**REPORT**

**PART 1**

The input for the above two models is the fashion\_mnist dataset. There are 70,000 images.

**Model 1:**

One hot encoding is applied on the target variables.

Convolution Layers: 3

Dropouts: 3 (One at each conv layer)

Pooling: 3 Max pooling layers (One at each conv layer)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

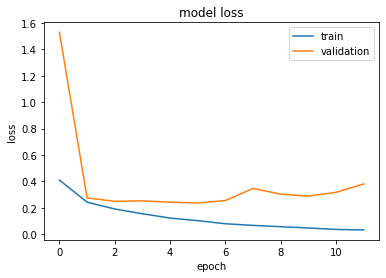
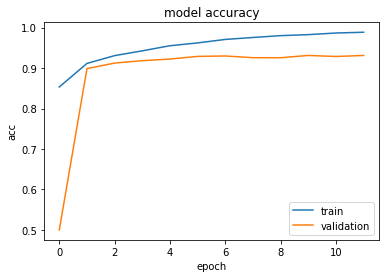
**Output:**

Validation loss: 0.38023622093992404

Validation accuracy: 0.9312499761581421

Test Accuracy Score: 0.9271

**Plots for accuracy and loss:**



Model 2:

One hot encoding is **not** applied on the target variables.

Convolution Layers: 3

Dropouts: 3 (One at each conv layer)

Pooling: 3 Max pooling layers (One at each conv layer)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

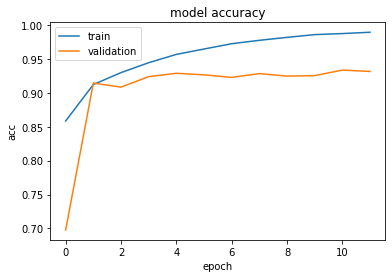
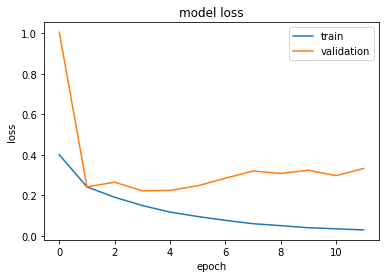
**Output:**

Validation loss: 0.33321774934371934

Validation accuracy: 0.9314166903495789

Test Accuracy Score: 0.928

**Plots for accuracy and loss:**

For parts 2 and 3, as the data set is big, the data is loaded into pickle files. To run the code download pickle files to your drive in the path ‘/content/drive/My Drive/’

Or download the files anywhere and mention the corresponding file path in the code in the line

pickle.load(open('/\_\_path\_\_/\_\_file\_name\_\_.p', 'rb'))

Links to the pickle files:

For part 3:

* **val.p** - <https://drive.google.com/file/d/1-LKpKrKVMUmYX5iTR4Z__Mvrqxzas6QV/view?usp=sharing>
* **train.p** - <https://drive.google.com/file/d/1-FJmP_0uNuP-lYxHGLW__IqYnfgdwZ2Q/view?usp=sharing>
* **test.p** - <https://drive.google.com/file/d/1-Lteu2gPPS1G7dNEI57AIEsQ3hlhGCLs/view?usp=sharing>

For part 2:

* **pickle.p -** <https://drive.google.com/file/d/19hG3os0q6kVGgNiUA9BLeVkiXqmaMmW-/view?usp=sharing>

**PART 2**

The data set for this part is augmented by rotating, flipping the images etc. We use ‘ImageDataGenerator’ to modify the images to generate more data for training and to avoid overfitting.

**Model 1:**

Convolution Layers: 4

Dropouts: 3 (One at each conv layer)

Pooling: 3 Max pooling layers (One at each conv layer)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

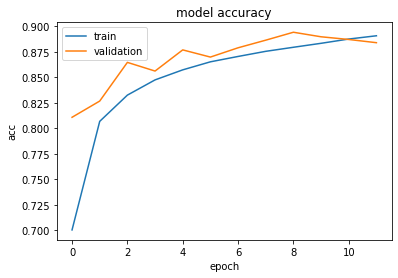
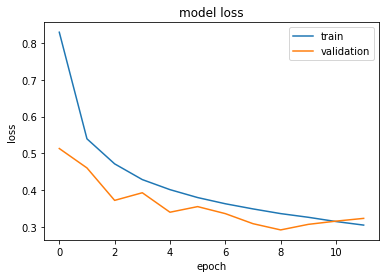
**Output:**

