GNR638

MINI PROJECT 1

GROUP:

SARVESH PATIL-22B2276

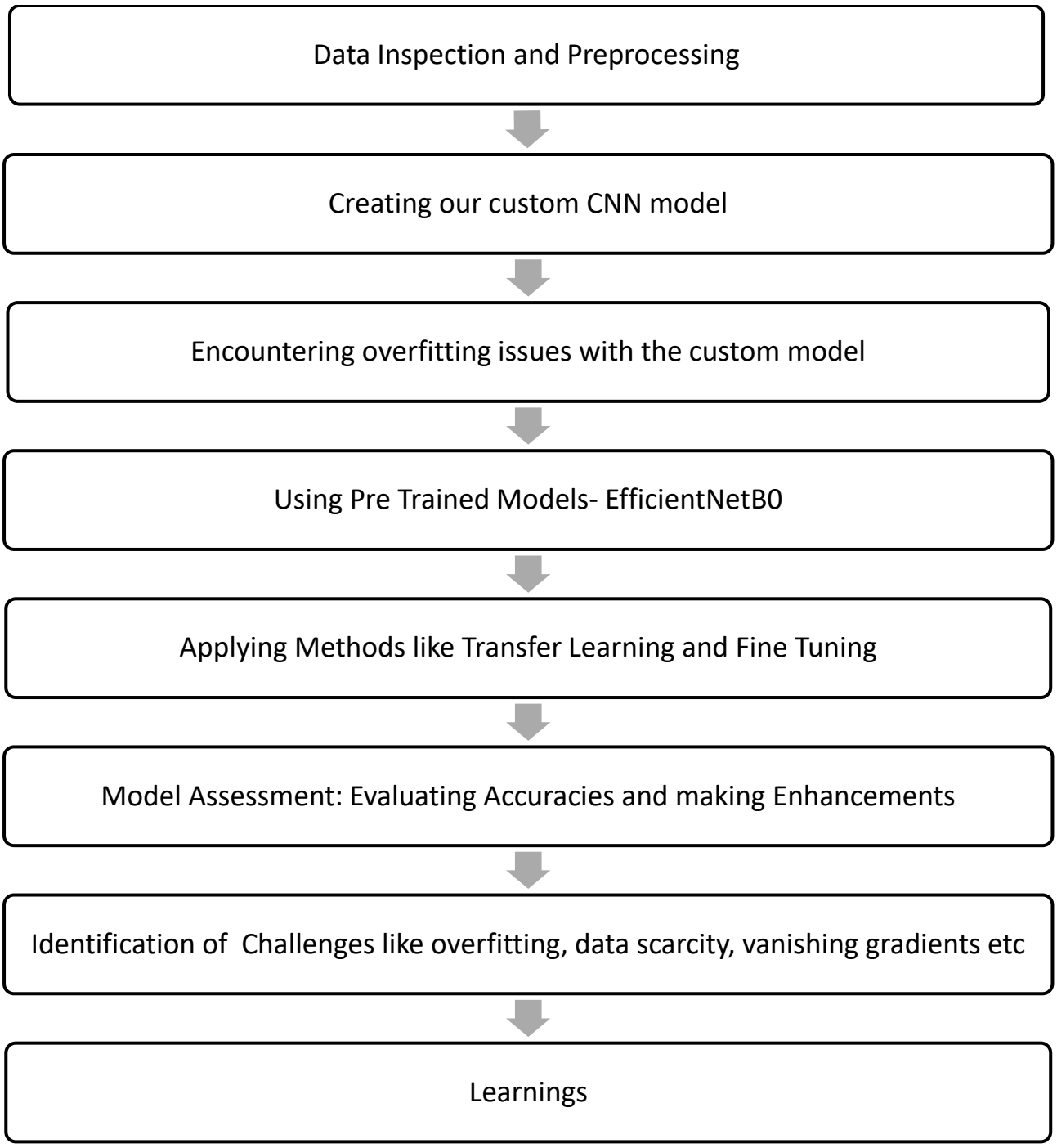
STUTI SINGLA-22B0046

INDEX:

INTRODUCTION OF THE PROBLEM STATEMENT:

We were given with a CUB dataset containing 11788 images belonging to 200 different classes. Our Problem Statement was to train a Convolutional Neural Network for the given dataset with less than 10 Million parameters.

FLOW OF THE PROJECT:



DETAILED PROJECT:

1. **Data Inspection:**

   We carefully inspected and tried to understand the data provided and how it was segregated.

2. **Data Preprocessing:**

   The images were provided to us in a folder named images containing 200 subfolders named by the classes, each containing about 60 pictures. Since we were asked to use the default test train split, provided in the "train_test_split.txt" file which contained 1s for train data indexes and 0s for test data indexes, we converted the data into a tensor, by reading the file and then used it to make two lists train_index and test_index containing the indexs at which we have train data points and test data points respectively.

   a) **Creating y_train and y_test:**

   We read the image_class_labels.txt into a tensor and then iterated over the tensor using a for loop. We appended the Class Label Number into y_train/test if the corresponding index was found in train_index/test_index.

   b) **Creating x_train and x_test:**

   Now, the links to the images were provided to us in a file named images.txt. Again we read the data and stored the same in a numpy array with !st dimension containing the sequence number and 2nd contating the link. We concatenated the link with a str ".//CUB_200_2011//images//" so as to make the link more accessible to us while reading the image files.

   Next, we initialised two empty lists: x_train and x_test; enumerated over the data_array containing the links, resized each individual list to (txt); appended the image to x_train if index in train_index.

   We then tried to normalise the images to range [0,1] which made us run into an error:

   ***ValueError**: setting an array element with a sequence. The requested array has an inhomogeneous shape after 3 dimensions. The detected shape was (5994, 64, 64) + inhomogeneous part.*

   c) **Deleting Inhomogeneuities and Normalizing the Data:**

   This lead us to visually inspect the data dimensions, where everything seemed fine, but on close inspection using a for loop we found some data points which did not have channels; we deleted those data points and finally normalised x_train and x_test; also converted y_train and y_test to one hot encoding.

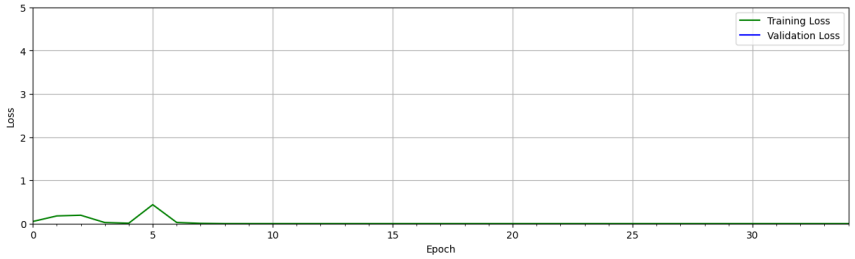
3. Implementing Our Own CNN Model:

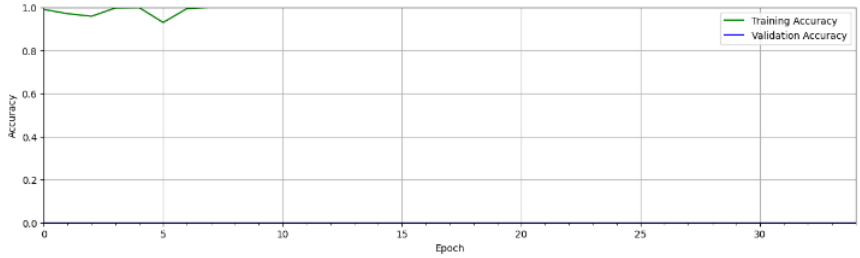
Model without Dropout:

