42028: Deep Learning and Convolutional Neural Network

Assignment -1

Student Name: VRAJ MEHTA

Student ID: 13488642

Introduction:

The report shows the various methods used to classify handwritten digits correctly. Given an image of a digit/number, the task of Machine learning model is to correctly classify it. MNSIT database of handwritten digits having fixed size images is been used for the experiments.

Raw data, Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) are been used for feature extraction. Whereas the classifiers are: Support Vector Machine (SVM), K-Nearest Neighbour (KNN) and Artificial Neural Network (ANN).

The report along with the python notebooks shows all the combinations possible with the mentioned features & classifiers. For instance, HOG, LBP & Raw Data with SVM. Comparison among which feature extraction method works best with SVM as classifier. Similarly, HOG, LBP & Raw Data with LBP and ANN.

A comparative study with discussion is also shown which feature extraction method and classifier performs better given the dataset provided.

Dataset:

The MNSIT Dataset provided has 60000 training samples and 10000 testing samples. The given data is already pre-processed. Each image is of the size 784 (1-D Array), which will be converted to 28X28 to form a 2-D Array so that image processing techniques can be used on the data.

There are four files provided for training and testing:

1.Train Images: train-images-idx3-ubyte.gz	(60000, 784)
2.Train Labels: train-labels-idx1-ubyte.gz	(60000,)
3.Test Images: t10k-images-idx3-ubyte.gz	(10000, 784)
4.Test Labels: t10k-labels-idx1-ubyte.gz	(10000,)

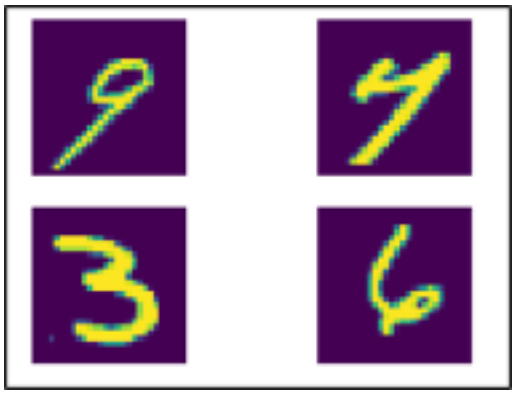


Fig. Train Image

Class Names or Label Names are the following as we are trying to classify single handwritten digit correctly.

class_names/labelNames =
["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"].

Experimental results and discussion:

a.Experimental settings:

->Classifiers (SVM, KNN, ANN)

(I) SUPPORT VECTOR MACHINE (SVM)

SVM Kernel Parameters Search C= [0.1,1,100,300], Gamma= [0.01,0.1,1,5]
& Kernel = [rbf, poly, linear]
The parameters leading to best results are:

- **Kernel** = 'RBF'
- **C** = 300
- **Gamma** = 5

(II) K - NEAREST NEIGHBOUR (KNN)

KNN Parameters Search k= [1,3,5,7,9]

Step 4: Training Classifier using KNN multiclass classifier

```
[12] for num in range(1,10,2):
       clf = KNeighborsClassifier(n_neighbors=num)

       ## 2.2 Fit the model to the training dataset
       clf.fit(data_train, labels_train)

       ## 3. Calculate the train set accuracy (~ 1 line)
       ## Hint use clf.score()
       acc_train = clf.score(data_train, labels_train)
       print('Neighbours:',num ,'Train set accuracy: ', acc_train)

     Neighbours: 1 Train set accuracy:  1.0
     Neighbours: 3 Train set accuracy:  0.9252
     Neighbours: 5 Train set accuracy:  0.9126333333333333
     Neighbours: 7 Train set accuracy:  0.9047833333333334
     Neighbours: 9 Train set accuracy:  0.9006666666666666
```

The figure shows the training set accuracy for different values k. Taking only odd values for number of neighbours so that the model doesn't confuse itself, if two points results in same distance.

Also, n_neighbours = 1 will result in under-fitting of the model and will result in poor test accuracy, though the train accuracy is high.
Hence, the best parameter is **k=3.**

(III) ARTIFICIAL NEURAL NETWORK (ANN)

The below figure is the configuration for the ANN classifier. There are three layers in ANN. The first layer is flatten with input shape as 28X28, since we are provided with 784 1-D image. The second layer are the number of nodes in it with the activations as 'relu'. The third layer as the class_names, since we are predicting number from '0' to '9', it should be 10.

```
[16] model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=[28,28]),
                                         tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                         tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

Step 5: Training the model

```
model.compile(optimizer = tf.optimizers.Adam(),
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

