Image Classification using ANNs

University of Cape Town
CSC3022F

ML Assignment 1: Handwritten digit recognition using machine and deep learning algorithms

Student num: MKWHAP002

Date: 06 May 2022

Introduction

Machine learning (ML) refers to various algorithms that are used to optimize a performance criterion using example data or past experience.

Learning can used when humans are unable to explain their expertise which is the case for handwritten digit recognition. We will be focusing on the type of learning known as Supervised Learning. Classification falls under Supervised Learning. You are given a set of labeled examples also commonly known as training data, each described by a set of features and labeled with a class. With classification, you need to find a model for the class attribute as a function of the values of other attributes.

An Artificial Neural Network is a computational model that imitates the way nerve cells work in the human brain.

ANNs use machine and deep learning algorithms to learn more about data.

ANNs have three or more layers that are connected. The first layer is made up of input neurons, these neurons pass data onto deeper layers which in turn pass the final output data to the last output layer.

Learning is done by iteratively modifying the weights to satisfy some learning rule. The layers learn about the input data by weighing it according to the ANN model.

The ANN can adjust its output results by taking errors into account through backpropagation, each time the output is labeled as an error during the training stage, data is sent backwards. Each weight is updated in correspondence to how it was liable for the error. ANN eventually learns how to minimize the chance for errors and unwanted results.

For this assignment I will be using MNIST (Modified National Institute of Standards and Technology database) dataset which contains 70,000 grayscale images of handwritten digits, dimensions[28x28] as input to my Artificial Neural Network (ANN) to classify hand written digits. The 60,000 of the MNIST dataset is the training set and 10,00 is the test set. Our focus will be based on the ability of a Neural Network to recognize handwritten digits from images and classify them into 10 classes {0,1,2,3,4,5,6,7,8,9}. The end goal is to predict which class an image falls under after completing training of data.

Figure 1. MNIST dataset of hand written digits.

(link to image : https://www.researchgate.net/figure/MNIST-dataset-of-handwritten-digits_fig1_324877673)

CONVOLUTIONAL NEURAL NETWORK Classifier

CNN is one deep learning algorithm popularly used for image recognition, classification including facial recognition. CNN image classification loads an input image and goes on to the process of processing data that has a grid-like topology, it is made up of a series of pixels that indicate how the brightness and color each pixel should be. CNNs usually have three layers: a convolutional layer, a pooling layer, and a fully connected layer. CNN uses an array of weights to extract features from the input image. With each layer, different activation functions are used to add some non-linearity.

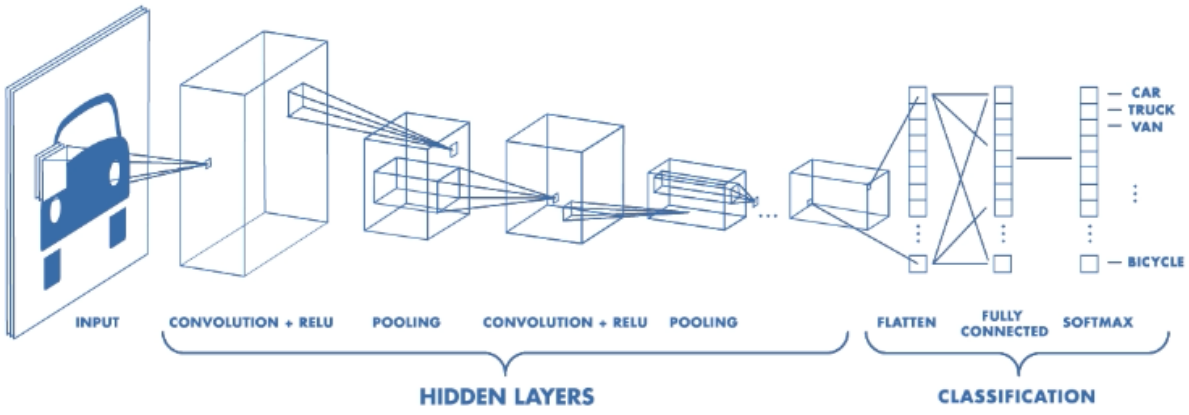


Figure 2. CNN Architecture(image link)

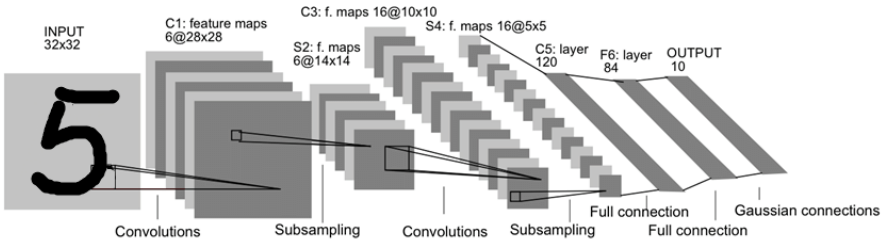
IMPLEMENTATION

Convolution Layer

Before we train any data and define the model, we need to make sure that data from MNIST is read correctly. The torchvision package allows accessing of the data from the local repository, shuffling it and any other pre-procesing requirements. The architecture I used for the CNN is LeNet 5 architecture. This architecture consists of 7 layers, 2 sets of Vonvolution layers and 2 sets of Average pooling layers , a flattening convolution layer, 2 dense fully connected layers and a softmax classifier.