We created a Sequential Model.
Total No. of layers: 12
Total No. of Parameters: 9,150,985 with input size (64x64x3)
Architecture of the Model:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	1,792
conv2d_1 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73,856
conv2d_3 (Conv2D)	(None, 32, 32, 128)	147,584
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	147,584
conv2d_5 (Conv2D)	(None, 16, 16, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1024)	8,389,632
dense_1 (Dense)	(None, 201)	206,025

We compiled the model using “rmsprop” as optimizer, “categorical_crossentropy” as loss and “accuracy” as metrics.
The above model reached 100% training accuracy in only 9 epochs.
The test accuracy using this model was 4.33%





Conclusion: The model was highly overfit.

a) Model Using Dropout Layer:

Sequential Model:

No. of Layers: 16

No. of Parameters: 9,150,985 with input size (64x64x3)

Architecture of the Model:

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 64, 64, 64)	1,792
conv2d_7 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout (Dropout)	(None, 32, 32, 64)	0
conv2d_8 (Conv2D)	(None, 32, 32, 128)	73,856
conv2d_9 (Conv2D)	(None, 32, 32, 128)	147,584
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_1 (Dropout)	(None, 16, 16, 128)	0
conv2d_10 (Conv2D)	(None, 16, 16, 128)	147,584
conv2d_11 (Conv2D)	(None, 16, 16, 128)	147,584
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_2 (Dropout)	(None, 8, 8, 128)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 1024)	8,389,632
dropout_3 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 201)	206,025

We again compiled the model using “rmsprop” as optimizer, “categorical_crossentropy” as loss and “accuracy” as metrics.
The above model reached 90% training accuracy in only 35 epochs.
The test accuracy was found to be: 6%
In our model, we observed a discrepancy between the high training accuracy of 90% achieved in just 35 epochs and the low test accuracy of 6%. We suspect that this difference may be due to vanishing gradients. Vanishing gradients occur when the gradients used to update the weights during training become extremely small as they propagate backward through the layers during backpropagation. As a result, the weights in the earlier layers of the network may not update effectively, leading to slow or stalled learning. This phenomenon can prevent the model from effectively learning complex patterns in the data, resulting in lower accuracy on the test set despite high accuracy during training. Therefore, we believe that vanishing gradients may be a contributing factor to the discrepancy between our training and test accuracy.

4. **Using a Pre trained Model:**

We decided to use EfficientNetB0 as our pretrained model for the following reasons:
EfficientNet is a family of convolutional neural network architectures designed to achieve state-of-the-art performance with a highly efficient use of computational resources, such as memory and processing power.
It has impressive performance on various computer vision tasks while being more resource-efficient compared to other architectures like ResNet or Inception.
We specifically used version B0 because it used less no. of parameters (around 4M) without the top.
Total parameters with the top: 9M

We started by using **Transfer Learning** methods used the EfficientNet Model’s feature extractor, flattened the output added a dense layer having a total of 1024 nodes, a dropout layer, and finally the final classification layer having 201 nodes.
We compiled the model using “RMS prop” with a learning rate of 0.0001 as optimiser, “Binary_crossentropy” as loss and “accuracy” as metrics.
We saw that the model was learning slowly, but steadily.

It had a decent test to train accuracy ratio, so we decided to finetune the model. Fine-tuning a model refers to the process of taking a pre-trained model (usually trained on a large dataset like ImageNet) and further training it on a smaller, task-specific dataset. Fine Tuning would leverage the knowledge learned by the pre-trained model to improve performance on the new dataset, with less data and computation than training a model from scratch.
Another thing we observed was that we were using “binary_crossentropy” as our loss. We changed it to “categorical_crossentropy” and saw almost and exponential growth in the training process as compared to when we used “binary_crossentropy”

a) No. of layers finetuned=7

```
Model: "model_13"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_14 (InputLayer)       [(None, 64, 64, 3)]       0         []

 stem_conv (Conv2D)          (None, 32, 32, 32)        864       ['input_14[0][0]']

 stem_bn (BatchNormalizatio  (None, 32, 32, 32)        128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 32, 32, 32)        0         ['stem_bn[0][0]']
 n)

 block1a_dwconv (DepthwiseC  (None, 32, 32, 32)        288       ['stem_activation[0][0]']
 onv2D)

 block1a_bn (BatchNormaliza  (None, 32, 32, 32)        128       ['block1a_dwconv[0][0]']
 tion)

 block1a_activation (Activa  (None, 32, 32, 32)        0         ['block1a_bn[0][0]']
 tion)

 block1a_se_squeeze (Global  (None, 32)                0         ['block1a_activation[0][0]']
 AveragePooling2D)
...
Total params: 9499493 (36.24 MB)
Trainable params: 6231369 (23.77 MB)
Non-trainable params: 3268124 (12.47 MB)
```

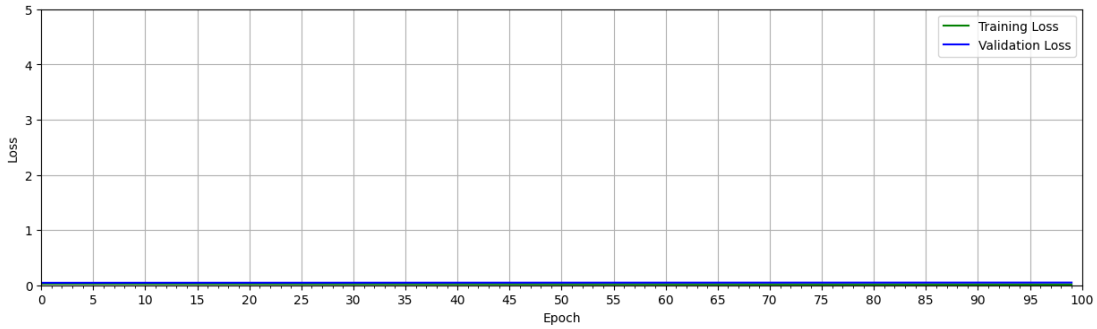
Total No. of Parameters= 9499493
No. of Trainable Parameters=6231369

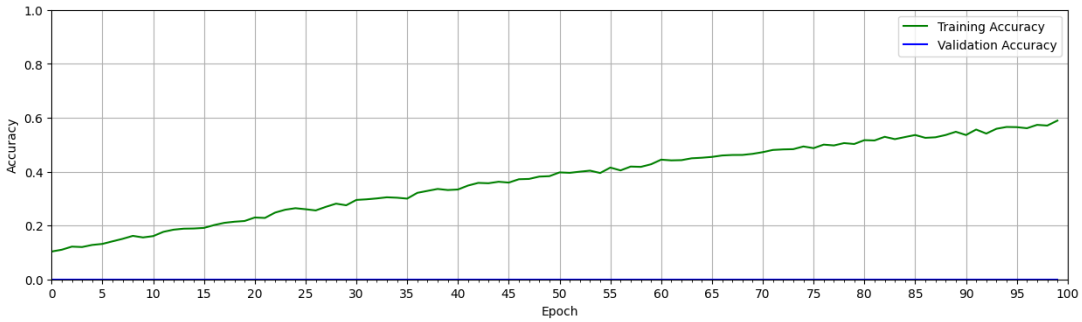
We compiled and trained the model with “RMS prop” lr=0.0001 as optimiser, “binary_crossentropy” as loss and “acc” as metrics.
Training Accuracy= 58.92%

