Neural Networks

Project Report

January 15, 2021

Performed by: Mansi Nanavati (EE19BTECH11036)
Nandita LT (CS19BTECH11051)

1 Feed Forward Neural Network

$$y(\mathbf{x}, \mathbf{w}) = f\Big(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\Big) \tag{1}$$

Linear Combinations of Input Data

$$a_j = \sum_{i=1}^{D} w_{ji}^1 x_i + w_{j0}^1 \tag{2}$$

Activation Function (Differentiable, Non-Linear)

$$z_j = h(a_j) \tag{3}$$

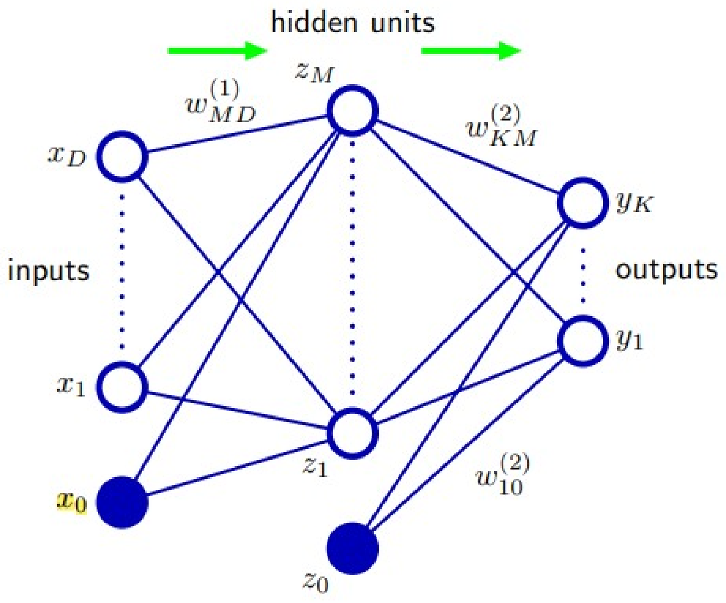


Figure 1: Two-Layer Feed Forward Neural Network

1.1 Weight Symmetries

By changing the signs of a particular group of weights (and a bias), the input–output mapping function represented by the network is unchanged.
For M hidden units, there will be M such ‘sign-flip’ symmetries, and thus any given weight vector will be one of a set 2^M equivalent weight vectors.

Interchanging the values of all of the weights (and the bias) leading both into and out of a particular hidden unit with the corresponding values of the weights (and bias) associated with a different hidden unit. Again,leaves the network input–output mapping function unchanged, but it corresponds to a different choice of weight vector.
For M hidden units, any given weight vector will belong to a set of M! equivalent weight vectors associated with this interchange symmetry, corresponding to the M! different orderings of the hidden units.

The network will therefore have an overall weight-space symmetry factor of M!2^M.

2 Convolutional Networks

Convolutional networks provide a way to specialize neural networks to work with data that has a clear grid-structured topology and to scale such models to very large size.
Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

2.1 Convolution Operator

We learnt that the convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the layer, and the different channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters. Filters encode specific aspects of the input data: at a high level, a single filter could encode the concept “presence of a face in the input,” for instance.

2.2 Features of CNN

- Sparse interactions
- Parameter sharing
- Equivariant Representation
- Pooling

We also went through the stages of pooling and understood the different kinds of pooling through the examples. We saw how max pooling works and how it is being majorly used now a days.

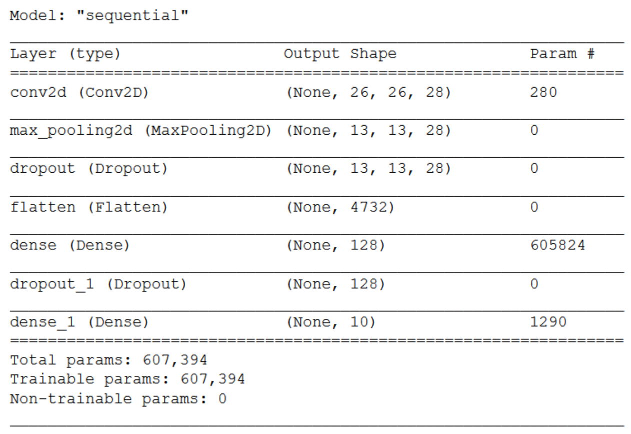
2.3 Kernels

CNNs have sparse interactions, accomplished by making the kernel smaller than the input, which makes them highly useful to highlight or suppress specific features while classifying. This makes it easier to visualize kernels and understanding their purpose.

