A Report on Convolutional Neural Networks

Prepared for

Dr. Navneet Goyal

Instructor In-charge, Machine Learning (BITS F464)
Department of Computer Science & Information Systems
Birla Institute of Technology & Science
PILANI, 333031 Rajasthan, INDIA

By

Aryan Madaan (2020A1PS0222P)
Rayman Singh Narwal (2020A1PS1058P)
Grandhi Surya Koteswara Rao (2021H1030125P)

Birla Institute of Technology & Science
PILANI, 333031 Rajasthan, INDIA

8th December 2022

Acknowledgment

We are very grateful to our professor Dr. Navneet Goyal who gave us a chance to work on this project. We would like to thank him for giving us valuable suggestions and ideas. We would also like to thank our college for providing all the necessary resources for the project. We would like to thank everyone involved in this project who helped us with their suggestions to improve the project.

Abstract

Convolutional neural network (CNN), a subclass of artificial neural networks that have gained dominance in a number of computer vision tasks, is gaining popularity in a number of fields, including radiology. Using a variety of building blocks, including convolution layers, pooling layers, and fully connected layers, CNN is intended to automatically and adaptively learn spatial hierarchies of features through backpropagation. This review article provides insight into CNN's fundamental ideas and how they apply to various radiological tasks. It also discusses its difficulties and potential applications in the future of radiology. This article will also discuss methods to reduce the problems of small datasets and overfitting when using CNN for radiological tasks. To fully utilize CNN's potential in diagnostic radiology and enhance radiologists' performance while enhancing patient care, it is crucial to be aware of its principles, benefits, and limitations.

Table of Contents

1. Convolutional Networks

Deep learning techniques are based on neural networks, a branch of machine learning. They are made up of node levels, each of which includes an input layer, one or more hidden layers, and an output layer. Each node has a threshold and weight that are connected to one another. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next tier.

There are other kinds of neural nets, which are utilized for diverse use cases and data types, while we mainly concentrated on feedforward networks in that article. Recurrent neural networks, for instance, are frequently used for speech and natural language processing, but convolutional neural networks (also known as CNNs or ConvNets) are more frequently employed for classification and computer vision applications. Before CNN's, identifying objects in images required the use of laborious, manual feature extraction techniques. Convolutional neural networks, on the other hand, now offer a more scalable method for classifying images and recognizing objects by using matrix multiplication and other concepts from linear algebra to find patterns in images. However, they can be computationally taxing, necessitating the use of graphics processing units (GPUs) when modeling them.

Convolutional networks sometimes referred to as convolutional neural networks, or CNNs (LeCun, 1989), are a specific type of neural network for data processing that have a recognized grid-like architecture. A few examples include time-series data, which may be visualized as a 1-D grid that collects samples at predetermined intervals, and picture data, which can be visualized as a 2-D grid of pixels. In practical applications, convolutional networks have been incredibly successful. The term "convolutional neural network" denotes the use of a mathematical procedure known as convolution. A specific sort of linear process is called convolution. Convolutional networks are simply neural networks with convolution used in at least one layer rather than standard matrix multiplication.

2. How do convolutional neural networks work?

Convolutional neural networks outperform other neural networks when given inputs such as images, voice, or audio, for example. There are three basic categories of layers in them:

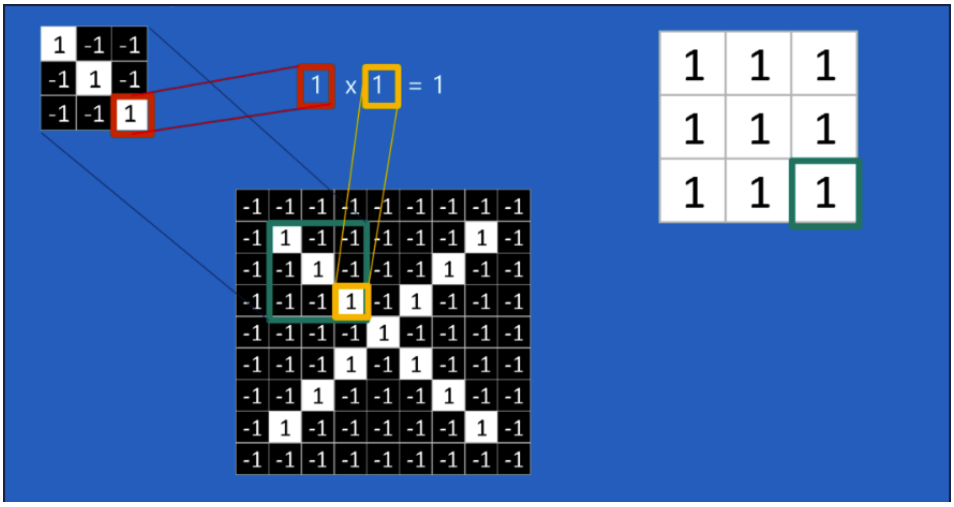
- Convolutional layer
- Pooling layer
- FC (fully connected) layer

A convolutional network's first layer is the convolutional layer. The fully-connected layer is the last layer, even though convolutional layers, further convolutional layers, or pooling layers, can come after it. The CNN becomes more complicated with each layer, detecting larger areas of the image. Early layers emphasize basic elements like colors and borders. The larger features or shapes of the object are first recognized when the visual data moves through the CNN layers, and eventually, the intended object is recognized.

