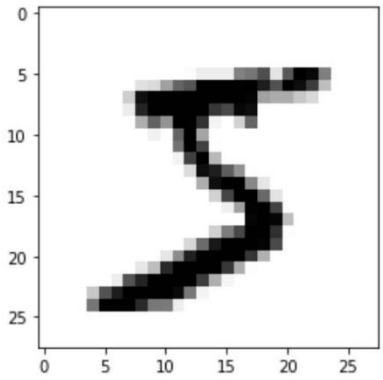
ENEE633 Statistical Pattern Recognition Project 2

UID:118339238 Young-Shiuan Hsu

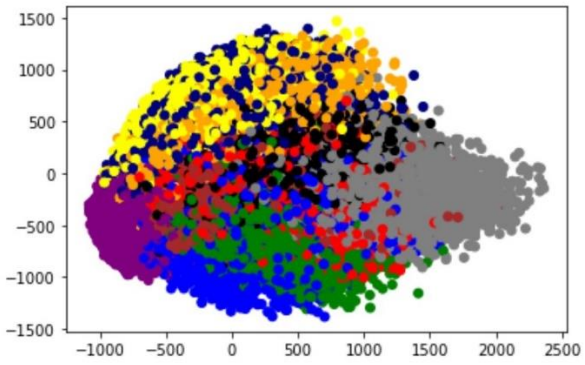
- **Part I : Hand-written Digit Recognition (MNIST dataset)**

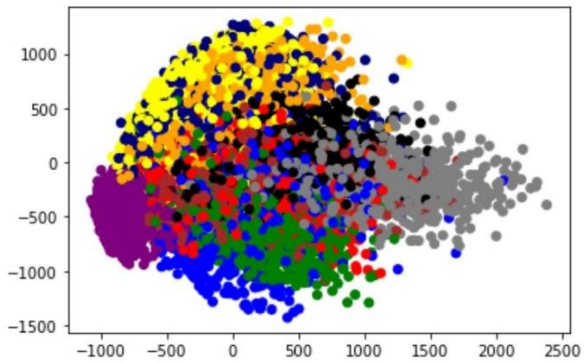
The MNIST hand written digit dataset is a dataset could be directly loaded with the function mnist.load_data() in python. It contains 60000 training samples and 10000 testing samples, and each sample is a 28*28 image that has no colors. For example, the data we are going to process in this section is going to look like the figure below.



1. SVM

In this section, we have to try different kernels such as the Linear, polynomial and RBF kernel. First, we have to reduce the dimension of the data set such that we could train the support vector machine more efficiently. I choose to us PCA and reduce the dimension to 2. However, since we are losing a lot of data, it is hard to get high accuracy in this case. The left figure below is the result of implementing PCA to the training data and the right figure is the result of implementing PCA to the testing data. Each color represents a class and there are 10 classes.





i. Different parameters for Linear kernel

Since the run time for Linear kernel is extremely large, I set the maximum iteration and by tuning different maximum iteration, we can get different results.

Max iterations	1e5	1e10	1e30
Accuracy (Score)	0.4519	0.4525	0.4544

ii. Different parameters for Polynomial kernel

By changing different gamma and degree, we are able to get different accuracy (scores). The maximum iteration is 1e5 in this experiment.

	Degree = 2	Degree=3	Degree=4
Gamma=0.5	0.3293	0.3287	0.1985
Gamma=1	0.3316	0.4408	0.3094
Gamma=3	0.3333	0.4423	0.3082

iii. Different parameters for RBF kernel

By tuning the gamma parameter, we are able to get different accuracy (score). The maximum iteration is 1e5 in this experiment.

	Gamma=0.008	Gamma=1	Gamma=7	Gamma=15
Accuracy	0.4545	0.4782	0.4808	0.3011

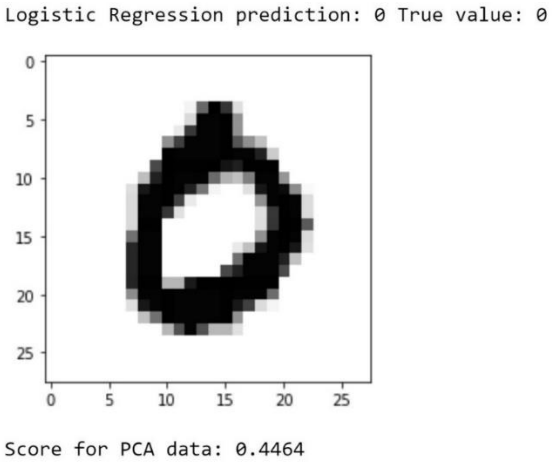
Obviously, we can see that if we increase the value of gamma, it will improve the training accuracy. However, when we pass a particular threshold, the model will overfit the training data which causes the accuracy to drop.

