============================] - 4s 14ms/step - loss:
22.2375 - accuracy: 0.0995 - val_loss: 20.7193 - val_accuracy: 0.0699
Epoch 3/500
313/313 [==============================] - 4s 14ms/step - loss:
21.7067 - accuracy: 0.0990 - val_loss: 20.7907 - val_accuracy: 0.0657
Epoch 4/500
313/313 [==============================] - 5s 14ms/step - loss:
21.3715 - accuracy: 0.0997 - val_loss: 20.7876 - val_accuracy: 0.0974
Epoch 5/500
313/313 [==============================] - 4s 14ms/step - loss:
21.1775 - accuracy: 0.1017 - val_loss: 20.6952 - val_accuracy: 0.1017
Epoch 6/500
313/313 [==============================] - 4s 14ms/step - loss:
21.0892 - accuracy: 0.1003 - val_loss: 21.2302 - val_accuracy: 0.0951
Epoch 7/500
313/313 [==============================] - 4s 14ms/step - loss:
21.0070 - accuracy: 0.1036 - val_loss: 20.7138 - val_accuracy: 0.0981
Epoch 8/500
313/313 [==============================] - 4s 14ms/step - loss:
20.9709 - accuracy: 0.0995 - val_loss: 20.7036 - val_accuracy: 0.1061
Epoch 9/500
313/313 [==============================] - 4s 14ms/step - loss:
20.9621 - accuracy: 0.1019 - val_loss: 20.6926 - val_accuracy: 0.0875
Epoch 10/500
```

```
313/313 [==============================] - 4s 14ms/step - loss:
20.9322 - accuracy: 0.0970 - val_loss: 20.7179 - val_accuracy: 0.0861
Epoch 11/500
313/313 [==============================] - 4s 14ms/step - loss:
20.9016 - accuracy: 0.1005 - val_loss: 20.6937 - val_accuracy: 0.1031
Epoch 12/500
313/313 [==============================] - 4s 14ms/step - loss:
20.9016 - accuracy: 0.0977 - val_loss: 20.7981 - val_accuracy: 0.0705
Epoch 13/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8463 - accuracy: 0.1005 - val_loss: 20.7550 - val_accuracy: 0.0720
Epoch 14/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8464 - accuracy: 0.1007 - val_loss: 20.6916 - val_accuracy: 0.0821
Epoch 15/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8485 - accuracy: 0.0980 - val_loss: 20.7123 - val_accuracy: 0.1050
Epoch 16/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8367 - accuracy: 0.0983 - val_loss: 20.7108 - val_accuracy: 0.1186
Epoch 17/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8323 - accuracy: 0.1041 - val_loss: 20.6980 - val_accuracy: 0.1079
Epoch 18/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8419 - accuracy: 0.1029 - val_loss: 20.9596 - val_accuracy: 0.1276
Epoch 19/500
313/313 [==============================] - 4s 14ms/step - loss:
20.8243 - accuracy: 0.1037 - val_loss: 20.7275 - val_accuracy: 0.1046
{'sparse_categorical_crossentropy': 0.814,
'mean_squared_logarithmic_error': 0.035, 'mean_squared_error': 0.075,
'hinge': 0.035, 'kullback_leibler_divergence': 0.073}
```

```
print("F1-score of different loss function:",
{'sparse_categorical_crossentropy': 0.814,
'mean_squared_logarithmic_error': 0.035, 'mean_squared_error': 0.075,
'hinge': 0.035, 'kullback_leibler_divergence': 0.073})
```

```
F1-score of different loss function:
{'sparse_categorical_crossentropy': 0.814,
'mean_squared_logarithmic_error': 0.035, 'mean_squared_error': 0.075,
'hinge': 0.035, 'kullback_leibler_divergence': 0.073}
```

After having tried a variety of loss function, the loss function sparse_categorical_crossentropy gives the highest f1-score of 81%. Hence, I will continue to use sparse_categorical_crossentropy in the later sections

```
CNN_8_v4=build_CNN_8()
optimizer = tf.keras.optimizers.Adam(lr=0.0004)
```

```
CNN_8_v4.compile(optimizer=optimizer,loss='sparse_categorical_crossent
ropy',metrics=['accuracy'])
CNN_8_v4_history=CNN_8_v4.fit(X_train, y_train, epochs=500,
batch_size=32, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))
```

```
Epoch 1/500
1250/1250 [==============================] - 9s 7ms/step - loss:
1.9175 - accuracy: 0.3552 - val_loss: 1.6902 - val_accuracy: 0.4035
Epoch 2/500
1250/1250 [==============================] - 8s 6ms/step - loss:
1.3492 - accuracy: 0.5243 - val_loss: 1.2262 - val_accuracy: 0.5685
Epoch 3/500
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1177 - accuracy: 0.6086 - val_loss: 0.9235 - val_accuracy: 0.6765
Epoch 4/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9929 - accuracy: 0.6556 - val_loss: 0.8302 - val_accuracy: 0.7084
Epoch 5/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9085 - accuracy: 0.6841 - val_loss: 0.8928 - val_accuracy: 0.6932
Epoch 6/500
1250/1250 [==============================] - 7s 6ms/step - loss:
0.8399 - accuracy: 0.7065 - val_loss: 0.7325 - val_accuracy: 0.7414
Epoch 7/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.7946 - accuracy: 0.7252 - val_loss: 0.6678 - val_accuracy: 0.7685
Epoch 8/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.7450 - accuracy: 0.7436 - val_loss: 0.6351 - val_accuracy: 0.7793
Epoch 9/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7095 - accuracy: 0.7564 - val_loss: 0.6085 - val_accuracy: 0.7880
Epoch 10/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.6812 - accuracy: 0.7633 - val_loss: 0.5747 - val_accuracy: 0.8000
Epoch 11/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.6545 - accuracy: 0.7721 - val_loss: 0.5645 - val_accuracy: 0.8053
Epoch 12/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.6278 - accuracy: 0.7829 - val_loss: 0.5794 - val_accuracy: 0.7982
Epoch 13/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5996 - accuracy: 0.7919 - val_loss: 0.5708 - val_accuracy: 0.8036
Epoch 14/500
1250/1250 [==============================] - 9s 8ms/step - loss:
0.5876 - accuracy: 0.7975 - val_loss: 0.5644 - val_accuracy: 0.8062
Epoch 15/500
```

```
1250/1250 [==============================] - 10s 8ms/step - loss:
0.5722 - accuracy: 0.8015 - val_loss: 0.5452 - val_accuracy: 0.8160
Epoch 16/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.5511 - accuracy: 0.8107 - val_loss: 0.5472 - val_accuracy: 0.8076
Epoch 17/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.5433 - accuracy: 0.8113 - val_loss: 0.5377 - val_accuracy: 0.8147
Epoch 18/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5281 - accuracy: 0.8176 - val_loss: 0.5180 - val_accuracy: 0.8231
Epoch 19/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5078 - accuracy: 0.8224 - val_loss: 0.5547 - val_accuracy: 0.8114
Epoch 20/500
1250/1250 [==============================] - 8s 7ms/step - loss:
0.5008 - accuracy: 0.8257 - val_loss: 0.5108 - val_accuracy: 0.8278
Epoch 21/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4893 - accuracy: 0.8316 - val_loss: 0.5268 - val_accuracy: 0.8237
Epoch 22/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4834 - accuracy: 0.8324 - val_loss: 0.5242 - val_accuracy: 0.8227
Epoch 23/500
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4699 - accuracy: 0.8372 - val_loss: 0.4884 - val_accuracy: 0.8349
Epoch 24/500
1250/1250 [==============================] - 9s 8ms/step - loss:
0.4675 - accuracy: 0.8365 - val_loss: 0.5001 - val_accuracy: 0.8308
Epoch 25/500
1250/1250 [==============================] - 9s 8ms/step - loss:
0.4557 - accuracy: 0.8427 - val_loss: 0.5068 - val_accuracy: 0.8304
Epoch 26/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4473 - accuracy: 0.8448 - val_loss: 0.5030 - val_accuracy: 0.8324
Epoch 27/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4366 - accuracy: 0.8479 - val_loss: 0.5022 - val_accuracy: 0.8296
Epoch 28/500
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4300 - accuracy: 0.8510 - val_loss: 0.5100 - val_accuracy: 0.8304
```

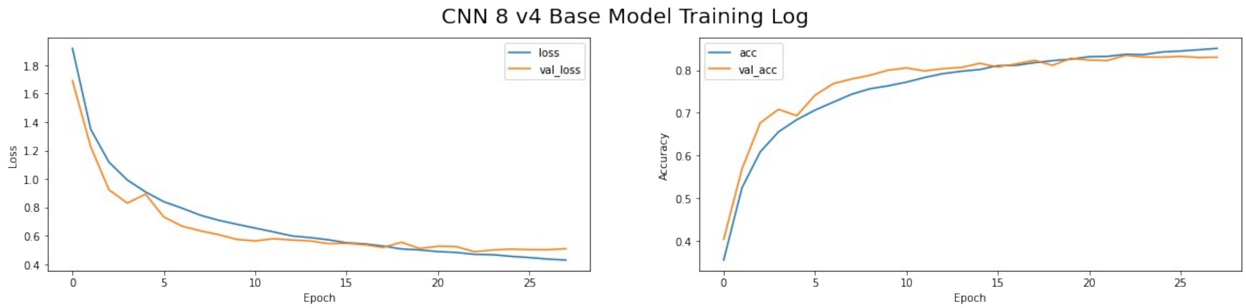
Now I will be using the sparse_categorical_crossentropy loss function to compile the built model to record this version of the model