3 Digit Recognition (MNIST Dataset)

3.1 Using Tensorflow

All the functions are predefined in the libraries, one only has to specify the input dataset and the model type. We experimented with various type of filters and number of neurons in tensorflow. It is quite fast and easy but it doesn't give an insight into the working of the neural network such as forward and back propagation, pooling, etc.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 28)        280
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 28)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 28)        0
_________________________________________________________________
flatten (Flatten)            (None, 4732)              0
_________________________________________________________________
dense (Dense)                (None, 128)               605824
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Summary of the model implemented

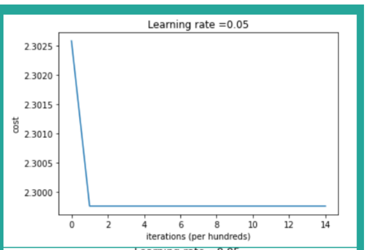
3.2 Using Numpy

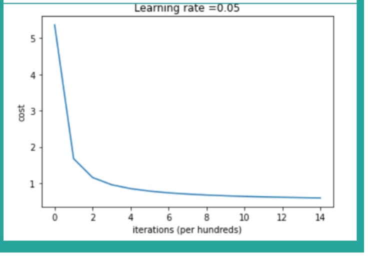
Various functions were created to perform operations such as creating a model, initializing parameters (weights and biases), forward propagation, activation functions such as soft-max and ReLU, computing error, updating parameters, and back propagation.
The MNIST dataset was used for this experiment, which comprised of both training and testing data. We could not use all the images in the dataset as the program would become extremely slow, hence we picked first 5000 images for training. The accuracy of the model has also been calculated. It was observed that the accuracy was quite lower than the model using Tensorflow. This is due to the reduced size of training dataset.
Another obvious observation was made that the model based on Numpy took quite a long time to train. As a result of this deeply nested iteration, processing one epoch of the MNIST data set with a one-ConvLayer CNN with 32 filters would require 28 trillion NumPy calculations. NumPy is indeed fast, but executing literally trillions of small NumPy calculations requires an absurdly long period of time.

3.2.1 Different parameters using Numpy

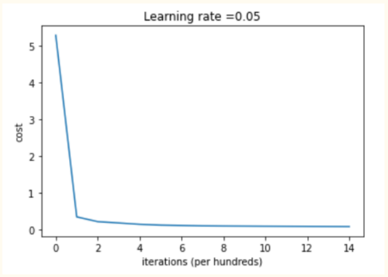
Initialization

- Case 1: Initializing all weights and biases with ones in matrices/vectors. The cost function remains the same beyond 100th iteration. Accuracy 11.26% and 12.6%
- Case 2: Giving heavier weight. Accuracy is terrible: 82.08% and 75.7%, compared to 97.82% and 87.2%





Learning rate

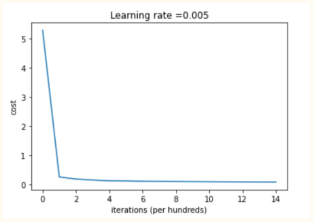


Cost function observed over 1500 iterations based on Original learning rate (0.05)

Learning rate

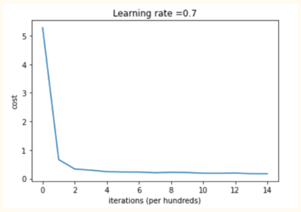
Learning rate = 0.005

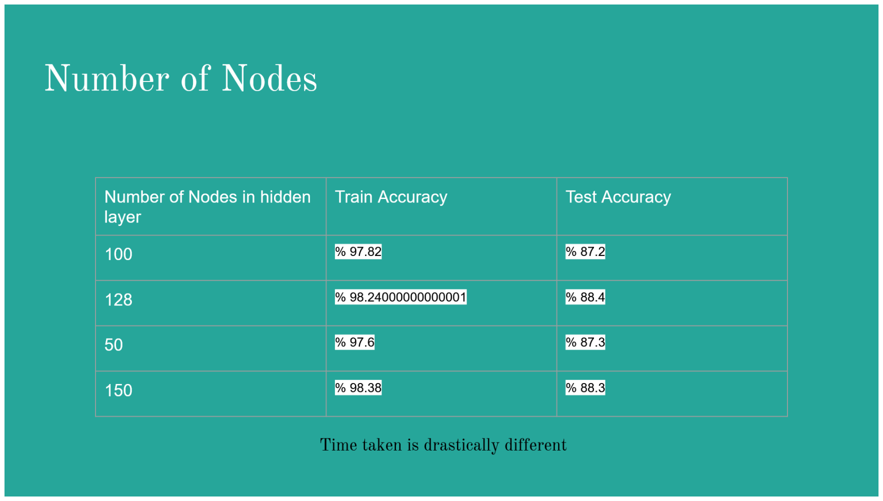
There is not much difference in this graph and the previous one. But with smaller learning rate, the model takes time to adapt to the problem. The train accuracy is good (98.08%) but test accuracy not so much (86.1%).



