Project Part 3: Classification using Neural Networks and Deep Learning

Data Set: SVHN Dataset

Shashank Davalgi Omprakash (1219510734)

Overview:

Building and Training a small convolutional neural network for a classification task.

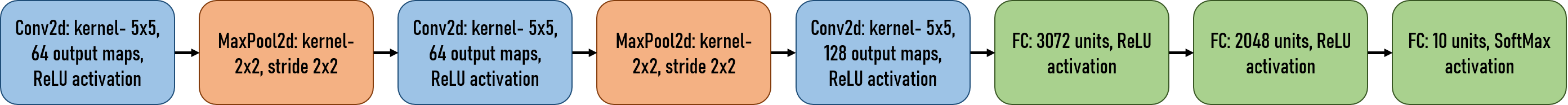
**Dataset:** SVHN Dataset

**Dataset Description:**

* It consists of printed digits cropped from the house plate pictures belonging to 10 classes.
  + Training Samples: 73,275 images
  + Testing Samples: 26,032 images
* The input image resolution is 32x32 and consists of 3 (RGB) channels.

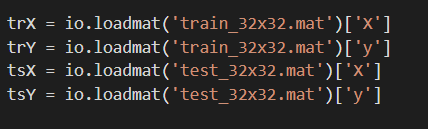
Architecture:

The below diagram shows the CNN architecture implemented in this phase of the project.

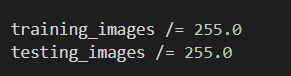


Implementation:

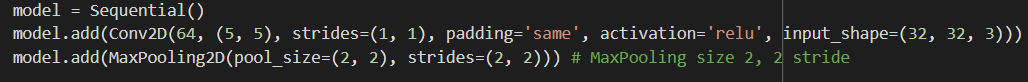
1. The training and testing images are loaded along with the training and testing labels as shown below:



1. The training and testing images are converted to ‘float64’ type and the training and testing labels are converted into ‘int64’ type.
2. We then normalize the data to bring it in the range of 0-1. It is done as shown below:

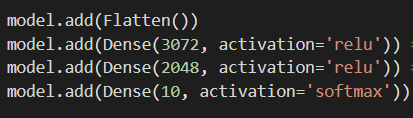


1. We build the model as per the architecture provided above using the Keras Sequential model.
2. The first block of the architecture is implemented as shown below:

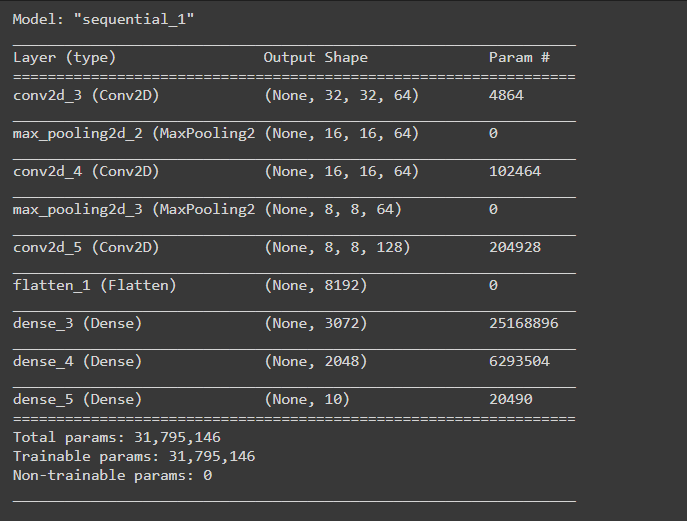


* The Sequential() function creates a Keras sequential model for 2D CNN.
* We add a 2D CNN to the model with output feature maps as 64, kernel size as 5x5, strides as 1.
* The padding=’same’ gives the output feature maps as the same size of input shape.
* ReLU activation function is used for this model with input shape = (32x32x3)

