AIP ASSIGNMENT - 1

Name:K.Kalyan Reddy
SR.No: 21361

2) Image Classification
A)Feature Extraction + KNN

I have implemented this in two .ipynb notebooks 1)Feature Extraction 2)KNN

I have saved the train features in 1st notebook and used them in the second notebook using np.save() and np.load()

This is implemented with reference to the tutorial 1 of AIP , some code used might resemble the code used in the tutorial.

Step 1:
Data transform and Loading the data

For training , the images are cropped to a random portion of image and resize it to a given size, i.e, 224 using `RandomResizedCrop(224)`

Then, `RandomHorizontalFlip()` to horizontally flip the given image randomly with a given probability(default=0.5)

This is done to ensure our model to be robust for any kind of Images (flipped ,cropped etc)

Then, `transforms.ToTensor()` to convert a Image or nd.array to tensor

Then, `transforms.Normalize()` to normalize the data with Imagenet data set mean and standard deviation

For testing, the images are resized to 256, center-cropped to 224 and normalized.

The train and test data are loaded using data_loader(Reference: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

Step 2:
Extracting features and saving it for KNN.ipynb

The images were loaded into the model and the features have been extracted from the last (flatten) layer , where each feature is a 512 dimensional vector .

The batch_size was set to 532 while loading data ,so that all the training images are used in one batch, and I did get a 532*512 train_feature tensor.

I also tried with batch_size =4 which returned a 4*512 train_feature tensor and I had to loop in for all the batches. Either way it is the same.

The train_features and train_labels have been saved for KNN.ipynb. Saved the model with torch.save() for using it in KNN.ipynb

The train_data features were(changing slightly every run time):

```
tensor([[0.9129, 0.6870, 1.3454,  ..., 0.1506, 1.1473, 0.5071],
        [1.7556, 0.5224, 0.3404,  ..., 0.0446, 0.7063, 0.1742],
        [0.0610, 1.7704, 1.2495,  ..., 0.0710, 0.9178, 0.5716],
        ...,
        [0.1395, 0.4687, 0.9779,  ..., 0.0488, 0.4937, 0.0170],
        [2.3035, 0.1009, 0.1535,  ..., 0.3850, 0.7634, 0.0000],
        [0.7542, 0.1819, 0.4553,  ..., 0.3978, 2.1255, 0.8106]],
       device='cuda:0')
```

Step 3: KNN

In this notebook, the test features are extracted for test images and implemented KNN.

Loaded the model saved in feature_extraction.ipynb with torch.load()

I have used KNNClassifier() from sklearn (Reference: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

I tried to get test features in the previous notebook itself, but cuda ran out of memory after extracting train_features. Hence I extracted the test images features in this notebook with the model loaded.

Results :

n_neighbours=k(euclidean distance)	Accuracy
1	90%
3	91.66%
5	95%
7	92.5%
9	91.66%

distance_metric(For k=5)	Accuracy
cityblock	89.16%
manhattan	89.16%
euclidean	95%
cosine	94.16%

KNN with euclidean distance for k=5 had obtained highest accuracy of 95% while with cosine similarity the accuracy was almost similar, 94.16%.

The above accuracies are varying a little but during runtime(don't know exactly why), each time I run the code the accuracies are varying by one or two percent. Probably this is because the feature vectors are changing slightly every time I run it.

Problems faced:

GPU CUDA memory ran out during runtime when I try to make it in a single file, hence did it in 2 files.
I figured that,this is because I had made batch size = 532 instead of smaller batches.

Observations:

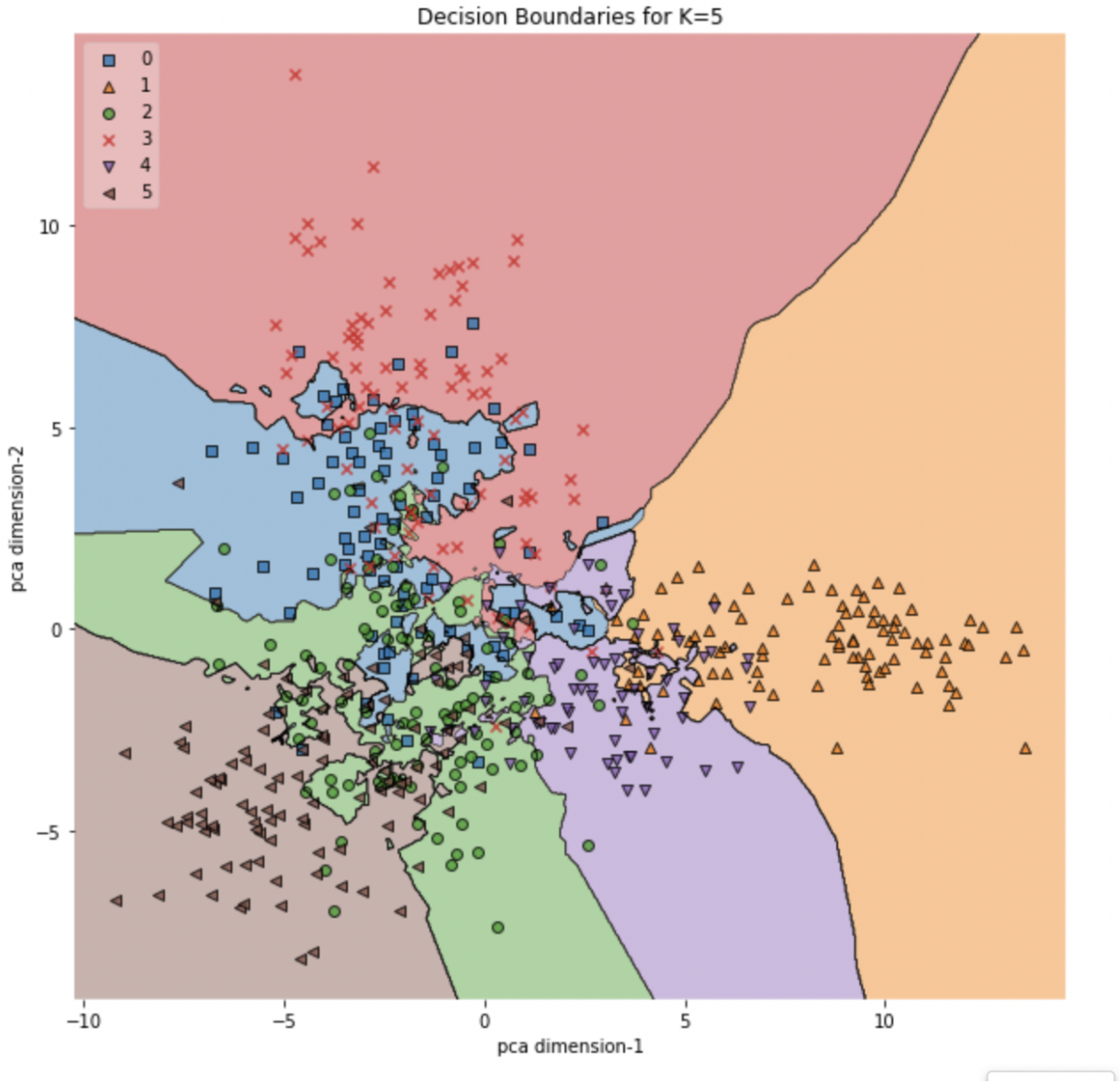
Classification Report:

```
Test Accuracy:
0.95
----------------------------
Classification Report:
              precision    recall  f1-score   support

        bear       0.95      1.00      0.98        20
   butterfly       1.00      1.00      1.00        20
       camel       1.00      0.90      0.95        20
       chimp       1.00      1.00      1.00        20
        duck       1.00      0.80      0.89        20
    elephant       0.80      1.00      0.89        20

    accuracy                           0.95       120
   macro avg       0.96      0.95      0.95       120
weighted avg       0.96      0.95      0.95       120

----------------------------
Confusion Matrix:
[[20  0  0  0  0  0]
 [ 0 20  0  0  0  0]
 [ 0  0 18  0  0  2]
 [ 0  0  0 20  0  0]
 [ 1  0  0  0 16  3]
 [ 0  0  0  0  0 20]]
```

Decision Boundaries :

PCA has been used to reduce the 512 dimensional feature vector into 2 dimensions for visualizing decision boundaries.
(Reference:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

The below figure is for k=5, Euclidean distance(After PCA):



AIP ASSIGNMENT - 1

Name: **K.Kalyan Reddy**
SR.No: **21361**

2)B) Transfer learning with Resnet18 Model:

Transfer_learning_Fine_Tuning.ipynb :

1) Preprocessed(Normalized) all the Images using ImageNet mean and standard deviation
2) Training data has been split into 2 folders in the ratio 80:20 , train and val, for Validation purpose.
3) Splitting resulted in 423 train Images and 109 validation Images (a total of 532 images)
4) Loaded train images and validation images with data_loader in batches of 4
5) Imported Resnet18 model
6) Freezed gradients using `param.requires_grad = False`
7) Added a fully connected layer with 6 output classes
8) Loss used: CrossEntropyLoss()
9) Optimizer used : SGD/Adam
10) Scheduler used: StepLR with a step size=7 and gamma=0.1