Learning rate = 0.7

The graph begins to converge faster than usual. But the accuracy is not good. It is 94.92% and 87.6% compared to 97.82% and 87.2%





Number of Nodes

Number of Nodes in hidden layer	Train Accuracy	Test Accuracy
100	% 97.82	% 87.2
128	% 98.24000000000001	% 88.4
50	% 97.6	% 87.3
150	% 98.38	% 88.3

Time taken is drastically different

4 Visualizing the Network

We broke down the components of the convolutional network by printing out the results of intermediate filters and layers. Visualizing intermediate activations consists of displaying the feature maps that are output by various convolution and pooling layers in a network, given a certain input (the output of the activation function). This gives a view into how an input is decomposed into the different filters learned by the network. You want to visualize feature maps with three dimensions: width, height, and depth (channels). Each channel encodes relatively independent features, so the proper way to visualize these feature maps is by independently plotting the contents of every channel as a 2D image.

4.1 Experimenting with Shapes and Digits

Initially, we followed the example of printing out intermediate layers of CNN based on shapes (square, triangle and circle) as mentioned in the book Deep Learning with Python by François Chollet. When input image is fed, the model returns the values of the layer activations in the original model and these values are converted to set of images as shown below.

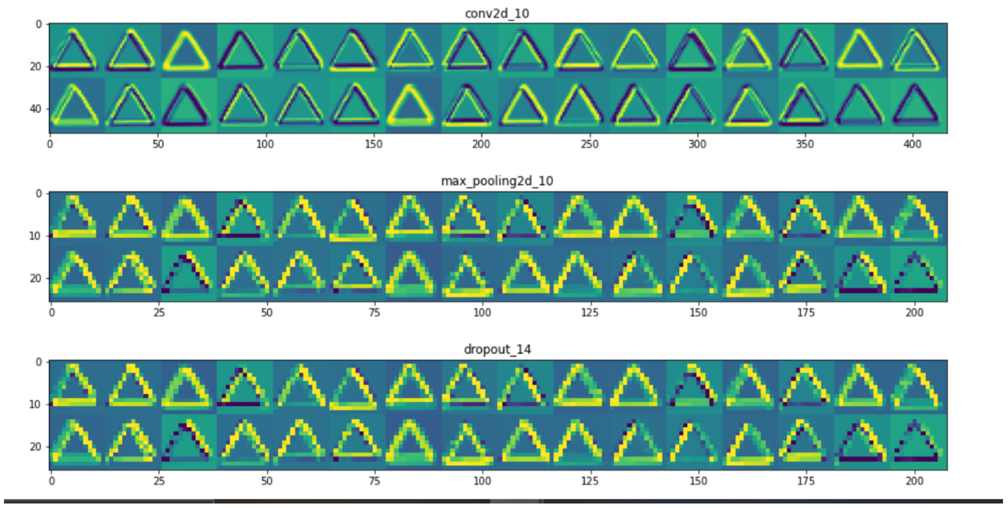


Figure 3

So the first observation we made was that in the first layer, the input image is sort of retained. And the output of each filter is highlighting a special feature individually.