```
preds = CNN_8_v4.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_8_v4.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
```

```
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      1000
           1       0.92      0.93      0.92      1000
           2       0.83      0.67      0.74      1000
           3       0.75      0.61      0.67      1000
           4       0.70      0.87      0.78      1000
           5       0.75      0.78      0.77      1000
           6       0.85      0.88      0.87      1000
           7       0.86      0.88      0.87      1000
           8       0.92      0.86      0.89      1000
           9       0.87      0.93      0.90      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.82     10000
weighted avg       0.83      0.83      0.82     10000

313/313 - 1s - loss: 0.5291 - accuracy: 0.8264
Accuracy: 82.63999819755554
Macro F1-score: 0.8245499349288821
```

```
loss = CNN_8_v4_history.history['loss']
val_loss = CNN_8_v4_history.history['val_loss']
acc = CNN_8_v4_history.history['accuracy']
val_acc = CNN_8_v4_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 8 v4 Base Model Training Log",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



This will be the final model training log after tuning all hyperparameters in the compile function. As shown there is very sligh underfitting towards the end which I will be testing out with data augmentation to solve the issue.

TO summarize, the best combination of hyperparamters would be:

1. Optimizer: Adam
2. Learning Rate: 0.0004
3. sparse_categorical_crossentropy
4. batch size: 32

Let's compare the summary of all model results:

Conclusion: CNN 8 v4[With Tuned Hyperparameters] F1-Score is 82%

3.4 Tuning Images with Data Augmentation

For this section, we will be using CNN 8 version 4 with the tuned parameters in the previous section.

In this section, I will be using data augementation, as it is useful a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.Image data augmentation is used to expand the training dataset in order to improve the performance and ability of the model to generalize.

```
def build_CNN_9():
    model=Sequential()
    model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,1)))
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.3))

    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
```

```
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.4))


    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2,2))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(128,activation='tanh'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(Dense(10,activation='softmax'))
    optimizer = tf.keras.optimizers.Adam(lr=0.0004)

model.compile(optimizer=optimizer,loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
    return model
```

These are the images before data augmentation

```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 6))
plt.suptitle("Images Before Augmentation",fontsize=20)
for i in range(21):
    plt.subplot(3,7,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(data.iloc[i,:-1].values.reshape(32,32),cmap="gray")
#reshaped matrix
    #plt.xlabel(class_names[y_train[i][0]])
plt.show()
```

Images Before Augmentation

First, I will be building the model with no data augmentation for better comparison in the later stage

```
CNN_9_v1_no_aug = build_CNN_9()

history_no_aug = CNN_9_v1_no_aug.fit(X_train, y_train, epochs=500,
batch_size=128, verbose=1, validation_split=0.2,
              callbacks=[es_callback],validation_data=(X_test,
y_test))

loss_no_aug, acc_no_aug = CNN_9_v1_no_aug.evaluate(X_test,  y_test)

Epoch 1/500
313/313 [==============================] - 5s 14ms/step - loss: 2.2105
- accuracy: 0.2857 - val_loss: 4.0425 - val_accuracy: 0.1335
Epoch 2/500
313/313 [==============================] - 4s 13ms/step - loss: 1.6681
- accuracy: 0.4110 - val_loss: 1.9409 - val_accuracy: 0.3568
Epoch 3/500
313/313 [==============================] - 4s 13ms/step - loss: 1.4548
- accuracy: 0.4828 - val_loss: 1.3865 - val_accuracy: 0.5066
Epoch 4/500
313/313 [==============================] - 4s 13ms/step - loss: 1.2827
- accuracy: 0.5455 - val_loss: 1.2886 - val_accuracy: 0.5457
Epoch 5/500
313/313 [==============================] - 4s 13ms/step - loss: 1.1718
- accuracy: 0.5881 - val_loss: 1.0920 - val_accuracy: 0.6149
Epoch 6/500
313/313 [==============================] - 4s 13ms/step - loss: 1.0912
- accuracy: 0.6197 - val_loss: 1.0606 - val_accuracy: 0.6252
Epoch 7/500
313/313 [==============================] - 4s 13ms/step - loss: 1.0251
```

```
- accuracy: 0.6420 - val_loss: 0.9748 - val_accuracy: 0.6567
Epoch 8/500
313/313 [==============================] - 4s 13ms/step - loss: 0.9734
- accuracy: 0.6615 - val_loss: 0.9064 - val_accuracy: 0.6839
Epoch 9/500
313/313 [==============================] - 4s 13ms/step - loss: 0.9241
- accuracy: 0.6783 - val_loss: 0.9581 - val_accuracy: 0.6693
Epoch 10/500
313/313 [==============================] - 4s 13ms/step - loss: 0.8920
- accuracy: 0.6879 - val_loss: 0.7878 - val_accuracy: 0.7242
Epoch 11/500
313/313 [==============================] - 4s 13ms/step - loss: 0.8599
- accuracy: 0.7022 - val_loss: 0.9474 - val_accuracy: 0.6758
Epoch 12/500
313/313 [==============================] - 4s 13ms/step - loss: 0.8294
- accuracy: 0.7115 - val_loss: 0.7480 - val_accuracy: 0.7408
Epoch 13/500
313/313 [==============================] - 4s 13ms/step - loss: 0.8023
- accuracy: 0.7214 - val_loss: 0.7638 - val_accuracy: 0.7398
Epoch 14/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7784
- accuracy: 0.7310 - val_loss: 0.7095 - val_accuracy: 0.7560
Epoch 15/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7499
- accuracy: 0.7403 - val_loss: 0.7710 - val_accuracy: 0.7413
Epoch 16/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7378
- accuracy: 0.7464 - val_loss: 0.7048 - val_accuracy: 0.7608
Epoch 17/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7204
- accuracy: 0.7511 - val_loss: 0.7966 - val_accuracy: 0.7309
Epoch 18/500
313/313 [==============================] - 4s 13ms/step - loss: 0.7010
- accuracy: 0.7566 - val_loss: 0.6830 - val_accuracy: 0.7634
Epoch 19/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6825
- accuracy: 0.7640 - val_loss: 0.7263 - val_accuracy: 0.7575
Epoch 20/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6690
- accuracy: 0.7703 - val_loss: 0.6309 - val_accuracy: 0.7817
Epoch 21/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6478
- accuracy: 0.7759 - val_loss: 0.6697 - val_accuracy: 0.7727
Epoch 22/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6337
- accuracy: 0.7794 - val_loss: 0.6920 - val_accuracy: 0.7659
Epoch 23/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6230
- accuracy: 0.7838 - val_loss: 0.6905 - val_accuracy: 0.7678
```

```
Epoch 24/500
313/313 [==============================] - 4s 13ms/step - loss: 0.6105
- accuracy: 0.7904 - val_loss: 0.7636 - val_accuracy: 0.7421
Epoch 25/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5990
- accuracy: 0.7933 - val_loss: 0.5976 - val_accuracy: 0.7953
Epoch 26/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5897
- accuracy: 0.7965 - val_loss: 0.6274 - val_accuracy: 0.7891
Epoch 27/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5808
- accuracy: 0.7986 - val_loss: 0.6574 - val_accuracy: 0.7753
Epoch 28/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5721
- accuracy: 0.8026 - val_loss: 0.5640 - val_accuracy: 0.8084
Epoch 29/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5577
- accuracy: 0.8090 - val_loss: 0.6135 - val_accuracy: 0.7894
Epoch 30/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5496
- accuracy: 0.8106 - val_loss: 0.5971 - val_accuracy: 0.8010
Epoch 31/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5343
- accuracy: 0.8171 - val_loss: 0.5835 - val_accuracy: 0.8045
Epoch 32/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5292
- accuracy: 0.8152 - val_loss: 0.5709 - val_accuracy: 0.8108
Epoch 33/500
313/313 [==============================] - 4s 13ms/step - loss: 0.5269
- accuracy: 0.8156 - val_loss: 0.5701 - val_accuracy: 0.8088
313/313 [==============================] - 1s 3ms/step - loss: 0.6011
- accuracy: 0.8050
```

