Homework 1

Step1: Data Preprocessing

At first we started with a simple CNN model designed from scratch. We tried two different techniques for data preprocessing, range normalization and standardization.In standardization the trained data statistics applied to the validation data. As shown in the table the performance on validation data is better with standardization.

	Validation Accuracy	Validation Loss
Without Preprocessing	0.4965	1.2948
Normalization	0.5910	1.0797
Standardization	0.6121	1.0331

Step2: Data Augmentation

Due to the limited amount of data, the model overfits easily. To solve this problem we tried different parameters for data augmentation to increase the amount of data. As figure 1 indicates the overfitting data augmentation reduces the overfitting and increases the validation accuracy.
Augmentation params: width_shift_range = 35, height_shift_range = 35, horizontal_flip = True, vertical_flip = True, fill_mode = 'reflect'

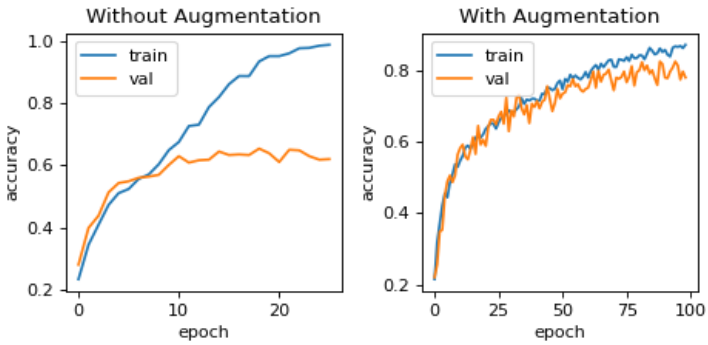

Figure 1: Data Augmentation Effect

Step3: Global Average Pooling instead of Flatten

As it would decrease the number of parameters vastly, it prevents overfitting. Moreover, it would increase robustness to spatial transformation. We checked this on our scratch model, the results are shown in the table. As it shows the model with GAP has much less parameters and it has a better accuracy.

	Validation Accuracy	Validation Loss	Trainable Params
Flatten	0.7707	0.6472	10,355,528
Global Average Pooling	0.8271	0.4895	1,180,488

Step4: Transfer Learning

As we have limited amount of data to learn good filters to extract good features, we decided to use the feature extracting part (conv and pooling layers) of pretrained models. At this step the weights of pretrained models were freeze.

For preprocessing data in this step, instead of standardization the specific preprocess function for each pretrained model has been used.

Different architectures for the fully connected layers have been tested. Also, different pretrained models have been tried. we tried efficientnets because they have small number of parameters, VGG16 because it had few layers and ResNets because they are powerful to the shortcut connections.

We trained our networks three times , every time with lower learning rate than previous one .

Table 3 shows the performance using different pretrained models. As VGG16 didn't do well we didn't use that in the next steps.

	Validation Accuracy	Validation Loss
Vgg16	0.6748	0.9072
ResNet50	0.7726	0.6292
ResNet152	0.7838	0.6108
EfficientNetB0	0.7368	0.7382
EfficientNetB1	0.7399	0.7462
EfficientNetB2	0.7928	0.7331

Step5: Fine Tuning

To customize the feature extracting part of the network we unfreeze conv layers in different manners to learn to extract features more specific to our problem.

Because we didn't want to change the ImageNet weights a lot we chose lower learning rates than transfer learning step, and we did the all the trainings three times with three learning rates, bigger learning rates at first so our networks don't get stuck in local Minimas and then training lower learning rates so our networks converge to global minima.

Finetuning ResNet: first we trained a small number of last layers and we saw increase in the results. Then we tried fine tuning bigger number of layers but it didn't led to any obvious improvements of validation accuracy, which in our opinion it was because number of trainable parameters was huge.

Then, starting from the last layers we unfroze about 5 million parameters at a time, and froze other layers of the ResNet. After training, froze those layers and unfroze the shallower layers of the model until half of the layers. By this method we improved the results more and reached validation accuracy of 87% with Resnet152, and 84 % with ResNet50.

Finetuning EfficientNet: At start we fine tune small number of last layers in efficientnets, then tried fine tuning bigger number of layers as an experiment, which led to better results, we continued to increase the number of unfroze layers until we tried fine tuning each of these efficientnets completely. As number of parameters in efficientsnets were a lot smaller than Resnets it worked well.

Model	Validation Accuracy
resnet50	0.84
resnet152	0.87
EfficientB1	0.89
EfficientB2	0.91

Step6: Ensemble Model

As the results with efficientnetB1 and efficientnetB2 was more satisfying we ensembled them to make our Final model.

We ensembled our two fine tuned efficientnet base models (B1 and B2) by concatenating their output layer(prediction) followed by a 8 neuron dense layer. The two models' weights are frozen.

The ensembled model (our final model) improved by 2-3 percent in the local validation accuracy and reached 94% locally also 88.4 on final test dataset .

The model has been plotted in figure 2.

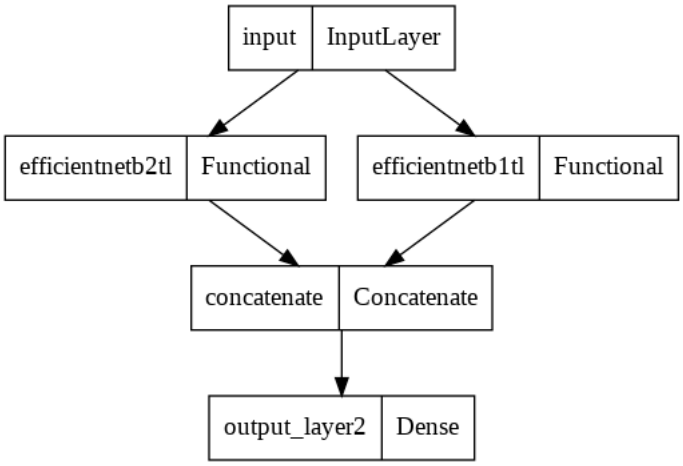

Figure 2: Ensemble Model