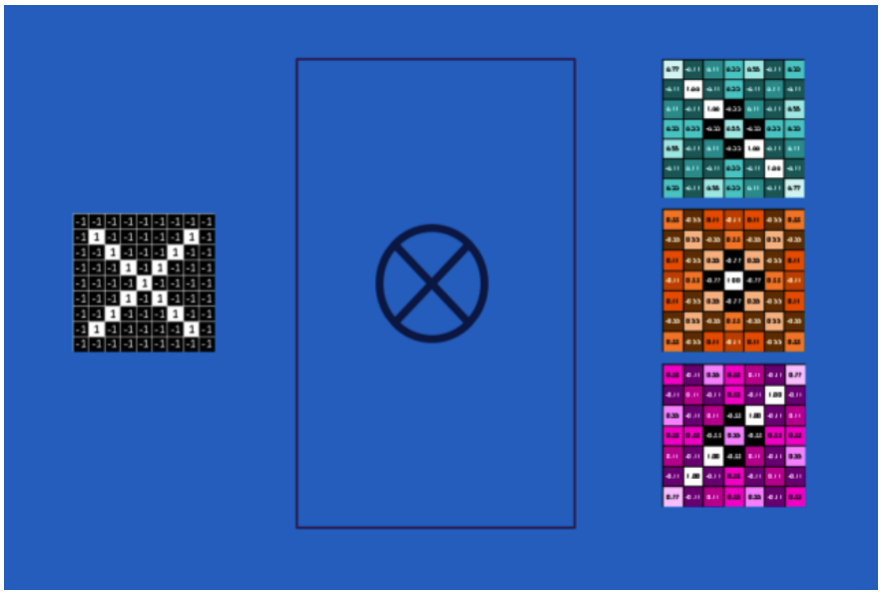
2.1 Convolutional Layer

When given a new image, CNN checks these traits everywhere and in every conceivable position because it is unsure of exactly where they will match. We turn a feature into a filter by determining the match to it across the entire image. Convolutional Neural Networks derive their name from the type of mathematics we employ to accomplish this.

A sixth-grader would not find the algebra underlying convolution to be difficult. Simply multiply each feature pixel by the value of the corresponding image pixel to see how well a feature matches a patch of the image. Subtract the entire number of pixels in the feature from the overall answer and add the results. 1 * 1 equals 1 if both pixels are white (1 value). The equation (-1) * (-1) Equals 1 if both are black. Each matched pixel yields a 1 in either case. Likewise, a -1 represents any mismatch. If every pixel in a feature is identical, adding all of those pixels together and dividing by the overall number of pixels results in a value of one. Similar to the last example, the response is a -1 if none of the pixels in a feature match the picture patch.



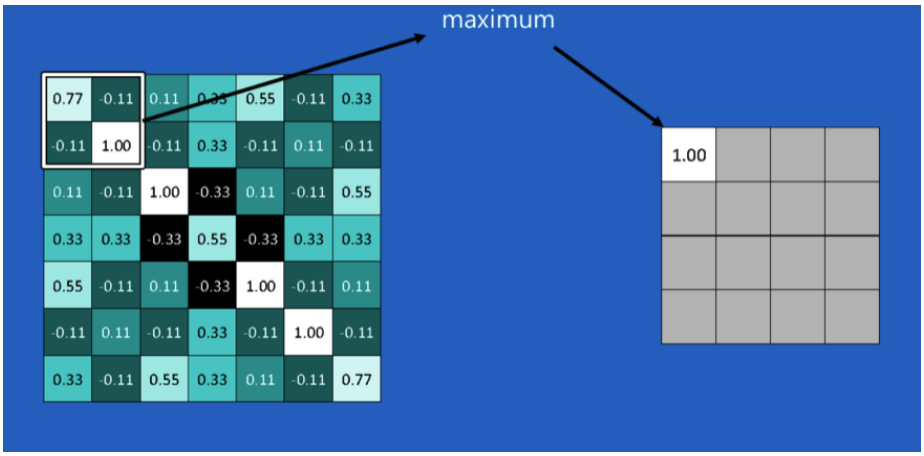
We continue this procedure in order to finish our convolution, aligning the feature with each potential image patch. On the basis of the location of each patch in the image, we may take the result of each convolution and create a new two-dimensional array from it. The image we started with has been filtered to create this map of matches. It shows the location of the feature within the image on a map. Strong matches are shown by values close to 1, strong matches for the photographic negative of our feature are indicated by values close to -1, and no matches of any kind are indicated by values close to zero.