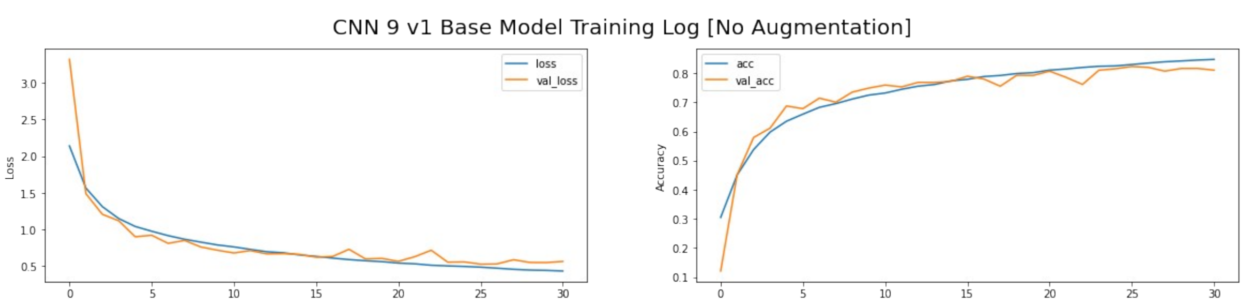
```
preds = CNN_9_v1_no_aug.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_9_v1_no_aug.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.90      0.75      0.82      1000
           1       0.94      0.89      0.92      1000
           2       0.82      0.66      0.73      1000
           3       0.67      0.64      0.65      1000
           4       0.71      0.85      0.77      1000
           5       0.71      0.76      0.74      1000
           6       0.77      0.90      0.83      1000
           7       0.91      0.83      0.87      1000
```

```
           8       0.84      0.93      0.88      1000
           9       0.88      0.91      0.90      1000

    accuracy                           0.81     10000
   macro avg       0.82      0.81      0.81     10000
weighted avg       0.82      0.81      0.81     10000

313/313 - 1s - loss: 0.5794 - accuracy: 0.8109
Accuracy: 81.08999729156494
Macro F1-score: 0.8103610834021039
```

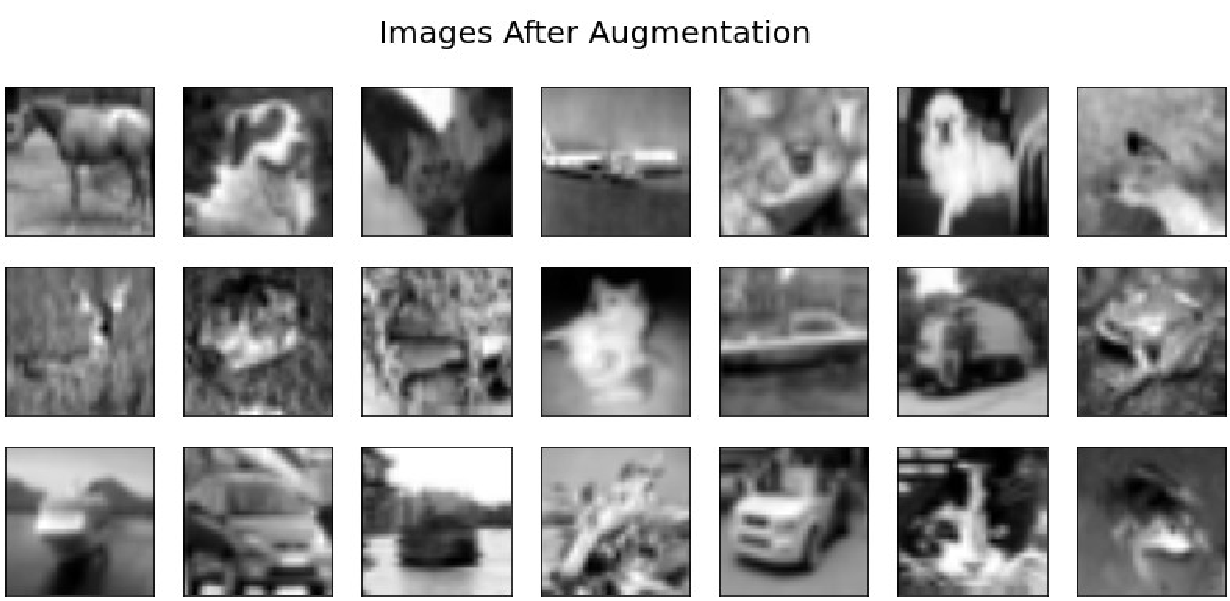
We can see from the results, that with no data augmentation, model results is at 81% for f1-score, accuracy, precision and recall. Now I will be augmenting the data, where I will be flipping the image with the Horizontal and Vertical Shift/Flip Augmentation

```
loss = history_no_aug.history['loss']
val_loss = history_no_aug.history['val_loss']
acc = history_no_aug.history['accuracy']
val_acc = history_no_aug.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 9 v1 Base Model Training Log [No
Augmentation]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



We can see the model training log is quite good with very slight fluctuation at some point

```
def visualize_data(images, categories, class_names):
    fig = plt.figure(figsize=(14, 6))
    plt.suptitle("Images After Augmentation",fontsize=20)
    fig.patch.set_facecolor('white')
    for i in range(3 * 7):
        plt.subplot(3, 7, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(images[i],cmap='gray')
        class_index = categories[i].argmax()
        #plt.xlabel(class_names[class_index])
    plt.show()
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']
width_shift = 1.0
height_shift = 1.0
flip = True

datagen = ImageDataGenerator(
    horizontal_flip=flip,
    width_shift_range=width_shift,
    height_shift_range=height_shift,
    )
datagen.fit(X_train)

it = datagen.flow(X_train, y_train, shuffle=True)
batch_images, batch_labels = next(it)
visualize_data(batch_images, batch_labels, class_names)
```



```
CNN_9_v1_with_aug = build_CNN_9()
datagen.fit(X_train)

history_aug = CNN_9_v1_with_aug.fit(X_train, y_train, epochs=50,
batch_size=32, verbose=1,
validation_split=0.2,validation_data=(X_test, y_test))
loss_aug, acc_aug = CNN_9_v1_with_aug.evaluate(X_test,  y_test)
```

```
Epoch 1/50
1250/1250 [==============================] - 12s 6ms/step - loss:
1.9941 - accuracy: 0.3242 - val_loss: 1.6944 - val_accuracy: 0.3993
Epoch 2/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.4991 - accuracy: 0.4637 - val_loss: 1.3555 - val_accuracy: 0.5066
Epoch 3/50
1250/1250 [==============================] - 7s 6ms/step - loss:
1.2612 - accuracy: 0.5567 - val_loss: 1.1094 - val_accuracy: 0.6053
Epoch 4/50
1250/1250 [==============================] - 7s 6ms/step - loss:
1.1307 - accuracy: 0.6076 - val_loss: 0.9817 - val_accuracy: 0.6575
Epoch 5/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0403 - accuracy: 0.6393 - val_loss: 0.9481 - val_accuracy: 0.6716
Epoch 6/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9805 - accuracy: 0.6604 - val_loss: 0.8269 - val_accuracy: 0.7110
Epoch 7/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.9320 - accuracy: 0.6780 - val_loss: 0.9518 - val_accuracy: 0.6738
Epoch 8/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8930 - accuracy: 0.6932 - val_loss: 1.0203 - val_accuracy: 0.6519
Epoch 9/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.8600 - accuracy: 0.7035 - val_loss: 0.7354 - val_accuracy: 0.7462
Epoch 10/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.8267 - accuracy: 0.7162 - val_loss: 0.7159 - val_accuracy: 0.7576
Epoch 11/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8021 - accuracy: 0.7258 - val_loss: 0.6952 - val_accuracy: 0.7592
Epoch 12/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.7749 - accuracy: 0.7358 - val_loss: 0.6948 - val_accuracy: 0.7643
Epoch 13/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.7557 - accuracy: 0.7429 - val_loss: 0.6744 - val_accuracy: 0.7731
Epoch 14/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.7437 - accuracy: 0.7469 - val_loss: 0.6531 - val_accuracy: 0.7783
```

