Title:

Assignment 1,

Classification of Persian digits & Classification of Fashion MNIST dataset

Department of Computer and Data Sciences

Advisor:

Dr. Kheradpisheh

Student:

Fatemeh Saberi Khomami

Student Number:

400422114

1. Introduction

Computer vision is a field of AI that enables computers to derive meaningful information from images, videos and other visual inputs, and take actions or make recommendations based on that information. Computer vision tries to work as same as human vision, So, we expect such systems to know how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image. In this project, we'll focus on the detection task.

In the Second part, we'll make a Convolutional Neural Network that can tell apart Persian digits. In the Third part, we'll make a Convolutional Neural Network that can detect the clothes in the MNIST dataset. Further, we will compare our selected model's performance with the RESNET-50 architecture.

2. The Persian Digit dataset

2.1. Getting to know the Persian Digit dataset

The training set includes 60,000 32*32 images, and the test set includes 20,000 32*32 images. Both data sets are provided with the labels. In below figures, we demonstrated the number of images in each class.

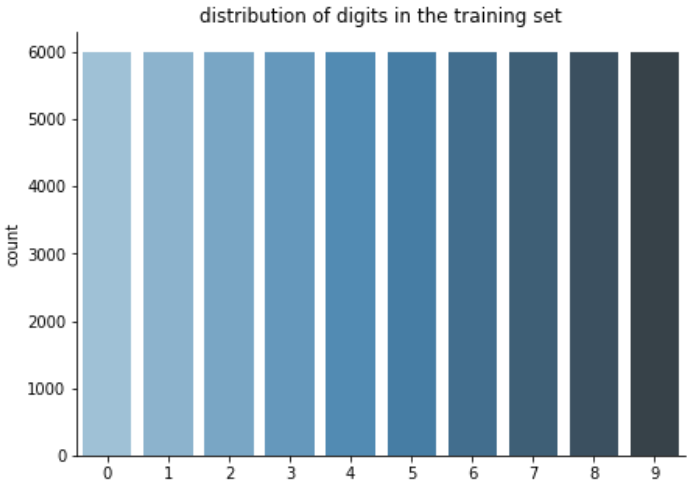

Figure (1): Bar Plot of Persian Digits in the training set

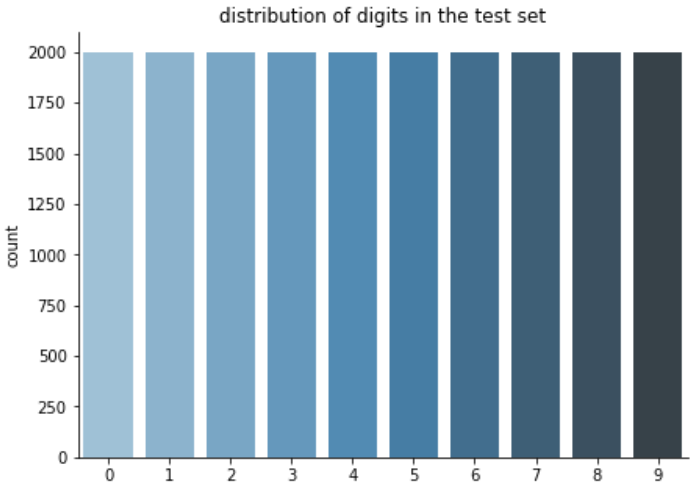

Figure (2): Bar Plot of Persian Digits in the test set

There is a remaining set which includes 22,352 images with the following distribution.

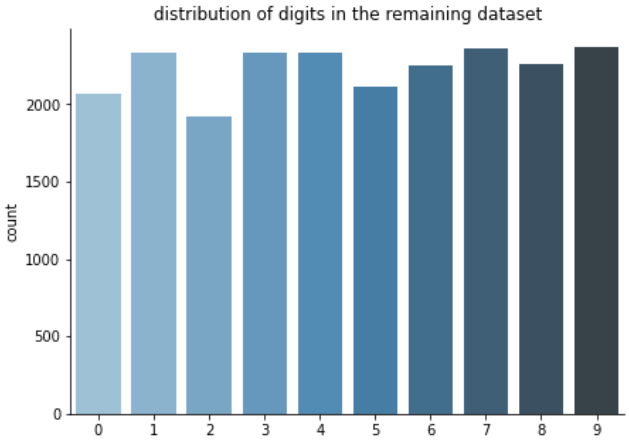

Figure (3): Bar Plot of Persian Digits in the remaining dataset

We extracted a validation set with 19,230 images out of the above data, with 1,923 data within each class. The following figure is the corresponding distribution.

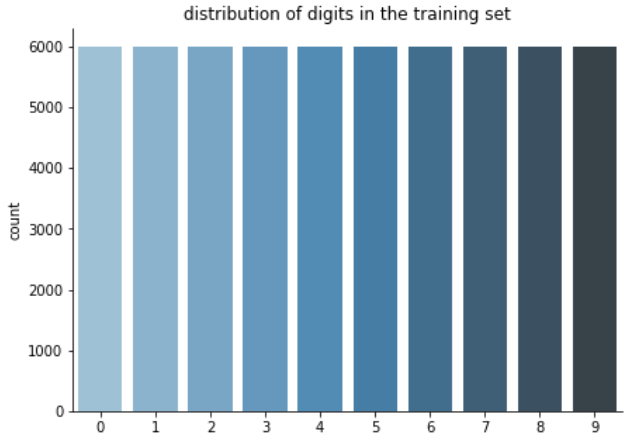

Figure (4): Bar Plot of Persian Digits in the validation set

2.2. Data cleaning

Regarding that the test and validation data were in order of the digits, we shuffled the mentioned datasets. Then, we normalized all the images in training, test, and validation sets by dividing the image matrices by 255.