The complete convolution procedure must then be repeated for each of the additional features. A collection of filtered photos, one for each of our filters, is the outcome. Conveying this entire set of convolution processes as a single processing step is practical. This is referred to as a convolution layer in CNNs, indicating that additional layers will be added to it soon.

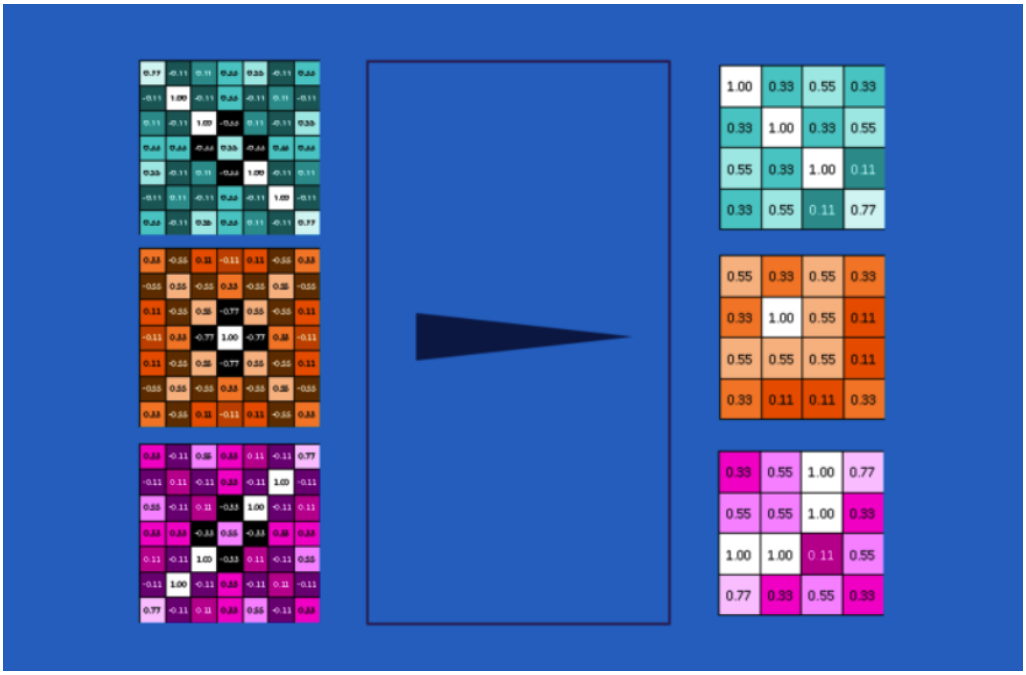
It's understandable how CNNs gained a reputation for being computation hogs. The number of additions, multiplications, and divisions can pile up quickly even though we can draw our CNN on the back of a napkin. They scale linearly with the number of pixels in the image, the number of pixels in each feature, and the total number of features, to use mathematical terminology. With so many contributing components, it's simple to multiply this issue by millions without breaking a sweat. It should come as no surprise that chip makers are now producing specialist processors in an effort to keep up with CNN needs.

2.2 Pooling Layer



Pooling is another effective strategy employed by CNN's. Pooling is a technique for taking huge photos and condensing them while retaining the most crucial details. The mathematics behind pooling is quite difficult. It involves moving a small window across an image while taking the maximum value out of the window at each step. In actuality, steps of 2 pixels and windows with 2 or 3 sides work effectively.

An image has around one-fourth the number of pixels after pooling. The best fits of each feature within the window are maintained because it retains the maximum value from each window. This indicates that it doesn't really care where the feature fits in the window as long as it does. As a result, CNN's can determine if a feature is present in an image without worrying about its location. This assists in resolving the issue of computers becoming overly literal.



A pooling layer is nothing more than the process of applying pooling to a single image or to a set of images. The result will be the same amount of photos, but with lower pixel counts. This aids in controlling the computational load. Life is much simpler for everyone downstream when an 8-megapixel image is reduced to a 2 megapixel image.

There are mainly 2 types of pooling:

- Max pooling: The filter chooses the pixel with the highest value to send to the output array as it advances across the input. As a side note, this method is applied more frequently than average pooling.
- Average pooling: The filter calculates the average value inside the receptive field as it passes across the input and sends that value to the output array.

The pooling layer loses a lot of information, but it also offers CNN a number of advantages. They lessen complexity, increase effectiveness, and lower the risk of overfitting.

