Project 3

Classification Using Neural Networks and Deep Learning (SVHN Dataset)

Sai Teja Vishal Jangala
sjangala@asu.edu
1218332037

I. Introduction:

This project is the implementation of Classification task using Convolution Neural Networks in Deep Learning. The dataset used here is a SVHN Dataset, which is Street View House Numbers dataset, which will contain all the picture of House Numbers collected from Google Street View images. There are 73257 training images and 26032 testing images, each image is a color image of the size 32x32. The Convolution Neural Networks, shortly called as CNN, are a part of Deep Learning algorithms which are very popular in the Image Classification. CNN depends on the concept of Convolution in mathematics and this very useful in obtaining the features from the image. These features are very important and are required in recognizing the image by the machine.

There are layers in CNN, they are convolution layer, ReLU layer (which is an activation function), Pooling layer and then finally comes the Fully Connecting Layer. CNN layer is different from a fully connected neural network, where a neuron is connected to every neuron in the layer before it. Where in CNN, a neuron in a layer is connected to a small region of the layer before it. The input images, which are part of training data is sent through each layer and the convolution happens at each layer followed by the pooling, and then to the fully connected layer. The architecture is explained in the further steps.

II. Features:

In the SVHN dataset, there are training samples and testing samples. Training samples are 73257 images, and the testing samples are 26032 images. For these samples, the labels are also provided. These images are RGB colored images of size 32x32. The labels will have 10 classes and each class denotes a digit. The dataset is loaded using scipy.io and then processed using Keras from Tensorflow.

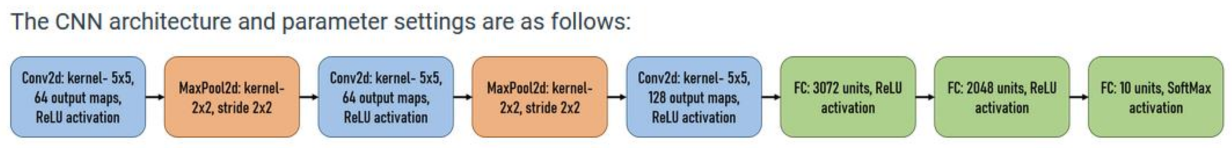
III. CNN Model and Training:

Important terms to be understood before going into the architecture are Convolution layer and Max Pooling layer.

Convolution Layer: A Convolution layer is a concept from mathematics that is used in Deep Learning to identify the features present in the image. This Convolution Layer is the application of the filter on the image, which is read as a matrix by the machine. This same filter is applied all over the image and creates a feature map.

Architecture:

The below is the Architecture Diagram:


The CNN architecture and parameter settings are as follows:
Conv2d: kernel- 5x5, 64 output maps, ReLU activation
MaxPool2d: kernel- 2x2, stride 2x2
Conv2d: kernel- 5x5, 64 output maps, ReLU activation
MaxPool2d: kernel- 2x2, stride 2x2
Conv2d: kernel- 5x5, 128 output maps, ReLU activation
FC: 3072 units, ReLU activation
FC: 2048 units, ReLU activation
FC: 10 units, SoftMax activation

There are 8 layers in the CNN model used in the project. These 8 hidden layers are added one after the other in a Sequential manner to create CNN architecture. Each of the hidden layer is explained below:

Hidden Layer 1: This layer is a Convolution layer, where 2D convolutions happen with a kernel size of 5x5 with 64 output maps, and with a stride of (1,1).
The activation function used here is ReLU same as discussed earlier.

Hidden Layer 2: This layer is a 2D Max-Pooling layer with a kernel size of 2x2 and a stride of 2x2.
The activation function is used here, and it is a ReLU activation.
Hidden Layer 3: This layer is a Convolution layer, where 2D convolutions happen with a kernel size of 5x5 with 64 output maps, and with a stride of (1,1).
The activation function used here is ReLU same as discussed earlier.

Hidden Layer 4: This layer is a 2D Max-Pooling layer with a kernel size of 2x2 and a stride of 2x2.
The activation function is used here, and it is a ReLU activation.

Hidden Layer 5: This layer is a Convolution layer, where 2D convolutions happen with a kernel size of 5x5 with 64 output maps, and with a stride of (1,1).
The activation function used here is ReLU same as discussed earlier.

Hidden Layer 6: This is a Fully Connected Layer, where neuron is connected to every neuron in the previous layer. This fully connected layer is implemented using a Dense layer with output space of 3072 units and ReLU activation.