```
Epoch 15/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.7144 - accuracy: 0.7560 - val_loss: 0.7564 - val_accuracy: 0.7461
Epoch 16/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.7038 - accuracy: 0.7568 - val_loss: 0.6615 - val_accuracy: 0.7747
Epoch 17/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.6862 - accuracy: 0.7666 - val_loss: 0.6238 - val_accuracy: 0.7851
Epoch 18/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.6751 - accuracy: 0.7700 - val_loss: 0.6446 - val_accuracy: 0.7799
Epoch 19/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.6657 - accuracy: 0.7731 - val_loss: 0.6205 - val_accuracy: 0.7886
Epoch 20/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.6529 - accuracy: 0.7774 - val_loss: 0.5976 - val_accuracy: 0.7991
Epoch 21/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.6479 - accuracy: 0.7801 - val_loss: 0.7041 - val_accuracy: 0.7707
Epoch 22/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.6335 - accuracy: 0.7847 - val_loss: 0.5885 - val_accuracy: 0.8022
Epoch 23/50
1250/1250 [==============================] - 7s 6ms/step - loss: 
0.6216 - accuracy: 0.7883 - val_loss: 0.6288 - val_accuracy: 0.7889
Epoch 24/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.6080 - accuracy: 0.7944 - val_loss: 0.6009 - val_accuracy: 0.7970
Epoch 25/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.6114 - accuracy: 0.7936 - val_loss: 0.5870 - val_accuracy: 0.8055
Epoch 26/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.5929 - accuracy: 0.7985 - val_loss: 0.5619 - val_accuracy: 0.8120
Epoch 27/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.5907 - accuracy: 0.7986 - val_loss: 0.5727 - val_accuracy: 0.8092
Epoch 28/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.5797 - accuracy: 0.8033 - val_loss: 0.6944 - val_accuracy: 0.7699
Epoch 29/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.5765 - accuracy: 0.8046 - val_loss: 0.5644 - val_accuracy: 0.8133
Epoch 30/50
1250/1250 [==============================] - 8s 6ms/step - loss: 
0.5707 - accuracy: 0.8059 - val_loss: 0.5894 - val_accuracy: 0.8065
Epoch 31/50
```

```
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5711 - accuracy: 0.8062 - val_loss: 0.5841 - val_accuracy: 0.8079
Epoch 32/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5579 - accuracy: 0.8100 - val_loss: 0.5529 - val_accuracy: 0.8162
Epoch 33/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5556 - accuracy: 0.8115 - val_loss: 0.5705 - val_accuracy: 0.8092
Epoch 34/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5459 - accuracy: 0.8154 - val_loss: 0.5875 - val_accuracy: 0.8061
Epoch 35/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5426 - accuracy: 0.8176 - val_loss: 0.5692 - val_accuracy: 0.8107
Epoch 36/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5386 - accuracy: 0.8171 - val_loss: 0.5545 - val_accuracy: 0.8163
Epoch 37/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5373 - accuracy: 0.8191 - val_loss: 0.5476 - val_accuracy: 0.8161
Epoch 38/50
1250/1250 [==============================] - 8s 7ms/step - loss:
0.5273 - accuracy: 0.8203 - val_loss: 0.5659 - val_accuracy: 0.8141
Epoch 39/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5223 - accuracy: 0.8236 - val_loss: 0.5512 - val_accuracy: 0.8166
Epoch 40/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5252 - accuracy: 0.8217 - val_loss: 0.6081 - val_accuracy: 0.8010
Epoch 41/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5151 - accuracy: 0.8249 - val_loss: 0.5592 - val_accuracy: 0.8155
Epoch 42/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5112 - accuracy: 0.8259 - val_loss: 0.5287 - val_accuracy: 0.8254
Epoch 43/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5050 - accuracy: 0.8275 - val_loss: 0.5427 - val_accuracy: 0.8192
Epoch 44/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.5017 - accuracy: 0.8282 - val_loss: 0.5483 - val_accuracy: 0.8223
Epoch 45/50
1250/1250 [==============================] - 7s 6ms/step - loss:
0.4979 - accuracy: 0.8291 - val_loss: 0.5401 - val_accuracy: 0.8241
Epoch 46/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.5007 - accuracy: 0.8312 - val_loss: 0.5389 - val_accuracy: 0.8243
Epoch 47/50
1250/1250 [==============================] - 8s 7ms/step - loss:
```

```
0.4925 - accuracy: 0.8323 - val_loss: 0.5295 - val_accuracy: 0.8247
Epoch 48/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.4908 - accuracy: 0.8331 - val_loss: 0.5991 - val_accuracy: 0.8040
Epoch 49/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4881 - accuracy: 0.8324 - val_loss: 0.5381 - val_accuracy: 0.8204
Epoch 50/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.4770 - accuracy: 0.8378 - val_loss: 0.5633 - val_accuracy: 0.8163
313/313 [==============================] - 1s 2ms/step - loss: 0.5775
- accuracy: 0.8151
```

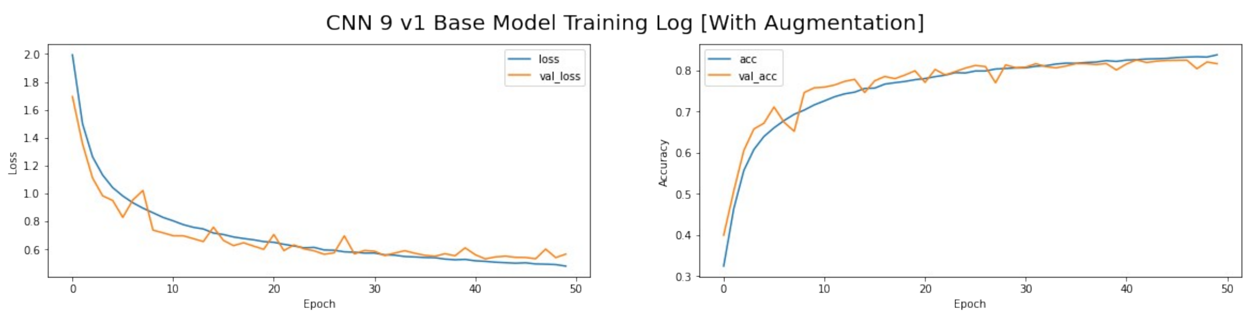
```
preds = CNN_9_v1_with_aug.predict(X_test)
print(classification_report(y_test,preds.argmax(axis=1)))
accuracy = CNN_9_v1_with_aug.evaluate(X_test, y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.90      0.68      0.78      1000
           1       0.94      0.92      0.93      1000
           2       0.76      0.71      0.73      1000
           3       0.71      0.64      0.67      1000
           4       0.67      0.89      0.76      1000
           5       0.80      0.72      0.76      1000
           6       0.84      0.88      0.86      1000
           7       0.86      0.88      0.87      1000
           8       0.83      0.94      0.88      1000
           9       0.90      0.90      0.90      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.81     10000
weighted avg       0.82      0.82      0.81     10000

313/313 - 1s - loss: 0.5775 - accuracy: 0.8151
Accuracy: 81.51000142097473
Macro F1-score: 0.8137635956327902
```

With data augmentation, results have improved by 0.003%. This is because the model is exposed to a wider variety of different images that allows model to learn better and generalize when faced with unseen images or images in another variation

```
loss = history_aug.history['loss']
val_loss = history_aug.history['val_loss']
acc = history_aug.history['accuracy']
val_acc = history_aug.history['val_accuracy']
epoch = range(len(loss))
```

```
plt.figure(figsize=(20, 4))
plt.suptitle("CNN 9 v1 Base Model Training Log [With
Augmentation]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