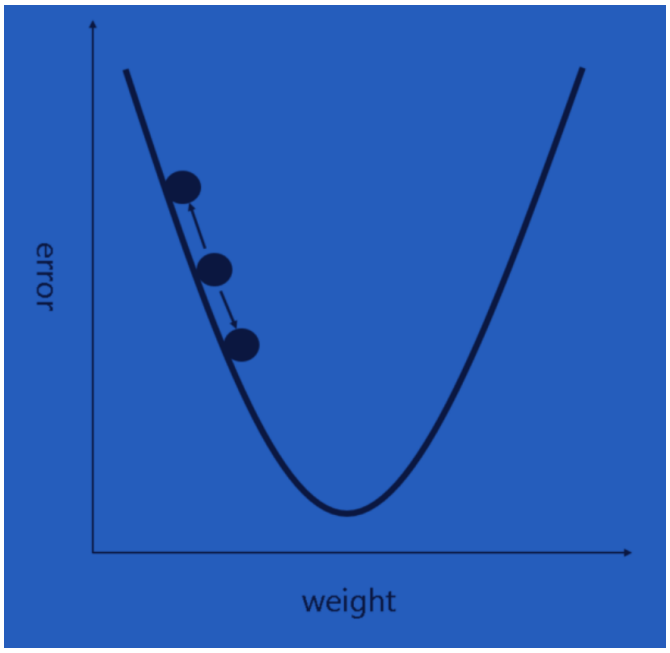
2.3 Fully Connected Layer

The fully-connected layer is exactly what its name implies. As was already noted, partially connected layers do not have a direct connection between the input image's pixel values and the output layer. In contrast, every node in the output layer of the fully-connected layer is directly connected to a node in the layer above it.

Based on the features that were retrieved from the preceding layers and their various filters, this layer conducts the classification operation. FC layers often utilize a softmax activation function to categorize inputs appropriately, producing a probability ranging from 0 to 1. Convolutional and pooling layers typically use ReLu functions.

Due to the fact that their outputs (a list of votes) closely resemble their inputs, fully connected layers, like the others, can be stacked (a list of values). In reality, many fully connected layers are frequently piled on top of one another, with each intermediary layer casting votes for fictitious, "hidden" categories. Effectively, as layers are added, the network is able to learn ever more complex feature combinations that aid in decision-making.

3. Backpropagation



Although our story is starting to take shape, there is a giant gaping hole—where do features originate from? Additionally, where do we find the weights in our fully connected layers? CNN's would be considerably less well-liked than they are if they had to be completely manually selected. Fortunately, a piece of machine learning magic known as backpropagation takes care of this for us.

We require a set of photos for which we already know the solution in order to conduct backpropagation. This indicates that a kind person carefully looked over tens of thousands of pictures and gave them the labels X or O. We utilize these with a CNN that has not been trained, thus each pixel of each feature and each weight in each fully connected layer is set to a random value. Then, one by one, we begin to pass images through it.

A vote is generated for each image that CNN processes. The degree of mistake in the vote, or wrongness, reveals the quality of our features and weights. Then, to reduce the inaccuracy, the features and weights can be changed. The new inaccuracy is calculated after each small upward and downward adjustment to each value. Whatever correction reduces the error is preserved. The new weights provide an answer that works marginally better for that image after performing this for each feature pixel in each convolutional layer and each weight in each fully connected layer. The following image in the collection of labeled images receives the same treatment. Unlike patterns that appear often throughout a large number of photographs, quirks that only appear in a single image are soon forgotten. These values stabilize to a set that performs admirably in a variety of situations if you have enough annotated images.

Backpropagation, as is undoubtedly clear, is another expensive processing phase and a further justification for specialist computing hardware.

You should note that backpropagation is a wonderfully local process. Every gate in a circuit design has some inputs and can immediately compute two things: 1. its output value and 2. the local gradient of its output with regard to its inputs. Notice how the gates, which are integrated into a complete circuit, are capable of performing this entirely independently. The gate will eventually discover the gradient of its output value on the overall circuit's final output during backpropagation, but, after the forward pass has ended. The chain rule dictates that the gate adds that gradient to each gradient it typically computes for all of its inputs.

By using the example once more, let's get a sense of how this functions. Inputs [-2, 5] were provided to the add gate, which calculated output 3. The gate has a local gradient of 1 for both of its inputs because it is computing the addition operation. The remainder of the circuit calculated the outcome, which is -12. The add gate (which is an input to the multiplication gate) learns that the gradient for its output was -4 during the backward pass, in which the chain rule is applied recursively backward through the circuit. If we anthropomorphize the circuit as desiring to output a higher value (which might aid in intuition), we can consider the circuit as "wanting" the add gate's output to be lower (owing to the negative sign) and with a force of four. The add gate takes that gradient and multiplies it by each of the local gradients for its inputs in order to chain the gradient and continue the recurrence (making the gradient on both x and y 1 * -4 = -4). The output of the add gate would decrease if x, and y decreased (responding to their negative gradient), which would cause the multiply gate's output to increase. This has the intended outcome.

