● *Image Classification*

● *K-NN*

Overview: Used a pretrained ResNet-18 model to build a new model which extracts features from the 'flatten' layer of the base ResNet model. Using this new model extracted feature vectors of the provided training dataset and test dataset images. Using the train data feature vectors trained KNeighborsClassifier(this classifier just remembers the training data) of sklearn library. Used this classifier and predicted the labels of test data feature vectors, which gave an **accuracy** of **more than 95%** for various values of k.

Implementation Details:

- *Data Pre-Processing:*

  In data pre-processing step, following transformations were applied on train data-

  i)`RandomResizedCrop(224)` - Crop a random portion of image resize it to a given size,i.e 224*224

  ii)`RandomHorizontalFlip` - Horizontally flip the given image randomly with a given probability.(default - 0.5)

  iii) `Normalise` - `Normalised the image with mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225].Above value are used as they are the mean and std of imagenet dataset on which ResNet-18 model is trained.`

- ***Dataset- Preparation:***
- Created a class, ***'ClassificationDataset'*** for custom dataset, (this can also be done using datasets.ImageFolder) and

implemented __init__, __len__ and __getitem__ functions in it.

- ***Model- Building:***
- Created a class, ***'ResNet18FeatureExtractor'*** which is a subclass of torch.nn.Module and implemented __init__ and forward function.
- In __init__ function, created a model from base model ResNet18 with pretrained weights and by using '*create_feature_extractor'* functions extracted output from *'flatten'* layer of ResNet-18 model.

- ***Feature- Extraction:***
  Extracted features from both train and test images by running the '*ResNet18FeatureExtractor'* in eval mode. These features are stored along with their labels for training and testing of KNN model.

- ***KNN Model:***
  Used the features extracted from train dataset images to fit to **KNeighborsClassifier** of sklearn library and predicted labels of test dataset images using this model.

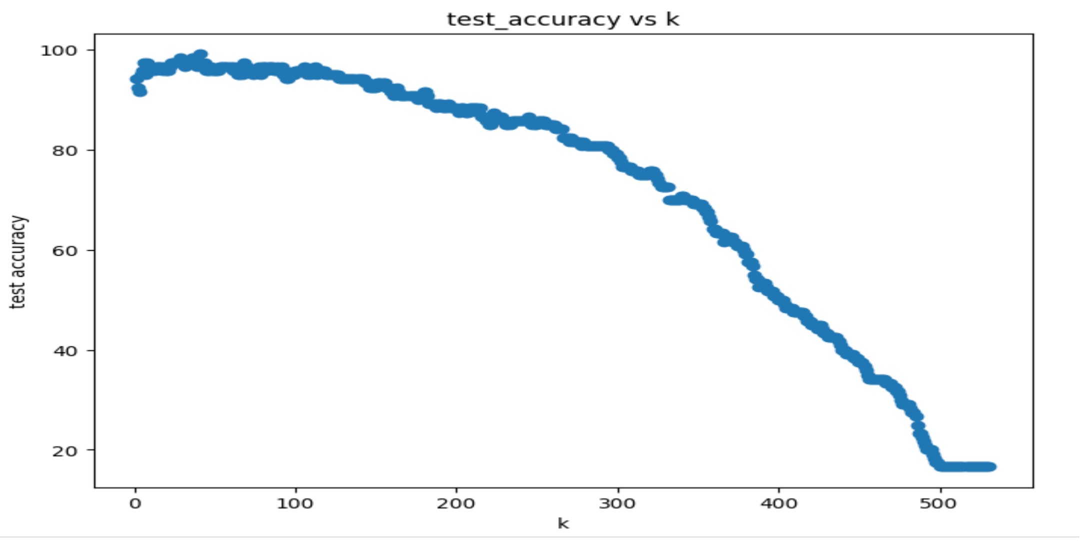
- ***Model_Testing:***
- Obtained a **test accuracy** of **96%** (For k = 10) and maximum accuracy of **97.5%** (for k around 40)