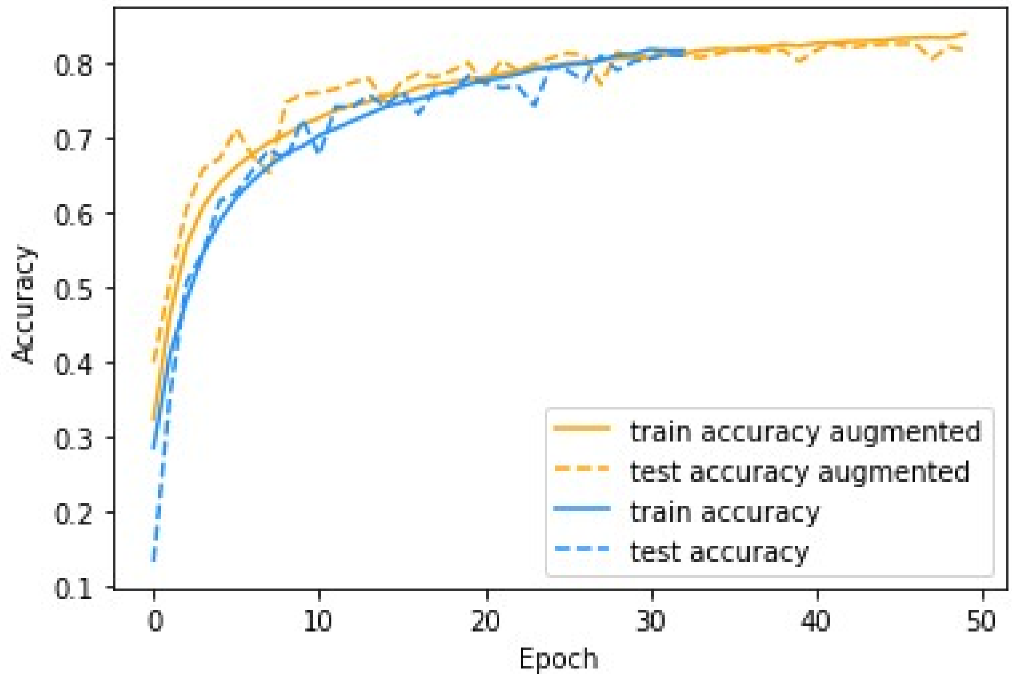


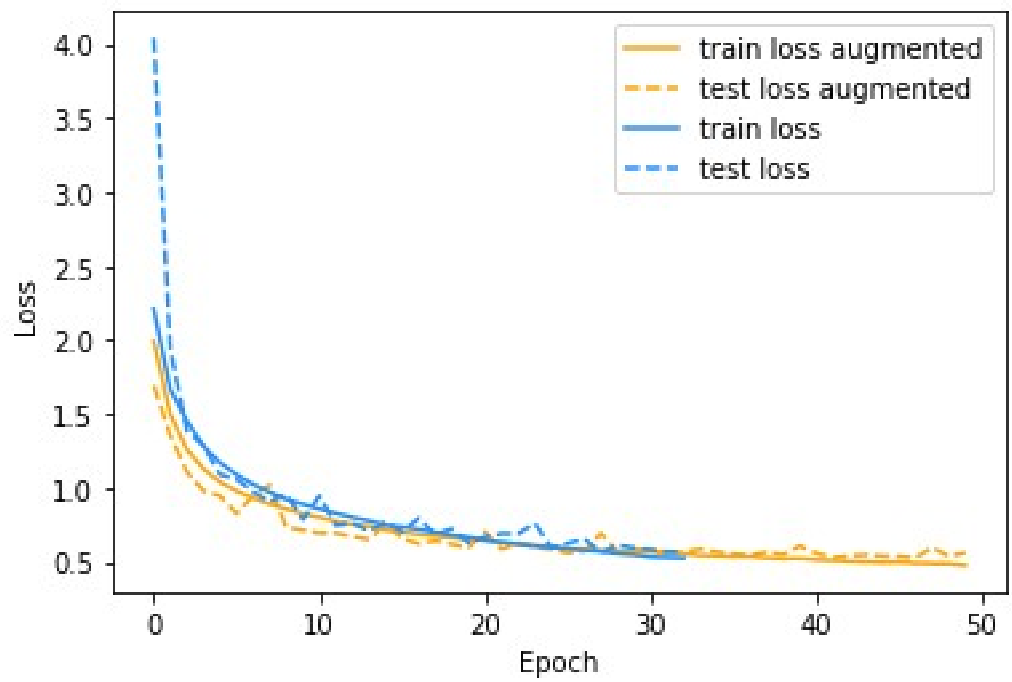
As shown in the model training log, there is no signs of underfitting or overfitting as the train loss augmented and the original train loss is in one single line. The loss is minimal and the the test accuracy of the augmented dataset is around the same value as the non-augmented dataset. Hence, the data augmentation can be keeped as the model performance and the training log is good.

Also, the model seems more stable as there are more data points of the validation loss/acc aligned with the training loss/acc

```
fig = plt.figure()
fig.patch.set_facecolor('white')

plt.plot(history_aug.history['accuracy'],
         label='train accuracy augmented',
         c='orange', ls='-')
plt.plot(history_aug.history['val_accuracy'],
         label='test accuracy augmented',
         c='orange',ls='--')

plt.plot(history_no_aug.history['accuracy'],
         label='train accuracy',
```

```
         c='dodgerblue', ls='-')
plt.plot(history_no_aug.history['val_accuracy'],
         label='test accuracy',
         c='dodgerblue', ls='--')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()


fig = plt.figure()
fig.patch.set_facecolor('white')

plt.plot(history_aug.history['loss'],
         label='train loss augmented',
         c='orange', ls='-')
plt.plot(history_aug.history['val_loss'],
         label='test loss augmented',
         c='orange',ls='--')

plt.plot(history_no_aug.history['loss'],
         label='train loss',
         c='dodgerblue', ls='-')
plt.plot(history_no_aug.history['val_loss'],
         label='test loss',
         c='dodgerblue', ls='--')

plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```





In conclusion, I have managed to achieve a relatively good model where the model has NO SIGNS of OVERFITTING or UNDERFITTING with a decent f1-score 81%.

Conclusion: CNN 9 v1[With Augmentation] F1-Score is 81%

3.5 Model Comparison

Let's compare the summary of all model results:

After much tuning, I have came to the conclusion that CNN 9 version 1 will be the model used for predictions as it has the highest model perfromance of 81% f1-score/accuracy/precision/recall. Even though some versions of the other model has a higher model performance but there was always some imperfections in the model training process. It is a trade off between the model training and the model performance.

Thus, CNN 9 v1 have managed to achieved a good performance of 81% for f1-score and there is not slight overfitting or underfitting throughout the training process. Hence, we will be using this will be the final model that I will settle with.

3.6 Conclusion

For PART A, I have gone through the whole data pipeline from data understanding, preparation,modelling, evaluation, prediction and finally comparison of models. To optimize the performance of the model, I have tried many tuning methodologeis to constantly balance the model performance and training. I hope this report will provide great value to you and assist you to find the best model that will derive the most accurate and precise predictions.

PART B - Cifar-10 Colored Dataset

1. Data Understanding of Cifar-10 Colored Dataset

```
import tarfile
tar = tarfile.open('cifar-10-python.tar.gz', "r:gz")
tar.extractall()
tar.close()
for i in tar:
    print(i)
```

```
<TarInfo 'cifar-10-batches-py' at 0x2102e01a040>
<TarInfo 'cifar-10-batches-py/data_batch_4' at 0x20ff3bdd280>
<TarInfo 'cifar-10-batches-py/readme.html' at 0x20ff3bddd00>
<TarInfo 'cifar-10-batches-py/test_batch' at 0x20ff3bdd040>
<TarInfo 'cifar-10-batches-py/data_batch_3' at 0x20ff3bdd100>
<TarInfo 'cifar-10-batches-py/batches.meta' at 0x20ff3bdd340>
<TarInfo 'cifar-10-batches-py/data_batch_2' at 0x20ff3bdd640>
<TarInfo 'cifar-10-batches-py/data_batch_5' at 0x20ff3bdd400>
<TarInfo 'cifar-10-batches-py/data_batch_1' at 0x20ff3bdd700>
```

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
colored_train_batch1=unpickle('cifar-10-batches-py/data_batch_1')
colored_train_batch2=unpickle('cifar-10-batches-py/data_batch_2')
colored_train_batch3=unpickle('cifar-10-batches-py/data_batch_3')
colored_train_batch4=unpickle('cifar-10-batches-py/data_batch_4')
colored_train_batch5=unpickle('cifar-10-batches-py/data_batch_5')
```

```
colored_test_batch=unpickle('cifar-10-batches-py/test_batch')
```