Therefore, backpropagation can be conceptualized as gates talking to one another (through the gradient signal) about whether and how strongly they want their outputs to increase or decrease in order to enhance the value of the final output.

4. About Code

The Modified National Institute of Standards and Technology dataset is referred to by the term MNIST. It is a collection of 60,000 handwritten single numbers between 0 and 9 that are 28 by 28 pixel small square grayscale images.
We are using around 40,000 datapoints for testing our program

4.1 Data Structures created:

- **Mat -** This is a 2-D Array, where we store input and use it as input for convolution and pooling layer
- **Flat -** This is a 1-D array to pass inputs to the dense layers
- **Layers** - Union structure that can hold either convolution, pooling, or dense layers
- **MyLayer** - Structure which maintains the order of the layers which are present in our CNN model. (Follows a structure similar to the doubly linked list)

4.2 Classes in the current implementation:

1) Convolution Class - input matrix, no. of kernels, kernel size, stride, error rates, del_weights(value by which we update the kernel values)
    - Constructor-It creates the kernels and it makes sure the kernels created are normalized using Standard Uniform distribution
    - Deflatten - If the convolution layer is right beside the dense layer during backpropagation it makes sure the error rates passed are distributed to the kernel values
    - Convolutional Operation - This applies the kernel to the input and gives the input to the next layer. The size of the output is defined by

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

    - Where W is the input size, F is the kernel size, P is padding(we are assuming it to be 0), and S is stride.
    - Error Rate: TBD
    - Update Kernel Weights - Here we are calculating the del weights by using the error rates and learning rate and updating the kernel values.

2) Pooling Class - input matrix, kernel size, stride, type of pooling, error rates

- Error propagation
- Deflatten - If the pooling layer is right beside the dense layer during backpropagation it makes sure the error rates passed are distributed to the kernel values
- Pooling operation - here based on the type of pooling we want we aggregate the input using aggregations functions of max, sum, average

3) Perceptron - input, weights, bias
    - Perceptron operation - we make sure that the weights of the perceptron follow uniform std distribution. Then we multiply them with the input value and add bias to the resulting sum. Finally, we apply the activation function (here we are using the sigmoid function).
    - Set Bias - to update the bias during the backdrop

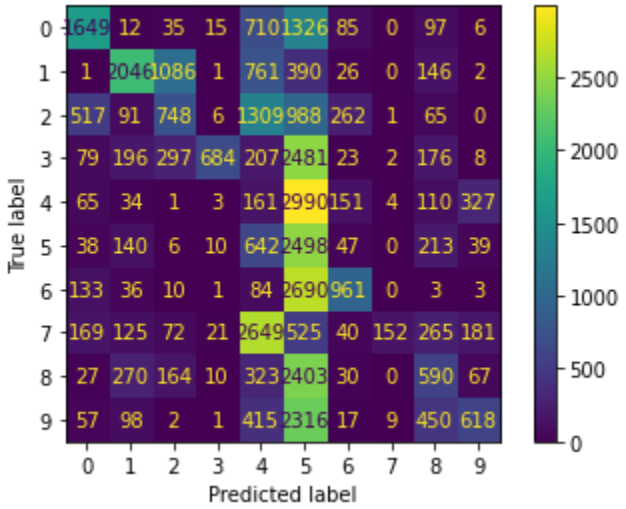
4) Dense - flattened input array, no. of perceptrons, error rates, flattened output array
    - Operation - here each input is passed to the perceptron in the dense layer, and perceptron operation is done at each perceptron and the output is stored in the dense object.
    - Update weights - Here we are updating the weights and the bias using the error rates obtained during the backpropagation and learning rate.

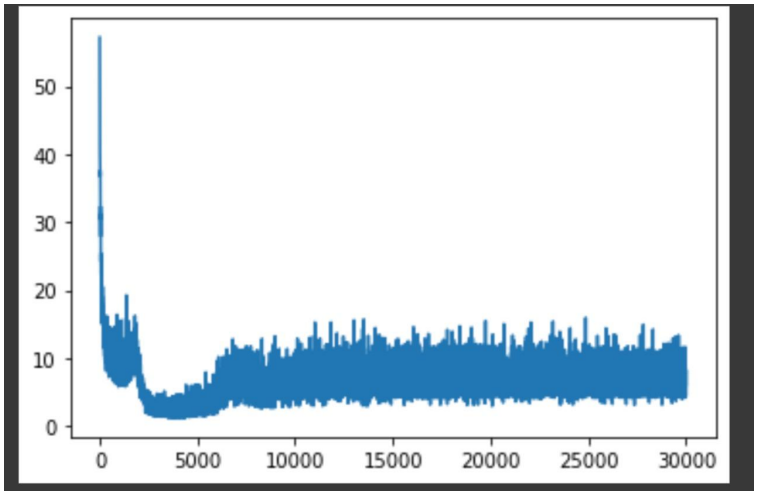
5) Layer Class - superclass which supports the creation of any type of layer, stores the error rates at each layer, and also has the pointers to maintain the order of the classes created
    - Constructor - takes the learning rate from the stdin
    - Create a layer - based on the type of layer we want the parameters are obtained from stdin and the object of the layer is created. This also has the doubly linked structure to maintain the order and once the layers we want are created
    - Layer Operation - based on the type of layer corresponding operation is called and for the last dense layer, we apply softmax.
    - Train - This is the backpropagation function of CNN. Here, we start from the last of the layers (tail) and calculate the error rate for the misclassified data items. Finally, we again start from the last of layers (tail) to update the weights at each layer.