H=model.fit(train_images, train_labels, epochs=10,validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.2086 - accuracy: 0.9387 - val_loss: 0.0978 - val_accuracy: 0.9710
Epoch 2/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0837 - accuracy: 0.9740 - val_loss: 0.0854 - val_accuracy: 0.9758
Epoch 3/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0536 - accuracy: 0.9832 - val_loss: 0.0706 - val_accuracy: 0.9802
Epoch 4/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0386 - accuracy: 0.9872 - val_loss: 0.0719 - val_accuracy: 0.9800
Epoch 5/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0266 - accuracy: 0.9913 - val_loss: 0.0780 - val_accuracy: 0.9762
Epoch 6/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.0765 - val_accuracy: 0.9804
Epoch 7/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0179 - accuracy: 0.9938 - val_loss: 0.0684 - val_accuracy: 0.9816
Epoch 8/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0133 - accuracy: 0.9955 - val_loss: 0.0763 - val_accuracy: 0.9794
Epoch 9/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0126 - accuracy: 0.9959 - val_loss: 0.0707 - val_accuracy: 0.9822
Epoch 10/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0118 - accuracy: 0.9960 - val_loss: 0.0858 - val_accuracy: 0.9800
```

Parameter tuning for best results:

- **No. Of Neurons/Nodes:** 512
- **Optimizer:** Adam (Gives better results than SGD = 0.01)
- **Epochs:** 10 (more epochs will usually result in better accuracy, but since the accuracy is already high in this case)

->Feature Extraction Method (LBP & HOG)

(I) HISTOGRAM OF ORIENTED GRADIENTS (HOG)

Parameter tuning results in:
- **Orientations**: 9
- **Pixels per cell:** (7, 7)
- **Cells per block:** (2, 2)
- **Transform sqrt:** True
- **Block Norm:** L2-Hys

Changing the Pixels per cell from (10,10) to (7,7), improved the accuracy by around ~10% in the case when SVM is used as classifier.

Also the difference in accuracies between train & test set reduced, which shows the model is not under-fitting or over-fitting.

(II) LOCAL BINARY PATTERN (LBP)

- LBP when used for feature extraction with **SVM (Best parameters for SVM)** as classifier, the parameters that results in best results are:

Number of Points: 12
Radius: 2

This increases the accuracy for both train and test datasets by around ~10%.

- LBP when used for feature extraction with **KNN (Best parameters for KNN)** as classifier, the parameters that results in best results are:

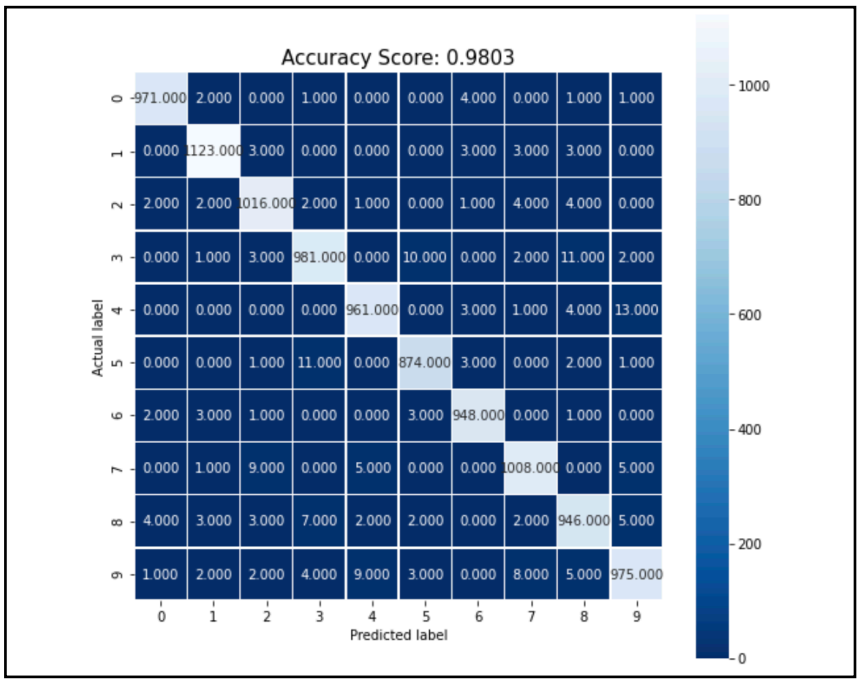
  Number of Points: 6
  Radius: 2

  This increases the accuracy for both train and test datasets by around ~10%.

B. Experimental Results

I. Confusion Matrix

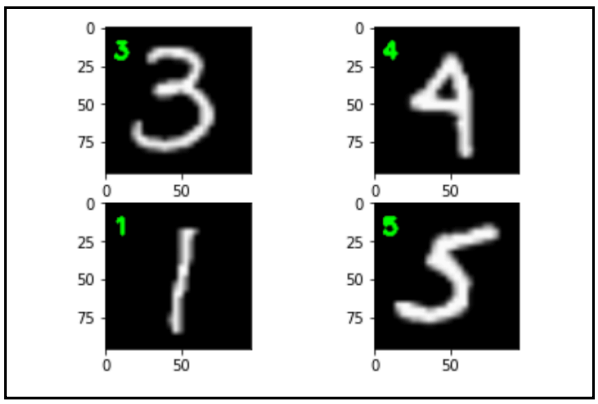
(I) SUPPORT VECTOR MACHINE (SVM)



The figure shows the Confusion Matrix for SVM.
Data used: HOG feature extracted

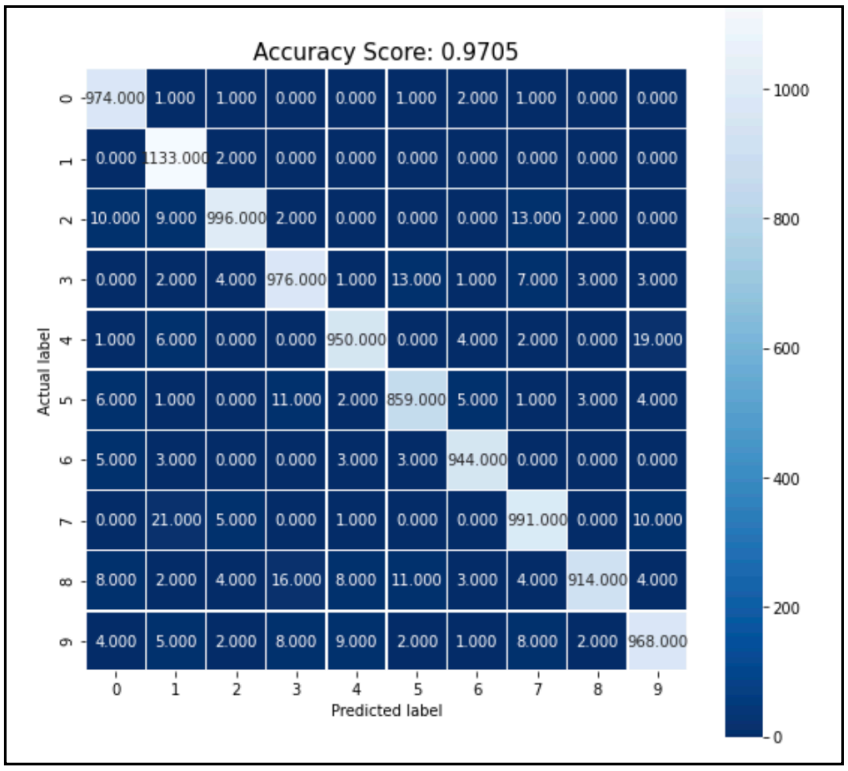
We can clearly see that with fine tuning the parameters for SVM, also using HOG for feature extraction around ~98% accuracy is achieved.

Number '4' is identified as Number '9', is the most incorrect with 13 times. Also '3' identified as '8' & '5' identified as '3' have >10 incorrect cases.



Some sample images of the results with their predicted output.

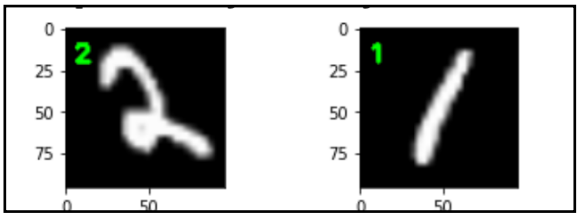
(II) K - NEAREST NEIGHBOUR (KNN)



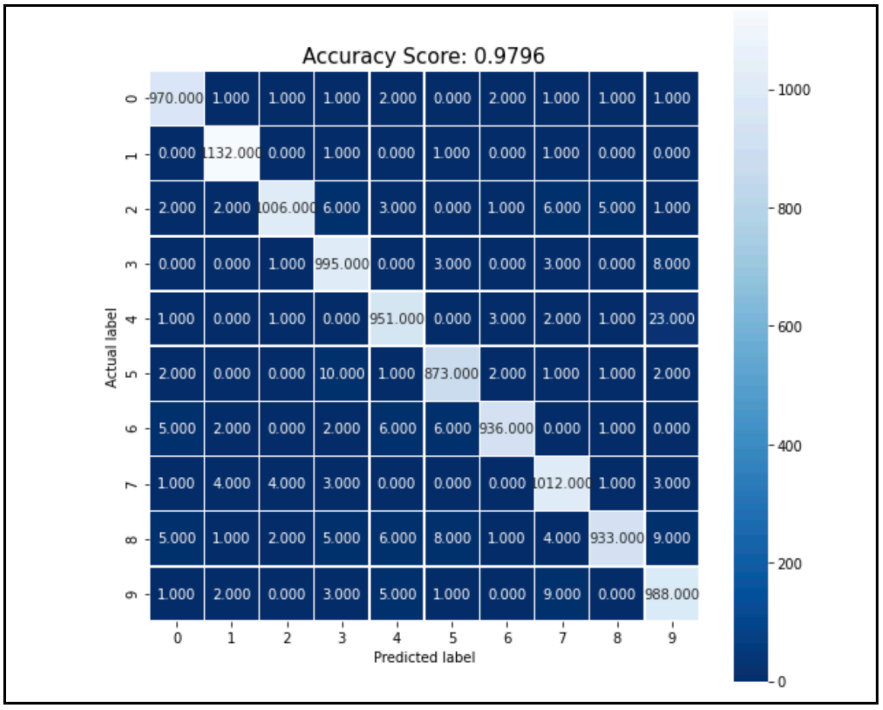
The figure shows the Confusion Matrix for KNN.
Data used: Raw Input

We can clearly see that with fine tuning the parameters for ANN, around ~97% accuracy is achieved.

Some samples images of the results with their predicted output.

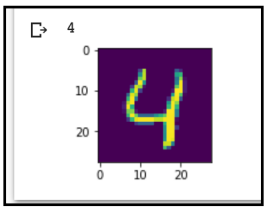


(III) ARTIFICIAL NEURAL NETWORK (ANN)



The figure shows the Confusion Matrix for ANN.
Data used: Raw Input

We can clearly see that with fine tuning the parameters for ANN, around ~98% accuracy is achieved.



The result image for ‘4’.

Number ‘4’ is identified as Number ‘9’, is the most incorrect with 23 times. Rest all the number, incorrectly labeled is less than 10.

II. Comparative Study

The results are with the best parameters fine tuned:

Classifier/Feature	LBP	HOG	Raw Input
SVM	60.83	98.03	98.33
KNN	49.52	86.76	97.05
ANN	NA	NA	97.96

Clearly SVM & ANN are performing better than KNN.
Also Raw Input in most of the cases performing better than HOG & LBP.

III. Discussion

Discussion with respect to the **classifiers**, in case of using SVM as classifier, the important task is to make a balance between 'C' the regularization parameter & 'Gamma'. Also Kernel is important factor to be taken into account.
The training (~100%) and testing (~98%) accuracy for HOG + SVM, is very similar, which results that the model is perfect. There is no under-fitting or over-fitting happening.

Regarding the results for **KNN**, it doesn't perform as accurate as SVM. LBP + KNN achieved ~49.5% accuracy while HOG + KNN had ~86.8% accuracy.
KNN only takes one parameter that is 'k' neighbours to consider while classification. This method clearly doesn't perform well when we have multiple classes/labels to predict.