Hidden Layer 7: Similar to Layer 6, this is also a Fully Connected Layer which is implemented using a Dense layer in Keras with output space of 2048 units. A ReLU activation is also used here.

Hidden Layer 8: This is the final layer in the architecture of our CNN model, which is also a Dense Layer in Keras implementation which has an output space of 10 units. However, a SoftMax function is used as Activation Function here.
The output space is 10 units, indicating that there are 10 outputs from this final/last layer. Each of the output unit indicates each class in the dataset (0-9 digits).

Parameters for Training:

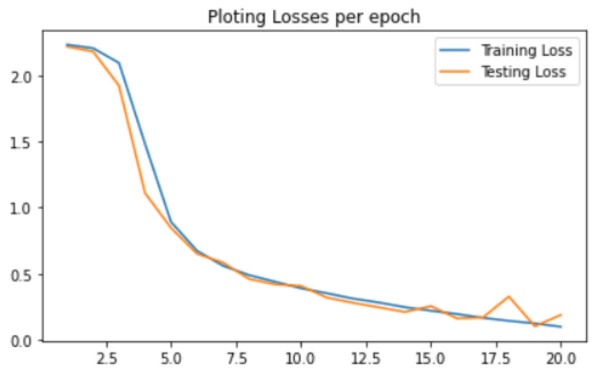
The below are the parameters used for training.
An SDG (Stochastic gradient descent) optimizer is used here for training the CNN model which is created from the 8 hidden layers as explained above.
The parameters are learning rate, which is 0.01 with momentum 0.5. With this learning rate, the model is trained with a batch size of 128 over 20 epochs. And at each epoch, the loss and accuracies are observed and recorded. The losses and accuracies are observed to reduce at each epoch for the training and validation dataset.
Now these recorded accuracies and losses for the learning and testing phases are plotted for each epoch which will be shown in the further in the results section.

IV. <u>Results and Plots:</u>

After evaluating the training, the model for 20 epochs, the losses and accuracies are printed for testing and training data.

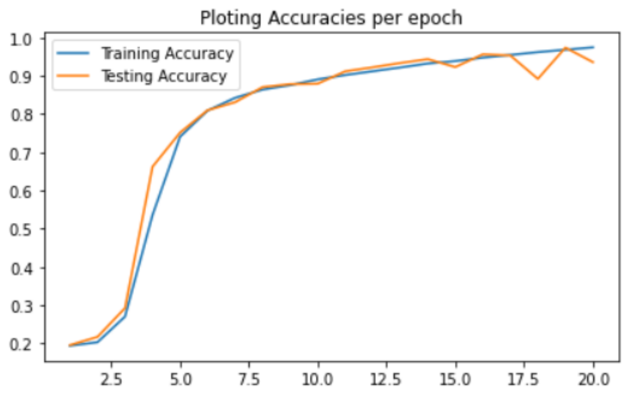
<u>The plots are displayed below:</u>

1.



X-axis represents number of epochs and y-axis represents the Loss. Blue line represents the Training loss, and the orange line indicated the Testing loss. It is observed that for every epoch, the loss is reduced.
At 20th epoch, the training loss is reported to be lesser than testing loss, by a very small amount.

2.



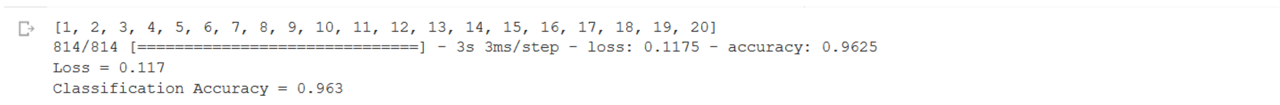
X-axis is number of epochs and y-axis represents the Accuracy. Blue line in the plot is the Training Accuracy, and orange line is Testing accuracy. In the plot above, we can see that for every epoch the accuracy increased and it at maximum in 20th epoch.
At 20th Epoch, the testing accuracy is reported to be greater than the testing accuracy, by a very small amount.

Results for Training data:

The Accuracy for Training Data:	0.9872
The Loss for Training Data:	0.0918

Results for Testing data:

The Accuracy for Testing Data:	0.9625
The Loss for Testing Data:	0.1175



```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
814/814 [==============================] - 3s 3ms/step - loss: 0.1175 - accuracy: 0.9625
Loss = 0.117
Classification Accuracy = 0.963
```