Sample Run - 1

Parameter passed:

Number of iterations	30000
Test Data Size	30
Learning Rate	0.5
Number of kernels	12
Size of the convolution kernel	8
Dense Layer 1 #No. Of perceptrons	15
Dense Layer 2 #No. Of perceptrons	10

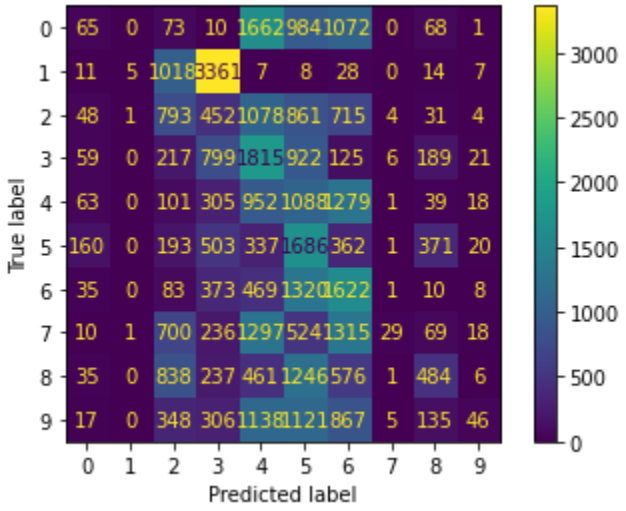


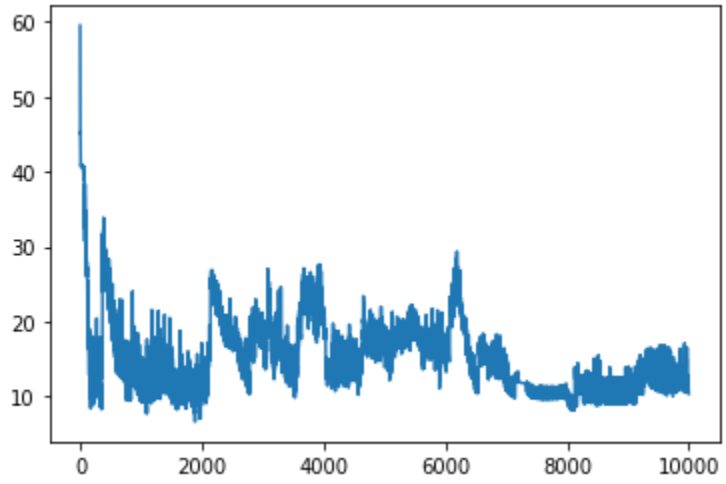


Sample Run - 2

Parameters passed:

Number of iterations	10000
Test Data Size	30
Learning Rate	0.5
Number of kernels	7
Size of the convolution kernel	5
Dense Layer 1 #No. Of perceptrons	15
Dense Layer 2 #No. Of perceptrons	10