Results/ Observations :

1)Test accuracy on test_data(120 images) was greater than 95%
2)Validation Accuracy was 95%+ almost all the time
3)Train Accuracy was low in Epoch 0 and increased to 80%+ by Epoch 6 and remained at around 80% all along training phase
4)Optimizer choice did not affect much, SGD and Adam gave almost similar results

References Used:

1)AIP Tutorial

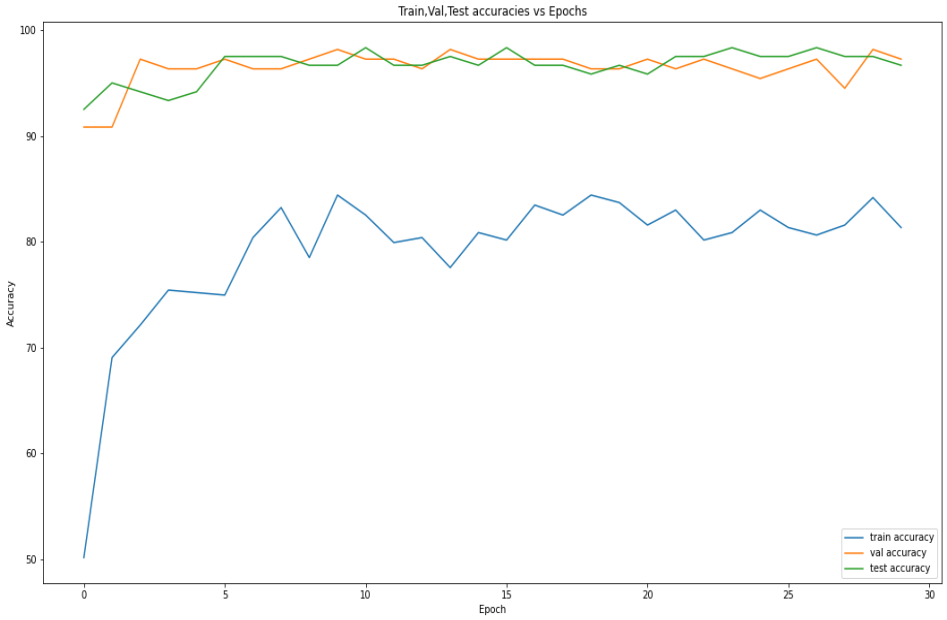
2)Pytorch documentation

3)sklearn documentation

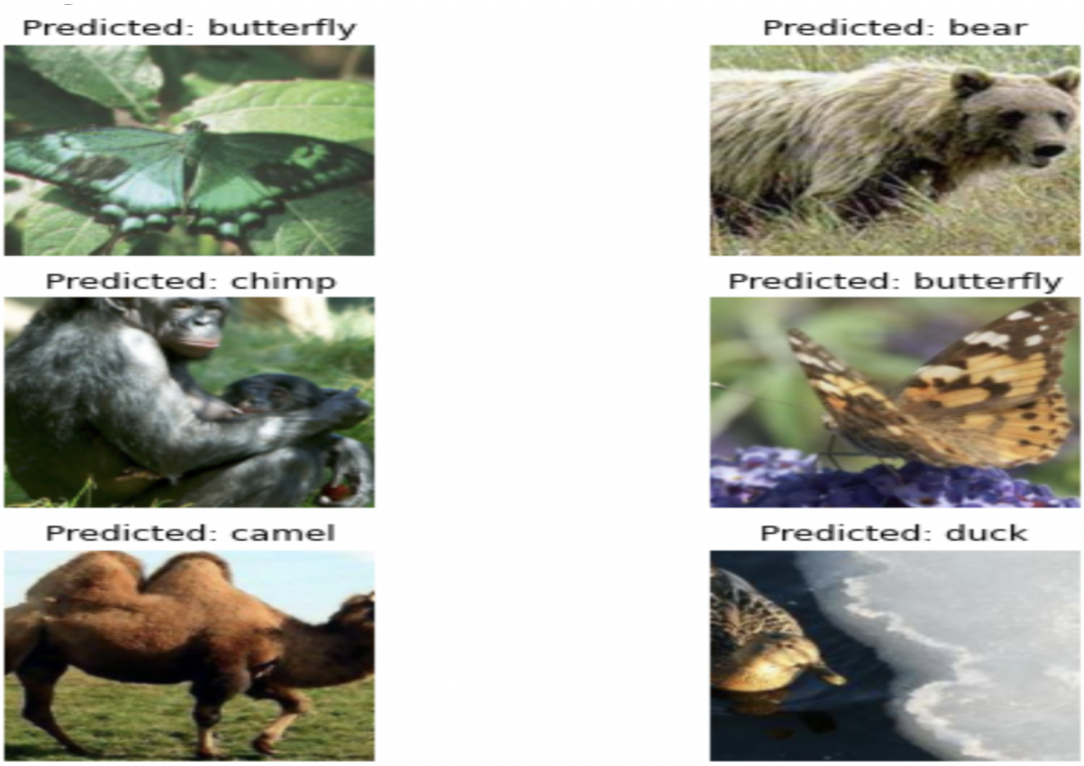
Classification Report:

```
Test Accuracy:
96.66666666666667
----------------------------
Classification Report:
              precision    recall  f1-score   support

        bear       1.00      0.90      0.95        20
   butterfly       1.00      1.00      1.00        20
       camel       0.91      1.00      0.95        20
       chimp       0.95      0.95      0.95        20
        duck       1.00      1.00      1.00        20
    elephant       0.95      0.95      0.95        20

    accuracy                           0.97       120
   macro avg       0.97      0.97      0.97       120
weighted avg       0.97      0.97      0.97       120

----------------------------
Confusion Matrix:
[[18  0  1  1  0  0]
 [ 0 20  0  0  0  0]
 [ 0  0 20  0  0  0]
 [ 0  0  0 19  0  1]
 [ 0  0  0  0 20  0]
 [ 0  0  1  0  0 19]]
<Figure size 1080x1080 with 0 Axes>
```

