Homework1

In all steps validation set is 10% of the entire data.

Step 1: We tried only dense layers for classification that faced underfitting, and the ACC (accuracy) reached 32.16% for validation data. After this, we put away pure dense layers for prediction and we focused on CNN layers.

CNN layers worked a bit well on this dataset. With the pure multi-CNN layers, we got about 30% ACC on the test set, then we improved the idea and used the Batch Normalization layer after each CNN and it worked and we raised ACC to 34%.

Step 2: The dataset is imbalanced and due to this the validation set will be imbalanced when we use splitting in Imagedatagenerator. We used a simple counter on the validation set and printed the number of samples in the validation set per class.

Counter ➔ Apple: 98, Blueberry: 46, Cherry: 58, Corn: 120, Grape: 145, Orange: 174, Peach: 97, Pepper: 76, Potato: 71, Raspberry: 26, Soybean: 161, Squash: 57, Strawberry: 67, Tomato: 569.

Because the imbalanced dataset could affect results and due to this in the next steps, we tried to use random noise layers and data augmentation and also TL (transfer learning).

For a complex Model, we tried data augmentation with several random parameters. After training several models with different arguments for data generation we reached 57% ACC. As a result of Chart1, only in 2 classes augmentation didn't work well in comparison to Pure CNN.

Augmentation params: rescale=1/255, rotation_range=30, shear_range = 0.2, height_shift_range=30, width_shift_range=30, zoom_range=0.1, horizontal_flip=True, vertical_flip=True, fill_mode='nearest'.

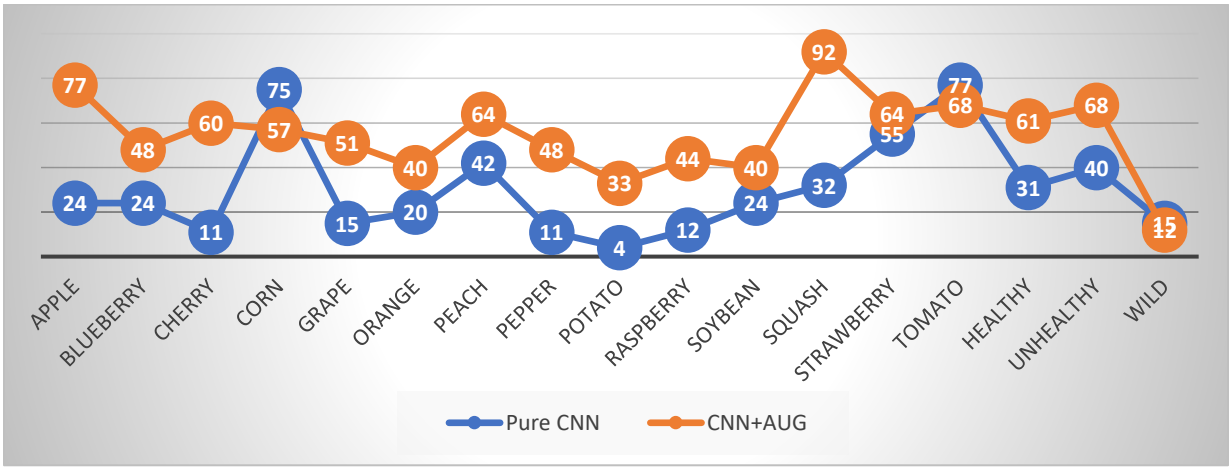

Chart 1: Comparing the pure CNN model with CNN + Augmentation.

Step 3: Unfortunately using noise layer (especially Gaussian Noise) didn't help us and on validation set we couldn't get good results.

The pure TL on VGG16 gave us 41% ACC. In some cases (Corn, Orange, and Tomato) the results are better than the augmentation model, then using TL as well as augmentation should be good.

VGG16 with augmentation had 54% ACC that is better than pure usage of VGG16 but this is 3% less than pure augmentation. Then the next idea is that to use other TL networks such as Resnet50, Resnet152, MobileNet, EfficientNet (B0 to B7), and Xception.