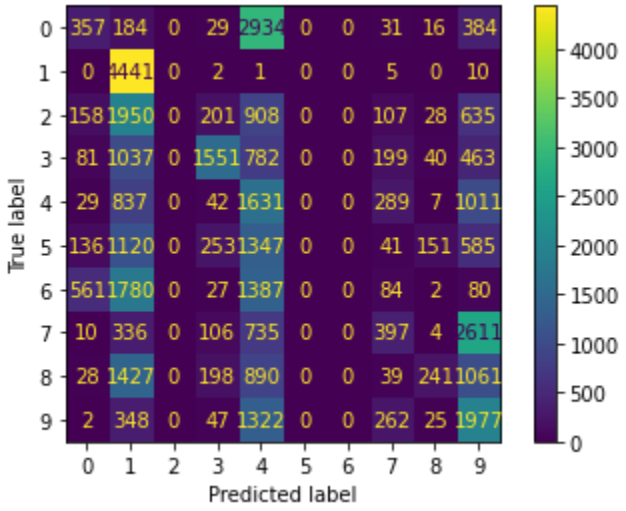


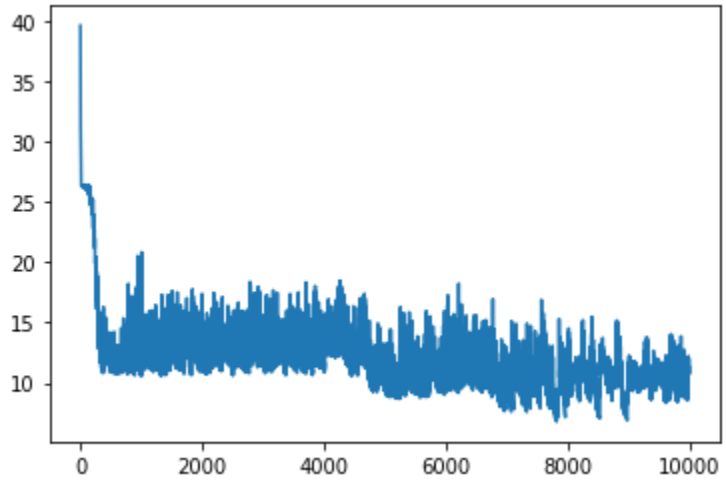


Sample Run - 3

Parameters passed:

Number of iterations	10000
Test Data Size	20
Learning Rate	0.5
Number of kernels	8
Size of the convolution kernel	7
Dense Layer 1 #No. Of perceptrons	15
Dense Layer 2 #No. Of perceptrons	10

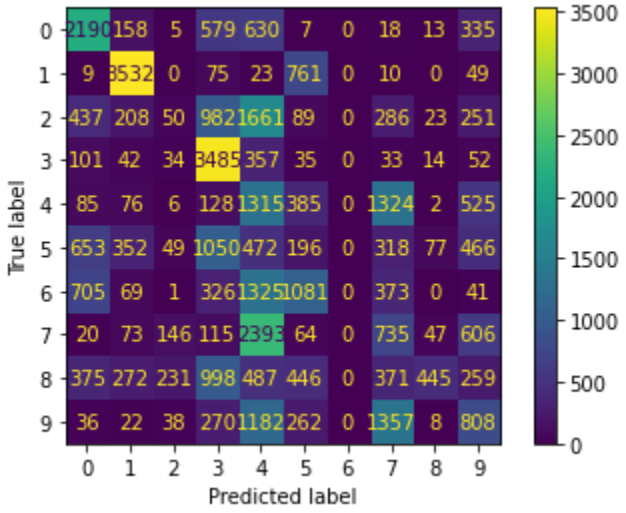


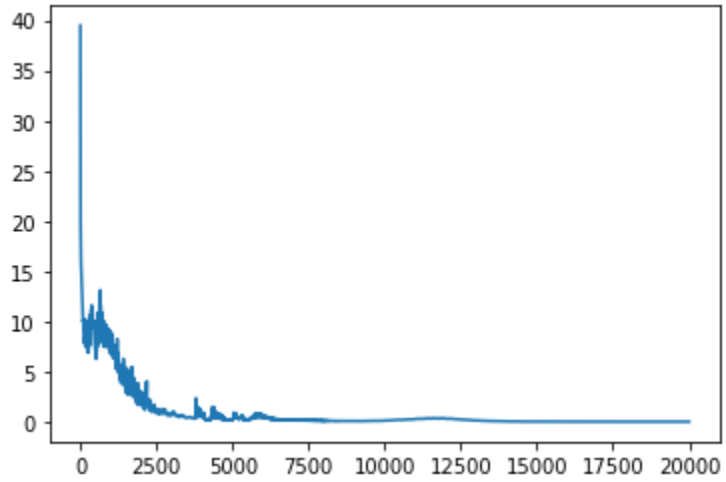


Sample Run - 4

Parameters passed:

Number of iterations	20000
Test Data Size	20
Learning Rate	0.05
Number of kernels	4
Size of the convolution kernel	7
Dense Layer 1 #No. Of perceptrons	15
Dense Layer 2 #No. Of perceptrons	10

