Computer Vision

Subhasish Basak (MDS201803)

Final Project 1 : Image Classification with Convolutional Neural Nets

April 13, 2020

1 Introduction

In this Assignment we implement a **Convolutional Neural Network** (CNN) using Keras and Tensorflow for Image Classification. Here we will use the Fashion MNIST dataset, imported from datasets library of Keras. The link to my **Colab Notebook** is as follows: Click to view Colab Notebook

2 Data Preprocessing

The Fashion-MNIST dataset is similar to the MNIST dataset in image size (28×28), number of classes (10), and the size of the training and test datasets ($60,000$ and $10,000$ images, respectively). We pre-process the data using the following steps :

- **Transforming into grayscale** : Since each pixel value in the images are in the range $[0, 255]$. We transform the images into Gray-scale by dividing each pixel value by 255.
- **Resizing** : We also need to reshape the training images in order to feed it into the CNN.

3 The CNN Architecture

Our Convolutional Neural network has the following configuration :

- Input Layer
  - **Shape** : each of the imput is a 28×28 image of pixels.
- 1st Convolution Layer
  - **Kernel Shape** : 3×3
  - **Depth** : 32
  - **Stride** : 1
  - **Activation** : **ReLU**
- Max Pooling Layer
  - **Kernel Shape** : 2×2
  - **Stride** : 2
- Dropout Layer
  - **Probability** : 0.3
- 2nd Convolution Layer
  - **Kernel Shape** : 3×3
  - **Depth** : 64
  - **Stride** : 1

- **Activation** : **ReLU**

- Max Pooling Layer
  - **Kernel Shape** : 2×2
  - **Stride** : 2
- Dropout Layer
  - **Probability** : 0.3
- Fully connected Layer
  - **Number of nodes** : 256
  - **Activation** : **ReLU**
- Dropout Layer
  - **Probability** : 0.3
- Output Layer
  - **Number of nodes** : 10
  - **Activation** : **Softmax**

Loss function : Sparse Categorical Cross Entropy
Optimizer : Adam
Number of trainable hyperparameters : 431242.

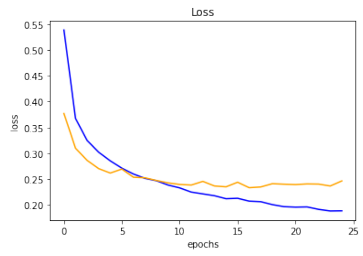
4 Observation & Findings

We first tried the given CNN configuration in the question but it gave training accuracy around 90.2%. We also tried with adding another Convolutional layer but it lead to *Overfitting* (yielding training accuracy around 96.8% and validation accuracy 90.8%). We have trained our network with 60,000 training images and validate the model on the 10,000 test images. Since we used **TPU** on Google colab it took 68 seconds per epoch. Given the above configuration we got the following results after full training:

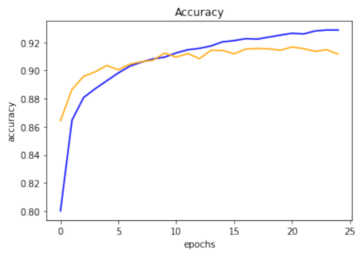
- **Training Accuracy** : 92.87%
- **Validation Accuracy** : 91.16%

5 The Evaluation plots

We plot the **Training accuracy** vs the **validation accuracy**. Also the **Training loss** vs the **Validation loss**. The plots are as follows: As we can see that in the loss curve, both Training loss and



(a) Loss plot



(b) Accuracy plot

Validation loss curves decreases along with the training epochs. After 25 epochs the loss comes below 0.18. From the accuracy plot we can observe that the model performs slightly better in the validation set than the training set. It also does not show any signs of *Overfitting*.