iv. Conclusion

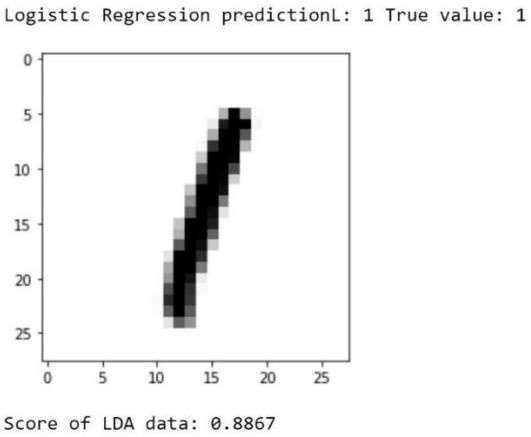
From the experiment, we can tell that the linear and RBF kernel performs better that the polynomial kernel. However, I could not find a parameter that could have results over 0.5. I am assuming that since we reduce the dimensionality to 2 that loss a lot of data information, it is hard for the model to do better.

2. Logistic Regression

First, we have to implement PCA and LDA and analyze the result of this two different dimension reduction method. For PCA, the method is the same as we did in SVM. For LDA is this experiment, we reduce the dimensionality to the number of classes-1, which is 9. For logistic regression, we use the 'newton-cg' as the solver parameter, and this is the only one that works in this large dataset. For the result of PCA we are only able to obtain the accuracy score of 0.4464, which is similar to SVM with linear and RBF kernel. The figure shown below is the result of PCA data and randomly pick a data to visualize the classification results.

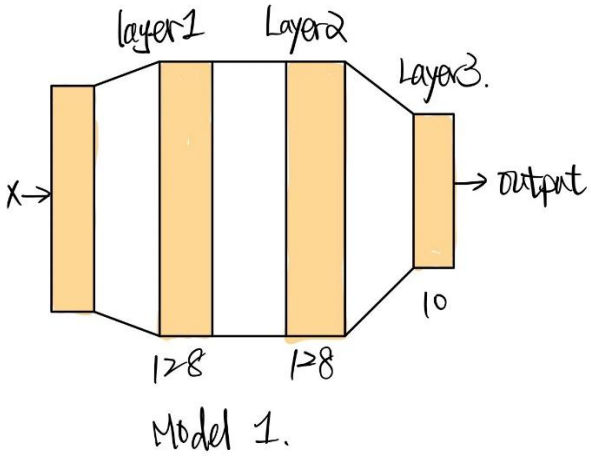


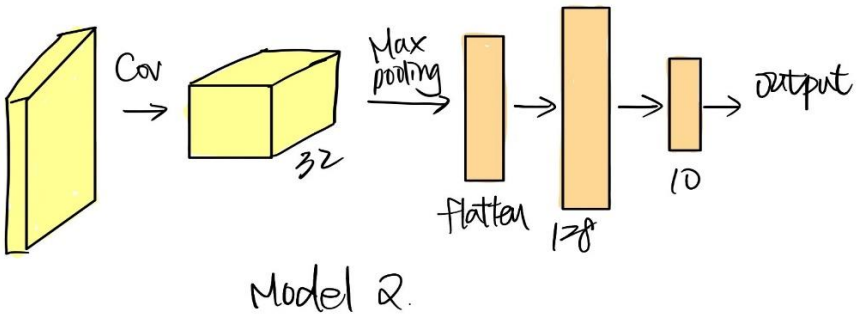
For the result of LDA we are able to obtain the accuracy score of 0.8867, which is higher that I expected. The figure shown below is the result of LDA data and randomly pick a data to visualize the classification results.



3. Deep Learning (CNN)

In this experiment, I tried two different structures of neural network to identify the hand-written digits. One is a completely flat layer that does not have any convolution layer and the activation function is a sigmoid function. The other model has one convolution layer and one max-pooling layer which has the relu function as the activation function.





Each model is trained with 10 epochs and has a very descent accuracy. For model 1 the training accuracy is 0.9322 and the testing accuracy (to check if the model overfits the training data set) of model 1 is 0.9298.

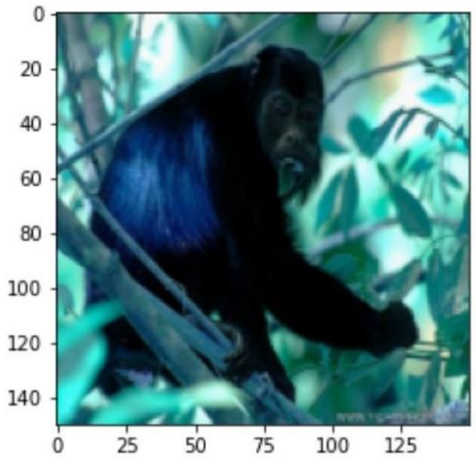
```
Epoch 1/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.5524 - accuracy: 0.8473
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3435 - accuracy: 0.8971
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3116 - accuracy: 0.9036
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3082 - accuracy: 0.9029
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2913 - accuracy: 0.9104
Epoch 6/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2715 - accuracy: 0.9155
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2436 - accuracy: 0.9245
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2382 - accuracy: 0.9254
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2342 - accuracy: 0.9263
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2193 - accuracy: 0.9322
```

