# Computer Vision & Deep Learning Picture Classification Student Project

Divya Sasidharan
*M.Sc. Digital Engineering*
*Otto von Guericke University*
Magdeburg, Germany
divya.sasidharan@st.ovgu.de

Sreekar Kumar Indarapu
*M.Sc. Digital Engineering*
*Otto von Guericke University*
Magdeburg, Germany
sreekar.indarapu@st.ovgu.de

***Abstract*—In this project we are implementing deep convolutional models to classify Imagenette and Imagewoof datasets. Both the datasets are a subset of 10 easily classified classes from Imagenet. Imagenette is less complicated compared to Imagewoof**

***Index Terms*—Computer Vision, Deep learning, Image classification, Imagenette, Imagewoof**

## I. Motivation

The work in this project is concerned with the development of simple deep learning (CNN) model for classification of the images in the datasets, imagenette and imagewoof. Image classification is a supervised machine learning technique to learn and predict the category of a given image. To meet this objective, the project was delivered in three phases 1) Data analysis 2) Implementation of classification techniques for imagenette dataset 3) Adaptation of the model developed in 2nd step to fit more complex dataset Imagewoof. Imagenette and Imagewoof dataset is available on *https://github.com/fastai/imagenette*.

## II. Data Load and Analysis

Imagenette is a subset of 10 easily classified classes from Imagenet (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute). Imagenette have a total number of 13,394 examples . The images are already divided into "train" and "validation" and contain 9469 and 3925 images respectively. We also recognize that the images contained in the dataset have different sizes. We're using full-size, full-color images, which are photos of objects of different sizes, in different orientations, in different lighting, and so forth
Imagewoof is a subset of 10 dog breed classes from Imagenet. The breeds are Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever and Old English sheepdog. Similar to Imagenette the size of Imagewoof dataset varies . The total number of examples present is 12954. The images here are also divided into "train" and "validation" and contain 9025 and 3929 images respectively. The main challenge of these datasets is that they are more diverse and less data samples available for each classes. There were around 13k-14k of images in total and these images have too much background noise. Some images contains humans in background eg.like a person holding a fish or dog. So in some cases it is very difficult to identify the class for the image even from a human perspective. Some examples from each dataset are below.


Figure 1: Imagenette example images

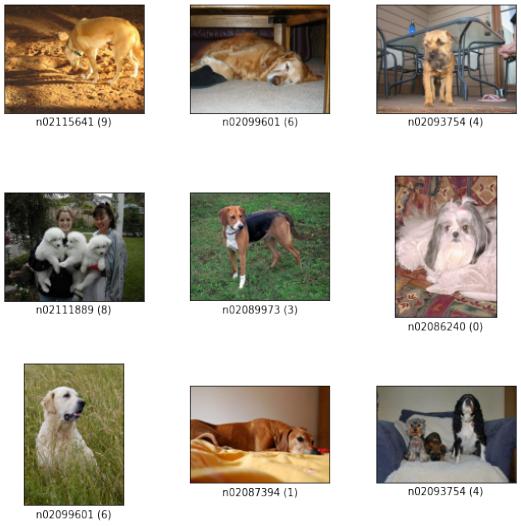
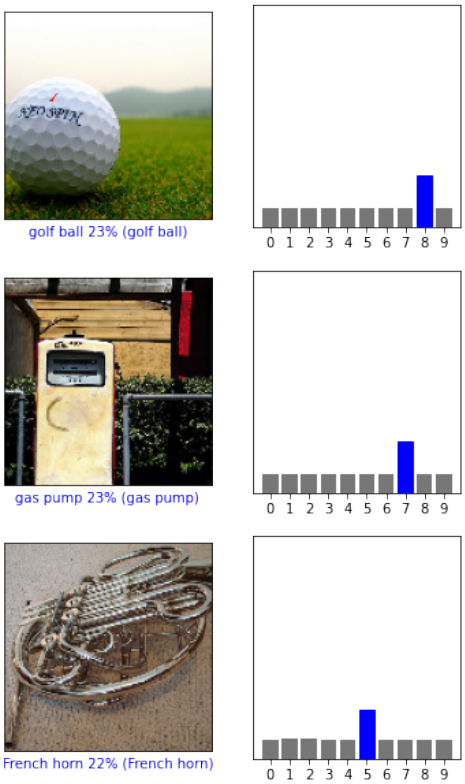



Figure 2: Imagewoof example images

Many useful features are present in different images from which the model could learn. Suppose for golf ball it could be the texture or the round shape of the object, for gas pump it could be the rectangular shape, for French horn, it could be the curved structure as shown in the below images that our model learns as features for a particular image which helps in classification. In Imagewoof the main difference between dog breed could be ear shape, color, hair length, etc.




## A. *Preprocessing*

The first step for classification is to preprocess the dataset with help of Keras image preprocessing API. We need to resize the image to a fixed height and weight. We choose the image size of 128 and resized each image input to a target height and width of (128,128,3) for the Imagenette dataset. In the case of Imagewoof, we choose the image size to be 224 as the images are complex compared to imagenette dataset.

The next basic preprocessing step is to normalize the pixel values of each image so that they are all in the range between 0 and 1. Normally the value of a pixel is between 0 and 255 (RGB color values), but the neural networks prefer to calculate with values between 0 and 1, so we will simply divide each pixel value by 255.

Since the dataset was small we wanted to avoid overfitting that might occur during training of the model. For this, we implemented a few data augmentation techniques. Data augmentation techniques helps in increasing the variation in a dataset by applying transformations to the original data. It is often used when the training data is limited and as a way of preventing overfitting.The below figure 3 shows the preprocessing steps and figure 4 , 5 shows the images after preprocessing of Imagenette and Imagewoof datasets.

```
[ ]  1 size_image = 128
     2 resize_and_rescale = tf.keras.Sequential([
     3   tf.keras.layers.experimental.preprocessing.Resizing(size_image, size_image,crop_to_aspect_ratio=True),
     4   tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
     5 ])
```