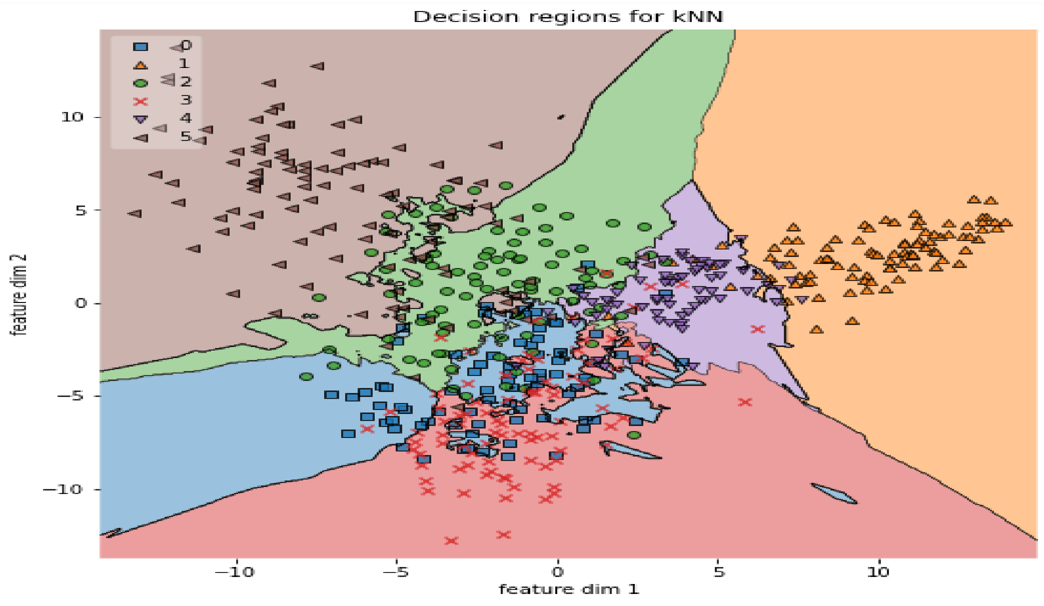
Observations/Results:

- Experimented with KNN Classifier by changing values of k (number of nearest neighbours), initially test accuracy increased and then after some point it started decreasing and reached a minimum accuracy of 16.66% when k was set to 532.

- **Possible Reason**: When k is larger, the distance is then larger, which defeats the principle behind kNN - that neighbours that are nearer have similar densities or classes.

- Reduced the dimension of the feature vectors to 2 using PCA to plot the decision boundary of each class. Below is the plot for k = 10 (Below image is different from the python notebook as it changes by rerunning)





● *Fine tuning*

Overview: Used a pretrained ResNet-18 model to build a new model by removing the classifier layer of ResNet model and adding a new classifier layer with 6 outputs. Freezed all parameters of the network except the weights of the newly added classifier layer and trained this network on the given data by tweaking various hyper-parameters and tested the model on the given test data.

Implementation Details:

- ***Data Pre-Processing:***

  In data pre-processing step, following transformations were applied on train data-

i)`RandomResizedCrop(224)`	-	Crop a random portion of image resize it to a given size,i.e 224*224
ii)`RandomHorizontalFlip`	-	Horizontally flip the given image randomly with a given probability.(default - 0.5)
iii) `Normalise`	-	`Normalised the image with mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225].Above value are used as they are the mean and std of imagenet dataset on which ResNet-18 model is trained.`

- ***Dataset- Preparation:***
- Split the training data into 2 sets, train and validation using ***'split-folder'*** library of python(this can also be done using ***'torch.utils.data.random_split'*** function of pytorch).
- ***Problem Faced***: splitfolders function splits and creates a new folder in the given directory, this consumes extra memory on the disk. To resolve this problem, I wrote a cleanup code, which deletes the directory at the end. A better solution is to use *'torch.utils.data.random_split'*

function of pytorch but this does not allow us to use different transforms for train and validation sets.

