CSE 575 – STATISTICAL MACHINE LEARNING

Class # 18194 – Spring 2021 Instructor **– Nupur Thakur**

PROJECT Part – 3

Deep Learning (Convolutional Neural Networks)

Report by - **Subba Raja Kashyap Saligrama** **(Asu Id:** 1219510851**)**

Dataset used :

The dataset given is SVHN (Street View House Numbers) dataset. It is provided in form of two mat files - 'train_32x32.mat', 'test_32x32.mat'.

Number of samples in training set:	73257	RGB images each of size (32 x 32 x 3)
Number of samples in testing set:	26032	RGB images each of size (32 x 32 x 3)

Algorithms Implemented :

- Implemented a Convolutional Neural Network (CNN) according to given architecture of layers.

Programming Language, workspace, software used :

- Python language is used to implement the source code of the given CNN architecture.
- Tensorflow platform is used to execute the program.
- Google Colaboratory (Colab) is used as the workspace to program the network.
- 'Scipy' library is used to import and read the mat file as the dataset is in form of a dictionary.
- 'Keras' library is used to build and compile all the layers & to train and test the network.
- 'Numpy' library is used to perform mathematical operations on the datasets.
- 'Matplotlib' library used to plot the graphs.

Importing and Reading Dataset :

Initially, we first import and load the data (training and testing) from the respective mat files (stored in Google Drive) with help of 'Scipy' library into the following labels –

trX – training samples	tsX – testing samples
trY – labels for training samples	tsY – labels for testing samples

Normalizing Sample Data :

We perform data normalization on both training and testing samples to bring them in the range of [0,1] , and update the above variables to reflect the new samples as follows –

$$trX = \frac{trX - trX.\mathrm{min}()}{trX.\mathrm{max}() - trX.\mathrm{min}()} \qquad tsX = \frac{tsX - tsX.\mathrm{min}()}{tsX.\mathrm{max}() - tsX.\mathrm{min}()}$$

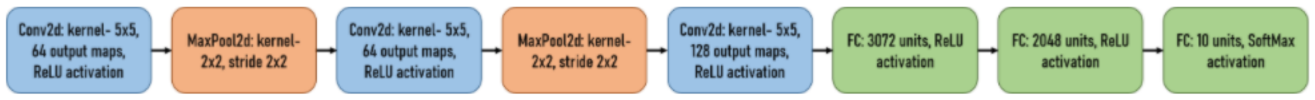
Encoding Label Data :

Then, we encode both training and testing labels using One-Hot Vector encoding.

```
trY = keras.utils.to_categorical(trY, num_classes, dtype='int')

tsY = keras.utils.to_categorical(tsY, num_classes, dtype='int')
```

Building CNN model :


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

Layer 1 – 2D Convolution layer *padding* – same *stride* – 1x1
Kernels – 64 kernels of size 5x5 *Activation function* – ReLU

Layer 2 – 2D MaxPooling (size 2x2) *padding* – same *stride* – 2x2

Layer 3 – 2D Convolution layer *padding* – same *stride* – 1x1
Kernels – 64 kernels of size 5x5 *Activation function* – ReLU

Layer 4 – 2D MaxPooling (size 2x2) *padding* – same *stride* – 2x2

Layer 5 – 2D Convolution layer *padding* – same *stride* – 1x1
Kernels – 128 kernels of size 5x5 *Activation function* – ReLU

Layer 6 - Fully Connected (FC) layer
Nodes - 3072 nodes *Activation function* – ReLU

Layer 7 - Fully Connected (FC) layer
Nodes - 2048 nodes *Activation function* – ReLU

Layer 8 - Fully Connected (FC) layer
Nodes - 10 output nodes *Activation function* – SoftMax

Note:

- padding is set to 'same' so that the inputs and outputs of all convolution and pooling layers will be same.
- A Flatten() layer is added between layers **5** and **6** in order to convert the output image from layer **5** into vector input to layer **6.**

Training the model :

Now, we train the above-built model using the training data (is passed in as batches).
Optimizer used here :– Stochastic Gradient Descent (SGD)
Loss function used :– Categorical Cross Entropy