Figure 3: Preprocessing steps using keras API

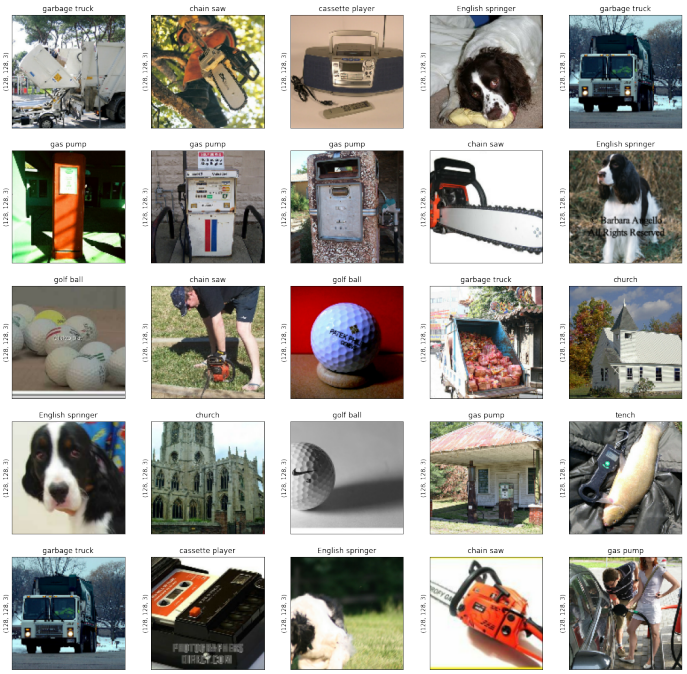


Figure 4: Preprocessed Imagenette

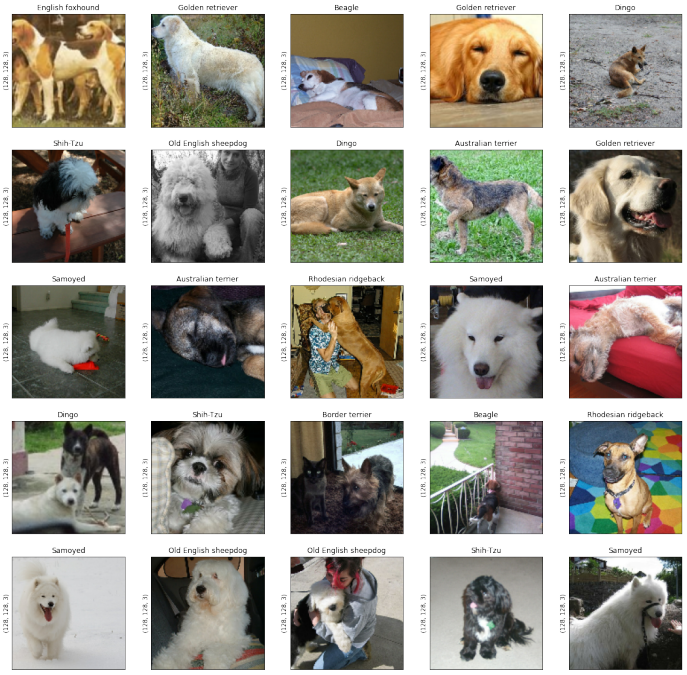


Figure 5: Preprocessed Imagewoof

## III. IMAGENETTE CLASSIFICATION

The implementation was carried out using Python 3,TensorFlow, and Keras framework. We used the Google colab for the development of the models. We have defined a simple CNN model with a kernel size of 3 and a relu activation function. The padding was set to 'same' to keep the output feature map size the same as the input. Max pooling after convolution downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window. The dropout layer randomly sets input units to 0 with a frequency of rate (0.5) at each step during training time, which helps prevent overfitting. We have a batch normalization layer that allows every layer of the network to do learning more independently. Using batch normalization learning becomes efficient and it avoids overfitting of the model. The learning rate used was 0.0001 for best results using the RMSprop optimizer. The final dense layer has a softmax activation function as this is a multi-class classification and the sum of all the probabilities will be 1.

Model

```
# patient early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32,(3,3),padding='same', activation=tf.keras.layers.ReLU(),input_shape=(128,128,3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(64,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(64,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(64,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64,activation=tf.keras.layers.ReLU()),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10,activation='softmax')])
```

Figure 6: Imagenette simple CNN model

### A. *Training Details and Experiments*

*1) With batch normalization:* The model was trained for 100 epochs with early stopping set on validation loss value with the patience of 10 epochs. The training continued for 30 epochs before the 'early stopping' was applied. At the end of 30 epochs, the validation accuracy was 73% and the validation loss was 0.8747. The training accuracy after early stopping was only 92%, while the training loss was as low as 0.2781.

```
Epoch 20/100
74/74 [==============================] - 17s 223ms/step - loss: 0.4658 - accuracy: 0.8611 - val_loss: 0.8616 - val_accuracy: 0.7310
Epoch 21/100
74/74 [==============================] - 17s 222ms/step - loss: 0.4510 - accuracy: 0.8622 - val_loss: 0.8878 - val_accuracy: 0.7205
Epoch 22/100
74/74 [==============================] - 17s 222ms/step - loss: 0.4223 - accuracy: 0.8707 - val_loss: 0.8802 - val_accuracy: 0.7233
Epoch 23/100
74/74 [==============================] - 16s 222ms/step - loss: 0.4088 - accuracy: 0.8775 - val_loss: 0.8675 - val_accuracy: 0.7292
Epoch 24/100
74/74 [==============================] - 17s 223ms/step - loss: 0.3743 - accuracy: 0.8900 - val_loss: 0.8652 - val_accuracy: 0.7327
Epoch 25/100
74/74 [==============================] - 16s 221ms/step - loss: 0.3600 - accuracy: 0.8948 - val_loss: 0.8935 - val_accuracy: 0.7164
Epoch 26/100
74/74 [==============================] - 16s 221ms/step - loss: 0.3455 - accuracy: 0.9014 - val_loss: 0.8800 - val_accuracy: 0.7225
Epoch 27/100
74/74 [==============================] - 16s 222ms/step - loss: 0.3264 - accuracy: 0.9054 - val_loss: 0.8850 - val_accuracy: 0.7254
Epoch 28/100
74/74 [==============================] - 17s 223ms/step - loss: 0.3095 - accuracy: 0.9122 - val_loss: 0.8832 - val_accuracy: 0.7254
Epoch 29/100
74/74 [==============================] - 17s 224ms/step - loss: 0.3013 - accuracy: 0.9115 - val_loss: 0.8705 - val_accuracy: 0.7322
Epoch 30/100
74/74 [==============================] - 16s 220ms/step - loss: 0.2781 - accuracy: 0.9225 - val_loss: 0.8747 - val_accuracy: 0.7320
Epoch 00030: early stopping
```

Figure 7: Training epochs

The below figures show the accuracy and loss plots for the training. From the plot, we can see that there is an overfitting issue for the model after the 8 epochs.