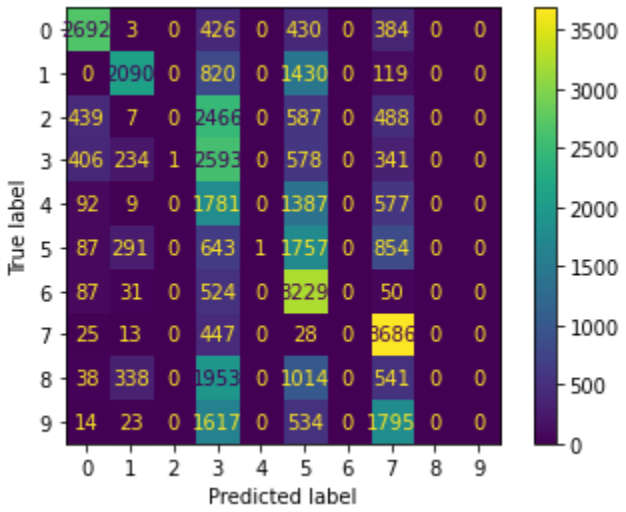


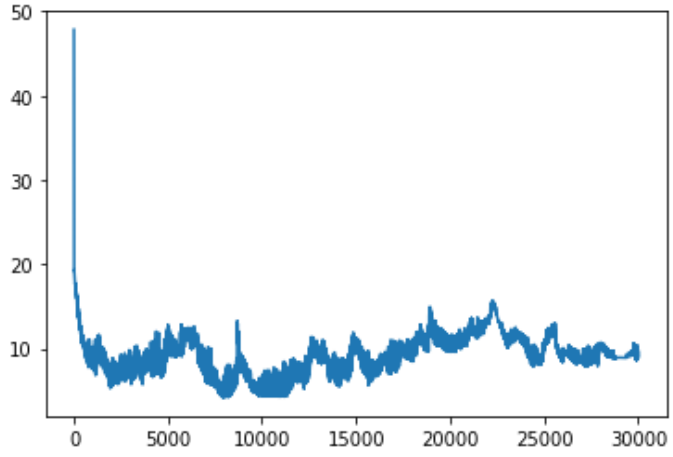


Sample Run - 5

Parameters passed:

Number of iterations	30000
Test Data Size	20
Learning Rate	0.01
Number of kernels	7
Size of the convolution kernel	7
Dense Layer 1 #No. Of perceptrons	20
Dense Layer 2 #No. Of perceptrons	15
Dense Layer 3 #No. Of perceptrons	10

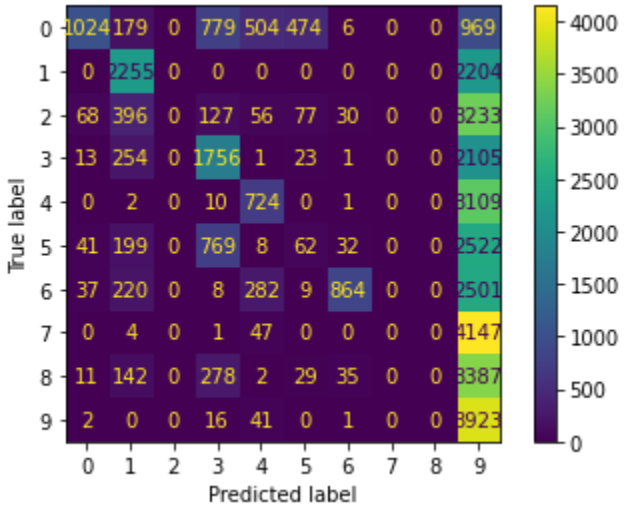


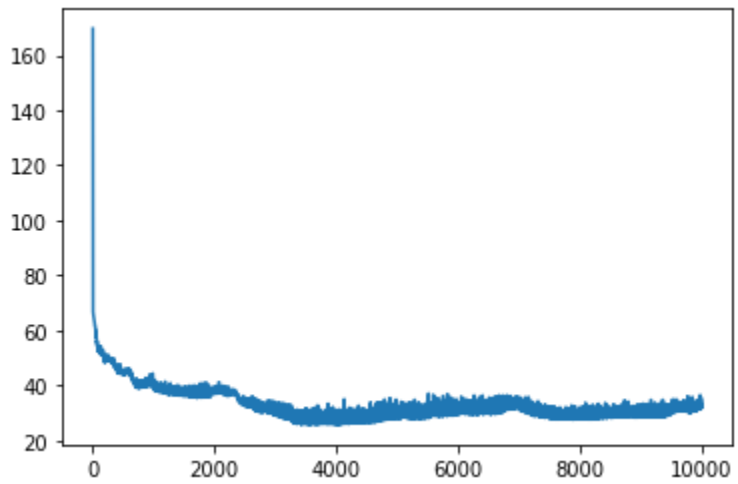


Sample Run - 6

Parameters passed:

Number of iterations	10000
Test Data Size	60
Learning Rate	0.05
Number of kernels	6
Size of the convolution kernel	5
Dense Layer 1 #No. Of perceptrons	5
Dense Layer 2 #No. Of perceptrons	10





References:

- https://www.ibm.com/cloud/learn/convolutional-neural-networks
- https://www.deeplearningbook.org/contents/convnets.html
- https://cs231n.github.io/convolutional-networks/#overview
- https://e2eml.school/how_convolutional_neural_networks_work.html
- https://cs231n.github.io/optimization-2/
- https://towardsdatascience.com/backpropagation-in-fully-convolutional-networks-fcns-1a13b75fb56a
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53