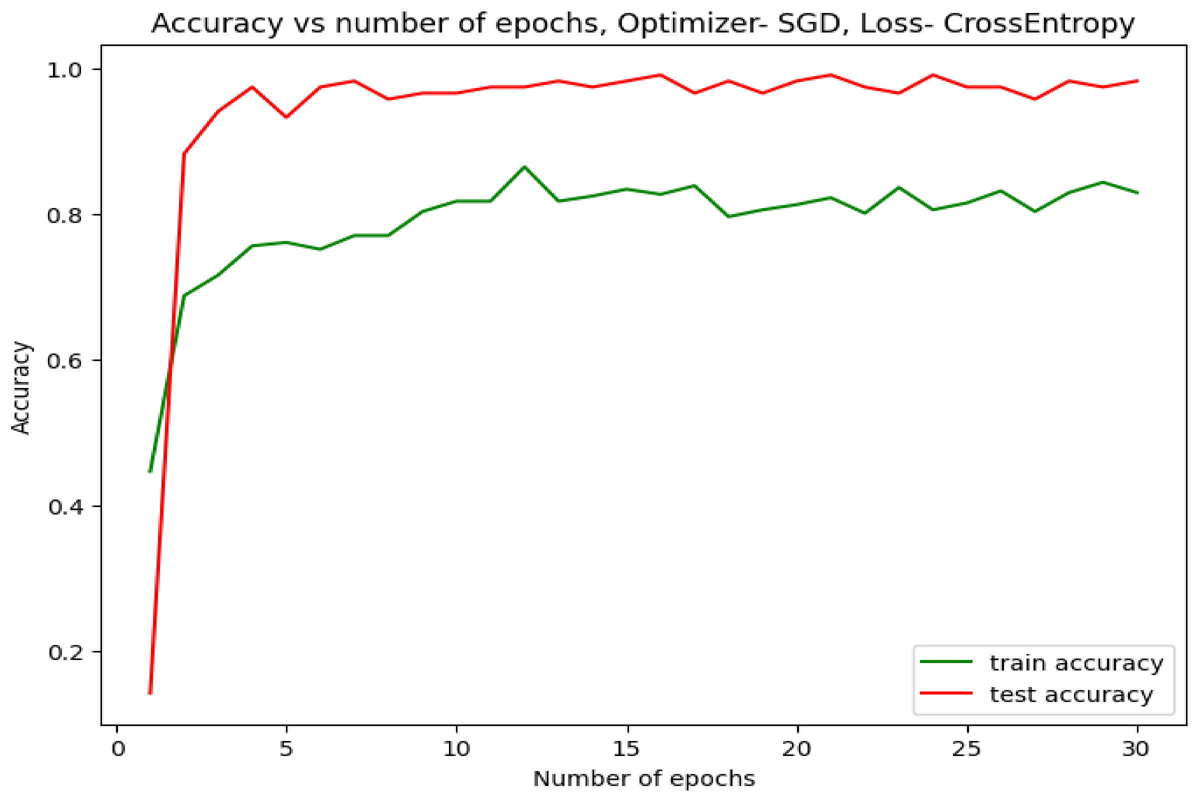
- ***Model- Building:***
- Created a class, ***'ResNet18FineTuned'*** which is a subclass of torch.nn.Module and implemented __init__ and forward function.
- In __init__ function, created a model from base model ResNet18 with pretrained weights.
- Then to freeze the parameters(from changing during training) ***'requires_grad'*** attribute of all parameters were set to ***false***. And then changed the last fully connected layer to give num_classes(6 in this case) outputs.
- Note that ***'requires_grad'*** was ***not*** set to **false** for the weights of this **fc layer**.

- ***Model Training:***
- Created a function, ***'train_model'*** which takes model, scheduler, dataloaders, criterion, optimizer, num_epochs as input.
- In the train_model function epoch runs in 2 phases, 'train' phase and 'val' phase.
- In train phase torch.set_grad_enable is set to true, so that grads are updated.
- And in val phase, best_model_weights is updated when the epoch accuracy is better than best_accuracy, and these best_model_weights are later loaded into the model.