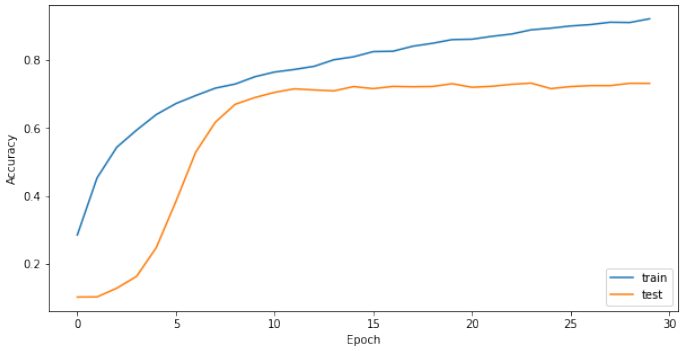



Figure 8: Accuracy

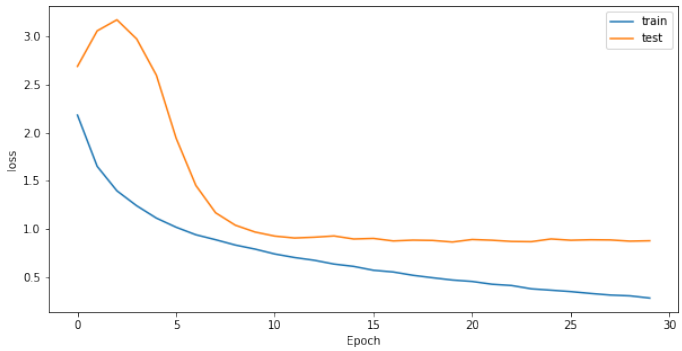



Figure 9: Loss

*2) Without batch normalization:* We trained the network without batch normalization and found that the results were the same as with batch normalization, but with more epochs(86 epochs) until the training ended due to callback early stopping.

```
74/74 [==============================] - 17s 224ms/step - loss: 0.5952 - accuracy: 0.8062 - val_loss: 0.8990 - val_accuracy: 0.7269
Epoch 76/100
74/74 [==============================] - 17s 226ms/step - loss: 0.5904 - accuracy: 0.8090 - val_loss: 0.8765 - val_accuracy: 0.7304
Epoch 77/100
74/74 [==============================] - 17s 223ms/step - loss: 0.5838 - accuracy: 0.8079 - val_loss: 0.9004 - val_accuracy: 0.7215
Epoch 78/100
74/74 [==============================] - 17s 226ms/step - loss: 0.5827 - accuracy: 0.8092 - val_loss: 0.9076 - val_accuracy: 0.7185
Epoch 79/100
74/74 [==============================] - 17s 227ms/step - loss: 0.5698 - accuracy: 0.8174 - val_loss: 0.9091 - val_accuracy: 0.7208
Epoch 80/100
74/74 [==============================] - 17s 228ms/step - loss: 0.5610 - accuracy: 0.8152 - val_loss: 0.9191 - val_accuracy: 0.7146
Epoch 81/100
74/74 [==============================] - 17s 227ms/step - loss: 0.5535 - accuracy: 0.8178 - val_loss: 0.9087 - val_accuracy: 0.7213
Epoch 82/100
74/74 [==============================] - 17s 225ms/step - loss: 0.5513 - accuracy: 0.8166 - val_loss: 0.9556 - val_accuracy: 0.7116
Epoch 83/100
74/74 [==============================] - 17s 224ms/step - loss: 0.5473 - accuracy: 0.8196 - val_loss: 0.9107 - val_accuracy: 0.7228
Epoch 84/100
74/74 [==============================] - 17s 224ms/step - loss: 0.5482 - accuracy: 0.8253 - val_loss: 0.8964 - val_accuracy: 0.7243
Epoch 85/100
74/74 [==============================] - 17s 227ms/step - loss: 0.5477 - accuracy: 0.8212 - val_loss: 0.9238 - val_accuracy: 0.7169
Epoch 86/100
74/74 [==============================] - 16s 222ms/step - loss: 0.5227 - accuracy: 0.8312 - val_loss: 0.8822 - val_accuracy: 0.7320
Epoch 00086: early stopping
```

Figure 10: Training epochs

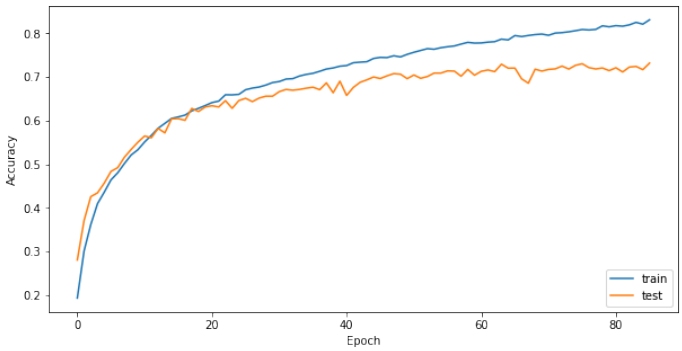



Figure 11: Accuracy

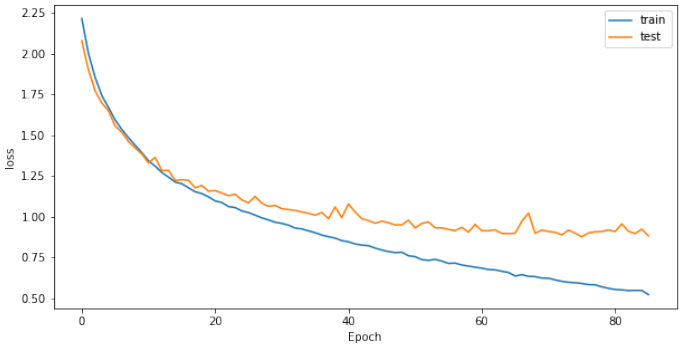



Figure 12: Loss

*3) With batch normalization and data augmentation:* As the next step of our study, we used data augmentation techniques like 'RandomFlip' and 'RandomRotation' to overcome the issue of overfitting. But the result was not satisfactory. The overfitting was slightly reduced and the validation accuracy dropped to 68% during the training time.

```
Epoch 327/2000
74/74 [==============================] - 29s 387ms/step - loss: 0.3934 - accuracy: 0.8658 - val_loss: 1.1328 - val_accuracy: 0.6887
Epoch 328/2000
74/74 [==============================] - 29s 388ms/step - loss: 0.3826 - accuracy: 0.8693 - val_loss: 1.0630 - val_accuracy: 0.7001
Epoch 329/2000
74/74 [==============================] - 29s 386ms/step - loss: 0.3905 - accuracy: 0.8678 - val_loss: 1.0958 - val_accuracy: 0.6920
Epoch 330/2000
74/74 [==============================] - 29s 390ms/step - loss: 0.3918 - accuracy: 0.8660 - val_loss: 1.1030 - val_accuracy: 0.6851
Epoch 331/2000
74/74 [==============================] - 29s 391ms/step - loss: 0.3892 - accuracy: 0.8689 - val_loss: 1.1338 - val_accuracy: 0.6792
Epoch 332/2000
74/74 [==============================] - 29s 387ms/step - loss: 0.3837 - accuracy: 0.8682 - val_loss: 1.3129 - val_accuracy: 0.6461
Epoch 333/2000
74/74 [==============================] - 29s 388ms/step - loss: 0.3887 - accuracy: 0.8684 - val_loss: 1.2768 - val_accuracy: 0.6550
Epoch 334/2000
74/74 [==============================] - 29s 390ms/step - loss: 0.3850 - accuracy: 0.8714 - val_loss: 1.1259 - val_accuracy: 0.6887
Epoch 335/2000
74/74 [==============================] - 29s 392ms/step - loss: 0.3939 - accuracy: 0.8656 - val_loss: 1.1439 - val_accuracy: 0.6808
Epoch 336/2000
74/74 [==============================] - 29s 393ms/step - loss: 0.3824 - accuracy: 0.8724 - val_loss: 1.1212 - val_accuracy: 0.6828
Epoch 337/2000
74/74 [==============================] - 28s 385ms/step - loss: 0.3755 - accuracy: 0.8686 - val_loss: 1.1137 - val_accuracy: 0.6902
Epoch 338/2000
74/74 [==============================] - 29s 389ms/step - loss: 0.3805 - accuracy: 0.8749 - val_loss: 1.1202 - val_accuracy: 0.6864
Epoch 00338: early stopping
```