Training Loss= 0.01

Test Accuracy=11.26%

Test Loss=0.0307





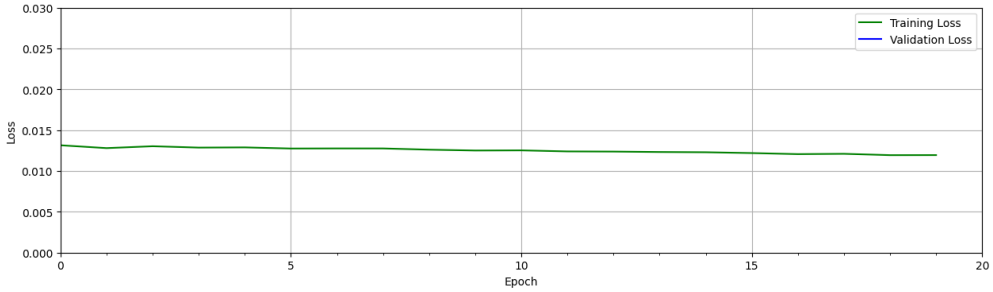
We decided to train the model using more epochs:

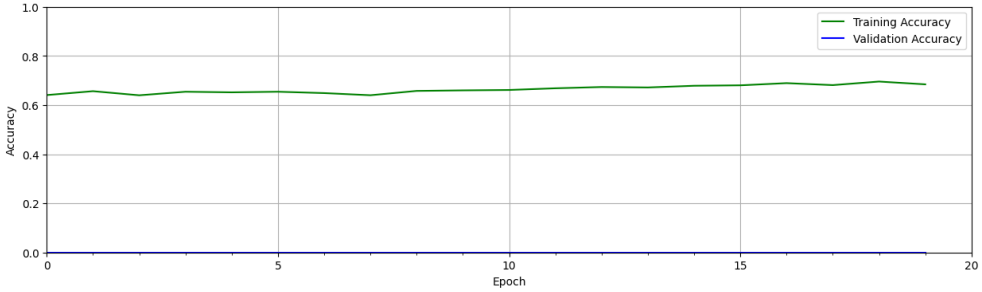
Training Accuracy= 68.43%

Training Loss= 0.0119

Test Accuracy=11.550%

Test Loss=0.0312





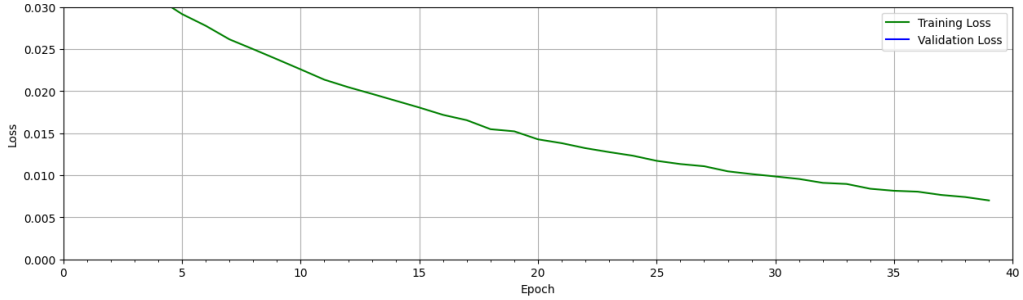
We trained the model again:

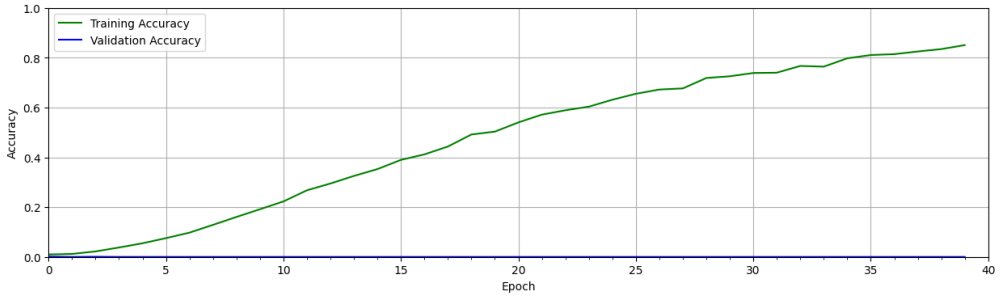
Training Accuracy= 85.06%

Training Loss= 0.007

Test Accuracy=14.934%

Test Loss=0.0341





The no. of layers in our model was 234.

We decided to finetune about 30% of it, ie, 70 layers

Also we used categorical_crossentropy as the loss this time around. This model was very clearly overfit. So we sticked to fine tuning 5 layers in our final model.

No. of layers Fine Tuned=5

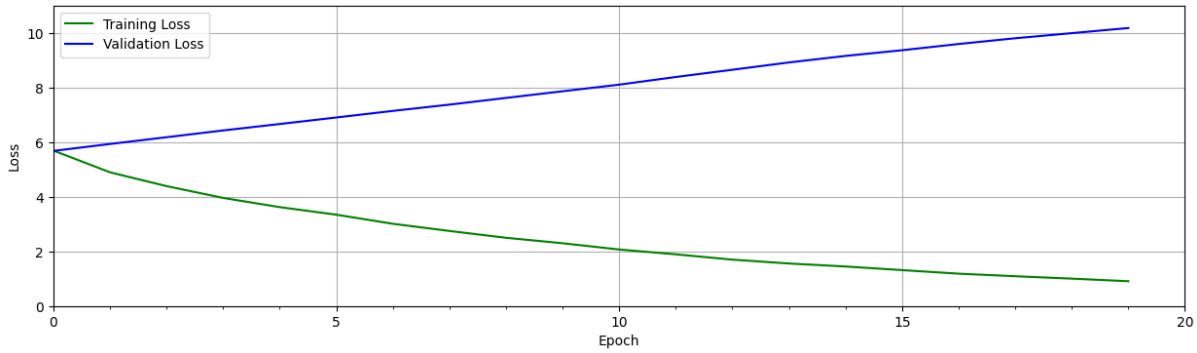
```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_4 (InputLayer)        [(None, 64, 64, 3)]       0         []

 stem_conv (Conv2D)          (None, 32, 32, 32)        864       ['input_4[0][0]']

 stem_bn (BatchNormalizatio  (None, 32, 32, 32)        128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 32, 32, 32)        0         ['stem_bn[0][0]']
 n)

 block1a_dwconv (DepthwiseC  (None, 32, 32, 32)        288       ['stem_activation[0][0]']
 onv2D)

 block1a_bn (BatchNormaliza  (None, 32, 32, 32)        128       ['block1a_dwconv[0][0]']
 tion)

 block1a_activation (Activa  (None, 32, 32, 32)        0         ['block1a_bn[0][0]']
 tion)

 block1a_se_squeeze (Global  (None, 32)                0         ['block1a_activation[0][0]']
 AveragePooling2D)
...
Total params: 9499493 (36.24 MB)
Trainable params: 5449929 (20.79 MB)
Non-trainable params: 4049564 (15.45 MB)
```

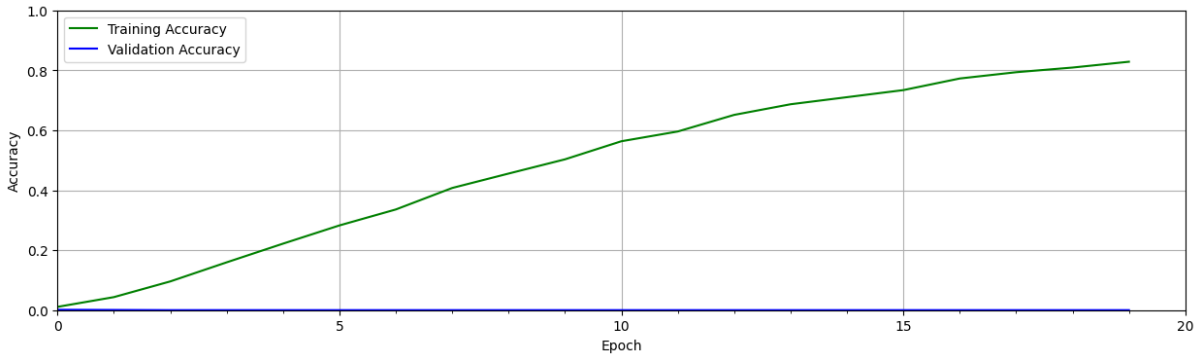
Train Accuracy: 82.85%

Train Loss: 0.90

Test Accuracy: 10.12%

Test Loss: 5.59





We witnessed that we were stuck with a test accuracy of around 10%. Even at different Training Accuracies. One of the reasons that this might have happened can be using a much smaller train data set than the one that EfficientNet model has been trained on.