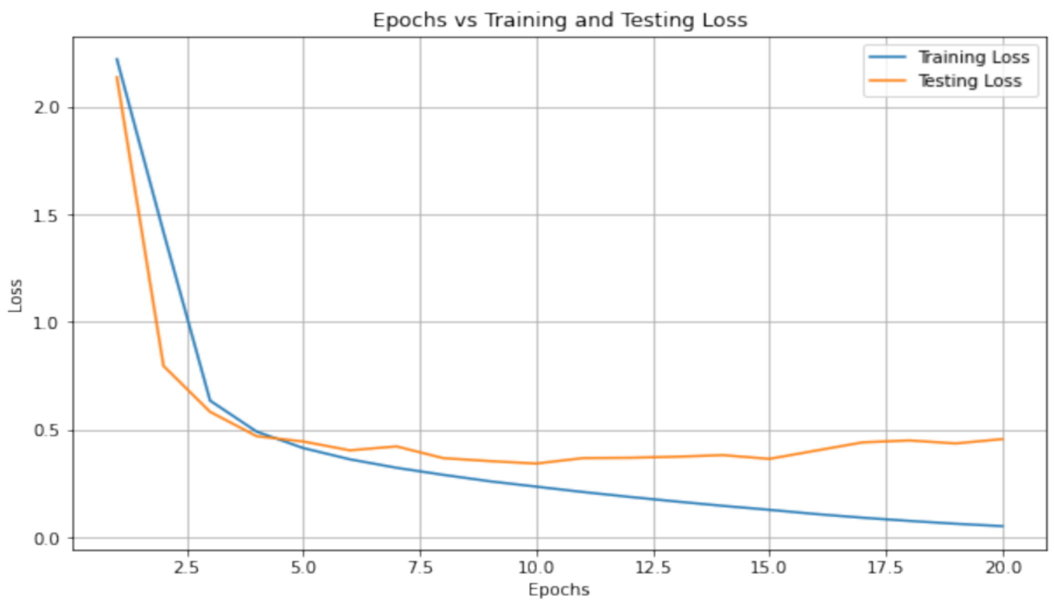
$$l(W) = \sum_{i=1}^{n}\sum_{j=1}^{c} \mathrm{I}_j(y_i) \log P(y_i = j \mid x_i) \qquad where, \quad \mathrm{I}_j(y_i) = \begin{cases} 1, & if\ y_i = j \\ 0, & otherwise \end{cases}$$

Parameters used :

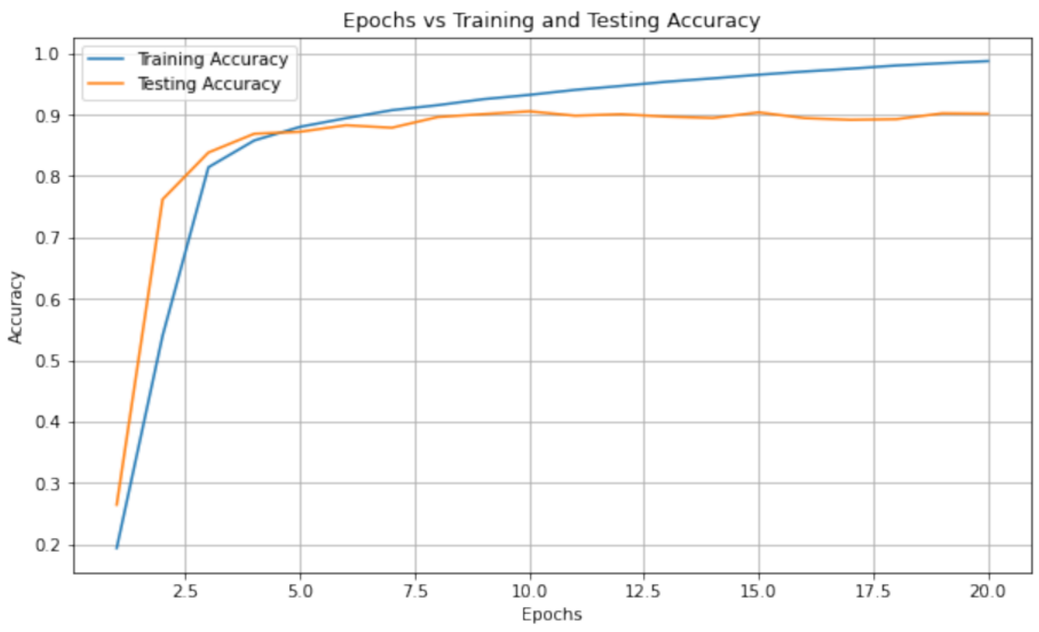
```
batch_size = 64     // Samples are fed in batches of this size per epoch
epochs = 20         // total no of epochs the model training is run for
learn_rate = 0.01   // learning rate η used in SGD to update weights
kernel_initializer = 'glorot_uniform'    // default one
```

Results :

Loss graph for Training and Testing –



Accuracy for Training and Testing –



Training Accuracy :- **98 %** *Testing Accuracy* :- **90.32 %**
Training Loss :**- 0.1326** *Testing Loss* :- **0.4783**

NOTE: Even though we achieved 90.32 % testing accuracy in the above-shown graph, this value may fluctuate between 88 % and 91 % on multiple executions without modifying the parameters.

Observations :

***Number of Epochs* –**

After crossing 5 or 6 epochs, the testing accuracy stabilizes at around 88 - 91 %. As training accuracy keeps increasing till 98 %, testing accuracy starts to deviate from training accuracy which implies that the model is starting to overfit.

***Learning Rate* –**

If learning rate = 0.001 is used, the model converges very slowly which is reflected in the slow variance in the testing loss.

If learning rate = 0.1 is used, the testing loss graph becomes zigzag which suggests it may be jumping the peak which makes it very difficult for the weights to converge.

***Dropout* –**

If we add Dropout layers between the two Dense (FC) layers, the variation between training and testing loss is very little. Also, the testing accuracy improved.

The testing accuracy improved and got closer to training accuracy as we increase the Dropout rate from 0.1 to 0.5.