Val loss: 0.2998387911717097

Val accuracy: 0.8934999704360962

Test Accuracy Score: 0.8833

**Plots for accuracy and loss:**

**Model 2:**

Convolution Layers: 6

Dropouts: 0

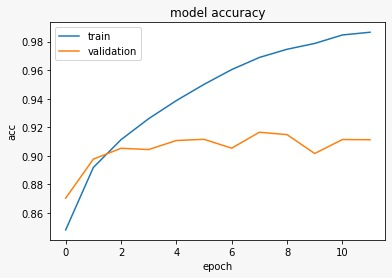
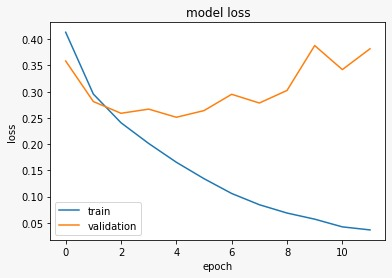
Pooling: 2 Max pooling layers (at conv layer 4 and 6)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

**Plots for accuracy and loss:**

**Model 3:**

Convolution Layers: 6

Dropouts: 3 (1 each after conv layers 2,4,6)

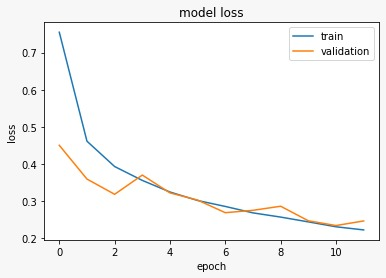
Pooling: 2 Max pooling layers (at conv layer 4 and 6)

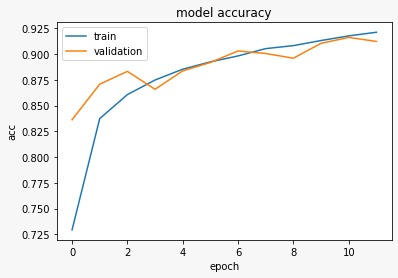
Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

**Plots for accuracy and loss:**





**Model 4:**

Convolution Layers: 7

Dropouts: 6 (1 each after conv layers 2,4,6)

Pooling: 3 Max pooling layers (at conv layers 3, 5 and 7)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

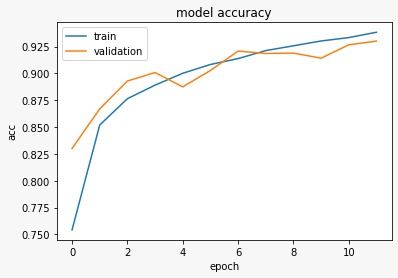
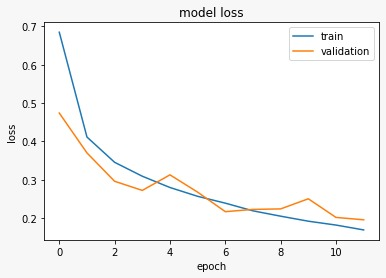
**Outputs:**

Validation loss: 0.19547169989041452

Validation accuracy: 0.9300535917282104

Test Accuracy Score: 0.9302285714285714

**Plots for accuracy and loss:**

**Observations:**

We have started with 3 convolution layers. The accuracy was in the range of 85-89. The layers were combinations of 32, 64, 128 filters. We implemented multiple models by gradually increasing the convolutional layers, pooling layers, neurons, dropouts and filters and made the following observations:

* As the width of hidden layers increases we observe a fluctuation in the accuracy. So, a general  
  trend could not be observed.
* As the number of epochs increase, we observe an increase in accuracy to an extent. After a point the accuracy fluctuates with increase in epochs giving approximately the same accuracy. At this point the data is being overfitted.
* To overcome this, we used early stopping with parameters ‘monitor=’val\_loss’ and patience = 8.
* Changing the number of hidden layers has the most impact on the performance. It has no generic trend.
* Accuracy of the CNN highly depended on number of convolution layers and filters in it. We observed a trend as the layers and filters increased, the accuracy was improving.
* **Optimizers:** We used Adadelta and Adam optimisers. No major changes were observed in the accuracy. Both the optimizers, performed very similarly.
* Accuracy of the neural network is strongly dependent on the number of neurons. As the number of hidden layers is increasing, we could observe that the accuracy on test data is fluctuating.