Data augmentation

One solution to that would be Data Augmentation. Data augmentation involves applying various transformations to the original images to create new training samples. It effectively increases the size of the training dataset, allowing the model to learn from a more diverse set of examples. This can lead to better generalization performance and improved accuracy when the model is tested on unseen data.

Keeping this in mind we applied 4 times Data Augmentation. {Rotated by 90, 180, 270 degrees}

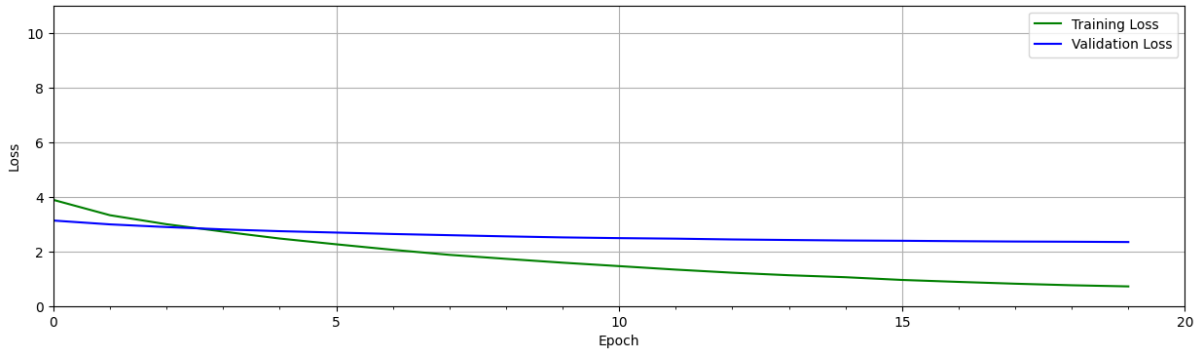
This increased the size of the train dataset from 5772, to 23952; which would allow the model to learn better.

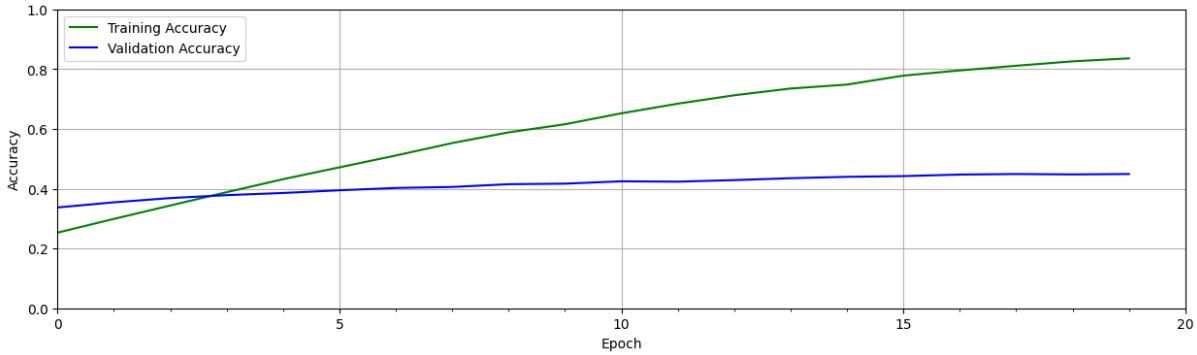
We also noticed that the train data that we were using was sorted, so we shuffled the dataset.

Data Parameters after Data Augmentation and Shuffling:

Training Accuracy: 82.58%

Training Loss: 0.7178
Test Accuracy: 11.654%
Test Loss: 4.8689





We were not able to increase accuracy beyond 12% even after data augmentation and random shuffling.

Since we were unable to obtain a test accuracy above 12% even after so many successive trials using EfficientNet; we decided to use another pretrained model.
We decided to go along with InceptionNet as our pretrained model but it had about 30M parameters.
Training Accuracy: 87%
Test Accuracy: 25%

Final Model:

For the final iteration of our model we decided to use the pretrained Model EfficientNetB0. We imported the feature extractor of the model and strapped it to our own extractor.

No. of Nodes in Dense Layer	Total No. Of Parameters
2048	14,949,228

1024	9499493
512	6774636
256	5412204

Hence we decided to go along with having 1024 nodes in the dense layer, added a dropout layer to avoid overfitting and a Final Dense classification layer having 201 nodes.

Architecture of The Model:

block6a_dwconv_pad (ZeroPadding2D)	(None, 7, 7, 672)	0	block6a_expand_a…
block6a_dwconv (DepthwiseConv2D)	(None, 2, 2, 672)	16,800	block6a_dwconv_p…
block6a_bn (BatchNormalizatio…	(None, 2, 2, 672)	2,688	block6a_dwconv[0…
block6a_activation (Activation)	(None, 2, 2, 672)	0	block6a_bn[0][0]
block6a_se_squeeze (GlobalAveragePool…	(None, 672)	0	block6a_activati…
block6a_se_reshape (Reshape)	(None, 1, 1, 672)	0	block6a_se_squee…
block6a_se_reduce (Conv2D)	(None, 1, 1, 28)	18,844	block6a_se_resha…
block6a_se_expand (Conv2D)	(None, 1, 1, 672)	19,488	block6a_se_reduc…
block6a_se_excite (Multiply)	(None, 2, 2, 672)	0	block6a_activati… block6a_se_expan…
block6a_project_co… (Conv2D)	(None, 2, 2, 192)	129,024	block6a_se_excit…

block5b_drop (Dropout)	(None, 4, 4, 112)	0	block5b_project_…
block5b_add (Add)	(None, 4, 4, 112)	0	block5b_drop[0][… block5a_project_…
block5c_expand_conv (Conv2D)	(None, 4, 4, 672)	75,264	block5b_add[0][0]
block5c_expand_bn (BatchNormalizatio…	(None, 4, 4, 672)	2,688	block5c_expand_c…
block5c_expand_act… (Activation)	(None, 4, 4, 672)	0	block5c_expand_b…
block5c_dwconv (DepthwiseConv2D)	(None, 4, 4, 672)	16,800	block5c_expand_a…
block5c_bn (BatchNormalizatio…	(None, 4, 4, 672)	2,688	block5c_dwconv[0…
block5c_activation (Activation)	(None, 4, 4, 672)	0	block5c_bn[0][0]
block5c_se_squeeze (GlobalAveragePool…	(None, 672)	0	block5c_activati…
block5c_se_reshape (Reshape)	(None, 1, 1, 672)	0	block5c_se_squee…
block5c_se_reduce (Conv2D)	(None, 1, 1, 28)	18,844	block5c_se_resha…
block5c_se_expand (Conv2D)	(None, 1, 1, 672)	19,488	block5c_se_reduc…
block5c_se_excite (Multiply)	(None, 4, 4, 672)	0	block5c_activati… block5c_se_expan…
block5c_project_co… (Conv2D)	(None, 4, 4, 112)	75,264	block5c_se_excit…
block5c_project_bn (BatchNormalizatio…	(None, 4, 4, 112)	448	block5c_project_…
block5c_drop (Dropout)	(None, 4, 4, 112)	0	block5c_project_…
block5c_add (Add)	(None, 4, 4, 112)	0	block5c_drop[0][… block5b_add[0][0]
block6a_expand_conv (Conv2D)	(None, 4, 4, 672)	75,264	block5c_add[0][0]
block6a_expand_bn (BatchNormalizatio…	(None, 4, 4, 672)	2,688	block6a_expand_c…
block6a_expand_act…	(None, 4, 4, 672)	0	block6a_expand_b…