Step 4: In this step, we used the Preprocess_input function (this is a function that there is in each TL model library) for preprocessing the inputs. After several trains with different parameters and classifiers (Table 1) finally, the ACC reached 86.2% with Resnet152.
For learning rate, we tested different values and finally we find that 0.001 works better.

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 256, 256, 3)]	0
resnet152 (Functional)	(None, 8, 8, 2048)	58370944
global_max_pooling2d (Global	(None, 2048)	0
Flattening (Flatten)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
reLU (ReLU)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
Output (Dense)	(None, 14)	7182

Table 1: The classifier that we used in TL models.

Model	ACC
ResNet 152	86.2
ResNet 152 V2----first try	77.5
ResNet 152 V2----second try	80.7
ResNet 152 V2----third try with dense 1024	83.2

EfficientNet B4---- first try	83.7
EfficientNet B4---- second try with dense 1024	83.5
EfficientNet B7	78.6
MobileNet	81.5
Xception	78.3

Table 2: ACC for different types of TL models in different runs with variate parameters. (These results are a summary of trains that we get good results and we didn't represent all results here)

According to table 2, it seems the best TL models for this dataset are Resnet152 and EfficientNet B4, and with these models, we could get the highest results on test data.

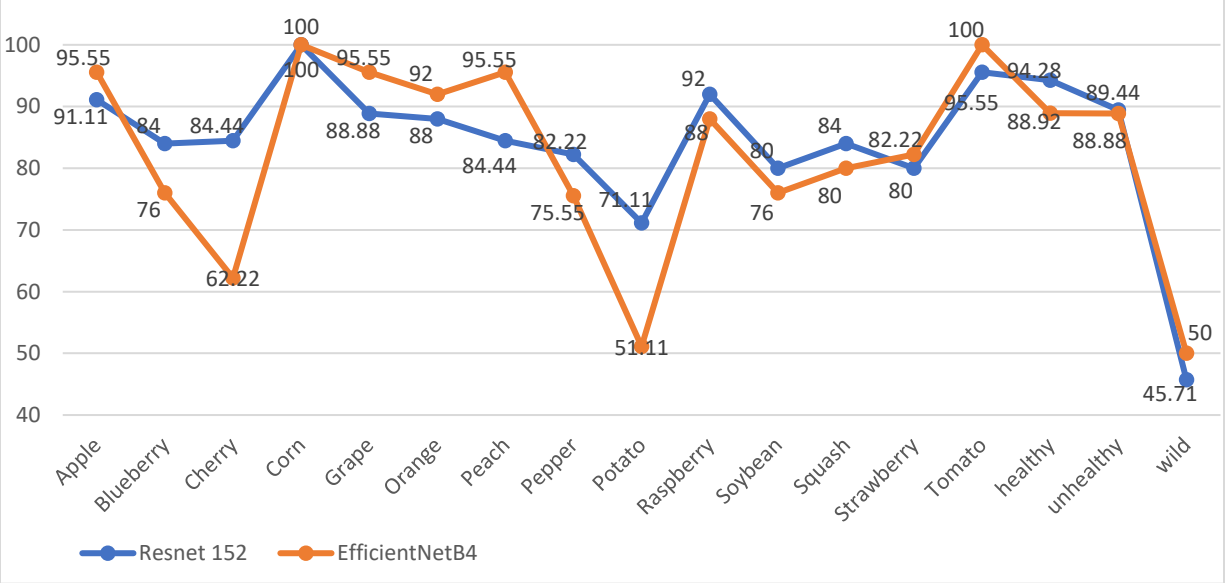

Chart 1: comparing the Resnet 152 and EfficientNetB4 result.

Final phase:

In the final phase, both EfficientNetB4 and Resnet152 worked on this dataset and we decided to submit the Resnet152 model because in the development phase Resnet152 had relatively the best results. It is obvious that changing the hyperparameters could affect results and because of limitations in time and GPU, we couldn't try variate hyperparameters. For these models we got 84.71% for EfficientNetB4 and 83.96% for Resnet152 ACC.