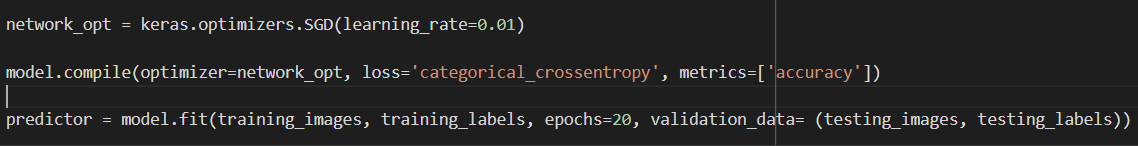
1. Next, we add the MaxPooling for the model after the first block with the pool\_size = 2x2 and strides = 2.
2. Similarly, the model is build based on the architecture by adding 2 more CNN layer with one more MaxPooling layer.
3. Once these layers are built, we build a Fully Connected Dense layer after the 3rd CNN layer with 3078 units and ReLU activation. Post that, we create 2 more Dense layers and output with the SoftMax activation function. It is implemented as shown below:



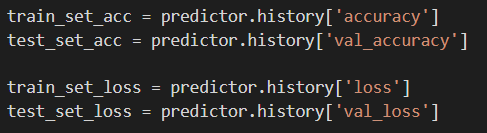
1. model.summary() gives the summary of the model and is shown below:



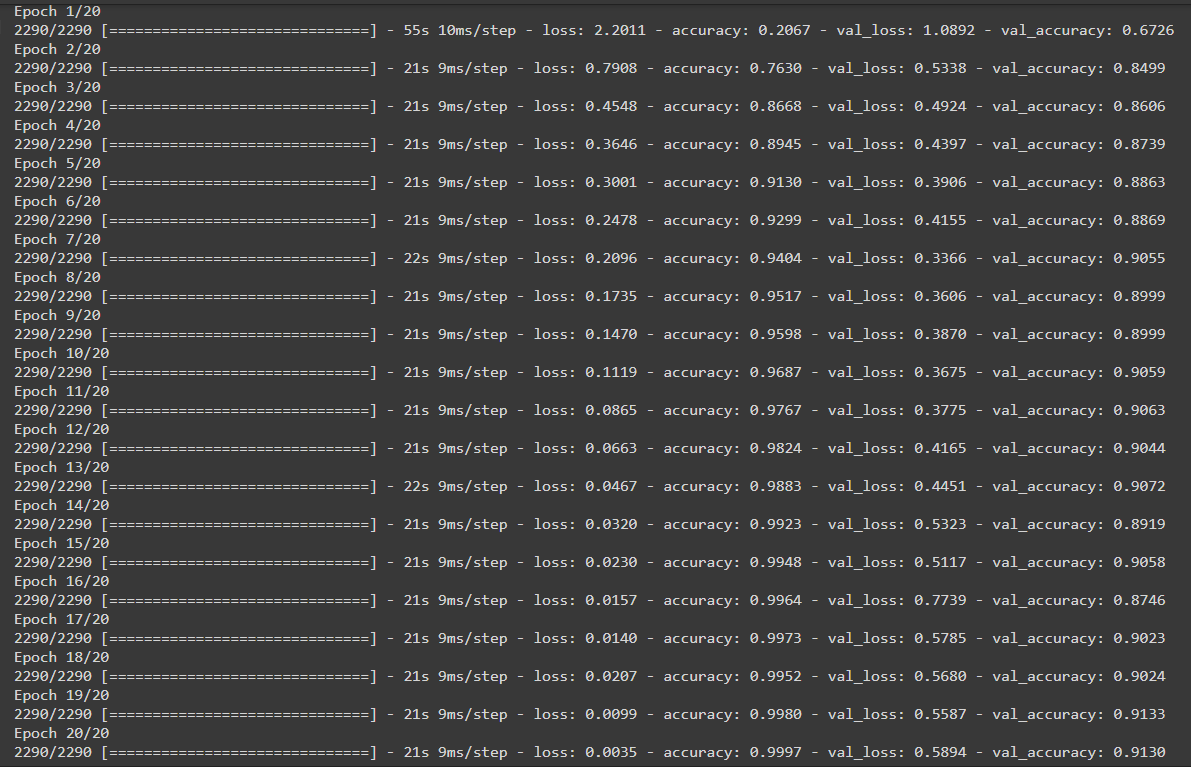
1. Then we build the model using the Stochastic Gradient Descent optimizer with a learning rate = 0.01
2. The model is then fit and validated on the training sample for 20 epochs. The implementation is :