2. Data Preparation/EDA of Cifar-10 Colored Dataset

```
x_data_temp=[]
y_data_temp=[]
x_test_data_temp=[]
y_test_data_temp=[]

x_data_temp.append(colored_train_batch1[b'data'])
y_data_temp.append(colored_train_batch1[b'labels'])
x_data_temp.append(colored_train_batch2[b'data'])
y_data_temp.append(colored_train_batch2[b'labels'])
x_data_temp.append(colored_train_batch3[b'data'])
y_data_temp.append(colored_train_batch3[b'labels'])
x_data_temp.append(colored_train_batch4[b'data'])
y_data_temp.append(colored_train_batch4[b'labels'])
x_data_temp.append(colored_train_batch5[b'data'])
y_data_temp.append(colored_train_batch5[b'labels'])

x_data=np.array(x_data_temp)
y_data=np.array(y_data_temp)

print(x_data.shape)
print(y_data.shape)

(5, 10000, 3072)
(5, 10000)

x_test_data_temp.append(colored_test_batch[b'data'])
y_test_data_temp.append(colored_test_batch[b'labels'])

x_test_data=np.array(x_test_data_temp)
y_test_data=np.array(y_test_data_temp)

print(x_test_data.shape)
print(y_test_data.shape)

(1, 10000, 3072)
(1, 10000)

x_train_1=x_data.reshape(x_data.shape[0]*x_data.shape[1],x_data.shape[
2])
y_train_1=y_data.reshape(y_data.shape[0]*y_data.shape[1])

x_test_1=x_test_data.reshape(x_test_data.shape[0]*x_test_data.shape[1]
,x_test_data.shape[2])
y_test_1=y_test_data.reshape(y_test_data.shape[0]*y_test_data.shape[1]
)
```

```
print(x_train_1.shape)
print(y_train_1.shape)
print(x_test_1.shape)
print(y_test_1.shape)

(50000, 3072)
(50000,)
(10000, 3072)
(10000,)

colored_X_train=x_train_1.reshape(x_train_1.shape[0],32,32,3)
colored_y_train=y_train_1

colored_X_test=x_test_1.reshape(x_test_1.shape[0],32,32,3)
colored_y_test=y_test_1

print(colored_X_train.shape)
print(colored_y_train.shape)
print(colored_X_test.shape)
print(colored_y_test.shape)

(50000, 32, 32, 3)
(50000,)
(10000, 32, 32, 3)
(10000,)

colored_X_train=colored_X_train.astype("float32")
colored_X_test=colored_X_test.astype("float32")
```

3.Modelling/Evaluation/Prediction of Cifar-10 Colored Dataset

3.1 Using and Tweaking previous model for input dataset on cifar-10 colored dataset

Part A: Building Model

Now i will be using the previous best performing model fo CNN_9_v1 and i will be tweaking the input shape from (32,32,1) to (32,32,3) as we are dealing colored images which have 3 channels this time round

```
model=Sequential()
model.add(Conv2D(32,(3,3),activation="relu",
padding='same',input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Conv2D(32,(3,3),activation="relu", padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
model.add(BatchNormalization())
```

```
model.add(Conv2D(64,(3,3),activation="relu", padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.4))


model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),activation="relu", padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128,activation='tanh'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(10,activation='softmax'))

optimizer = tf.keras.optimizers.Adam(lr=0.0004)
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
model_history=model.fit(colored_X_train, colored_y_train, epochs=50,
batch_size=32, verbose=1,
validation_split=0.2,validation_data=(colored_X_test, colored_y_test))
```

```
Epoch 1/50
1250/1250 [==============================] - 9s 7ms/step - loss:
2.1070 - accuracy: 0.2809 - val_loss: 1.6908 - val_accuracy: 0.3788
Epoch 2/50
1250/1250 [==============================] - 8s 7ms/step - loss:
1.6703 - accuracy: 0.3908 - val_loss: 1.5200 - val_accuracy: 0.4483
Epoch 3/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.5265 - accuracy: 0.4512 - val_loss: 1.5438 - val_accuracy: 0.4499
Epoch 4/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.4402 - accuracy: 0.4827 - val_loss: 1.4436 - val_accuracy: 0.4948
Epoch 5/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.3657 - accuracy: 0.5131 - val_loss: 1.4427 - val_accuracy: 0.4915
Epoch 6/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.3131 - accuracy: 0.5346 - val_loss: 1.2824 - val_accuracy: 0.5381
Epoch 7/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.2645 - accuracy: 0.5530 - val_loss: 1.3672 - val_accuracy: 0.5138
Epoch 8/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.2180 - accuracy: 0.5717 - val_loss: 1.1098 - val_accuracy: 0.6112
```

```
Epoch 9/50
1250/1250 [==============================] - 10s 8ms/step - loss:
1.1842 - accuracy: 0.5859 - val_loss: 1.4486 - val_accuracy: 0.5165
Epoch 10/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1478 - accuracy: 0.5975 - val_loss: 1.1143 - val_accuracy: 0.6082
Epoch 11/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1241 - accuracy: 0.6056 - val_loss: 1.0742 - val_accuracy: 0.6242
Epoch 12/50
1250/1250 [==============================] - 9s 7ms/step - loss:
1.0990 - accuracy: 0.6174 - val_loss: 1.0691 - val_accuracy: 0.6248
Epoch 13/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0783 - accuracy: 0.6267 - val_loss: 1.0752 - val_accuracy: 0.6263
Epoch 14/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0544 - accuracy: 0.6350 - val_loss: 1.0895 - val_accuracy: 0.6157
Epoch 15/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0399 - accuracy: 0.6378 - val_loss: 1.0694 - val_accuracy: 0.6230
Epoch 16/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0188 - accuracy: 0.6482 - val_loss: 1.0007 - val_accuracy: 0.6508
Epoch 17/50
1250/1250 [==============================] - 8s 6ms/step - loss:
1.0065 - accuracy: 0.6503 - val_loss: 0.9783 - val_accuracy: 0.6628
Epoch 18/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9887 - accuracy: 0.6566 - val_loss: 1.0199 - val_accuracy: 0.6499
Epoch 19/50
1250/1250 [==============================] - 9s 8ms/step - loss:
0.9740 - accuracy: 0.6621 - val_loss: 0.9986 - val_accuracy: 0.6494
Epoch 20/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.9621 - accuracy: 0.6677 - val_loss: 0.9515 - val_accuracy: 0.6662
Epoch 21/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.9489 - accuracy: 0.6719 - val_loss: 0.9686 - val_accuracy: 0.6672
Epoch 22/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.9368 - accuracy: 0.6769 - val_loss: 0.9503 - val_accuracy: 0.6731
Epoch 23/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.9312 - accuracy: 0.6772 - val_loss: 1.0034 - val_accuracy: 0.6545
Epoch 24/50
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9110 - accuracy: 0.6864 - val_loss: 1.0066 - val_accuracy: 0.6592
Epoch 25/50
```

```
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9028 - accuracy: 0.6869 - val_loss: 0.9964 - val_accuracy: 0.6573
Epoch 26/50
1250/1250 [==============================] - 8s 7ms/step - loss:
0.8981 - accuracy: 0.6883 - val_loss: 0.9340 - val_accuracy: 0.6760
Epoch 27/50
1250/1250 [==============================] - 8s 7ms/step - loss:
0.8887 - accuracy: 0.6917 - val_loss: 0.8890 - val_accuracy: 0.6948
Epoch 28/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8731 - accuracy: 0.6991 - val_loss: 0.9200 - val_accuracy: 0.6850
Epoch 29/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8744 - accuracy: 0.6964 - val_loss: 0.8912 - val_accuracy: 0.6881
Epoch 30/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8657 - accuracy: 0.7009 - val_loss: 1.0274 - val_accuracy: 0.6529
Epoch 31/50
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8580 - accuracy: 0.7042 - val_loss: 0.8885 - val_accuracy: 0.6937
Epoch 32/50
1250/1250 [==============================] - 8s 7ms/step - loss:
0.8517 - accuracy: 0.7072 - val_loss: 0.8799 - val_accuracy: 0.6956
Epoch 33/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8480 - accuracy: 0.7079 - val_loss: 0.9136 - val_accuracy: 0.6869
Epoch 34/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8370 - accuracy: 0.7116 - val_loss: 0.8573 - val_accuracy: 0.7070
Epoch 35/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8290 - accuracy: 0.7130 - val_loss: 0.8455 - val_accuracy: 0.7066
Epoch 36/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8230 - accuracy: 0.7164 - val_loss: 1.0472 - val_accuracy: 0.6466
Epoch 37/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8159 - accuracy: 0.7200 - val_loss: 0.9341 - val_accuracy: 0.6870
Epoch 38/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8100 - accuracy: 0.7232 - val_loss: 0.8638 - val_accuracy: 0.7040
Epoch 39/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8104 - accuracy: 0.7220 - val_loss: 0.8911 - val_accuracy: 0.6964
Epoch 40/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.7950 - accuracy: 0.7275 - val_loss: 0.8275 - val_accuracy: 0.7171
Epoch 41/50
1250/1250 [==============================] - 9s 7ms/step - loss:
```

```
0.7899 - accuracy: 0.7285 - val_loss: 0.8767 - val_accuracy: 0.7027
Epoch 42/50
1250/1250 [==============================] - 11s 9ms/step - loss:
0.7946 - accuracy: 0.7285 - val_loss: 0.8233 - val_accuracy: 0.7183
Epoch 43/50
1250/1250 [==============================] - 13s 10ms/step - loss:
0.7861 - accuracy: 0.7306 - val_loss: 0.8148 - val_accuracy: 0.7180
Epoch 44/50
1250/1250 [==============================] - 12s 9ms/step - loss:
0.7811 - accuracy: 0.7326 - val_loss: 0.8052 - val_accuracy: 0.7284
Epoch 45/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.7764 - accuracy: 0.7319 - val_loss: 0.8556 - val_accuracy: 0.7095
Epoch 46/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7663 - accuracy: 0.7359 - val_loss: 0.9030 - val_accuracy: 0.6972
Epoch 47/50
1250/1250 [==============================] - 9s 8ms/step - loss:
0.7665 - accuracy: 0.7351 - val_loss: 0.8204 - val_accuracy: 0.7203
Epoch 48/50
1250/1250 [==============================] - 10s 8ms/step - loss:
0.7654 - accuracy: 0.7388 - val_loss: 0.8798 - val_accuracy: 0.7072
Epoch 49/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7614 - accuracy: 0.7386 - val_loss: 0.8510 - val_accuracy: 0.7146
Epoch 50/50
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7509 - accuracy: 0.7395 - val_loss: 0.8026 - val_accuracy: 0.7264
```