Accuracy vs Epochs:



Some Predicted Images:



AIP ASSIGNMENT - 1

Name: **K.Kalyan Reddy**
SR.No: **21361**

2)C)Neural Network from Scratch :

Neural_Net.ipynb :

1) Preprocessed(Normalized) all the Images using train data mean and standard Deviation

Extracting Mean and Standard deviation from train_data for data normalization.

This is implemented using cv2(opencv) packages and the mean and std vectors of train_data came out to be

Mean of Train data: **[0.47949112 0.46650227 0.40862144]**
Standard dev of Train data: [**0.28497931 0.27706485 0.28491034]**

This was quite surprising as the mean and std vectors here as very close to ImageNet dataset mean and std which are:

Mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]

2) Training data has been split into 2 folders in the ratio 80:20 , train and val, for Validation purpose.

3) Splitting resulted in 423 train Images and 109 validation Images (a total of 532 images)

4) Loaded train images and validation images with data_loader in batches of 4

5)Loss used: CrossEntropyLoss()

6)Optimizer used : SGD/Adam

7)Scheduler used: StepLR with a step size=7 and gamma=0.1

Convolution Neural network Architecture:

Layer 1: **Conv2d**(3,16) with filter size=5, stride=1, padding=0
Batch Norm Layer
Activation : ReLU
Pooing : Maxpool with kernel_size=2 ,stride =2

Layer 2: **Conv2d**(16,32) with filter size=5, stride=1, padding=0
Batch Norm Layer
Activation : ReLU
Pooing : Maxpool with kernel_size=2 ,stride =2

Layer 3: **Conv2d**(32,64) with filter size=5, stride=1, padding=0
Batch Norm Layer
Activation : ReLU
Pooing : Adaptive average pool with kernel_size=2 ,stride =2

Layer 4: **Linear FC(**64**)**
Fully connected layer with 64 inputs and 6 outputs.

Results :

