Computer Vision

Chennai Mathematical Institute

Project- Part I (Notebook Link)

Task

Here our task is to design and train deep convolutional network for image classification using Keras and TensorFlow.

Data

Here we build and train a deep neural networks to work on the **Fashion MNIST dataset**.

The Fashion MNIST dataset contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels) as in the following -

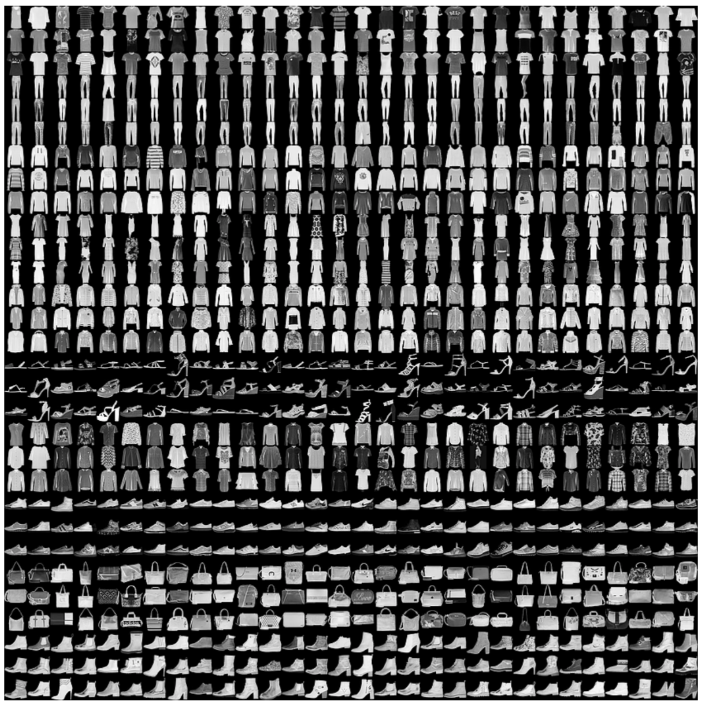


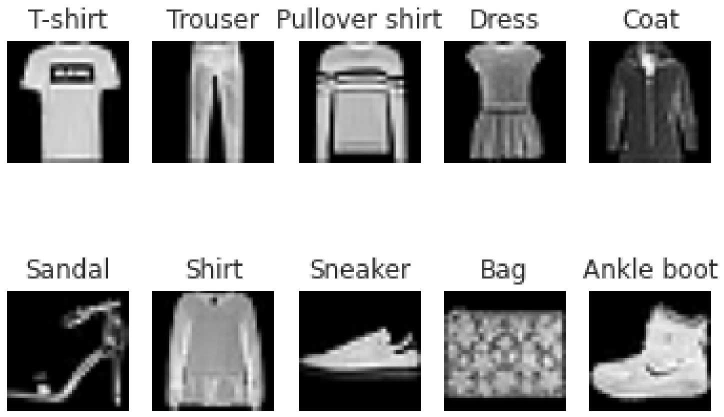
Figure 1. Fashion-MNIST samples (by Zalando, MIT License).

- 60,000 training Images
- 10,000 test Images
- 28 $\times$ 28 gray scale images
- 10 categories

Here are all 10 categories, "T-shirt","Trouser","Pullover shirt","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"

Visualization:

Here we visualize image from each of the categories -



Procedure:

- We read the dataset.
- We visualize each category images. We also check the count of training data points for each category.
- We convert the labels to one hot vectors. We have 10 categories. So size each vector is 10 with all zeroes except 1 in the position of label value of the one hot vector. For e.g. - If a training data had label 2 then now it is converted to (0,0,1,0,0,0,0,0,0,0).
- We normalize the data by dividing each pixel value by 255.
- We reshape the images.Shape of a input before reshaping: (28, 28). Shape of a input after reshaping: (28, 28, 1)
- Then we fit our model on the training data.
- We then evaluate the model on test data.
- We then plot the train,test loss and accuracy over the epochs and check how it behaves with the no of epochs.
- We plot the confusion matrix for the test data to check for the misclassifications.

Model

Here we are implementing Convolutional Neural Network for image classification.

Here we have the following type of layers -

- Input Layer
- Convolution Layer
- Pooling Layer
- Fully Connected Layer

Output layer is also a fully connected layer.

Our model architecture is as follows -

1. **Input Layer** - Shape of our input - $(28 \times 28 \times 1)$

2. **2D Convolution layer** - • Kernel size - (5×5) • Stride = 1 • No of filters - 64 • Padding - Zero padding with size 1 • "RELU" activation function is used.