Part B: Model Evaluation

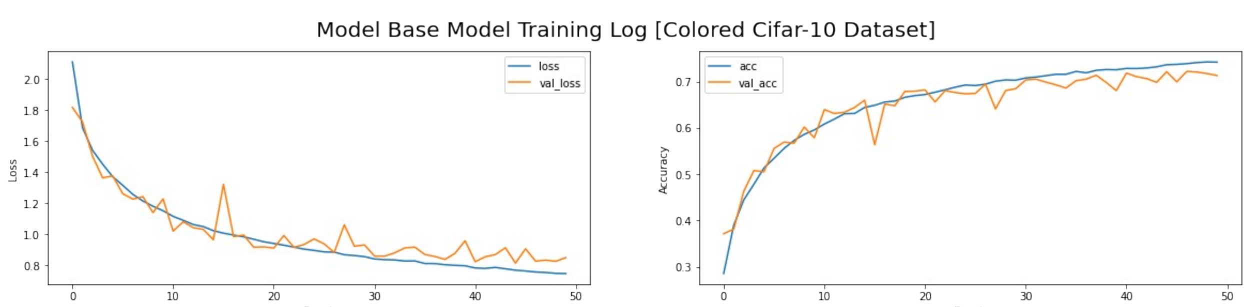
```
preds = model.predict(colored_X_test)
print(classification_report(colored_y_test,preds.argmax(axis=1)))
accuracy = model.evaluate(colored_X_test, colored_y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(colored_y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.74      0.77      0.76      1000
           1       0.78      0.87      0.82      1000
           2       0.58      0.63      0.61      1000
           3       0.53      0.45      0.48      1000
           4       0.72      0.59      0.65      1000
           5       0.58      0.70      0.63      1000
           6       0.68      0.85      0.75      1000
           7       0.78      0.74      0.76      1000
           8       0.89      0.74      0.81      1000
```

```
            9       0.87      0.75      0.80      1000

    accuracy                          0.71     10000
   macro avg       0.71      0.71      0.71     10000
weighted avg       0.71      0.71      0.71     10000

313/313 - 1s - loss: 0.8741 - accuracy: 0.7078
Accuracy: 70.77999711036682
Macro F1-score: 0.706955065217025
```

From the model performance, when we apply the previous model on the cifar-10 colored dataset, the model perfromance dropped 40% around around 80% to 70% which is 10% drop.

```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
acc = model_history.history['accuracy']
val_acc = model_history.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("Model Base Model Training Log [Colored Cifar-10
Dataset]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Although there are no overfitting or underfitting signs, however, there are multiple fluctuations in the model training process

Conclusion: Colored Model F1-Score is 71%

Part C: Analyzing and Explaining the performance

There may be 2 reasons for the drop in the model performance Firstly, there are more noise in colored dataset as given that there is colored images, there are more features, edges and higher image complexity for the model to train during the process. Hence there is more noise while model is training and may affect Secondly, model is fine tuned to the gray scale dataset hence, most of the parameterss in the CNN model is tailored for classifying the image qualities and features in the gray scale. Hence when the model is applied on the colored images, it will affect the model performance slightly as the model may not be trained as well to recognize the colored images.

Part D: Tuning with Data Augmentation

Now, I will be trying out with data augmentation to increase the model performance

```
width_shift = 1.0
height_shift = 1.0
flip = True

datagen = ImageDataGenerator(
    horizontal_flip=flip,
    width_shift_range=width_shift,
    height_shift_range=height_shift,
    )
datagen.fit(colored_X_train)
it = datagen.flow(colored_X_train, colored_y_train, shuffle=True)
batch_images, batch_labels = next(it)

optimizer = tf.keras.optimizers.Adam(lr=0.0004)
datagen.fit(colored_X_train)
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])
colored_history_aug = model.fit(colored_X_train, colored_y_train,
epochs=60, batch_size=32, verbose=1,
validation_split=0.2,validation_data=(colored_X_test,colored_y_test))
```

```
Epoch 1/60
1250/1250 [==============================] - 10s 7ms/step - loss:
2.0763 - accuracy: 0.2904 - val_loss: 1.7061 - val_accuracy: 0.3976
Epoch 2/60
1250/1250 [==============================] - 8s 7ms/step - loss:
1.6631 - accuracy: 0.3984 - val_loss: 1.6195 - val_accuracy: 0.4296
Epoch 3/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.5132 - accuracy: 0.4575 - val_loss: 1.5322 - val_accuracy: 0.4578
Epoch 4/60
1250/1250 [==============================] - 8s 6ms/step - loss:
```

```
1.4272 - accuracy: 0.4897 - val_loss: 1.3220 - val_accuracy: 0.5236
Epoch 5/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.3584 - accuracy: 0.5145 - val_loss: 1.2898 - val_accuracy: 0.5380
Epoch 6/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.3007 - accuracy: 0.5405 - val_loss: 1.3517 - val_accuracy: 0.5268
Epoch 7/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.2524 - accuracy: 0.5577 - val_loss: 1.7758 - val_accuracy: 0.4149
Epoch 8/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.2092 - accuracy: 0.5757 - val_loss: 1.1889 - val_accuracy: 0.5817
Epoch 9/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1712 - accuracy: 0.5884 - val_loss: 1.1206 - val_accuracy: 0.6079
Epoch 10/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1382 - accuracy: 0.6033 - val_loss: 1.1021 - val_accuracy: 0.6102
Epoch 11/60
1250/1250 [==============================] - 8s 6ms/step - loss:
1.1085 - accuracy: 0.6122 - val_loss: 1.0663 - val_accuracy: 0.6244
Epoch 12/60
1250/1250 [==============================] - 10s 8ms/step - loss:
1.0864 - accuracy: 0.6208 - val_loss: 1.0785 - val_accuracy: 0.6186
Epoch 13/60
1250/1250 [==============================] - 9s 7ms/step - loss:
1.0700 - accuracy: 0.6281 - val_loss: 1.0094 - val_accuracy: 0.6464
Epoch 14/60
1250/1250 [==============================] - 10s 8ms/step - loss:
1.0479 - accuracy: 0.6344 - val_loss: 0.9900 - val_accuracy: 0.6532
Epoch 15/60
1250/1250 [==============================] - 10s 8ms/step - loss:
1.0307 - accuracy: 0.6423 - val_loss: 1.0686 - val_accuracy: 0.6318
Epoch 16/60
1250/1250 [==============================] - 11s 9ms/step - loss:
1.0081 - accuracy: 0.6507 - val_loss: 0.9666 - val_accuracy: 0.6627
Epoch 17/60
1250/1250 [==============================] - 11s 9ms/step - loss:
0.9937 - accuracy: 0.6547 - val_loss: 0.9520 - val_accuracy: 0.6659
Epoch 18/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.9826 - accuracy: 0.6600 - val_loss: 0.9407 - val_accuracy: 0.6729
Epoch 19/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9720 - accuracy: 0.6636 - val_loss: 0.9919 - val_accuracy: 0.6575
Epoch 20/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9563 - accuracy: 0.6704 - val_loss: 0.9403 - val_accuracy: 0.6708
```