block5a_se_squeeze (GlobalAveragePool…	(None, 480)	0	block5a_activati…
block5a_se_reshape (Reshape)	(None, 1, 1, 480)	0	block5a_se_squee…
block5a_se_reduce (Conv2D)	(None, 1, 1, 20)	9,620	block5a_se_resha…
block5a_se_expand (Conv2D)	(None, 1, 1, 480)	10,080	block5a_se_reduc…
block5a_se_excite (Multiply)	(None, 4, 4, 480)	0	block5a_activati… block5a_se_expan…
block5a_project_co… (Conv2D)	(None, 4, 4, 112)	53,760	block5a_se_excit…
block5a_project_bn (BatchNormalizatio…	(None, 4, 4, 112)	448	block5a_project_…
block5b_expand_conv (Conv2D)	(None, 4, 4, 672)	75,264	block5a_project_…
block5b_expand_bn (BatchNormalizatio…	(None, 4, 4, 672)	2,688	block5b_expand_c…
block5b_expand_act… (Activation)	(None, 4, 4, 672)	0	block5b_expand_b…
block5b_dwconv (DepthwiseConv2D)	(None, 4, 4, 672)	16,800	block5b_expand_a…
block5b_bn (BatchNormalizatio…	(None, 4, 4, 672)	2,688	block5b_dwconv[0…
block5b_activation (Activation)	(None, 4, 4, 672)	0	block5b_bn[0][0]
block5b_se_squeeze (GlobalAveragePool…	(None, 672)	0	block5b_activati…
block5b_se_reshape (Reshape)	(None, 1, 1, 672)	0	block5b_se_squee…
block5b_se_reduce (Conv2D)	(None, 1, 1, 28)	18,844	block5b_se_resha…
block5b_se_expand (Conv2D)	(None, 1, 1, 672)	19,488	block5b_se_reduc…
block5b_se_excite (Multiply)	(None, 4, 4, 672)	0	block5b_activati… block5b_se_expan…
block5b_project_co… (Conv2D)	(None, 4, 4, 112)	75,264	block5b_se_excit…
block5b_project_bn (BatchNormalizatio…	(None, 4, 4, 112)	448	block5b_project_…

block4c_project_co… (Conv2D)	(None, 4, 4, 80)	38,400	block4c_se_excit…
block4c_project_bn (BatchNormalizatio…	(None, 4, 4, 80)	320	block4c_project_…
block4c_drop (Dropout)	(None, 4, 4, 80)	0	block4c_project_…
block4c_add (Add)	(None, 4, 4, 80)	0	block4c_drop[0][… block4b_add[0][0]
block5a_expand_conv (Conv2D)	(None, 4, 4, 480)	38,400	block4c_add[0][0]
block5a_expand_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block5a_expand_c…
block5a_expand_act… (Activation)	(None, 4, 4, 480)	0	block5a_expand_b…
block5a_dwconv (DepthwiseConv2D)	(None, 4, 4, 480)	12,000	block5a_expand_a…
block5a_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block5a_dwconv[0…
block5a_activation (Activation)	(None, 4, 4, 480)	0	block5a_bn[0][0]

block4c_expand_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block4c_expand_c…
block4c_expand_act… (Activation)	(None, 4, 4, 480)	0	block4c_expand_b…
block4c_dwconv (DepthwiseConv2D)	(None, 4, 4, 480)	4,320	block4c_expand_a…
block4c_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block4c_dwconv[0…
block4c_activation (Activation)	(None, 4, 4, 480)	0	block4c_bn[0][0]
block4c_se_squeeze (GlobalAveragePool…	(None, 480)	0	block4c_activati…
block4c_se_reshape (Reshape)	(None, 1, 1, 480)	0	block4c_se_squee…
block4c_se_reduce (Conv2D)	(None, 1, 1, 20)	9,620	block4c_se_resha…
block4c_se_expand (Conv2D)	(None, 1, 1, 480)	10,080	block4c_se_reduc…
block4c_se_excite (Multiply)	(None, 4, 4, 480)	0	block4c_activati… block4c_se_expan…

block4b_se_reduce (Conv2D)	(None, 1, 1, 20)	9,620	block4b_se_resha…
block4b_se_expand (Conv2D)	(None, 1, 1, 480)	10,080	block4b_se_reduc…
block4b_se_excite (Multiply)	(None, 4, 4, 480)	0	block4b_activati… block4b_se_expan…
block4b_project_co… (Conv2D)	(None, 4, 4, 80)	38,400	block4b_se_excit…
block4b_project_bn (BatchNormalizatio…	(None, 4, 4, 80)	320	block4b_project_…
block4b_drop (Dropout)	(None, 4, 4, 80)	0	block4b_project_…
block4b_add (Add)	(None, 4, 4, 80)	0	block4b_drop[0][… block4a_project_…
block4c_expand_conv (Conv2D)	(None, 4, 4, 480)	38,400	block4b_add[0][0]
block4c_expand_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block4c_expand_c…

block4a_project_bn (BatchNormalizatio…	(None, 4, 4, 80)	320	block4a_project_…
block4b_expand_conv (Conv2D)	(None, 4, 4, 480)	38,400	block4a_project_…
block4b_expand_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block4b_expand_c…
block4b_expand_act… (Activation)	(None, 4, 4, 480)	0	block4b_expand_b…
block4b_dwconv (DepthwiseConv2D)	(None, 4, 4, 480)	4,320	block4b_expand_a…
block4b_bn (BatchNormalizatio…	(None, 4, 4, 480)	1,920	block4b_dwconv[0…
block4b_activation (Activation)	(None, 4, 4, 480)	0	block4b_bn[0][0]
block4b_se_squeeze (GlobalAveragePool…	(None, 480)	0	block4b_activati…
block4b_se_reshape (Reshape)	(None, 1, 1, 480)	0	block4b_se_squee…

block4a_dwconv (DepthwiseConv2D)	(None, 4, 4, 240)	2,160	block4a_dwconv_p…
block4a_bn (BatchNormalizatio…	(None, 4, 4, 240)	960	block4a_dwconv[0…
block4a_activation (Activation)	(None, 4, 4, 240)	0	block4a_bn[0][0]
block4a_se_squeeze (GlobalAveragePool…	(None, 240)	0	block4a_activati…
block4a_se_reshape (Reshape)	(None, 1, 1, 240)	0	block4a_se_squee…
block4a_se_reduce (Conv2D)	(None, 1, 1, 10)	2,410	block4a_se_resha…
block4a_se_expand (Conv2D)	(None, 1, 1, 240)	2,640	block4a_se_reduc…
block4a_se_excite (Multiply)	(None, 4, 4, 240)	0	block4a_activati… block4a_se_expan…
block4a_project_co… (Conv2D)	(None, 4, 4, 80)	19,200	block4a_se_excit…