**PART 3:**

All images are taken from the ImageNet dataset. The images are downloaded using the downloader.py script. This script selects 20 classes and downloads the images for each of these classes. The size of the images and the image set is too huge to process, hence we resize the image to a much small scale of 224x224. Also, we consider only 500 images for each class. This is done by the image\_net\_preprocessing.py script.

Finally, the data is now available in the form of images. We now split the entire dataset into train, test and validation data. Once the data is split, all the images from the three sets are converted to arrays and stored different pickle files.

Data is loaded from the pickle files when needed and is passed on to the model. The model learns to classify the image into one of the 20 classes.

**Model 1:**

Convolution Layers: 9

Dropouts: 5 (4 at alternate conv layers and one after last conv layer)

Pooling: 5 Max pooling layers (One before each drop out later)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

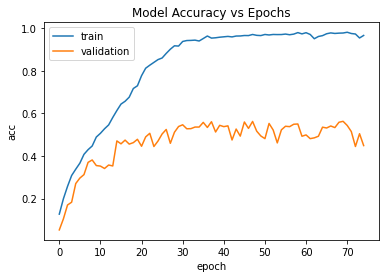
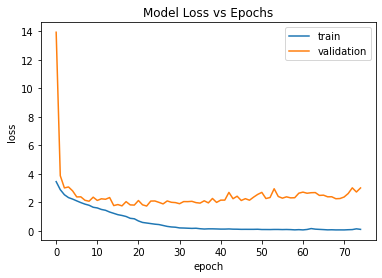
**Outputs:**

Validation loss: 3.0115849661827085

Validation accuracy: 0.44874998927116394

Test Accuracy Score: 0.448

**Plots for accuracy and loss:**

**Model 2:**

Convolution Layers: 9

Dropouts: 7 (4 at alternate conv layers and one after last conv layer)

Pooling: 5 Max pooling layers (One before each drop out later)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

Early stopping: ‘monitor=’val\_loss’ and patience = 8.

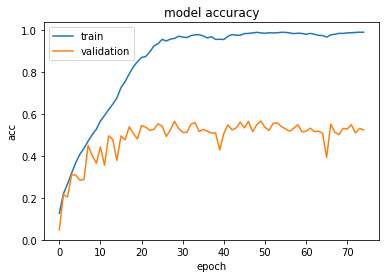
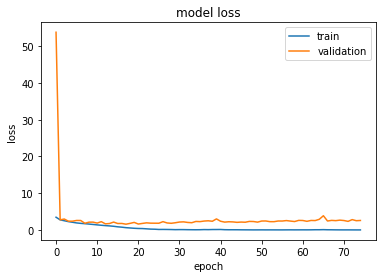
**Output:**

Validation loss: 1.8854860424995423

Validation accuracy: 0.5274999737739563

Test Accuracy Score: 0.5345

**Plots for accuracy and loss:**

** **

**Model 3:**

Convolution Layers: 10

Dropouts: 5 (At alternate conv layers)

Pooling: 5 Max pooling layers (At alternate conv laters)

Kernel size: 3x3

Fully Connected layers: 2

Activation functions: ReLU and softmax at output layer.

Early stopping: ‘monitor=’val\_loss’ and patience = 8.

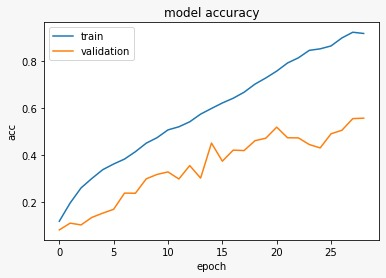
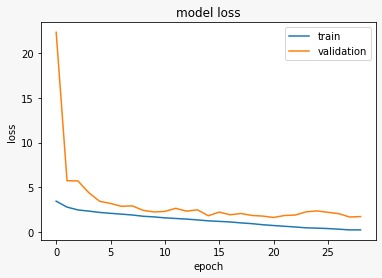
**Outputs**:

Validation loss: 1.7463324427604676

Validation accuracy: 0.5575000047683716

Test Accuracy Score: 0.5635

**Plots for accuracy and loss:**

**References:**

1. <https://www.pluralsight.com/guides/importing-image-data-into-numpy-arrays>
2. <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>
3. <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>
4. downloader.py – Class demo by Nitin Kulkarni
5. image\_net\_preprocessing.py – Class demo by Nitin Kulkarni