Figure 13: Training epochs

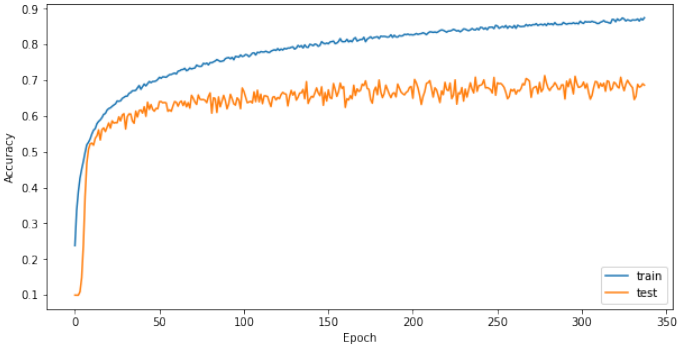



Figure 14: Accuracy

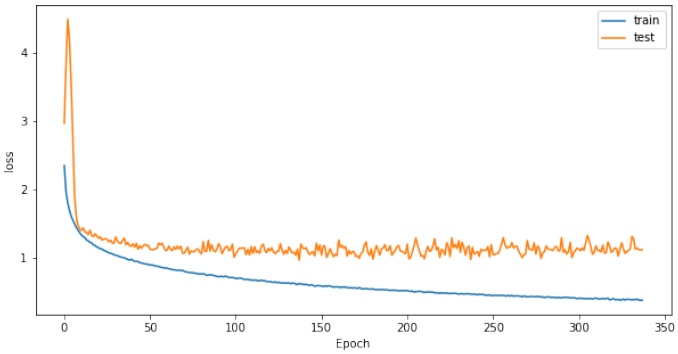



Figure 15: Loss

## B. *Test Results*

Our experiment shows that the test accuracy(73%) was best with minimum test loss(0.8747) during the scenario of using only batch normalization. When the batch normalization was not used both the loss and accuracy were not affected. But the training epochs required was more. When the data augmentation was applied to the training set, the test accuracy decreased to 68%, and test loss was also high compared to the other two cases.

*1) With batch normalization:*

```
[ ]  1 test_acc = model.evaluate(test_batch, verbose=2)
     2 print('Test accuracy: ',test_acc[1])
     3 print('Test loss: ',test_acc[0])
     4

31/31 - 5s - loss: 0.8747 - accuracy: 0.7320 - 5s/epoch - 150ms/step
Test accuracy:  0.7319745421409607
Test loss:  0.8747273087501526
```

Figure 16: Evaluation with Batch Normalization

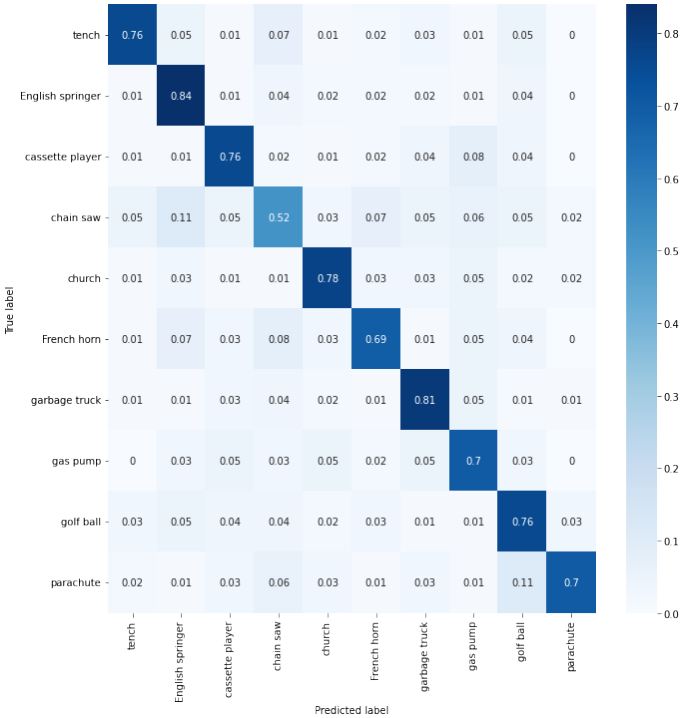



Figure 17: Confusion Matrix with Batch Normalization

*2) Without batch normalization:*

```
[ ]  1 test_acc = model.evaluate(test_batch, verbose=2)
     2 print('Test accuracy: ',test_acc[1])
     3 print('Test loss: ',test_acc[0])
     4
```

```
31/31 - 5s - loss: 0.8822 - accuracy: 0.7320 - 5s/epoch - 150ms/step
Test accuracy:  0.7319745421409607
Test loss:  0.8822094798088074
```

