ELEC 6910T Deep Learning
Assignment 1

Frederick Ziyang HONG
20528735

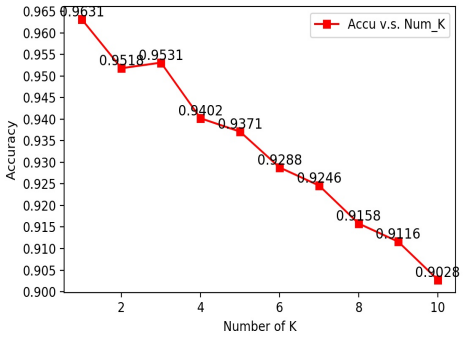
MNIST (100 points)

K Nearest Neighbors (KNN) (25 points)

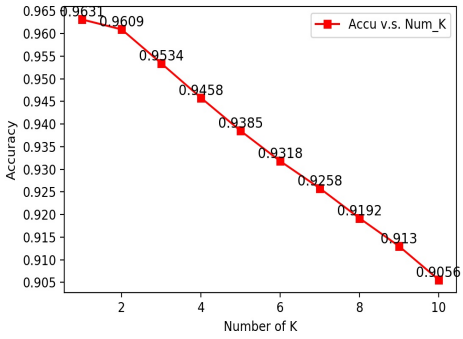
We will use the sum of absolute difference (SAD) on all the pixels to measure the similarity between two images. What will be the accuracy if we find the nearest neighbor? What will be the accuracy if we find K neighbors? When we find K, we choose the label that appears most among the KNN images (if there is a tie, we pick the one with the smallest aggregated SAD). Plot a curve of accuracy versus K from 1 to 10.

Answer

Hereinafter for all the answers, we train the model with 50,000 samples, validate it with 10,000 samples from the MNIST training dataset, and test it with 10,000 samples from MNIST testing dataset.
In this part, firstly we implement the K Nearest Neighbors(KNN) methods from scratch, i.e., implementing the SAD as similarity and then take K nearest neighbours for consideration. Note that when there is a tie between the prediction and the 2^{nd} candidate, we pick the one with the smallest aggregated SAD (as suggested above). In this case, the accuracy-K curve is shown as Figure 1(a). The curve indicates that with smallest aggregated SAD, the accuracy will be largest when we only consider nearest neighbors instead of K nearest. Even though we abandon the smallest aggregated operation, the accuracy remains largest when $K = 1$, as shown in Figure 1(b). I suppose it is caused by the differences between testset and training set.
However, when we import KNN from *scikit-learn*, the accuracy is actually improved. The Figure 2(a) shows that when $K = 2$, the accuracy is the highest as 0.9705 in the 10,000 samples MNIST testset. The Figure 2(b) indicates that, on such a testset, the testing time of *scikit-learn*-KNN is around 600 seconds while the KNN made from scratch is around 2,000 seconds. I suppose the *scikit-learn* package have decent code optimization thus significantly cut development time.

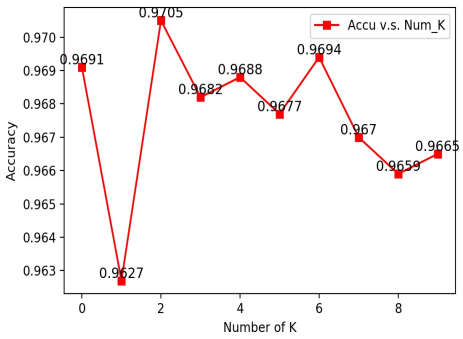


(a) Accu v.s. K with aggregated SAD



(b) Accu v.s. K without aggregated SAD

Figure 1: Accuracy v.s. K curves for my KNN



(a) Accu v.s. K with *scikit-learn*-KNN

```
neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(train_img, train_label)

for i in range(test_img.shape[0]):
    if neigh.predict(np.reshape(test_img[i],(1,784))) == test_label[i]
        opt+=1

accu = opt/test_img.shape[0]
toc = timeit.default_timer()
print ('Elapsed: %s'%(toc - tic))
```

```
Elapsed: 605.7476664999999
```

```
...
```

```
    if answer1.K_count(test_img[i], train_img, train_label, 1)[0][0] =
        opt+=1
accu = opt/test_img.shape[0]
toc = timeit.default_timer()
print ('Elapsed: %s'%(toc - tic))
```

```
Elapsed: 2049.7374494
```

(b) Running time for 2 KNNs

Figure 2: Comparision between *scikit-learn*-KNN and my KNN

Multilayer Perceptron (MLP) (25 points)

What is the accuracy of such a model? Plot a curve of accuracy versus the number of neurons: 4, 8, 16, 32, 64, 128, and 256. We assume both hidden layers have the same neurons.