4.2 Analyse Different Filter Outputs

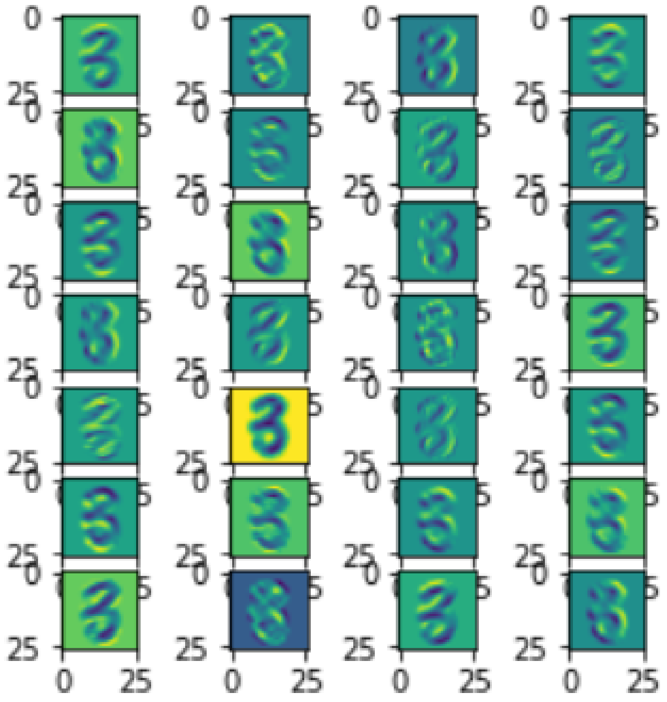


Figure 4: Output of the set of filters in first layer

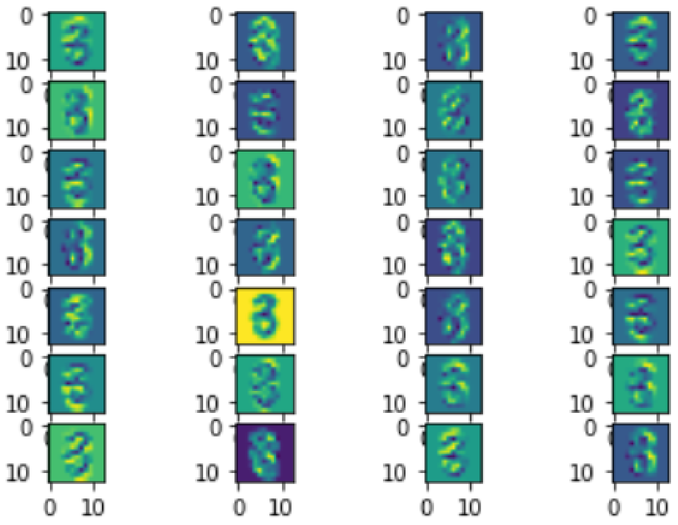
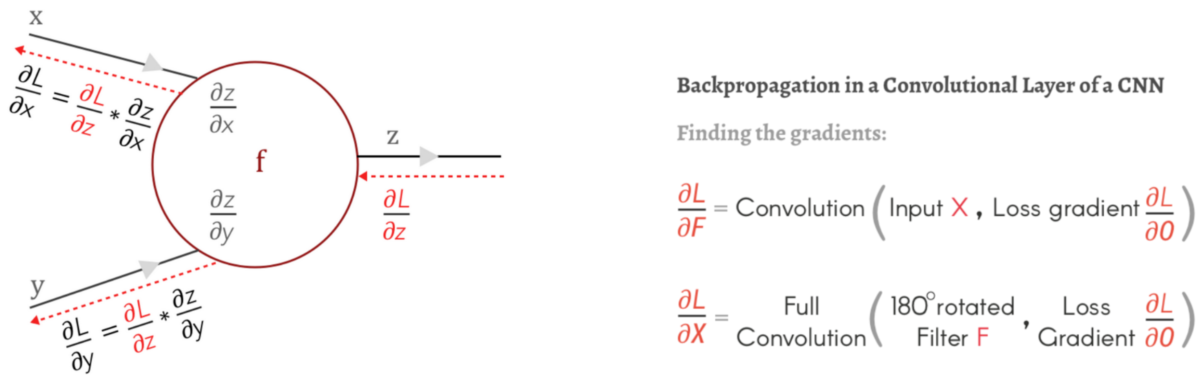


Figure 5: Output of the set of filters in second layer

4.3 Back Propagation

We also tried to visualize the layers and back-prop more. While reading, we realized that the back-propagation in CNNs could also be viewed as a convolution by a 180 degree flip of the filter.


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial y}$
Backpropagation in a Convolutional Layer of a CNN
Finding the gradients:
$\frac{\partial L}{\partial F}$ = Convolution (Input X, Loss gradient $\frac{\partial L}{\partial O}$)
$\frac{\partial L}{\partial X}$ = Full Convolution (180° rotated Filter F, Loss Gradient $\frac{\partial L}{\partial O}$)

(a) (b)

Figure 6: Back-propagation visualized

5 DFT Spectrum

The Discrete Fourier Transform of an image represents the image in the Fourier or frequency domain. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image. Fast Fourier Transform (FFT) is employed instead of Discrete Fourier Transform as it has lesser computation and is faster.

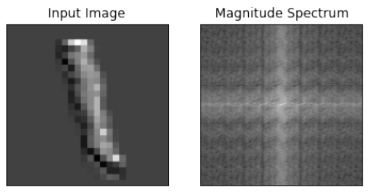
5.1 Spectrum of Output at each Layer

We printed out the DFT spectrum of a single filter at the 2nd layer of the digits as mentioned in the figure below. Certain observations were made:

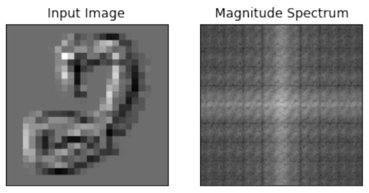
a. DFT spectrum of images with similar properties such as a slant line (in digits 4 and 7) exhibit similarities in the background.

b. DFT spectrum of images with completely opposite features (example: digits 8 and 1 or 8 and 7) have absolutely no similarity.