1)Test accuracy on test_data(120 images) was around 45%

2)Validation Accuracy was around 40% almost all the time

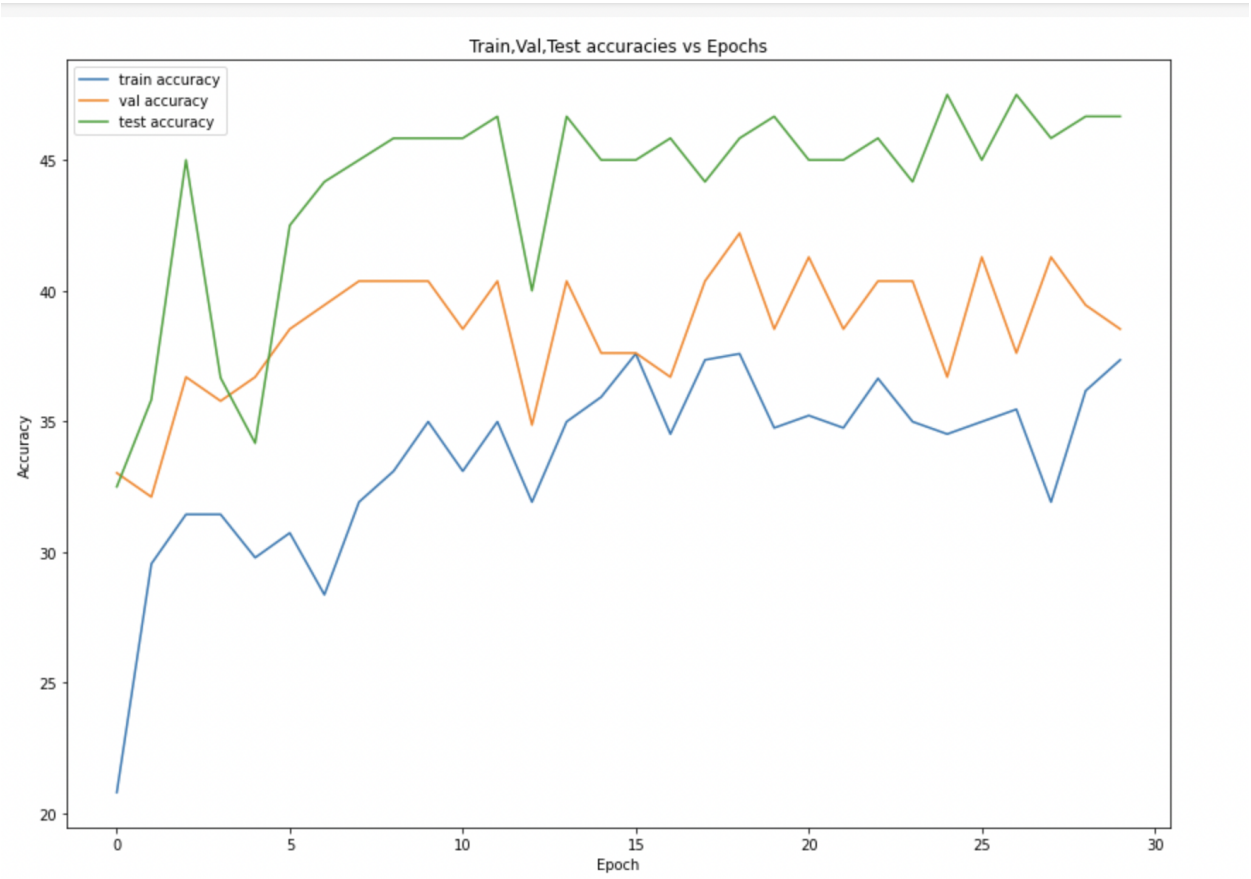
3)Train Accuracy was low, around 30% all the time

4)Optimizer choice did not affect much, SGD and Adam gave almost similar results

Classification Report:

```
Accuracy:
0.4583333333333333
---------------------------
Classification Report:
              precision    recall  f1-score   support

        bear       0.33      0.10      0.15        20
   butterfly       0.54      0.70      0.61        20
       camel       0.44      0.80      0.57        20
       chimp       0.58      0.55      0.56        20
        duck       0.50      0.20      0.29        20
    elephant       0.32      0.40      0.36        20

    accuracy                           0.46       120
   macro avg       0.45      0.46      0.42       120
weighted avg       0.45      0.46      0.42       120

---------------------------
Confusion Matrix:
[[ 2  3  6  2  1  6]
 [ 0 14  2  1  1  2]
 [ 1  2 16  0  0  1]
 [ 1  3  0 11  0  5]
 [ 1  4  6  2  4  3]
 [ 1  0  6  3  2  8]]
<Figure size 1080x1080 with 0 Axes>
```

Accuracy vs Epochs:



Observations:

1)Very low accuracy is observed compared to KNN model & fine-tuned model . This is perhaps because low training data ,the neural network was trained on almost 70 images per class which seems to be insufficient for the network

2)Also increasing layers did not increase much accuracy, tried multiple hyper parameters as well to bring up test accuracy which includes batchnorm layers, pooling type and filter size, stride etc

3)Most of the combinations of hyper parameters resulted in accuracy around 40%

Comparison between 3 models:

	KNN	Transfer_Learning (Fine Tuning)	Neural_Net from scratch
Test Accuracy	**95%**	**96.6%**	**45.83%**

- Clearly, Neural Network from scratch performed worse while KNN and fine tuned performed almost similar.

- Adam and SGD performed almost similarly in both fine tuning and neural network from scratch.