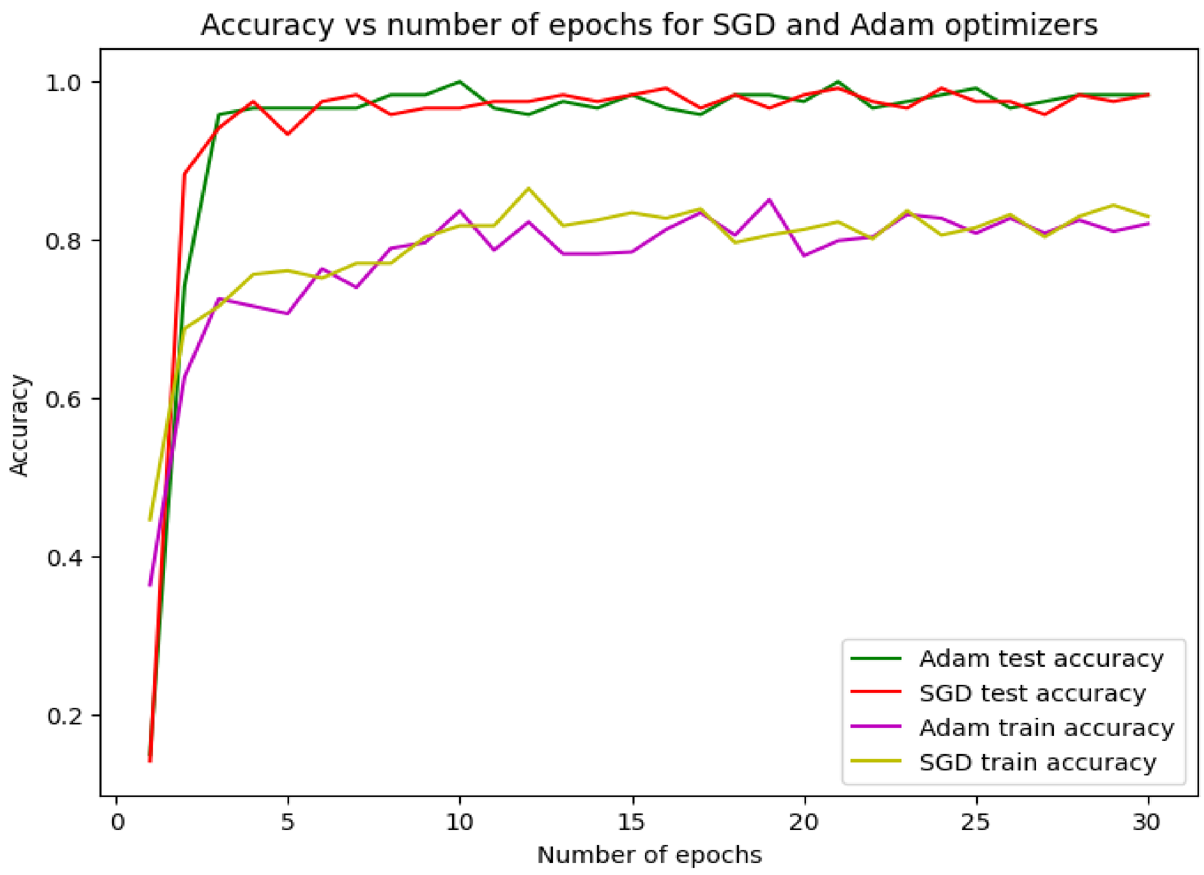
- ***Model_Testing:***
- Created a function, ***'test_model'*** which runs the model in eval mode and returns the test accuracy.
- Obtained a **test accuracy** of **99.16%** (used **cross entropy loss, SGD optimizer** and **25 epochs** for training)

Observations:

- Plotted test accuracy and train accuracy by varying number of epochs, observed that both test and train accuracies are increasing with number of epochs.
- Also observed that test accuracy is higher than train accuracy, this is not normal..
- **Possible Reason**: Generally this is a sign of underfitting, but in this case as we are using pretrained ResNet-18 model and training only the last layer, the weights are pretty good. But in model.train mode due to dropout and some other factors some of these weights are dropped, this causes the train accuracy to go down. In the eval mode all the weights are used which make test accuracy very high.



- Trained the model by using SGD optimizer and Adam optimizer. Did not observe any significant changes.



- ## *Simple CNN:*

  Overview: Built a convolution neural network from scratch(with 2 convolutional layers and 1 fully connected layer) and trained it on the given training set. Test this trained model and got an accuracy of about **22**% which is much lower than the pretrained model.