INPUT LAYER

The input layer of a CNN is the first layer get attributes from an input image[28x28]. It uses a small quantity of the input data. It can be defined as a mathematical operation that takes two inputs. Two image matrices [5x5] and [3x3] can be multiplied with each other and the convolution of that multiplication is called" Feature Map". The grayscale image that is fed as input passes through the first convolution layer with the 6 feature maps/filter kernel and stride=1. The input pixels are normalized making mean approx. as 0 and variance approx. as 1. Strides are the number of pixels shifts over the input matrix. Stride is equated to the factor you wish to move the filter by, to move it by 1 pixel, stride =1.



An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

Figure.3: The architecture of LeNet 5(link)

FIRST LAYER

The dimensions change from [32x32x1] to [28x28x6] and the first layer is produced. See link: link

A [32x32x1] image is used as the input image, convolution happens the th size changes as 1 channel is now changed to 6 channels as the filters were applied so the new dimensions are [28x28x6]

In the case where the filter doesn't fit the image correctly, I used padding by padding the image with 0s so it fits or perform valid padding.

It has 1 layer, and the output has 6 layers, kernel size =5 and the padding =2

Size after convolution= (N-f)/S+1 = (32-5)1+28 = 14. The new image as dimensions [28x28x6] .

SECOND LAYER

Pooling layers reduce the number of parameters when the images are too large. I implemented an average pooling layer with a filter of [5x5] i.e. kernal size =5, stride=1 and padding =0. It has 6 layers and the output has 16 layers

Size after pooling = (N-f)/S+1 = (14-5)1+1 = 14. The new image as dimensions [10x10x16] .

THIRD LAYER

We flattened our matrix into a vector with the fully connected layers and combined these attributes together to create a model. The result was then passed to ReLU()

Non-Linearity(ReLU) - Rectified LInear for a non-linear

output : f(x) = max(0,x)

The ReLu is responsible for making sure that there is non-linearity in our ConvNet, which is basically having more than one layer consisting of neurons in the network or ensuring that the activation functions result in weight non-linearities.

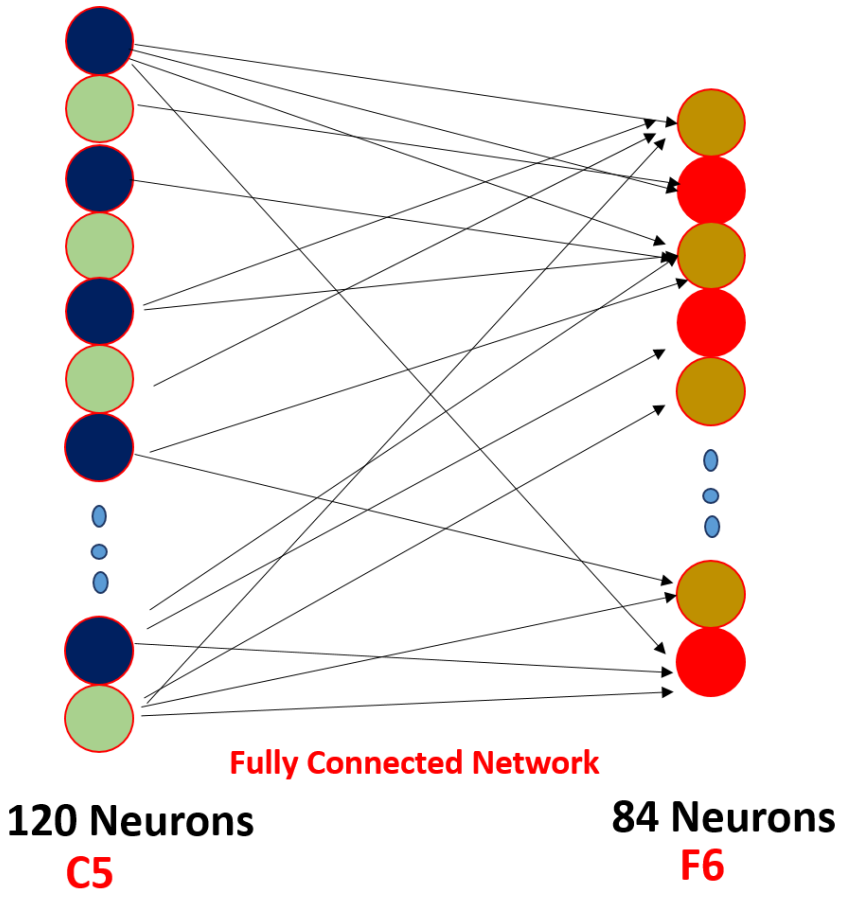


Figure 4: A fully connect layer flattened after pooling

I used Cross Entropy Loss as my loss function because it minimizes the distance between the predicted and the actual output. It is a good loss function for classification in that regard.