Figure 18: Evaluation without Batch Normalization

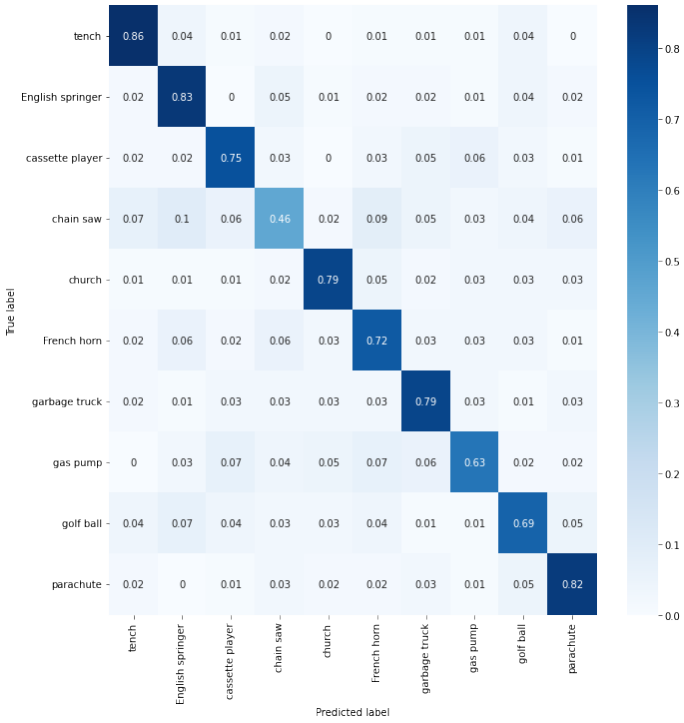



Figure 19: Confusion Matrix without Batch Normalization

*3) With batch normalization and data augmentation:*

```
[ ]  1 test_acc = model.evaluate(test_batch, verbose=2)
     2 print('Test accuracy: ',test_acc[1])
     3 print('Test loss: ',test_acc[0])
     4
```

```
31/31 - 5s - loss: 1.1202 - accuracy: 0.6864 - 5s/epoch - 151ms/step
Test accuracy:  0.6863694190979004
Test loss:  1.120245099067688
```

Figure 20: Evaluation with Batch Normalization and Data augmentation

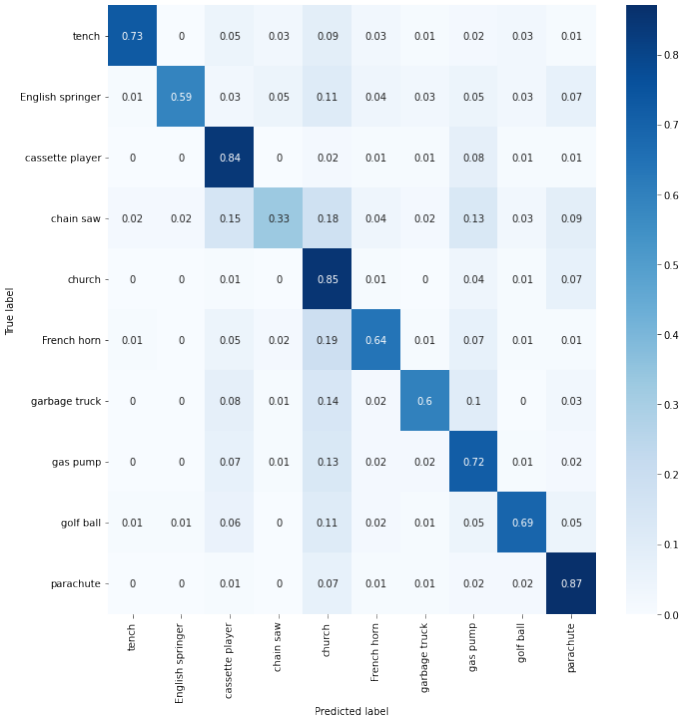



Figure 21:Confusion Matrix with Batch Normalization and Data augmentation

## IV. Imagewoof Classification

We started the Imagewoof classification using the base model from Imagenette classification. The model performance decreased mainly due to the complexity of the Imagewoof dataset.

The dataset consists of 10 breed dog classes. The model as shown in figure 22, gave a test accuracy of just 45%, with a test loss of 1.691.

As a next step to improve the accuracy, We increased the convolution layers and the channels to enable the network to learn more complex features during the training. The modified model produced better test accuracy of 63% and test loss was 1.5061. Since the accuracy have improved but overfitting was still an issue for the current model, we decided to use the data augmentation techniques as part of preprocessing similar to the experiment done for the imagenette dataset which resulted in improved accuracy.The test accuracy increased to 66% and test loss was 1.1665.

```
[16] # patient early stopping
     es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
     model = tf.keras.models.Sequential([
         tf.keras.layers.Conv2D(32,(3,3),padding='same', activation=tf.keras.layers.ReLU(),input_shape=(size_image,size_image,3)),
         tf.keras.layers.Conv2D(32,(3,3),padding='same', activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Conv2D(64,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(64,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Conv2D(128,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(128,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(128,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Conv2D(256,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(256,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(256,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.Conv2D(512,(3,3),padding='same',activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.MaxPooling2D((2,2),strides=2),
         tf.keras.layers.Flatten(),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(64,activation=tf.keras.layers.ReLU()),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.Dense(10,activation='softmax')])
```

Figure 22: Imagewoof modified CNN model

## A. *Training Details and Experiments*

*1) With the previous model used for Imagenette data classification:* The model was trained for 100 epochs with early stopping set on validation loss value with a patience of 10 epochs. The training continued for 33 epochs before the 'early stopping' was applied. at the end of 38 epochs the validation accuracy was 45% and the validation loss was 1.70. The training accuracy during early stopping had reached 80.80%, while the training loss was 0.6180.

```
Epoch 26/100
71/71 [==============================] - 20s 286ms/step - loss: 0.9063 - accuracy: 0.6993 - val_loss: 1.6415 - val_accuracy: 0.4505
Epoch 27/100
71/71 [==============================] - 20s 284ms/step - loss: 0.8765 - accuracy: 0.7157 - val_loss: 1.6246 - val_accuracy: 0.4599
Epoch 28/100
71/71 [==============================] - 20s 287ms/step - loss: 0.8593 - accuracy: 0.7170 - val_loss: 1.5979 - val_accuracy: 0.4663
Epoch 29/100
71/71 [==============================] - 20s 284ms/step - loss: 0.8227 - accuracy: 0.7370 - val_loss: 1.6198 - val_accuracy: 0.4614
Epoch 30/100
71/71 [==============================] - 20s 284ms/step - loss: 0.8004 - accuracy: 0.7404 - val_loss: 1.6558 - val_accuracy: 0.4556
Epoch 31/100
71/71 [==============================] - 20s 285ms/step - loss: 0.7835 - accuracy: 0.7458 - val_loss: 1.6736 - val_accuracy: 0.4505
Epoch 32/100
71/71 [==============================] - 20s 285ms/step - loss: 0.7533 - accuracy: 0.7562 - val_loss: 1.6721 - val_accuracy: 0.4597
Epoch 33/100
71/71 [==============================] - 20s 285ms/step - loss: 0.7327 - accuracy: 0.7664 - val_loss: 1.6522 - val_accuracy: 0.4650
Epoch 34/100
71/71 [==============================] - 20s 287ms/step - loss: 0.7127 - accuracy: 0.7723 - val_loss: 1.6467 - val_accuracy: 0.4599
Epoch 35/100
71/71 [==============================] - 20s 285ms/step - loss: 0.6822 - accuracy: 0.7799 - val_loss: 1.6817 - val_accuracy: 0.4500
Epoch 36/100
71/71 [==============================] - 20s 284ms/step - loss: 0.6567 - accuracy: 0.7908 - val_loss: 1.8386 - val_accuracy: 0.4235
Epoch 37/100
71/71 [==============================] - 20s 284ms/step - loss: 0.6375 - accuracy: 0.8004 - val_loss: 1.6537 - val_accuracy: 0.4653
Epoch 38/100
71/71 [==============================] - 20s 284ms/step - loss: 0.6180 - accuracy: 0.8080 - val_loss: 1.6914 - val_accuracy: 0.4561
Epoch 00038: early stopping
```

Figure 23: Training epochs

The below figure shows the accuracy and loss plots for the training. From the plot, we can see that there is an overfitting issue for the model after 8 epochs.