Implementation Details:

- ***Data Pre-Processing:***
  In data pre-processing step, following transformations were applied on train data-

i)`RandomResizedCrop(224)`	-	Crop a random portion of image resize it to a given size,i.e 224*224
ii)`RandomHorizontalFlip`	-	Horizontally flip the given image randomly with a given probability.(default - 0.5)
iii) `Normalise`	-	`Normalize the image with mean and std of the given data.Above Values are used as the model is Built from scratch and trained on the given training data, but the mean and std of this data are close to that of imagenet dataset.`

- ***Dataset- Preparation:***
- Split the training data into 2 sets, train and validation using '*split-folder*' library of python(this can also be done using '*torch.utils.data.random_split*' function of pytorch).
- **Problem Faced**: splitfolders function splits and creates a new folder in the given directory, this consumes extra

memory on the disk and also consumes a lot of time. To resolve this problem, I wrote a cleanup code, which deletes the directory at the end. A better solution is to use '*torch.utils.data.random_split*' function of pytorch but this does not allow us to use different transforms for train and validation sets.

- ***Model- Building:***
- Created a class, ***'ConvModel'*** which is a subclass of torch.nn.Module and implemented __init__ and forward function.
- In __init__ function, created a CNN from scratch which has 2 convolutional layers followed by 1 fully connected layer.
  Used ReLu activation function.
- In forward function, define the steps to be followed during forward pass.
- **Model:**

```
 (layer1): Sequential(
           Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
           BatchNorm2d(32, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
           ReLU()
           MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
 )

 (layer2):Sequential(
           Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
           BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          ReLU()
          MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
 )
```

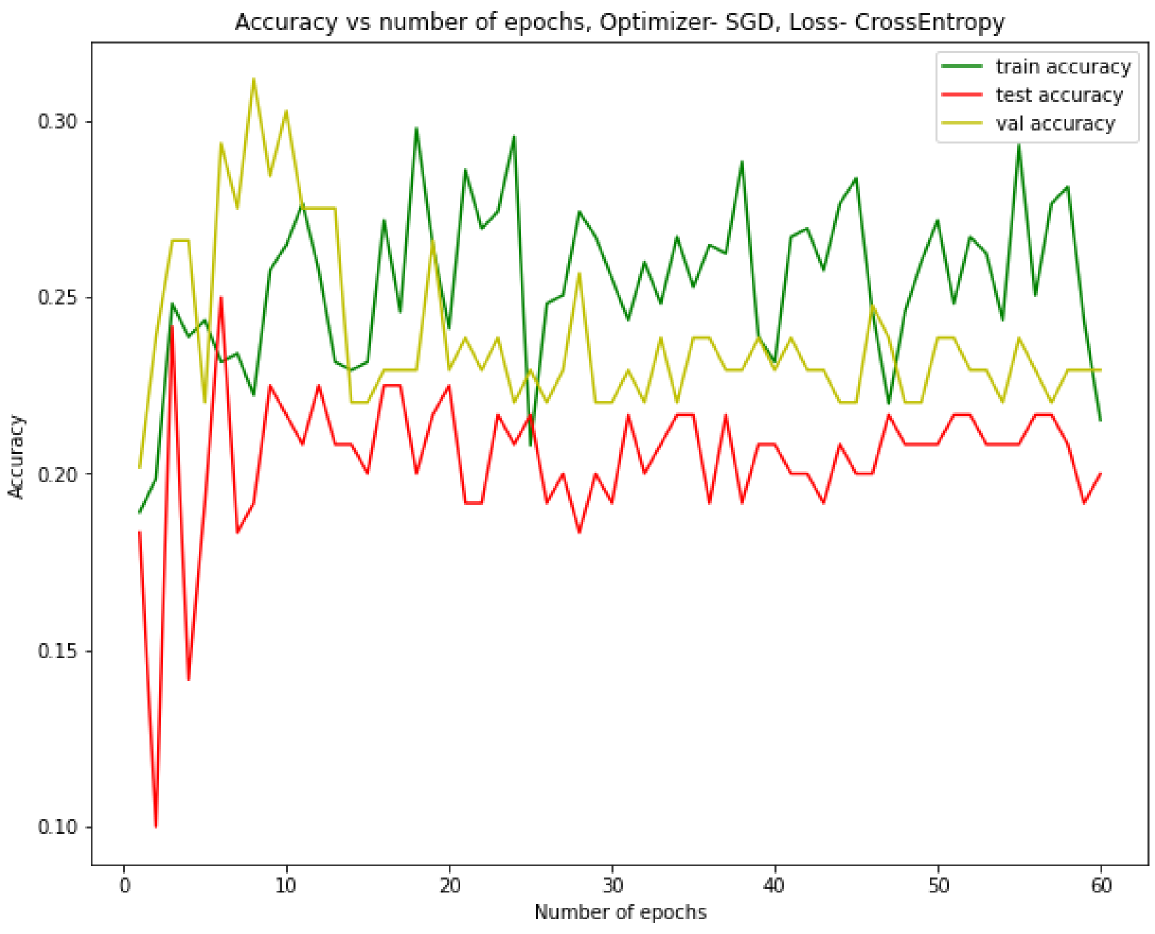
```
  (fc1): Sequential(
            Linear(in_features=200704,  out_features=6,
bias=True)
  )
)
```