I have also chosen the Adam optimizer for this model because it can handle sparse gradients and noisy problems. It is computationally efficient, uses less memory and a good fit for problems use large data.

2ND NEURAL NETWORK

Using Pytorch, I created a class that inherents from the nn.Module, using __init__ function , we will use it to initialize layers of the network.

By initializing nn.Flatten layer, we are converting the [28x28] image into an array of 784 pixel values, using the Linear layer, we perform linear transformations on the input using its weights and biases. The steps are repeated for the 3 layers but the 2nd layer and third use different parameters.

We used the same understand of ReLU from the previous model , The ReLu is responsible for making sure that there is non-linearity in our model. We used they linked layer on our matrix into a vector with the fully connected layers and combined these attributes together to create a model. The result was then passed to ReLU()

We used Cross Entropy Loss as well in this model, for the same reasons I mentioned above.

I chose to use SGD as an optimizer in this case because SGD is known for generalizing well than other optimizers and improves final performance really well.

TRAINING

Classifier{1} .py (defined as 2nd NEURAL NETWORK in this report):

I train my model over a number of epochs, the model then learns the parameters to make improved guesses.

For training the model, we take into consideration the number of epochs, batch size & learning rate, initialize the parameters just mentioned then we can then train and optimize the model as we iterate through the optimization loop. Epoch refers to each iteration of the optimization loop. Epochs will go through the loop over the training dataset to get close to optimal parameters

For the test set, we will loop over the test dataset to check if the model performance is getting better

Call optimizer.zero_grad() to reset the gradients of model parameters, backpropagate the prediction using loss.backward()

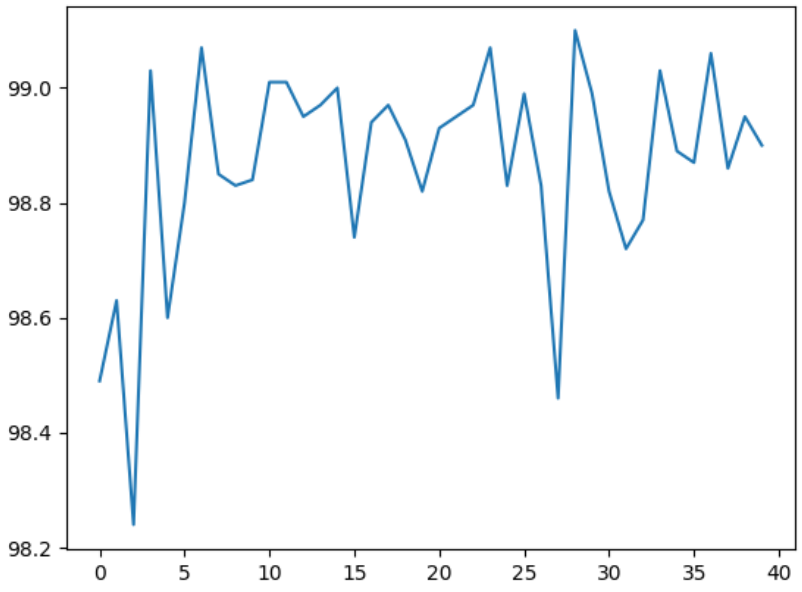
Classifier{2}.py

I train my model over a number of epochs, the model then learns the parameters to make improved guesses.

For training the model, we take into consideration the number of epochs & learning rate, initialize the parameters just mentioned then we can then train and optimize the model as we iterate through the optimization loop. Epochs will go through the loop over the training dataset to get close to optimal parameters as defined by the training method. Within that loop exists another loop that iterates over the training data and with every iteration increases the number of epochs) . We main to use to model that optimized so we will keep max_accuracy variable and keep updating it if there's a new higher accuracy than the one stored. The validate method will use the validation set of handwritten digits and compute how many images are predicted right out of the total number of images.