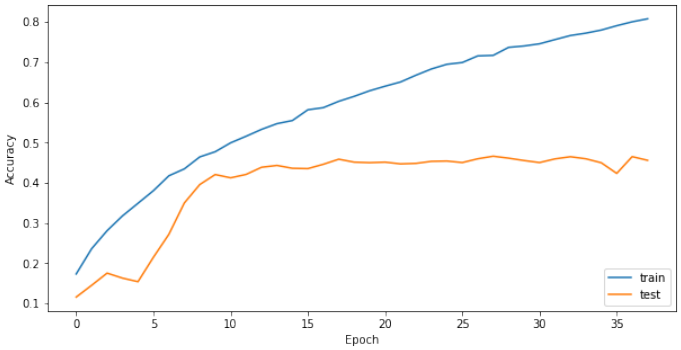



Figure 24: Accuracy

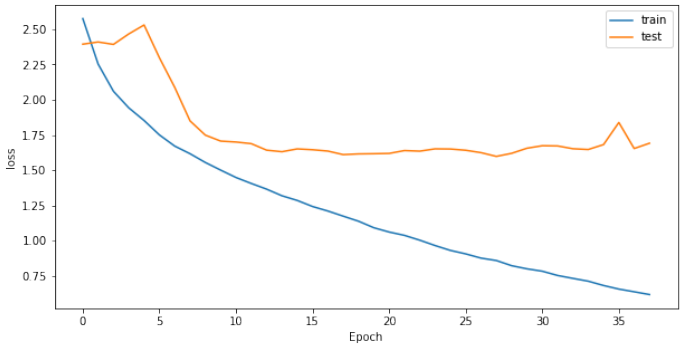



Figure 25: Loss

*2) With modifications to the previous model:* Now we modified the model to increase the accuracy to 63% by adding more convolution layers to learn complex features . We also increased the size of the image to 224 and the batch size was reduced from 128 to 64. After training the test accuracy was increased by 18% and test loss was slightly reduced to 1.5061. But the overfitting issue was persisting.

```
Epoch 8/50
142/142 [==============================] - 35s 245ms/step - loss: 0.9110 - accuracy: 0.6864 - val_loss: 1.4530 - val_accuracy: 0.5129
Epoch 9/50
142/142 [==============================] - 35s 244ms/step - loss: 0.7801 - accuracy: 0.7330 - val_loss: 1.4739 - val_accuracy: 0.5256
Epoch 10/50
142/142 [==============================] - 35s 246ms/step - loss: 0.6340 - accuracy: 0.7823 - val_loss: 1.2482 - val_accuracy: 0.6060
Epoch 11/50
142/142 [==============================] - 35s 246ms/step - loss: 0.4982 - accuracy: 0.8336 - val_loss: 1.6773 - val_accuracy: 0.4907
Epoch 12/50
142/142 [==============================] - 35s 244ms/step - loss: 0.3803 - accuracy: 0.8757 - val_loss: 1.6416 - val_accuracy: 0.5424
Epoch 13/50
142/142 [==============================] - 35s 244ms/step - loss: 0.3061 - accuracy: 0.9008 - val_loss: 1.3721 - val_accuracy: 0.6009
Epoch 14/50
142/142 [==============================] - 35s 244ms/step - loss: 0.2352 - accuracy: 0.9273 - val_loss: 1.7064 - val_accuracy: 0.5230
Epoch 15/50
142/142 [==============================] - 35s 245ms/step - loss: 0.1776 - accuracy: 0.9467 - val_loss: 1.4756 - val_accuracy: 0.5867
Epoch 16/50
142/142 [==============================] - 35s 245ms/step - loss: 0.1605 - accuracy: 0.9493 - val_loss: 1.8667 - val_accuracy: 0.5582
Epoch 17/50
142/142 [==============================] - 35s 245ms/step - loss: 0.1359 - accuracy: 0.9602 - val_loss: 1.9805 - val_accuracy: 0.5498
Epoch 18/50
142/142 [==============================] - 35s 244ms/step - loss: 0.1172 - accuracy: 0.9650 - val_loss: 1.9748 - val_accuracy: 0.5434
Epoch 19/50
142/142 [==============================] - 35s 244ms/step - loss: 0.1109 - accuracy: 0.9668 - val_loss: 2.2950 - val_accuracy: 0.5172
Epoch 20/50
142/142 [==============================] - 35s 243ms/step - loss: 0.1116 - accuracy: 0.9655 - val_loss: 1.5062 - val_accuracy: 0.6315
Epoch 00020: early stopping
```

Figure 26: Training epochs without data augmentation

The below figure shows the accuracy and loss plots for the training without data augmentation. We can see that the accuracy is increased, but we are not able to reduce the overfitting problem after modifying the model.

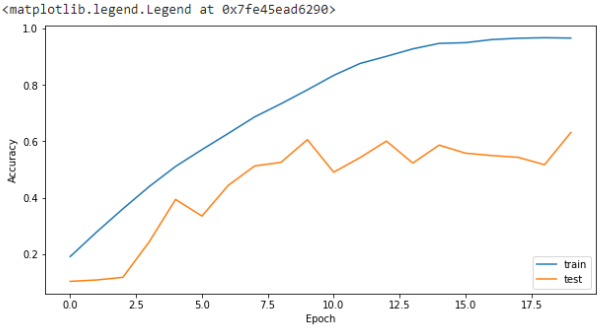



Figure 27: Accuracy

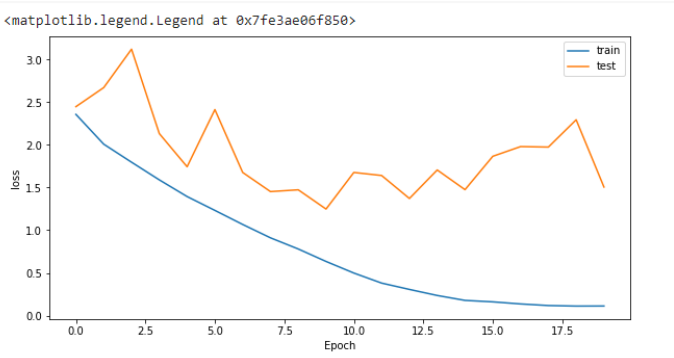



Figure 28: Loss

*3) Model with data augmentation:* To overcome the issue of overfitting, we incorporated data augmentation as part of preprocessing step. With data augmentation, the results were good and we could attain test accuracy of 66% with less overfitting.

```
Epoch 36/50
142/142 [==============================] - 79s 553ms/step - loss: 0.6924 - accuracy: 0.7642 - val_loss: 1.3940 - val_accurac
y: 0.6129
Epoch 37/50
142/142 [==============================] - 79s 560ms/step - loss: 0.6758 - accuracy: 0.7661 - val_loss: 1.2184 - val_accurac
y: 0.6124
Epoch 38/50
142/142 [==============================] - 81s 568ms/step - loss: 0.6702 - accuracy: 0.7712 - val_loss: 1.1666 - val_accurac
y: 0.6633
Epoch 00038: early stopping
```

Figure 29: Training epochs with data augmentation

The below figure shows the accuracy and loss plots for the training. From the plot, we can see that there is a less overfitting issue for the model.