For model 2 has a training accuracy of 0.9854 and the testing accuracy of 0.9737.

```
Epoch 1/10
1875/1875 [==============================] - 7s 4ms/step - loss: 3.5275 - accuracy: 0.7157
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.6849 - accuracy: 0.8171
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3189 - accuracy: 0.9246
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1872 - accuracy: 0.9560
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1328 - accuracy: 0.9679
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1026 - accuracy: 0.9747
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0921 - accuracy: 0.9783
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0760 - accuracy: 0.9812
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0626 - accuracy: 0.9840
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0578 - accuracy: 0.9854
```

- **Transfer Learning**

In this section, we are using the Kaggle 10-monkey species database. The objection is to classify the 10 different monkey species given a small training dataset. First, we have to load the dataset. There will be in total 1098 images in the training data set and 272 testing data set with the size 150*150*3 of each image. The image below is to visualize the data that we are going to work on.



After loading the dataset, we construct a neural network with only 3 layers hoping to distinguish between different monkey species. However, the training dataset is not sufficient in this case, therefore, the training accuracy is extremely low and obviously stuck in a local minimum in this case. (The figure below)

```
Epoch 1/25
34/34 [==============================] - 2s 15ms/step - loss: 2.8350 - accuracy: 0.0956
Epoch 2/25
34/34 [==============================] - 1s 15ms/step - loss: 2.4013 - accuracy: 0.0965
Epoch 3/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3597 - accuracy: 0.1020
Epoch 4/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3627 - accuracy: 0.0993
Epoch 5/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3359 - accuracy: 0.0947
Epoch 6/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3254 - accuracy: 0.1029
Epoch 7/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3153 - accuracy: 0.0792
Epoch 8/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3142 - accuracy: 0.0993
Epoch 9/25
34/34 [==============================] - 0s 15ms/step - loss: 2.3072 - accuracy: 0.1029
Epoch 10/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3099 - accuracy: 0.0902
Epoch 11/25
34/34 [==============================] - 0s 15ms/step - loss: 2.3065 - accuracy: 0.0920
Epoch 12/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3079 - accuracy: 0.0956
Epoch 13/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3080 - accuracy: 0.0856
Epoch 14/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3024 - accuracy: 0.1111
Epoch 15/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3022 - accuracy: 0.1111
Epoch 16/25
34/34 [==============================] - 1s 15ms/step - loss: 2.3021 - accuracy: 0.1111
```

Therefore, it is impossible to train a useable model with a raw model. Therefore, we have to introduce the pretrained model to improve the results. In this case, I am using the pretrained model VGG16. By discarding the last layer and add the layers that we want to train to the new model. I freeze the pretrained model instead of the last five layers and retrained the dataset. In 50 epochs, we are able to obtain an accuracy of 1 that greatly improved the model we trained previously.

```
Epoch 38/50
34/34 [==============================] - 5s 150ms/step - loss: 0.0071 - accuracy: 0.9991
Epoch 39/50
34/34 [==============================] - 5s 150ms/step - loss: 0.0057 - accuracy: 0.9991
Epoch 40/50
34/34 [==============================] - 5s 149ms/step - loss: 0.0107 - accuracy: 0.9953
Epoch 41/50
34/34 [==============================] - 5s 150ms/step - loss: 0.0129 - accuracy: 0.9962
Epoch 42/50
34/34 [==============================] - 5s 150ms/step - loss: 0.0116 - accuracy: 0.9962
Epoch 43/50
34/34 [==============================] - 5s 149ms/step - loss: 0.0036 - accuracy: 1.0000
Epoch 44/50
34/34 [==============================] - 5s 149ms/step - loss: 0.0020 - accuracy: 1.0000
Epoch 45/50
34/34 [==============================] - 5s 150ms/step - loss: 6.8007e-04 - accuracy: 1.0000
Epoch 46/50
34/34 [==============================] - 5s 149ms/step - loss: 8.0406e-04 - accuracy: 1.0000
Epoch 47/50
34/34 [==============================] - 5s 149ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 48/50
34/34 [==============================] - 5s 150ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 49/50
34/34 [==============================] - 5s 150ms/step - loss: 6.9397e-04 - accuracy: 1.0000
Epoch 50/50
34/34 [==============================] - 5s 149ms/step - loss: 4.7308e-04 - accuracy: 1.0000
```

The issue in this section is that my laptop has a limited GPU memory that I could not train other models. For instance, if I train a model by changing values or load different models, the system will break down and I have to restart everything. I tried my best to meet the minimum requirements in this section and I am only able to try the VGG16 pretrained data.