Programming Assignment 2

Katarina Vuckovic
CAP5415 Computer Vision
Nov 15, 2021
Instructor Dr. Yogesh Singh Rawat

I. Part 1: K-Nearest Neighbor Classification

A. Introduction

Nearest neighbor is a classification technique which computes the l2-norm between the test sample and all training samples in the dataset. The class of closest training sample (the sample that results in a minimum distance between the test sample) is assigned to the test sample. If we consider only the closest sample, this is referred to as $K = 1$ nearest neighbor. We may want to consider K closest samples where $K > 1$. In this case, the class with more points win. This is called K-Nearest Neighbor (KNN). In binary classification, it is common practice to select an odd number for K to avoid a tie between two classes. This technique does not require any training and is simple to implement. The disadvantage of this technique is becomes slow as the number of samples and number of features increases.

The objective of this section is to implement a KNN that classifies the images in the MNIST dataset. The KNN method is tested at different K-values ranging from 1 to 7.

B. Dataset

The project uses the NMIST digit dataset for evaluation. The dataset consists of images with digits from 0 to 9 (i.e. 10 classes). The size of the images are 28x28 and the total number of samples in the dataset is 6000 training samples and 1000 testing samples. The NMIST dataset is available from *sklearn* library.

C. Results and Discussion

The classification accuracy for different K values is presented in Table I. For $K = 1$, a high accuracy of 96% is achieved. Since MNIST is a relatively simple dataset, high accuracy can be achieved with $K = 1$. The improvement in increasing the K value is negligible. The accuracy improves by a fraction of a percent for $K = 3$ after which the accuracy starts slowly decreasing. This concludes that for this specific dataset, the highest possible accuracy using KNN classifier is 96.8% which is achieved when $K = 3$.

TABLE I: KNN Classifiers

K	Accuracy (%)
1	96.0
3	96.8
5	96.6
7	95.5

II. Autoencoders

A. Introduction

Autoencoders (AE) are a neural network structure that learn the features by compressing the data. AE networks consist of two parts, the encoder and decoder. The encoder compresses the representation of the original input and the decoder reconstructs the original input. It is often used for data compression but can also be use to effectively learn features in computer vision applications such as anomaly detection, image colorization, or image classification.

In this section we design two AEs, one using a fully connected (FC) layers and another one using convolutions neural networks (CNNs). The objective is to implement the two AEs on the MNIST dataset and to compare their performance.

B. Dataset

The project uses the NMIST digit dataset for evaluation. The dataset consists of images with digits from 0 to 9 (i.e. 10 classes). The size of the images are 28x28 and the total number of samples in the dataset is 6000 training samples and 1000 testing samples. The NMIST dataset is available from the *torchvision* library.

C. Fully Connected Autoencoder

The encoder consists of 2 layers with 256 and 128 neurons. The decoder also has two layers with 256 and 784 neurons. The encoder has a total of 233472 parameters. The decoder has the same number of parameters as the encoder. The FC AE is trained over 10 epochs using a batch of 100 samples. Furthermore mean square error (MSE) loss criteria is imposed and adam optimizer with a learning rate of 10^{-3} is employed.

Fig. 1 and 2 show two different sample outputs for digits 0 to 9. The top row in each figures shows the original input images that are being encoded and the bottom image are the decoded image. From the results we can observe that all output images clearly identify the digit, however, the decoded images have some black pixels where the white pixels should be.

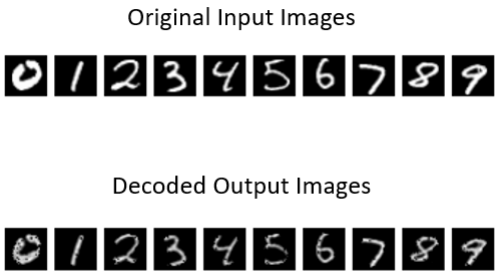

Fig. 1: First set of original (encoded) images vs decoded output images for digits 0-9 using FC AE

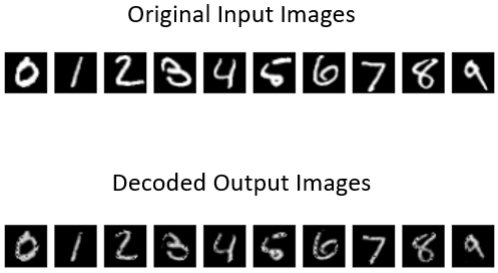

Fig. 2: Second set of original (encoded) image vs decoded output image for digits 0-9 using FC AE

D. CNN Autoencoder

The encoder consists of two convolutional layers with 3x3 kernel size, padding of 1 and ReLU activation function. Each convolutional layer is followed by a maxpool layer. The decoder consists of three convolutional layers with 3x3 kernels, padding of 1 and ReLU activation function. The first two layers are followed by a upsampling function that upsamples by a factor of 2. The last covolutional layer is followed by a sigmoid function instead of a ReLU function. The encoder and decoder each have a total of 233472 parameters. This is on the same order as the FC AE. The FC AE is trained over 10 epochs using a batch of 100 samples. Furthermore MSE loss criteria is imposed and adam optimizer using learning rate of 10^{-3} is employed.

Similarly to the FC AE, Fig. 3 and 4 show two different sample outputs for digits 0 to 9. In this case the digits seem to look thicker in the decoded images when compared to the original images. Comparing the output of the FC AE to the CNN AE, it is difficult to tell which one performs better as the outputs have different characteristics.

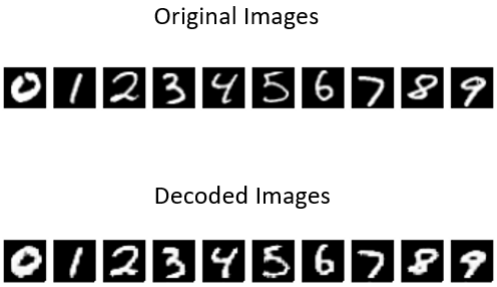

Fig. 3: First set of original (encoded) images vs decoded output images for digits 0-9 using CCN AE

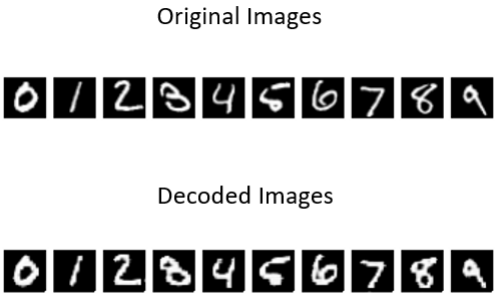

Fig. 4: Second set of original (encoded) image vs decoded output image for digits 0-9 using CNN AE