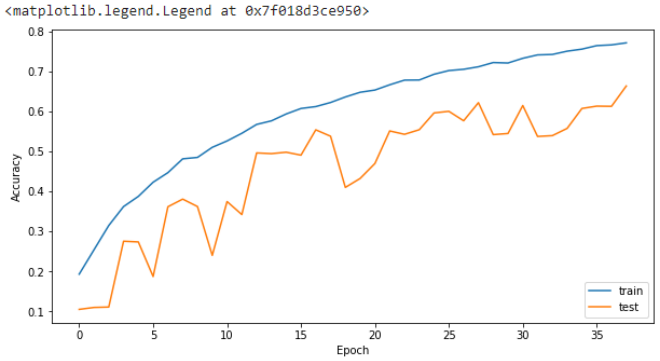



Figure 30: Accuracy

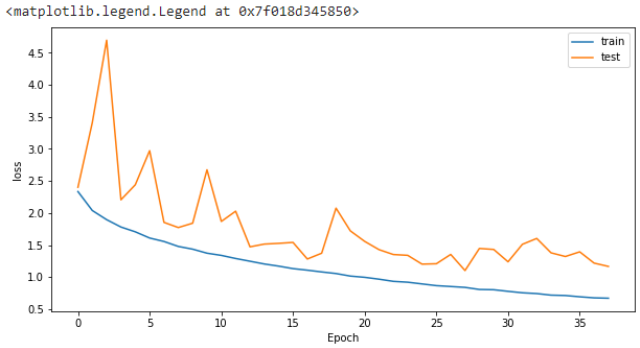



Figure 31: Loss

## B. *Test Results*

Our experiment shows that the test accuracy was best with minimum test loss during the scenario of using data augmentation for the imagewoof dataset. The data augmentation could reduce the issue of overfitting and the accuracy was brought up by 4% compared to the other models. The confusion matrix below shows the accuracy per class level for each experiments conducted during the training.

*1) With previously trained model on Imagenette data:*

```
test_acc = model.evaluate(test_batch, verbose=2)
print('Test accuracy: ',test_acc[1])
print('Test loss: ',test_acc[0])

31/31 - 6s - loss: 1.6914 - accuracy: 0.4561 - 6s/epoch - 190ms/step
Test accuracy:  0.45609569549560547
Test loss:  1.6914212703704834
```

Figure 32: Evaluation with Previously trained model on Imagenette data

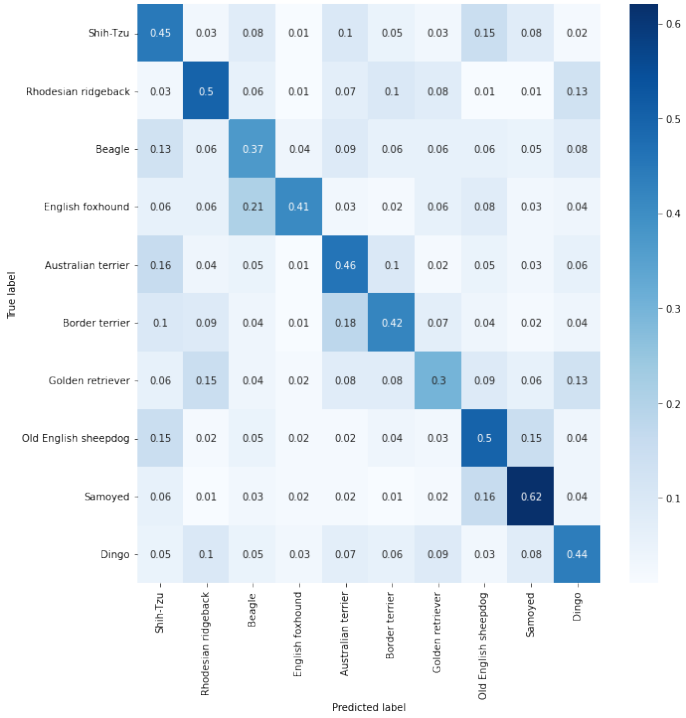



Figure 33: Confusion Matrix

*2) With modifications to the previous model:*

```
test_acc = model.evaluate(test_batch, verbose=2)
print('Test accuracy: ',test_acc[1])
print('Test loss: ',test_acc[0])

62/62 - 7s - loss: 1.5062 - accuracy: 0.6315 - 7s/epoch - 115ms/step
Test accuracy:  0.6314584016799927
Test loss:  1.5061798095703125
```