While SVM performs better than KNN, it comes with an **increased cost** of training time. KNN is much faster than SVM. Also, fine tuning the parameters for KNN is achieved in O(n) time as it has only one parameter. While fine tuning the parameters for SVM is achieved in O(n^2) time as it has only two parameters.

Discussion with respect to the **feature extracting methods**, clearly HOG is performing far better than LBP. HOG works on the algorithm that is good in detecting edges, while LBP is good for texture detection. The problem that we are trying to solve is detecting numbers in an image. The numbers present in the image can be easily separated from the background, it is a clear edge detection problem. That is the reason HOG performs better than LBP.

Talking about ANN, it is much better in terms of accuracy and computational time to classify the MNSIT digit dataset. Neural Networks have their own feature extraction methods that is the in-between layers & neurons present in them. With less time spend on parameter tuning, also the fast computation makes ANN the best classifier for this task.

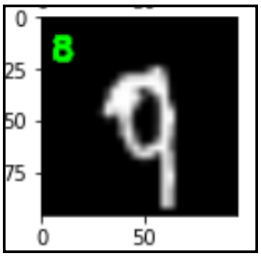


Image of '8' incorrectly classified as '9'.

Conclusion:

The MNSIT digit dataset is an interesting and realistic problem to solve which can have further applications in real life scenarios. Three different types of classifiers were utilised and comparison was done on their performance with respect to accuracy and compute time.
SVM, KNN & ANN are the three classifiers used for the experiment. It is clear that ANN is the most accurate and fastest, while SVM performs similar in-terms of accuracy but is expensive in compute time. KNN performs the worst in accuracy but takes relatively less compute time than SVM.
Comparing the feature extracting methods, HOG is based on edge detection while LBP is a texture based feature extractor. The MNSIT digit dataset performs better with HOG as feature extractor, as it is not a texture based image.

This experiment clearly shows, which classifier and feature extracting methods should be used depends on the given dataset. We can't deploy SVM or KNN in real time situations, as they are too slow. ANN can be used to solve real world problems, as it both fast and accurate. But given the internal working of SVM & KNN baseline models being quite less complex, they still perform well having ~98% accuracy.