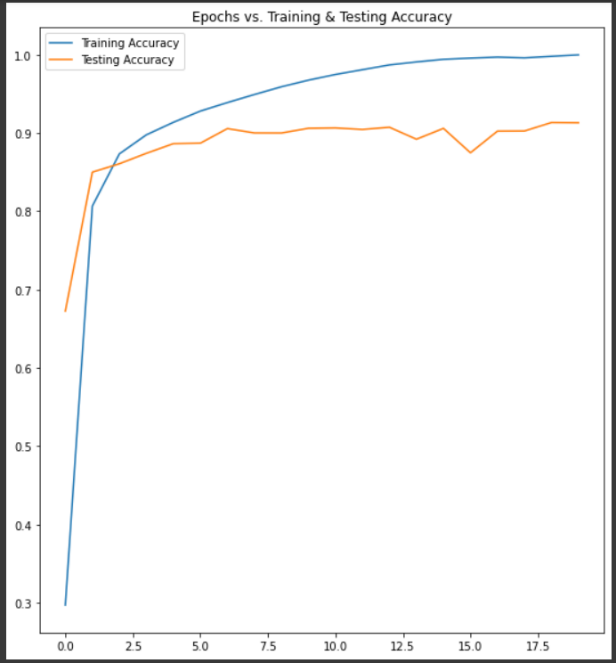
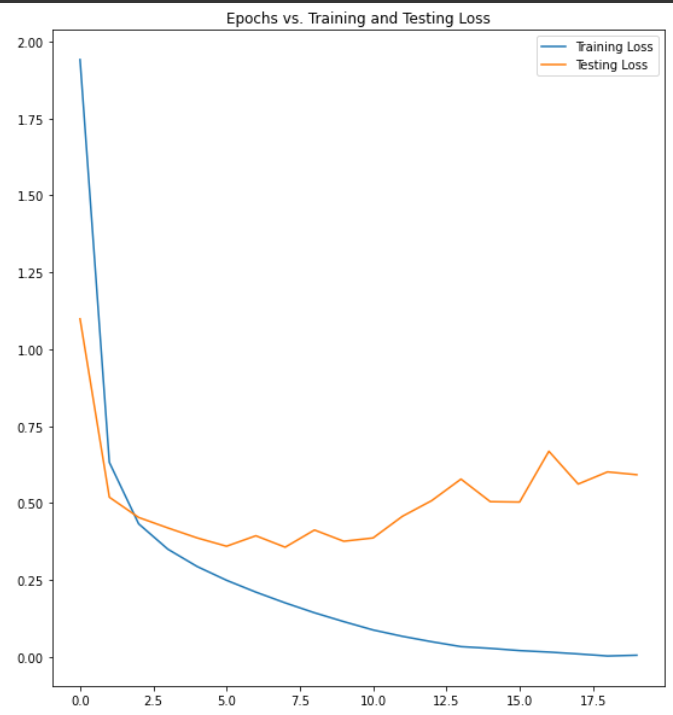
1. Once the model is executed, we find the training and testing accuracies and losses.



1. The output of the 20 epochs is shown below:



1. Using the results from the above functions, we plot the epochs against the training and testing accuracies and epochs against training and testing losses. The graphs are shown below:

Results and Accuracy:

The Training model accuracy is **99.93%**

The Testing model accuracy is **91.14%**

The training model loss is very negligible.

The testing model loss is **59.18%**