- ***Model Training:***
- Created a function, ‘train_model’ which takes model, scheduler, dataloaders, criterion, optimizer, num_epochs as Input.
- In the train_model function epoch runs in 2 phases, ‘train’ phase and ‘val’ phase. In train phase torch.set_grad_enable Is set to true, so that grads are updated.
- And in val phase, best_model_weights is updated when the epoch accuracy is better than best_accuracy, and these best_model_weights are later loaded into the model.

- ***Model_Testing:***
- Created a function, ‘test_model’ which runs the model in eval mode and returns the test accuracy.
- Obtained a **test accuracy** of **22.0%** (used cross entropy loss, SGD optimizer and 25 epochs for training)

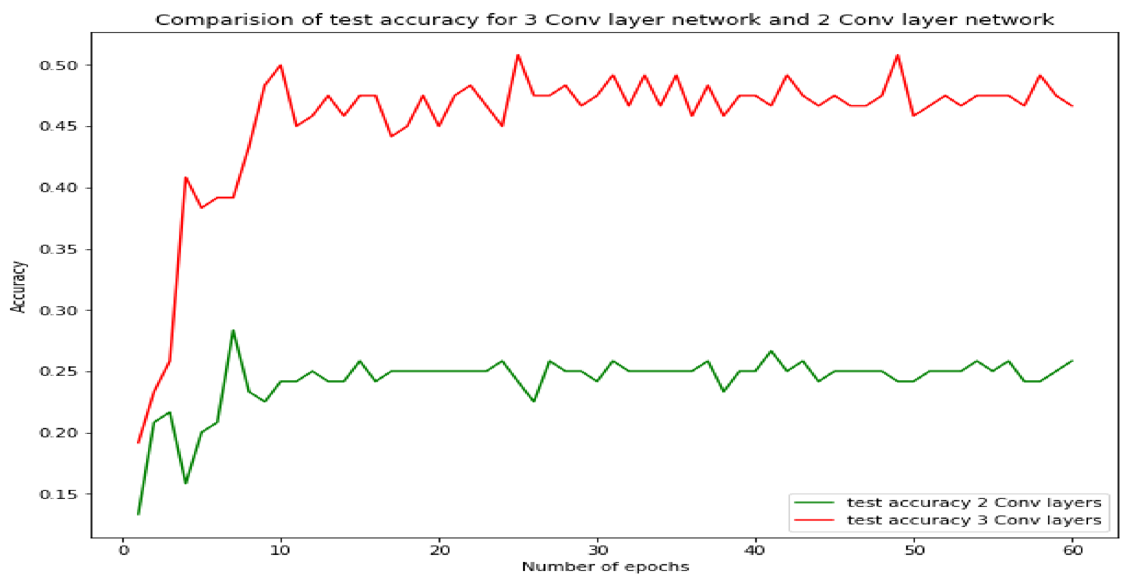
Observations/Results:

- Plotted train, test and validation accuracy for different epochs, as expected train accuracy> val_acuracy~test accuracy.
- But this kept changing with multiple trials, not sure of the reason for this.(Below plot is different from the plot in jupyter notebook)