block3b_se_excite (Multiply)	(None, 8, 8, 240)	0	block3b_activati… block3b_se_expan…
block3b_project_co… (Conv2D)	(None, 8, 8, 40)	9,600	block3b_se_excit…
block3b_project_bn (BatchNormalizatio…	(None, 8, 8, 40)	160	block3b_project_…
block3b_drop (Dropout)	(None, 8, 8, 40)	0	block3b_project_…
block3b_add (Add)	(None, 8, 8, 40)	0	block3b_drop[0][… block3a_project_…
block4a_expand_conv (Conv2D)	(None, 8, 8, 240)	9,600	block3b_add[0][0]
block4a_expand_bn (BatchNormalizatio…	(None, 8, 8, 240)	960	block4a_expand_c…
block4a_expand_act… (Activation)	(None, 8, 8, 240)	0	block4a_expand_b…
block4a_dwconv_pad (ZeroPadding2D)	(None, 9, 9, 240)	0	block4a_expand_a…

block3b_expand_bn (BatchNormalizatio…	(None, 8, 8, 240)	960	block3b_expand_c…
block3b_expand_act… (Activation)	(None, 8, 8, 240)	0	block3b_expand_b…
block3b_dwconv (DepthwiseConv2D)	(None, 8, 8, 240)	6,000	block3b_expand_a…
block3b_bn (BatchNormalizatio…	(None, 8, 8, 240)	960	block3b_dwconv[0…
block3b_activation (Activation)	(None, 8, 8, 240)	0	block3b_bn[0][0]
block3b_se_squeeze (GlobalAveragePool…	(None, 240)	0	block3b_activati…
block3b_se_reshape (Reshape)	(None, 1, 1, 240)	0	block3b_se_squee…
block3b_se_reduce (Conv2D)	(None, 1, 1, 10)	2,410	block3b_se_resha…
block3b_se_expand (Conv2D)	(None, 1, 1, 240)	2,640	block3b_se_reduc…

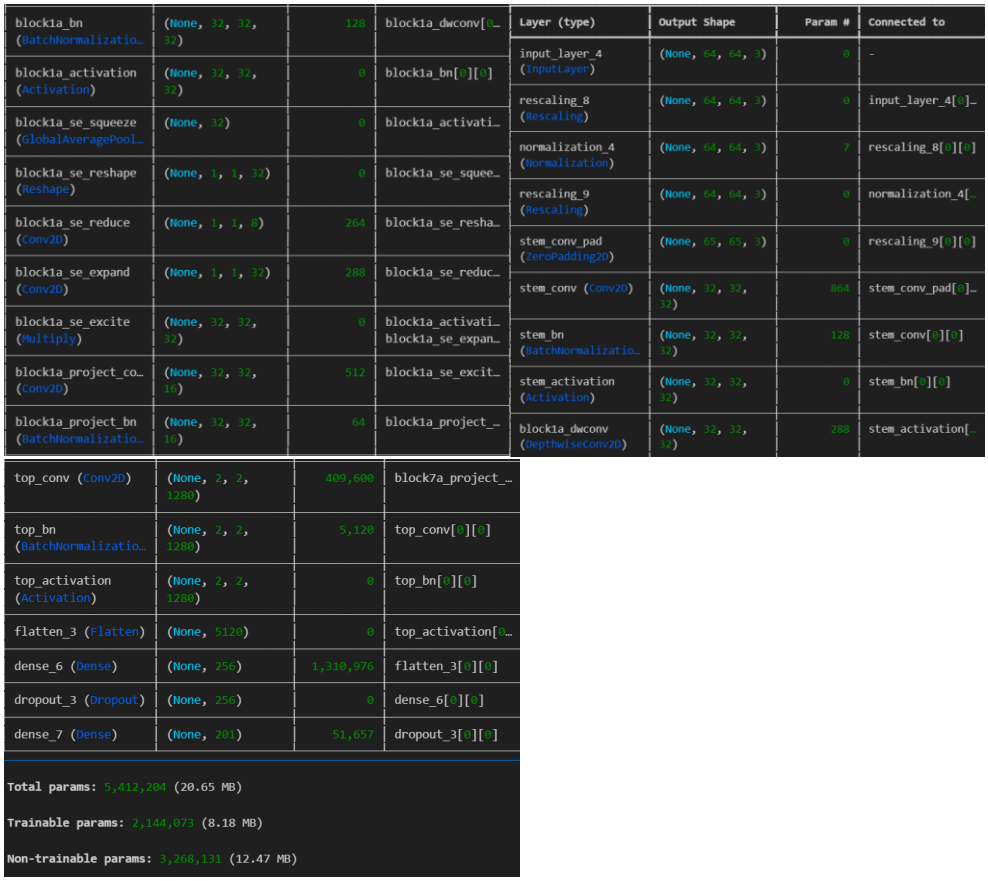
block3a_activation (Activation)	(None, 8, 8, 144)	0	block3a_bn[0][0]
block3a_se_squeeze (GlobalAveragePool…	(None, 144)	0	block3a_activati…
block3a_se_reshape (Reshape)	(None, 1, 1, 144)	0	block3a_se_squee…
block3a_se_reduce (Conv2D)	(None, 1, 1, 6)	870	block3a_se_resha…
block3a_se_expand (Conv2D)	(None, 1, 1, 144)	1,008	block3a_se_reduc…
block3a_se_excite (Multiply)	(None, 8, 8, 144)	0	block3a_activati… block3a_se_expan…
block3a_project_co… (Conv2D)	(None, 8, 8, 40)	5,760	block3a_se_excit…
block3a_project_bn (BatchNormalizatio…	(None, 8, 8, 40)	160	block3a_project_…
block3b_expand_conv (Conv2D)	(None, 8, 8, 240)	9,600	block3a_project_…

block2b_project_bn (BatchNormalizatio…	(None, 16, 16, 24)	96	block2b_project_…
block2b_drop (Dropout)	(None, 16, 16, 24)	0	block2b_project_…
block2b_add (Add)	(None, 16, 16, 24)	0	block2b_drop[0][… block2a_project_…
block3a_expand_conv (Conv2D)	(None, 16, 16, 144)	3,456	block2b_add[0][0]
block3a_expand_bn (BatchNormalizatio…	(None, 16, 16, 144)	576	block3a_expand_c…
block3a_expand_act… (Activation)	(None, 16, 16, 144)	0	block3a_expand_b…
block3a_dwconv_pad (ZeroPadding2D)	(None, 19, 19, 144)	0	block3a_expand_a…
block3a_dwconv (DepthwiseConv2D)	(None, 8, 8, 144)	3,600	block3a_dwconv_p…
block3a_bn (BatchNormalizatio…	(None, 8, 8, 144)	576	block3a_dwconv[0…

block2b_dwconv (DepthwiseConv2D)	(None, 16, 16, 144)	1,296	block2b_expand_a…
block2b_bn (BatchNormalizatio…	(None, 16, 16, 144)	576	block2b_dwconv[0…
block2b_activation (Activation)	(None, 16, 16, 144)	0	block2b_bn[0][0]
block2b_se_squeeze (GlobalAveragePool…	(None, 144)	0	block2b_activati…
block2b_se_reshape (Reshape)	(None, 1, 1, 144)	0	block2b_se_squee…
block2b_se_reduce (Conv2D)	(None, 1, 1, 6)	870	block2b_se_resha…
block2b_se_expand (Conv2D)	(None, 1, 1, 144)	1,008	block2b_se_reduc…
block2b_se_excite (Multiply)	(None, 16, 16, 144)	0	block2b_activati… block2b_se_expan…
block2b_project_co… (Conv2D)	(None, 16, 16, 24)	3,456	block2b_se_excit…