```
Epoch 21/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9408 - accuracy: 0.6753 - val_loss: 1.1053 - val_accuracy: 0.6245
Epoch 22/60
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9332 - accuracy: 0.6780 - val_loss: 1.0886 - val_accuracy: 0.6304
Epoch 23/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9229 - accuracy: 0.6811 - val_loss: 0.8898 - val_accuracy: 0.6935
Epoch 24/60
1250/1250 [==============================] - 8s 6ms/step - loss:
0.9131 - accuracy: 0.6847 - val_loss: 0.9691 - val_accuracy: 0.6699
Epoch 25/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.9064 - accuracy: 0.6853 - val_loss: 0.9839 - val_accuracy: 0.6613
Epoch 26/60
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8921 - accuracy: 0.6924 - val_loss: 0.8859 - val_accuracy: 0.6937
Epoch 27/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8825 - accuracy: 0.6966 - val_loss: 0.9624 - val_accuracy: 0.6762
Epoch 28/60
1250/1250 [==============================] - 12s 9ms/step - loss:
0.8730 - accuracy: 0.6975 - val_loss: 0.8930 - val_accuracy: 0.6909
Epoch 29/60
1250/1250 [==============================] - 13s 11ms/step - loss:
0.8733 - accuracy: 0.6982 - val_loss: 0.9038 - val_accuracy: 0.6889
Epoch 30/60
1250/1250 [==============================] - 11s 8ms/step - loss:
0.8615 - accuracy: 0.7030 - val_loss: 0.8854 - val_accuracy: 0.6960
Epoch 31/60
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8556 - accuracy: 0.7061 - val_loss: 1.4291 - val_accuracy: 0.5568
Epoch 32/60
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8450 - accuracy: 0.7097 - val_loss: 0.8630 - val_accuracy: 0.7061
Epoch 33/60
1250/1250 [==============================] - 11s 9ms/step - loss:
0.8391 - accuracy: 0.7125 - val_loss: 0.9040 - val_accuracy: 0.6942
Epoch 34/60
1250/1250 [==============================] - 11s 9ms/step - loss:
0.8400 - accuracy: 0.7132 - val_loss: 0.9671 - val_accuracy: 0.6709
Epoch 35/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.8255 - accuracy: 0.7154 - val_loss: 0.8539 - val_accuracy: 0.7100
Epoch 36/60
1250/1250 [==============================] - 10s 8ms/step - loss:
0.8293 - accuracy: 0.7137 - val_loss: 0.9368 - val_accuracy: 0.6869
Epoch 37/60
```

```
1250/1250 [==============================] - 11s 9ms/step - loss:
0.8124 - accuracy: 0.7201 - val_loss: 0.8616 - val_accuracy: 0.7051
Epoch 38/60
1250/1250 [==============================] - 8s 6ms/step - loss:
0.8068 - accuracy: 0.7229 - val_loss: 0.8645 - val_accuracy: 0.7061
Epoch 39/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7946 - accuracy: 0.7266 - val_loss: 0.8652 - val_accuracy: 0.7074
Epoch 40/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.8032 - accuracy: 0.7251 - val_loss: 0.8767 - val_accuracy: 0.7002
Epoch 41/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7960 - accuracy: 0.7275 - val_loss: 1.4759 - val_accuracy: 0.5497
Epoch 42/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7950 - accuracy: 0.7256 - val_loss: 0.8315 - val_accuracy: 0.7179
Epoch 43/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7840 - accuracy: 0.7291 - val_loss: 0.8115 - val_accuracy: 0.7243
Epoch 44/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7803 - accuracy: 0.7330 - val_loss: 0.8335 - val_accuracy: 0.7209
Epoch 45/60
1250/1250 [==============================] - 8s 6ms/step - loss:
0.7729 - accuracy: 0.7329 - val_loss: 0.9303 - val_accuracy: 0.6933
Epoch 46/60
1250/1250 [==============================] - 10s 8ms/step - loss:
0.7688 - accuracy: 0.7361 - val_loss: 0.8479 - val_accuracy: 0.7128
Epoch 47/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7685 - accuracy: 0.7361 - val_loss: 0.8335 - val_accuracy: 0.7164
Epoch 48/60
1250/1250 [==============================] - 9s 8ms/step - loss:
0.7614 - accuracy: 0.7395 - val_loss: 0.8285 - val_accuracy: 0.7226
Epoch 49/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7562 - accuracy: 0.7396 - val_loss: 0.8037 - val_accuracy: 0.7282
Epoch 50/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7549 - accuracy: 0.7405 - val_loss: 0.8550 - val_accuracy: 0.7188
Epoch 51/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7515 - accuracy: 0.7417 - val_loss: 0.8221 - val_accuracy: 0.7267
Epoch 52/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7456 - accuracy: 0.7452 - val_loss: 0.8651 - val_accuracy: 0.7130
Epoch 53/60
1250/1250 [==============================] - 8s 7ms/step - loss:
```

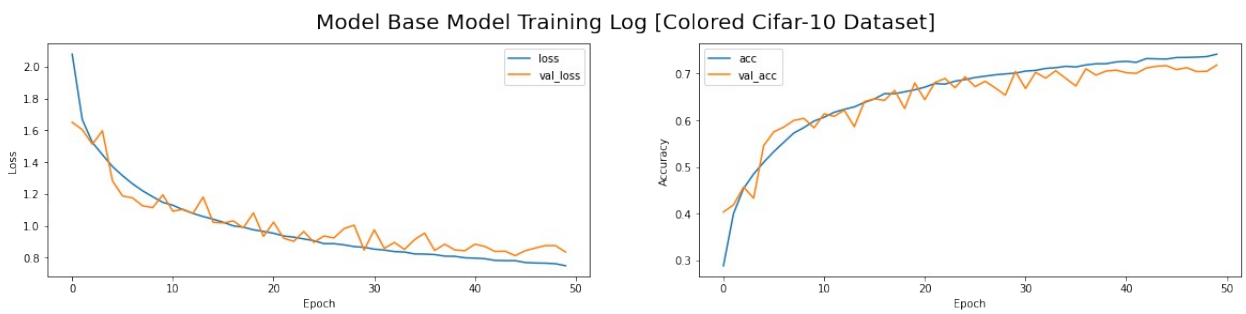
```
0.7394 - accuracy: 0.7457 - val_loss: 0.8028 - val_accuracy: 0.7284
Epoch 54/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7366 - accuracy: 0.7465 - val_loss: 0.8196 - val_accuracy: 0.7283
Epoch 55/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7325 - accuracy: 0.7517 - val_loss: 0.7945 - val_accuracy: 0.7326
Epoch 56/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7332 - accuracy: 0.7494 - val_loss: 0.8149 - val_accuracy: 0.7274
Epoch 57/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7271 - accuracy: 0.7502 - val_loss: 0.8295 - val_accuracy: 0.7234
Epoch 58/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7247 - accuracy: 0.7512 - val_loss: 0.7960 - val_accuracy: 0.7374
Epoch 59/60
1250/1250 [==============================] - 8s 7ms/step - loss:
0.7222 - accuracy: 0.7509 - val_loss: 0.8189 - val_accuracy: 0.7258
Epoch 60/60
1250/1250 [==============================] - 9s 7ms/step - loss:
0.7199 - accuracy: 0.7533 - val_loss: 0.7983 - val_accuracy: 0.7297
```

```python
###### preds = model.predict(colored_X_test)
print(classification_report(colored_y_test,preds.argmax(axis=1)))
accuracy = model.evaluate(colored_X_test, colored_y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
print('Macro F1-
score:',f1_score(colored_y_test,preds.argmax(axis=1),average="macro"))
```

```
              precision    recall  f1-score   support

           0       0.79      0.75      0.77      1000
           1       0.87      0.82      0.84      1000
           2       0.71      0.51      0.59      1000
           3       0.51      0.49      0.50      1000
           4       0.62      0.72      0.67      1000
           5       0.59      0.65      0.62      1000
           6       0.70      0.82      0.75      1000
           7       0.77      0.78      0.78      1000
           8       0.85      0.83      0.84      1000
           9       0.80      0.81      0.80      1000

    accuracy                           0.72     10000
   macro avg       0.72      0.72      0.72     10000
weighted avg       0.72      0.72      0.72     10000

313/313 - 1s - loss: 0.8283 - accuracy: 0.7210
Accuracy: 72.10000157356262
Macro F1-score: 0.7166387065975547
```

After data augmentation the f1-score has increased by 0.01 from 71% to 72%.

```
loss = colored_history_aug.history['loss']
val_loss = colored_history_aug.history['val_loss']
acc = colored_history_aug.history['accuracy']
val_acc = colored_history_aug.history['val_accuracy']
epoch = range(len(loss))
plt.figure(figsize=(20, 4))
plt.suptitle("Model Base Model Training Log [Colored Cifar-10
Dataset]",fontsize=20)
plt.subplot(1, 2, 1)
plt.plot(epoch,loss,label='loss')
plt.plot(epoch,val_loss,label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epoch,acc,label='acc')
plt.plot(epoch,val_acc,label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



As for the model training log, there is slight difference from the non-augmented training log where towards the end where there is less fluctuations/deviations from validation and training towards the ends of the model training/

Conclusion: Colored Model with Data Augmentation F1-Score is 73%