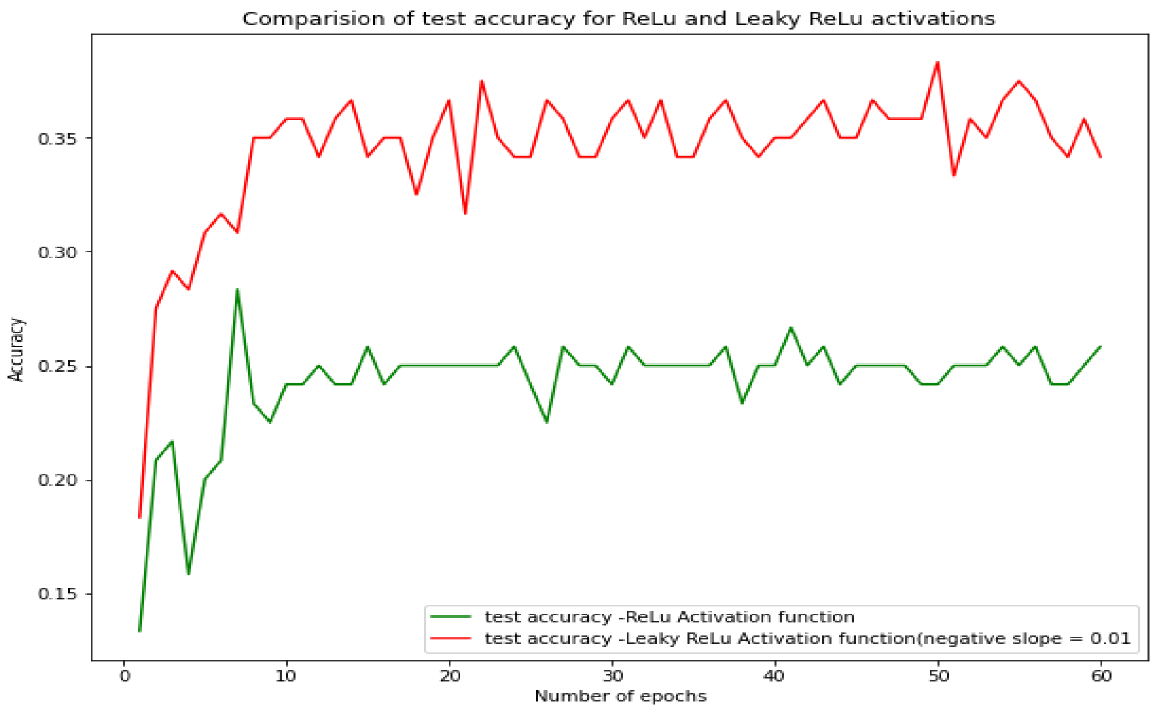
- Comparing models by **increasing number of conv layers**:

Increased the number of convolutional layers by 1(i.e total **3 conv**



layers), this increased the test accuracy from around **22% to 45%.**

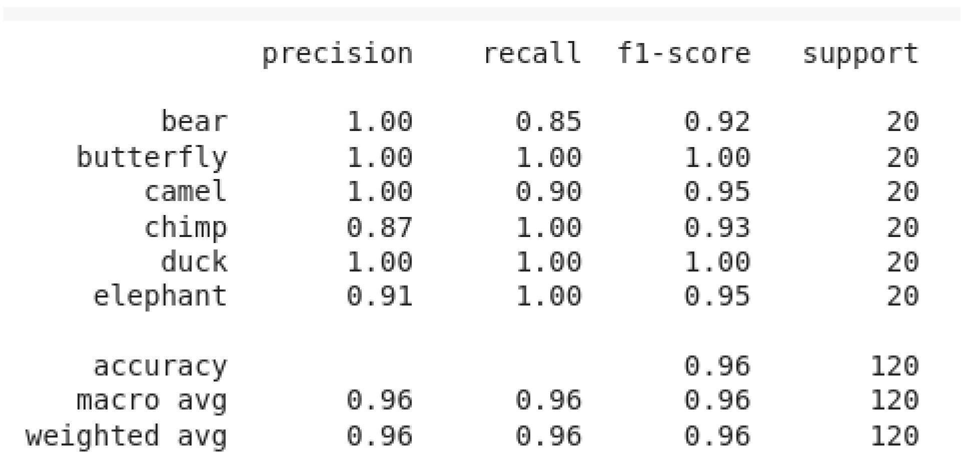
- **Comparing test accuracies by changing activation functions(ReLu and Leaky ReLu):**
  As expected leaky ReLu(negative slope- 0.01) performed better than ReLu and there is about 10% increase in test accuracy