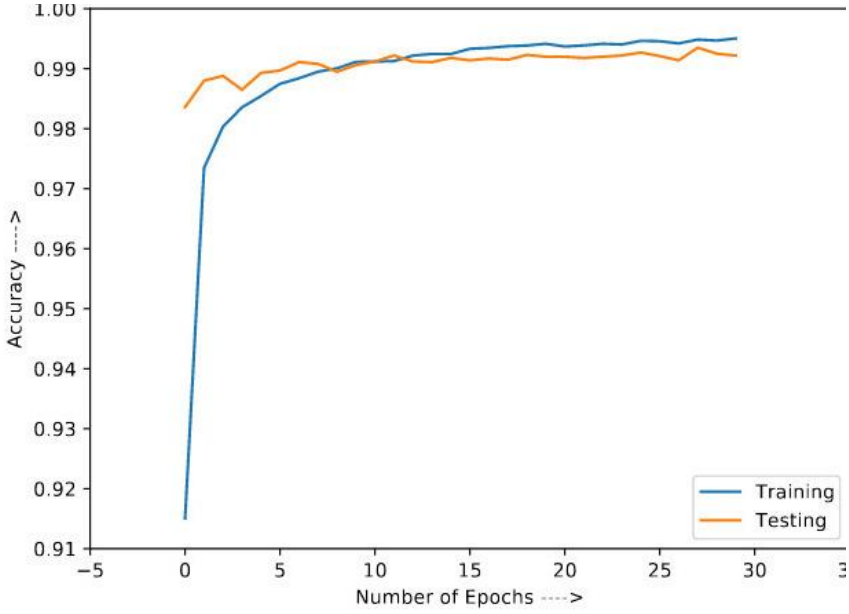
RESULTS

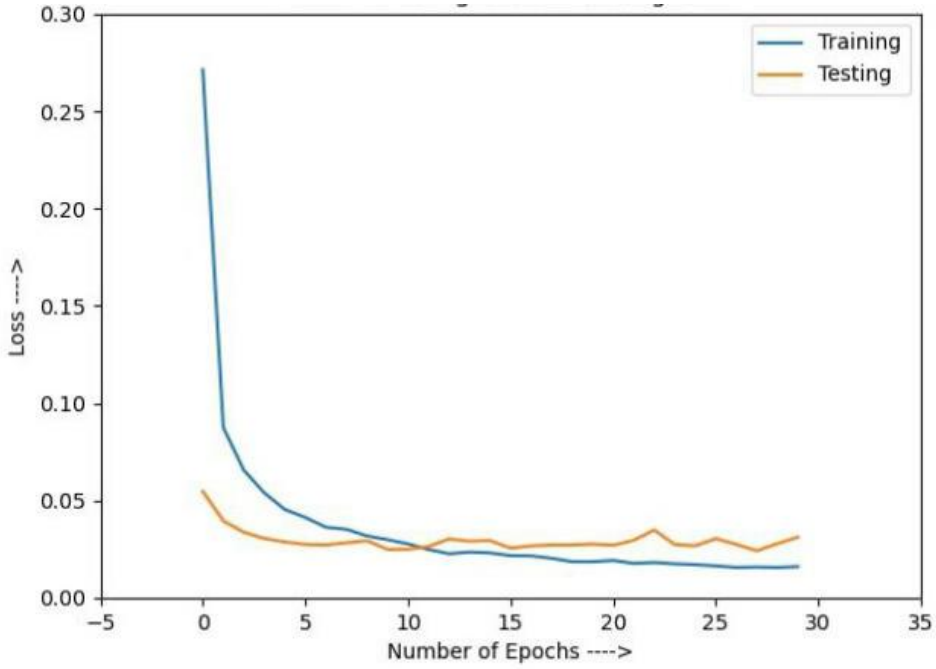
To main accuracy, I have used the same batch_size for both of the models and epochs



Graph representing accuracies of Classifier{2} .py over some time

CNN behaviour as number of epochs increases





Conclusion

The CNN model performs better than the standard NN, a few things may be a result of this. This model uses 2D convolutional layers which makes it way easier to process and manipulate data. CNNs are able to reduce the number of parameters without losing the quality of the model. The model also iterates quite thoroughly over the training data which makes way easy for it to make correct predictions and fully learn the model. Adam optimizer is also quite of help in assisting with the model's efficiency. Its accuracy is approximately 99% which shows that its good & reliable network and its most likely to predict the correct data

Classifier{1}.py begins with a very small accuracy but the model builds up quite impressively and predicts data correctly. It proves that SGD does generalize data in a much better way. SGD's convergence path is noiser. The accuracy isnt too high and its not too low. The data is trained well over the num_of epochs .

Both models used Cross Entropy Loss and had the sane number of epoch and batch_size , after considering this. It is safe to say Classifier{2} is a much more efficient model.

References :

1. Understanding of Convolutional Neural Network (CNN) — Deep Learning(Prabhu, 4 March 2018) Prabhu's article

2. The Architecture and Implementation of LeNet-5(Vaibhav Khandelwal, 29 July 2020) link

3. https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

4. https://medium.com/@krishna.ramesh.tx/training-a-cnn-to-distinguish-between-mnist-digits-using-pytorch-620f06aa9ffa