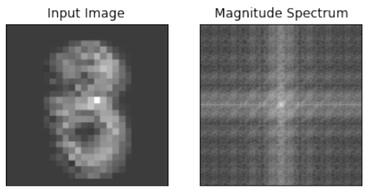
To explain our observations, we found out that certain features such as curves and lines can be identified by the DFT spectrum as these features leave certain marks on the spectrum. We also came across a paper Spectral Representations for Convolutional Neural Networks by Oren Rippel and Jasper Snoek where it is mentioned that spectral pooling using Fourier Transforms could be employed for faster convergence of CNN while training.



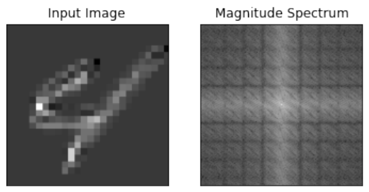
(a)



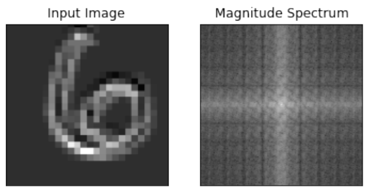
(b)



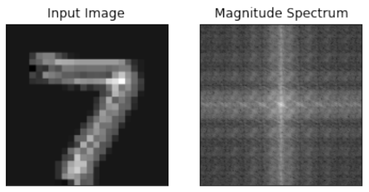
(c)



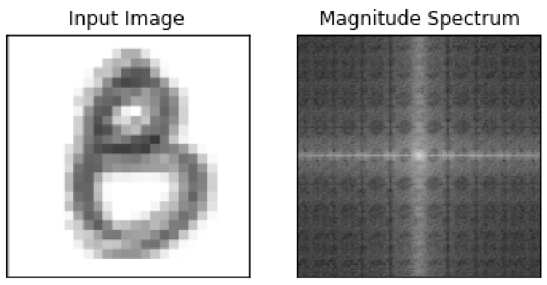
(d)



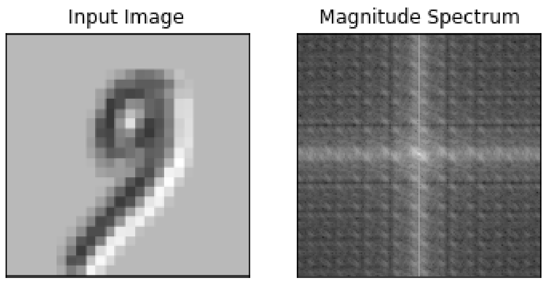
(e)



(f)



(g)



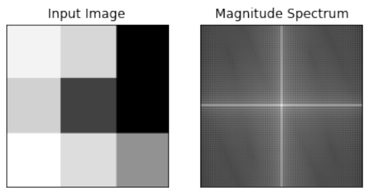
(h)

Figure 7: DFT spectrum of the outputs at the 2nd layer

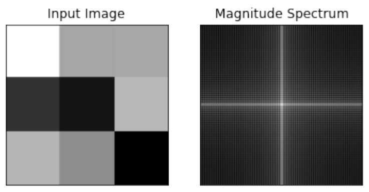
All the results show that the images contain components of all frequencies, but that their magnitudes get smaller for higher frequencies. Hence, low frequencies contain more image information than the higher ones. The transform images also tell us that there are two dominating directions in the Fourier images, one passing vertically and one horizontally through the center. These originate from the regular patterns in the background of the original images.

5.2 Spectrum of Filter

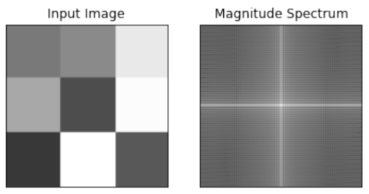
By visualising the learned weights we can get understand how well our network has learned. For example, if there are a lot of zeros, then we'll know we have many dead filters that don't improve the network, hence with this observation we understand if the model needs some pruning for model compression. Here is the DFT spectrum of certain filters in the model in 1st layer.



(a)



(b)



(c)

Figure 8: DFT spectrum of the filters at 1st layer

The Github repo for all the experiments can be viewed here.

6 References

a. Deep Learning Book

b. Deep Learning with Python by François Chollet

c. CNN using Numpy

d. Fourier Transform

e. Visualizing Filters