- ***Comparison between KNN, Fine-Tuning and Simple CNN:***

Classification Reports for each of the three cases:

a) **KNN: For k =10**

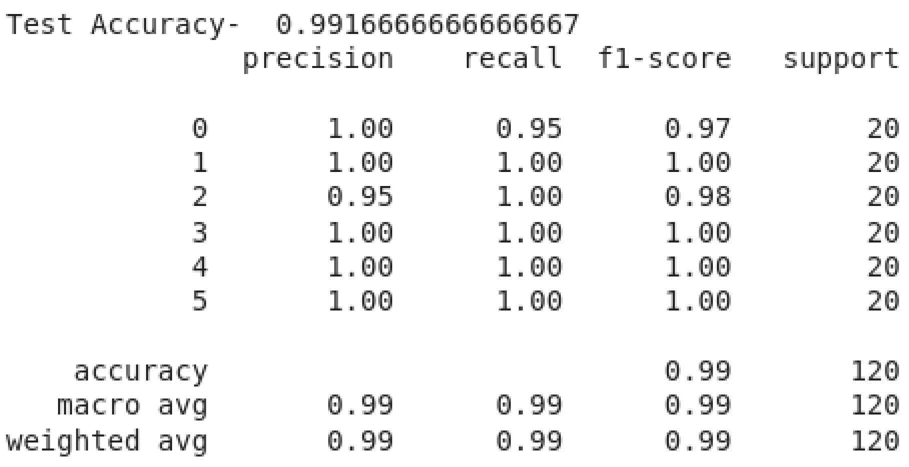


```
              precision    recall  f1-score   support

        bear       1.00      0.85      0.92        20
   butterfly       1.00      1.00      1.00        20
       camel       1.00      0.90      0.95        20
       chimp       0.87      1.00      0.93        20
        duck       1.00      1.00      1.00        20
    elephant       0.91      1.00      0.95        20

    accuracy                           0.96       120
   macro avg       0.96      0.96      0.96       120
weighted avg       0.96      0.96      0.96       120
```

b) **Fine-Tuned ResNet-18:**

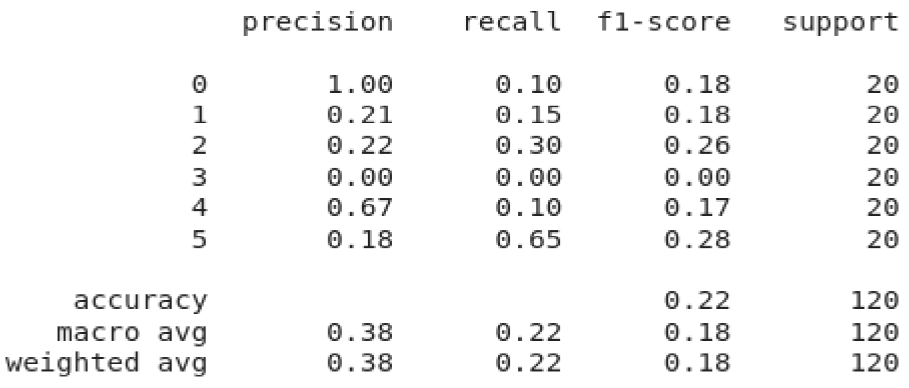


```
Test Accuracy-  0.9916666666666667
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        20
           1       1.00      1.00      1.00        20
           2       0.95      1.00      0.98        20
           3       1.00      1.00      1.00        20
           4       1.00      1.00      1.00        20
           5       1.00      1.00      1.00        20

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

c) **Simple CNN:**



```
              precision    recall  f1-score   support

           0       1.00      0.10      0.18        20
           1       0.21      0.15      0.18        20
           2       0.22      0.30      0.26        20
           3       0.00      0.00      0.00        20
           4       0.67      0.10      0.17        20
           5       0.18      0.65      0.28        20

    accuracy                           0.22       120
   macro avg       0.38      0.22      0.18       120
weighted avg       0.38      0.22      0.18       120
```

- From the above three classification reports, it is clear that KNN (features extracted from pretrained ResNet-18) and Fine-Tuned ResNet-18 are performing much better than a simple CNN built from scratch.

- This is to be expected as ResNet-18 is trained on a huge amount of data compared to our training dataset.

- Between KNN and Fine-Tuned models there is not much difference, KNN accuracy reached to a maximum of 97.5% by varying the values of K(around k=40 maximum is observed).

- **References:**
- **https://pytorch.org/tutorials/** (Most of the code snippets used are from these tutorials)