Answer

We start a Multilayer Perceptron(MLP) model with two hidder layers, 4 neurons for each hidden layer, ReLU as activation, cross-entropy as loss function. The accuracy of this model is not good enough, stuck at 0.7. After we increase the number of neurons of each hidden layers, the accuracy is up to 0.9831 when we finally put 256 neurons for each layer. Figure 3 shows the relationship between

accuracy and number of neurons.

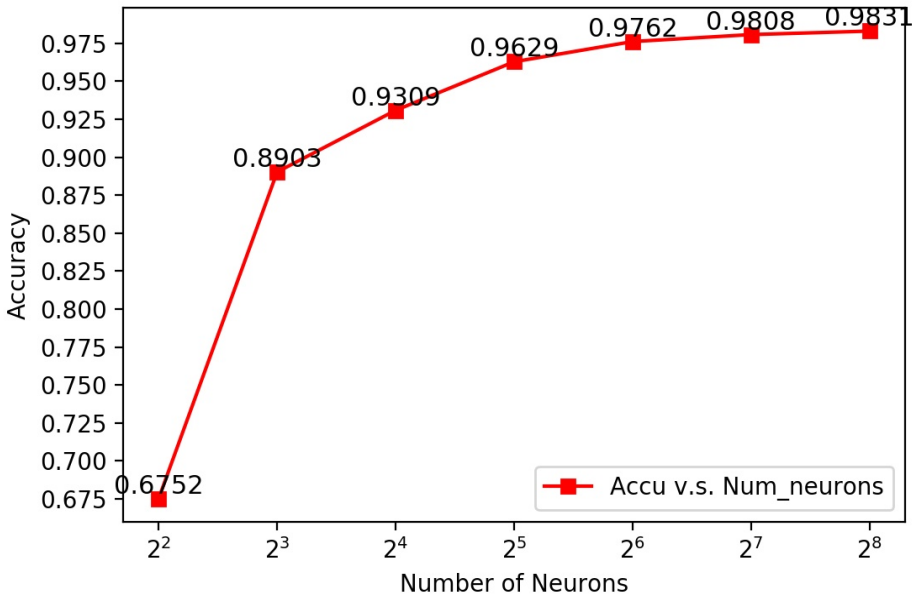


Figure 3: Accu v.s. Numbers of Neurons with MLP

Note that, dropout layer is only a regularization technique which we use to prevent over-fitting. Chances are fully-connected network are easier to be over-fitting, so we might need inserting dropouts. However, for the MLP on MNIST-digits dataset, we did not encounter over-fitting. Therefore we do not need to insert dropout layers here.

Convolutional Neural Networks (CNN) (25 points)

Please use input image size 28×28 in your model, although the input image size in the original LeNet-5 model is 32×32. What is the accuracy of the LeNet-5 model?

Answer

In LeNet, we should deal with the convolutional kernel size, strides, the number of filters, and the pooling layers. It can make the model more generalized. For the sake of avoiding feature shrinking, we use Zero Padding here, a technique that allows us preserve the original input size in the CNN. In a word, when we pass in the argument 'valid', the padding will not affect; when we pass in the argument 'same', the Zero Padding works, and it will figure out how much sizes

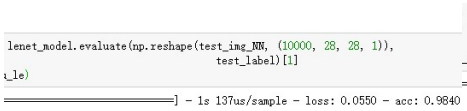


Figure 4: The accuracy of LeNet-5

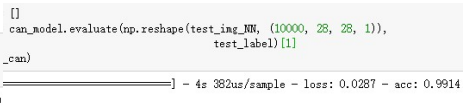


Figure 5: The accuracy of CAN with dilated convs

of borders need to be appended. The output size of convolution layers would be maintained by using Zero Padding.
The accuracy of the LeNet-5 model is up to 0.9840 after 20 epochs with 64 batch size as shown in the Figure 4.

Context Aggregation Networks (CAN) (25 points)

We can have a model with a large receptive field with the same spatial resolution in the hidden layers. See the network architecture in the Table. What is the accuracy of such a CAN model? What will be the accuracy if we use a different number of feature channels instead of 32?

Answer

We follow the table to build up the CNN with dilated convolutions as the CAN model. It can aggregate the context with the dilated filter. In this case, the accuracy of the CAN model is 0.9914 as indicated in Figure 5.
If we change the number of feature channels of the model, say we put it as 16 and 64 respectively. After the same epochs and batch sizes, the former one's accuracy is 0.9891 while the latter one's is 0.992. Since the accuracy for 10,000 samples will be affected by the random initialization of weights and biases, we can infer that with feature channel number as 32, the model is already with good generalizatin capability for the MNIST dataset.