3. **Pooling Layer** - • Max pooling is done. • Kernel size - (2×2) • Stride = 2 • Padding Size - 0

4. **2D Convolution Layer** - • Kernel size - (3×3) • Stride = 1 • No of filters - 32 • Padding - Zero padding with size 1 • "RELU" activation function is used.

5. **Pooling Layer** - • Max pooling is done. • Kernel size - (2×2) • Stride = 2 • Padding Size - 0

6. **Fully Connected Layer** - • No of nodes - 256 • "RELU" activation function is used.

7. **Output Layer**(also a fully connected layer) - • No of nodes = No of categories = 10 • "Softmax" activation function is used.

Dropout: We use "Dropout" in the hidden layers. It is one of the regularization technique, it is the method of dropping out neurons and it helps us in reducing over-fitting.

Optimizer: Several optimizers are tried out. ADAM and RMSprop both worked well as optimizer. But using ADAM was the best choice here. Changing the learning rate and other hyper-parameters has some effect in the result.

Batch Normalization: This is a technique for training deep neural networks. It helps us to stabilize the training process and reduce the no of training epochs required to train a model by reducing the problem of internal co-variance shift. It standardizes the inputs to a layer for each mini-batch.

Activation Function: "RELU" activation function is used in the convolution layers and "Softmax" activation function is used in the output layer.

Loss Function: Categorical Cross-entropy is used as Loss function. For one input data the loss can be calculated using the following formula -

$$L(y, p) = -\sum_{i \in C} y_i \log(p_i)$$

Here y is the true label(one hot encoded) and p is the vector of predicted probabilities corresponding to each class.

Data Augmentation: Data Augmentation is a special technique to increase the amount of data from whatever data available. Here we don't collect data. This helps us in better generalization of the model. To capture enough variation, we may need to generate data from the available data. Some of the Data Augmentation techniques are - cropping, rotating, adding noise, flipping, different kind of normalization, resizing, height-width shifting etc.

Keras provide the image data generator function for data augmentation. It generally helps us to increase performance of the model. In this case with those parameters using some of the data augmentation technique didn't give a significant increase in performance.

Parameters:

- No of parameters in convolution layer - (Height of the kernel × Width of the kernel × no of input image channels + 1) × Number of kernels. (1 correspond to the bias parameter)
- No of parameters in a pooling layer - zero. It just calculate a specific number.

- No of parameters in a fully connected layer - (No of nodes in the layer $\times$ No of nodes in the previous layer) + No of nodes in the layer(Corresponding to the bias parameter)

So total no of parameters -
In the first Convolution layer - $(5 \times 5 \times 1 + 1) \times 64 = 1664$

In the 2nd Convolution layer - $(3 \times 3 \times 64 + 1) \times 32 = 18464$

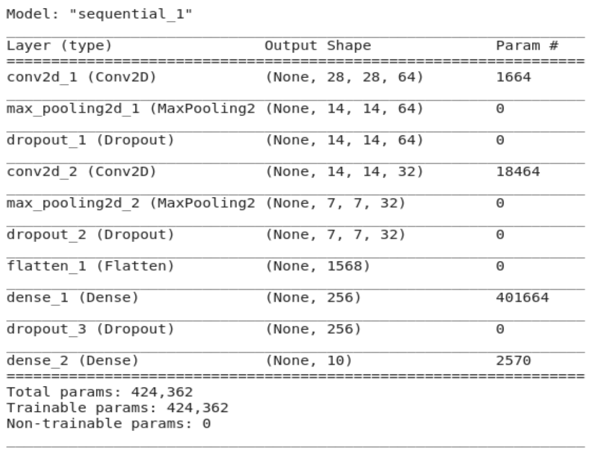
In the first fully connected layer -

After flattening the output of previous layer we have the length of previous layer output is - $(7 \times 7 \times 32) = 1568$. So total parameters in this layer - $(256 \times 1568) + 256 = 401664$

Now for the output layer - $(10 \times 256) + 10 = 2570$

So total trainable Parameters - 424362
If we choose less no of nodes in the fully connected layer, we can significantly decrease the number of parameters.