Figure 34: Evaluation with modified model

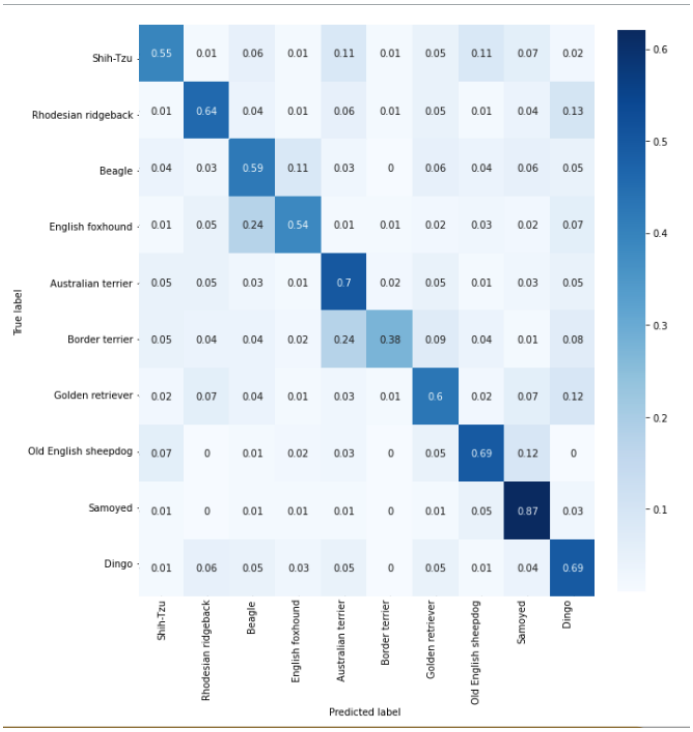



Figure 35: Confusion Matrix of the modified model

*3) With modifications and data augmentation:*

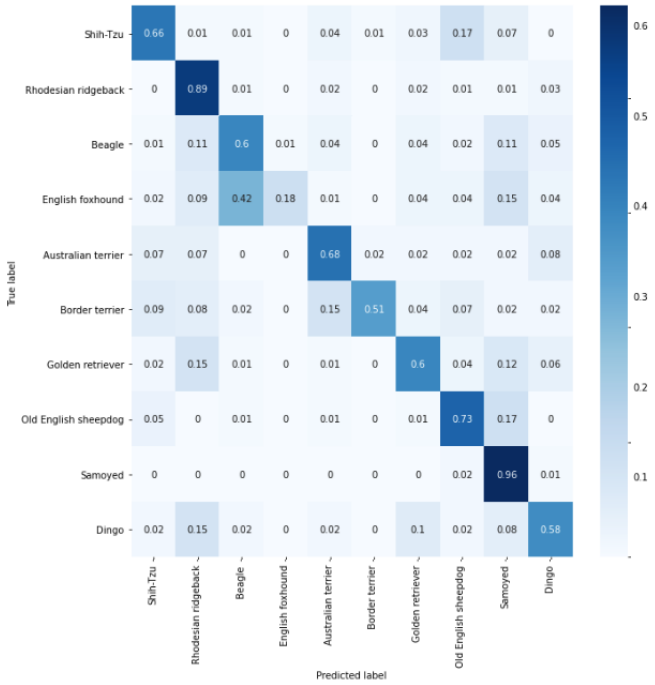



Figure 36: Confusion Matrix of the modified model with data augmentation

```
test_acc = model.evaluate(test_batch, verbose=2)
print('Test accuracy: ',test_acc[1])
print('Test loss: ',test_acc[0])
```

```
62/62 - 9s - loss: 1.1666 - accuracy: 0.6633 - 9s/epoch - 150ms/step
Test accuracy:  0.6632730960845947
Test loss:  1.1665747165679932
```

Figure 37: Evaluation with data augmentation

## V. Conclusion

In this project, we present the classification problem for imagenette and imagewoof complex datasets using simple CNN models, without the help of transfer learning. This classification was challenging since the images were too complex for a simple CNN to learn the features and classify them correctly.We started with a simple CNN model for the Imagenette dataset. The maximum accuracy we could attain was 73%. As part of second stage of classification challenge we used Imagewoof dataset and increased the model complexity to get better classification accuracy.The best accuarcy we could attain with CNN model with more layers was 63% without data augmentation and 66% with data augmentation.

Now we wanted to verify a few images from the test dataset. Let's plot several images with their predictions. Note that the model can be wrong even when very confident. Correct prediction labels are blue and incorrect prediction labels are red. The number gives the percentage (out of 100) for the predicted label.

### *A. imagenette predictions*

You can see that predictions for the class french horn were miss classified as chain saw. Even for the human eye, the image is noisy and the french horn is not the main focus in the image. Similar is the case for the gas pump, it is miss classified as a cassette player. But the model has learned the features of the cassette player, which is mainly rectangular shapes occurring in the images. (Figure 38 & 39)

### *B. imagewoof predictions*

In Figures 40 & 41, you can see that predictions for the class depend on the clarity of the images of the dog. The images where the dog features are not clearly visible are sometimes miss classified. Also many images have humans in the background and the dogs are not the main subject of that images.


Figure 38: imagenette correct classification

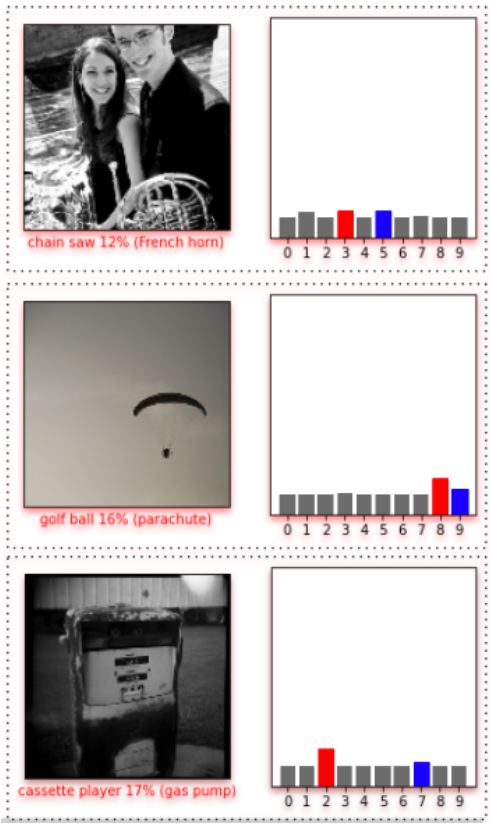



Figure 39: imagenette Incorrect classification


Figure 40: imagewoof Incorrect classification

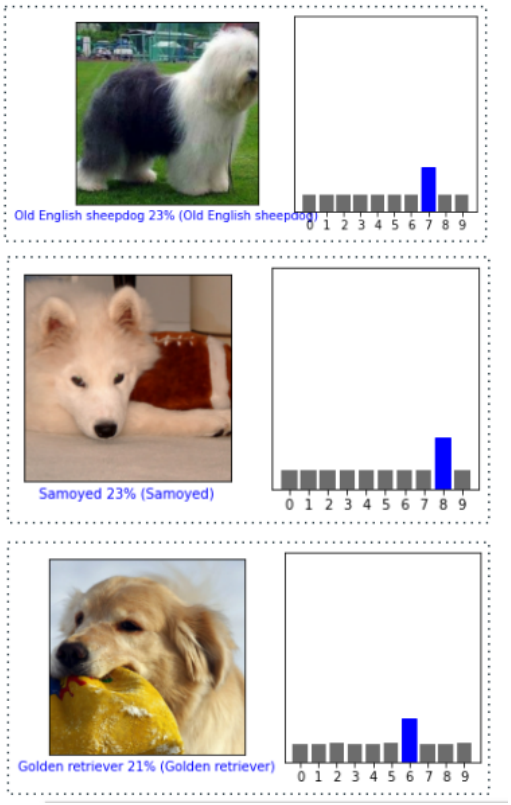



Figure 41: imagewoof correct classification

## REFERENCES


[1] https://github.com/fastai/imagenette

[2] https://www.tensorflow.org/

[3] https://www.tensorflow.org/datasets/catalog/imagenette