block2a_se_reshape (Reshape)	(None, 1, 1, 96)	0	block2a_se_squee…
block2a_se_reduce (Conv2D)	(None, 1, 1, 4)	388	block2a_se_resha…
block2a_se_expand (Conv2D)	(None, 1, 1, 96)	480	block2a_se_reduc…
block2a_se_excite (Multiply)	(None, 16, 16, 96)	0	block2a_activati… block2a_se_expan…
block2a_project_co… (Conv2D)	(None, 16, 16, 24)	2,304	block2a_se_excit…
block2a_project_bn (BatchNormalizatio…	(None, 16, 16, 24)	96	block2a_project_…
block2b_expand_conv (Conv2D)	(None, 16, 16, 144)	3,456	block2a_project_…
block2b_expand_bn (BatchNormalizatio…	(None, 16, 16, 144)	576	block2b_expand_c…
block2b_expand_act… (Activation)	(None, 16, 16, 144)	0	block2b_expand_b…

block2a_expand_conv (Conv2D)	(None, 32, 32, 96)	1,536	block1a_project_…
block2a_expand_bn (BatchNormalizatio…	(None, 32, 32, 96)	384	block2a_expand_c…
block2a_expand_act… (Activation)	(None, 32, 32, 96)	0	block2a_expand_b…
block2a_dwconv_pad (ZeroPadding2D)	(None, 33, 33, 96)	0	block2a_expand_a…
block2a_dwconv (DepthwiseConv2D)	(None, 16, 16, 96)	864	block2a_dwconv_p…
block2a_bn (BatchNormalizatio…	(None, 16, 16, 96)	384	block2a_dwconv[0…
block2a_activation (Activation)	(None, 16, 16, 96)	0	block2a_bn[0][0]
block2a_se_squeeze (GlobalAveragePool…	(None, 96)	0	block2a_activati…
block2a_se_reshape (Reshape)	(None, 1, 1, 96)	0	block2a_se_squee…



Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 64, 64, 3)	0	-
rescaling_8 (Rescaling)	(None, 64, 64, 3)	0	input_layer_4[0]…
normalization_4 (Normalization)	(None, 64, 64, 3)	7	rescaling_8[0][0]
rescaling_9 (Rescaling)	(None, 64, 64, 3)	0	normalization_4[…
stem_conv_pad (ZeroPadding2D)	(None, 65, 65, 3)	0	rescaling_9[0][0]
stem_conv (Conv2D)	(None, 32, 32, 32)	864	stem_conv_pad[0]…
stem_bn (BatchNormalizatio…	(None, 32, 32, 32)	128	stem_conv[0][0]
stem_activation (Activation)	(None, 32, 32, 32)	0	stem_bn[0][0]
block1a_dwconv (DepthwiseConv2D)	(None, 32, 32, 32)	288	stem_activation[…
block1a_bn (BatchNormalizatio…	(None, 32, 32, 32)	128	block1a_dwconv[0…
block1a_activation (Activation)	(None, 32, 32, 32)	0	block1a_bn[0][0]
block1a_se_squeeze (GlobalAveragePool…	(None, 32)	0	block1a_activati…
block1a_se_reshape (Reshape)	(None, 1, 1, 32)	0	block1a_se_squee…
block1a_se_reduce (Conv2D)	(None, 1, 1, 8)	264	block1a_se_resha…
block1a_se_expand (Conv2D)	(None, 1, 1, 32)	288	block1a_se_reduc…
block1a_se_excite (Multiply)	(None, 32, 32, 32)	0	block1a_activati… block1a_se_expan…
block1a_project_co… (Conv2D)	(None, 32, 32, 16)	512	block1a_se_excit…
block1a_project_bn (BatchNormalizatio…	(None, 32, 32, 16)	64	block1a_project_…
top_conv (Conv2D)	(None, 2, 2, 1280)	409,600	block7a_project_…
top_bn (BatchNormalizatio…	(None, 2, 2, 1280)	5,120	top_conv[0][0]
top_activation (Activation)	(None, 2, 2, 1280)	0	top_bn[0][0]
flatten_3 (Flatten)	(None, 5120)	0	top_activation[0…
dense_6 (Dense)	(None, 256)	1,310,976	flatten_3[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_6[0][0]
dense_7 (Dense)	(None, 201)	51,657	dropout_3[0][0]

Total params: 5,412,204 (20.65 MB)

Trainable params: 2,144,073 (8.18 MB)

Non-trainable params: 3,268,131 (12.47 MB)

We fine tuned the final 5 layers of the feature extractor. Which left us with:

Total parameters: 5,412,204 (20.65 MB)

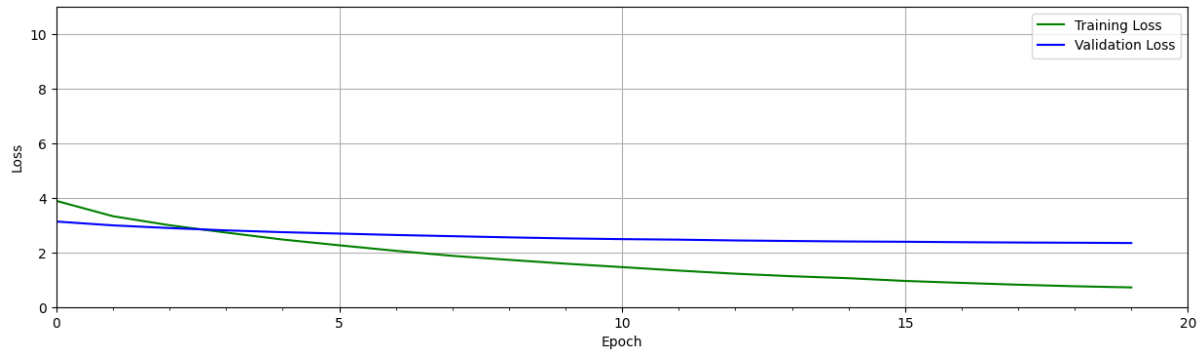
Trainable parameters: 2,144,073 (8.18 MB)

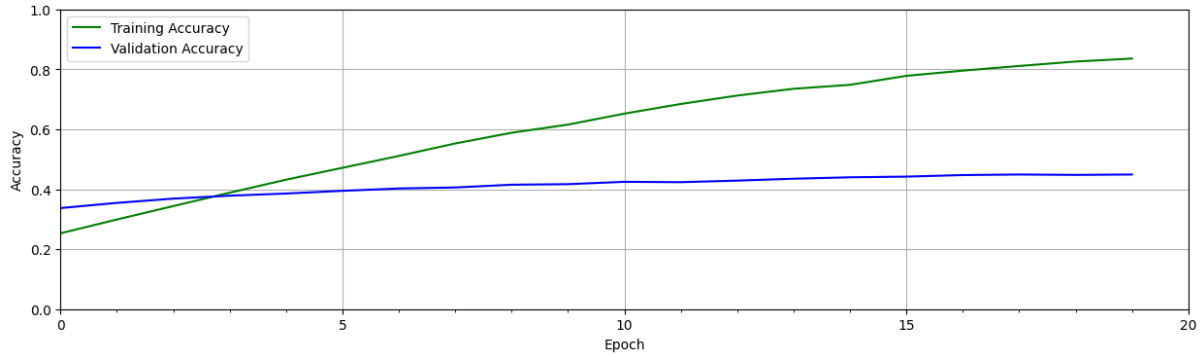
Non-trainable parameters: 3,268,131 (12.47 MB)

We trained the model using the augmented, shuffled dataset, using 20 epochs.