Here is the **Model Summary** :



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        1664
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        18464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               401664
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 424,362
Trainable params: 424,362
Non-trainable params: 0
_________________________________________________________________
```

Experiments

Here we run the model for 50 epochs with a batch size of 512. Lot of experiments were done by changing the hyper-parameters, adding convolution, pooling or fully connected layers. There wasn't significant change all the time.

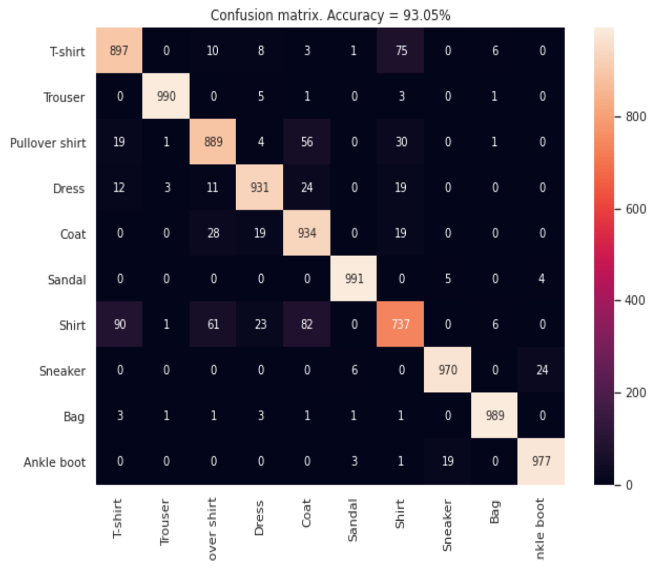
- Using Data Augmentation didn't help much in this case.
- Batch normalization is also tried out. There wasn't significant difference in the results after 50 epochs but using batch normalization the highest accuracy is reached faster in terms of epochs.
- Different optimizers were tried out. ADAM was the best choice here.
- Different learning rate were tried out in the optimizers. There were a little up and down in the accuracy for the change.
- Used different no of kernels and different kernel sizes and strides. Accuracy got changed based on the combination of these parameters.
- Adding convolution, pooling and fully connected layers to the final model didn't help much to increase accuracy.
- Increasing no of nodes in the fully connected layer gave a slight increase in the accuracy.
- Dropout helped in reducing over-fitting.Different probabilities were tried out.

Results

So we train our model on 60,000 training images and then test it on the 10,000 test images. The target is to classify an test image to one of the 10 categories of clothes we have.

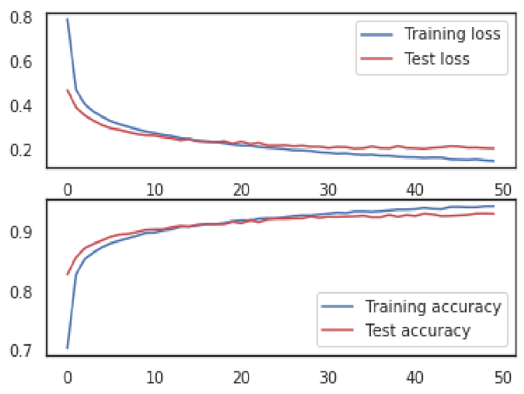
- No of epochs: 50
- Training:
  - Loss: 0.1473
  - Accuracy: 94.3 %
- Test:
  - Loss: 0.204
  - Accuracy: 93.05 %

Confusion matrix for test data:



	T-shirt	Trouser	over shirt	Dress	Coat	Sandal	Shirt	Sneaker	Bag	nkle boot
T-shirt	897	0	10	8	3	1	75	0	6	0
Trouser	0	990	0	5	1	0	3	0	1	0
Pullover shirt	19	1	889	4	56	0	30	0	1	0
Dress	12	3	11	931	24	0	19	0	0	0
Coat	0	0	28	19	934	0	19	0	0	0
Sandal	0	0	0	0	0	991	0	5	0	4
Shirt	90	1	61	23	82	0	737	0	6	0
Sneaker	0	0	0	0	0	6	0	970	0	24
Bag	3	1	1	3	1	1	1	0	989	0
Ankle boot	0	0	0	0	0	3	1	19	0	977

Test, training loss and accuracy over epochs:



So the model worked well on the test data. It achieved 93.05 % accuracy which is really good. If we see the accuracy and loss plots over epochs, we see that with the increase in number of epochs, the training and test loss both decreases and the accuracy increases. Finally both the loss and accuracy converges around some point. So the model has learned well over time with the increase in no of epochs.The confusion matrix gives us a clear idea about the output. We can check the misclassifications here. Most of the misclassifications occurred in case of shirts and T-shirts. Many shirts are predicted as T-shirts and many T-shirts are predicted as shirts. Also a large no of pullover shirts and coats are classified as shirts. There are some other misclassifications but those are in small numbers. Now based on these misclassifications we can think of adapting techniques (like possible data augmentation based on these, collecting more data for these categories etc) to increase the accuracy of the model.