It resulted in a train accuracy of 83.58% and a test accuracy of 11.64%.

```
Epoch 1/20
71/71 [==============================] - 112s 1s/step - loss: 3.8848 - acc: 0.2529 - val_loss: 3.1285 - val_acc: 0.3372
Epoch 2/20
71/71 [==============================] - 90s 1s/step - loss: 3.3235 - acc: 0.2989 - val_loss: 2.9881 - val_acc: 0.3544
Epoch 3/20
71/71 [==============================] - 90s 1s/step - loss: 2.9934 - acc: 0.3437 - val_loss: 2.8904 - val_acc: 0.3686
Epoch 4/20
71/71 [==============================] - 91s 1s/step - loss: 2.7244 - acc: 0.3883 - val_loss: 2.8050 - val_acc: 0.3784
Epoch 5/20
71/71 [==============================] - 99s 1s/step - loss: 2.4702 - acc: 0.4318 - val_loss: 2.7391 - val_acc: 0.3856
Epoch 6/20
71/71 [==============================] - 84s 1s/step - loss: 2.2592 - acc: 0.4714 - val_loss: 2.6885 - val_acc: 0.3946
Epoch 7/20
71/71 [==============================] - 83s 1s/step - loss: 2.0571 - acc: 0.5110 - val_loss: 2.6361 - val_acc: 0.4025
Epoch 8/20
71/71 [==============================] - 81s 1s/step - loss: 1.8732 - acc: 0.5526 - val_loss: 2.5927 - val_acc: 0.4058
Epoch 9/20
71/71 [==============================] - 82s 1s/step - loss: 1.7267 - acc: 0.5883 - val_loss: 2.5493 - val_acc: 0.4150
Epoch 10/20
71/71 [==============================] - 85s 1s/step - loss: 1.5863 - acc: 0.6155 - val_loss: 2.5116 - val_acc: 0.4168
Epoch 11/20
71/71 [==============================] - 83s 1s/step - loss: 1.4601 - acc: 0.6521 - val_loss: 2.4849 - val_acc: 0.4247
Epoch 12/20
71/71 [==============================] - 85s 1s/step - loss: 1.3324 - acc: 0.6843 - val_loss: 2.4657 - val_acc: 0.4235
Epoch 13/20
...
71/71 [==============================] - 80s 1s/step - loss: 0.7599 - acc: 0.8258 - val_loss: 2.3529 - val_acc: 0.4477
Epoch 20/20
71/71 [==============================] - 91s 1s/step - loss: 0.7178 - acc: 0.8358 - val_loss: 2.3420 - val_acc: 0.4491
INFO:tensorflow:Assets written to: efficientnet_WB_aug\assets
```





SUMMARY OF THE PROJECT:

Our project began by carefully examining the dataset, which consisted of 11,788 bird images categorized into 200 different species. We organized the

data into training and testing sets using provided text files. To prepare the images for processing, we resized and normalized them, ensuring consistency across the dataset.

Initially, we developed a custom CNN model but encountered overfitting issues. To address this, we integrated dropout layers, yet our model's performance remained modest due to vanishing gradients. Recognizing the limitations, we turned to pretrained models, opting for EfficientNetB0. By leveraging transfer learning, we fine-tuned the model for our bird classification task. Switching the loss function to categorical crossentropy significantly improved model training. Despite our efforts, test accuracy remained subpar, likely due to the smaller training dataset compared to the pretraining data.

To combat data scarcity, we explored data augmentation techniques, rotating images and shuffling them to diversify the training dataset. However, even with these enhancements, the test accuracy plateaued below 12%. In response, we transitioned to using the InceptionNet model, despite its higher parameter count. This decision proved fruitful, leading to a substantial increase in test accuracy to 25%.

Our project demonstrated a systematic approach to solving a complex problem, encompassing data preprocessing, model development, and experimentation with various architectures and techniques. Despite initial setbacks, our persistence and adaptability ultimately yielded significant improvements in model performance, culminating in a satisfactory test accuracy with the InceptionNet model.

LEARNINGS FROM THE PROJECT:

Throughout the project, we gained several key insights that have deepened our understanding of convolutional neural networks (CNNs) and their application to image classification tasks.

Firstly, we learned the importance of data preprocessing. Understanding the structure of the dataset and properly organizing the training and testing sets is crucial for model performance. Additionally, preprocessing steps such as resizing and normalization ensure consistency and aid in model convergence.

Secondly, we discovered the limitations of custom CNN models, particularly in handling overfitting issues. Integrating dropout layers helped alleviate overfitting to some extent, but we realized the significance of choosing appropriate architectures and regularization techniques to combat this common challenge.

Thirdly, our exploration of pretrained models underscored the power of transfer learning. Leveraging models like EfficientNetB0 allowed us to benefit from their prelearned features, saving computation time and improving performance. Adjusting loss functions and fine-tuning parameters further enhanced model training and adaptation to our specific task.

Moreover, our experience with data augmentation highlighted its importance in mitigating the effects of data scarcity. By augmenting the training dataset with various transformations, we effectively increased its diversity, leading to better generalization performance and improved accuracy.

Finally, our experimentation with different architectures, techniques, and hyperparameters emphasized the iterative nature of deep learning projects. Through continuous refinement and adaptation, we were able to overcome challenges and achieve significant improvements in model performance.

Overall, our project served as a valuable learning experience, providing practical insights into the complexities of training CNNs for image classification tasks and highlighting the importance of experimentation, adaptation, and perseverance in the pursuit of better model performance.

PROBLEMS WE FACED / REASONS WE COULD NOT FURTHER INCREASE THE ACCURACY:

Overfitting: Our custom CNN model initially suffered from overfitting, where it performed well on the training data but poorly on unseen test data. This

problem arose due to the model memorizing the training data rather than learning generalizable patterns.

Limited Dataset: The dataset provided for training the models was relatively small compared to the complexity of the task. This limited dataset size hindered the ability of our models to learn diverse and representative features, leading to suboptimal performance.

Vanishing Gradients: During training, we experienced issues with vanishing gradients, particularly in the custom CNN model. This phenomenon occurs when the gradients become extremely small, causing the model parameters to update very slowly or not at all, hindering learning.

Data Augmentation Challenges: While data augmentation helped increase the diversity of our training dataset, it did not fully address the limitations of the original dataset. Additionally, determining the appropriate augmentation techniques and parameters required experimentation and fine-tuning.

Loss Function Selection: Initially using the wrong loss function, such as "binary_crossentropy" instead of "categorical_crossentropy," negatively impacted model training and performance. Choosing the correct loss function is critical for guiding the optimization process towards the desired outcome.

Model Complexity and Resource Constraints: As we explored more advanced pretrained models like EfficientNet and InceptionNet, we encountered challenges related to model complexity and resource constraints. These models required more computational resources for training and inference, which may not have been readily available.

SUMMARY:

Model developement

We tried using efficientnetB0 as pretrained model. On top of it we added 1 flat layer and two dense layer to get final output. To avoid overfitting, we tried to use dropout, data augmentation and random shuffling. We used SoftMax at the end to get probability profile. We tried both RMSprop and Adam optimizer. We found that Adam is performing better in training as well as testing. For fine-tuning, we did unfreeze last few layers of efficientnet. First training happened with 20 epochs with 20sec for each epoch. Total training time found to 40s on my device. (Training time can vary device to device)

In conclusion, we firstly tried to build our own CNN model test accuracy of which was around 4%.

After that, we tried to explore few pretrained models like VGG, inceptionnet and efficientnet.

Accuracy of inceptionnet was found to be around 22% but number of parameters were around 24M. Hence, we rejected that model.

Our final submission includes trained model using efficient net having 9.1M parameters with test accuracy 11.6%

*We are attaching zip file of all models which we tried.

AKNOWLEDGEMENT:

We wanted to express our heartfelt gratitude for assigning us the project as part of our course GNR638: Deep Learning. This project has been an invaluable learning experience, allowing us to delve deeper into the world of deep learning and apply our knowledge in a practical setting.

We are truly grateful for the opportunity to work on this project under your mentorship. Thank you for your encouragement, patience, and dedication to our learning journey. We look forward to applying the skills and knowledge gained from this project in our future endeavours.

With sincere appreciation,

Stuti Singla(22B0046)
Sarvesh Patil(22B2276)