We converted the labels to one-hot vectors. Providing that there were no missing pixels or images, we went straight forward to build image generators for the training and validation data.

We created new images by randomly rotate images through any degree between 0 and 5.

We created new images by randomly shift images 0.1 horizontally or vertically or both.

We created new images by randomly zoom in the image by 0.1 magnitude.

We created new images by randomly shearing the image by 10%.

2.3. Designing the model and start learning

The first model that we trained had the below architecture.

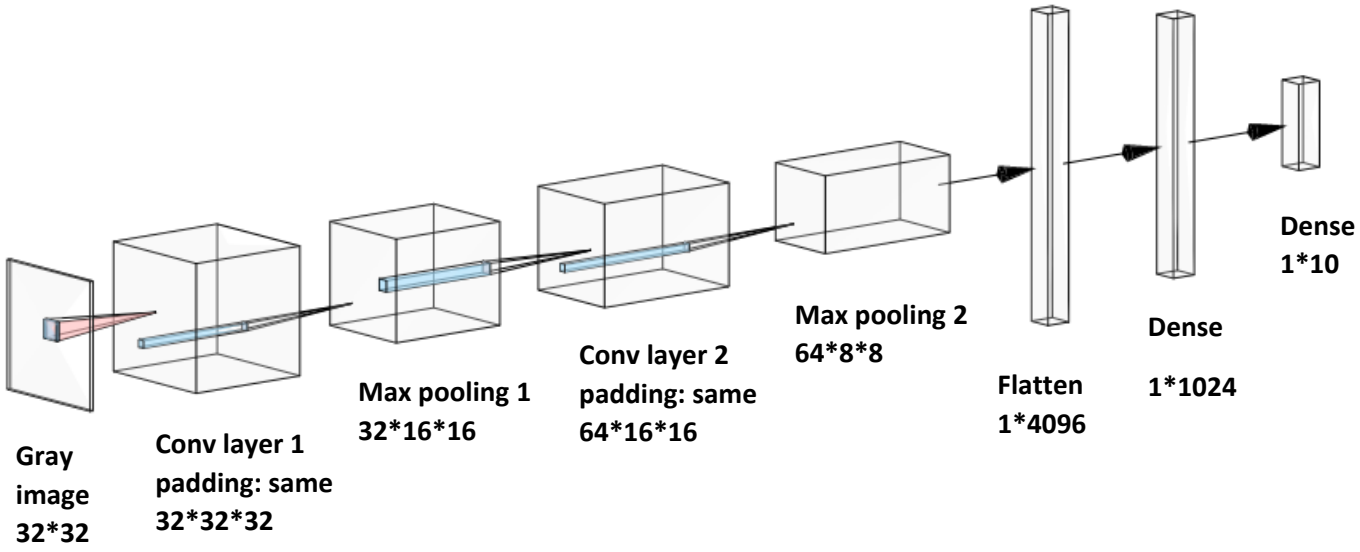

Figure (5): Architecture of the first trained model

The training was 10 epochs long, optimized by Adam with 0.001 as the learning rate.

The last epoch had the following results:

loss: 0.0417 - accuracy: 0.9888 - val_loss: 0.1883 - val_accuracy: 0.9496

It appears that we are overfitting the training data a little.

The performance of the model is depicted in the following figure.

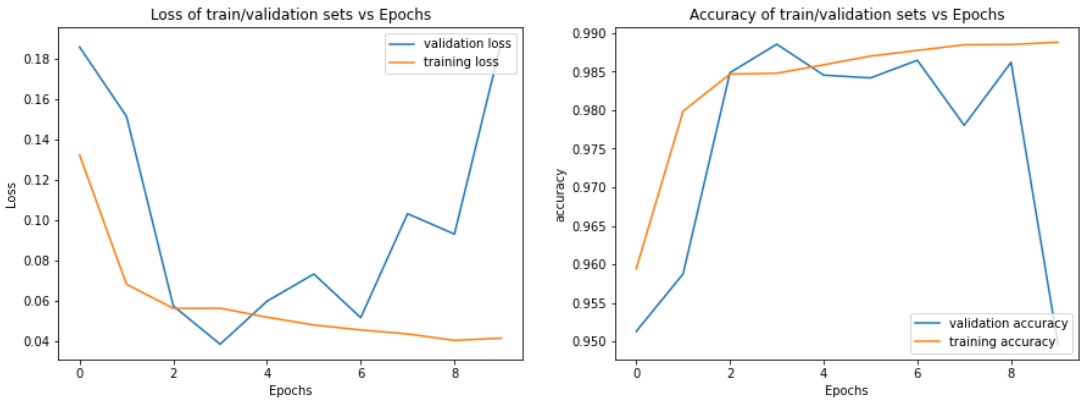

Figure (6): Loss and accuracy of validation set vs training set in model 1

Let's proceed to the second model.

We used the same architecture and configuration, but decreased the learning rate to 0.00085 and began the training with 8 epochs. The last epoch had the following results:
loss: 0.0422 - accuracy: 0.9882 - val_loss: 12.8313 - val_accuracy: 0.3155

Th e 8th epoch acted strangely as the 7th epoch had the following results:

loss: 0.0382 - accuracy: 0.9894 - val_loss: 0.0532 - val_accuracy: 0.9868

As you can see, the results at the 7th epoch of the second model was better than the performance of the last epoch in the first model. The performance of the model is depicted in the following figure.

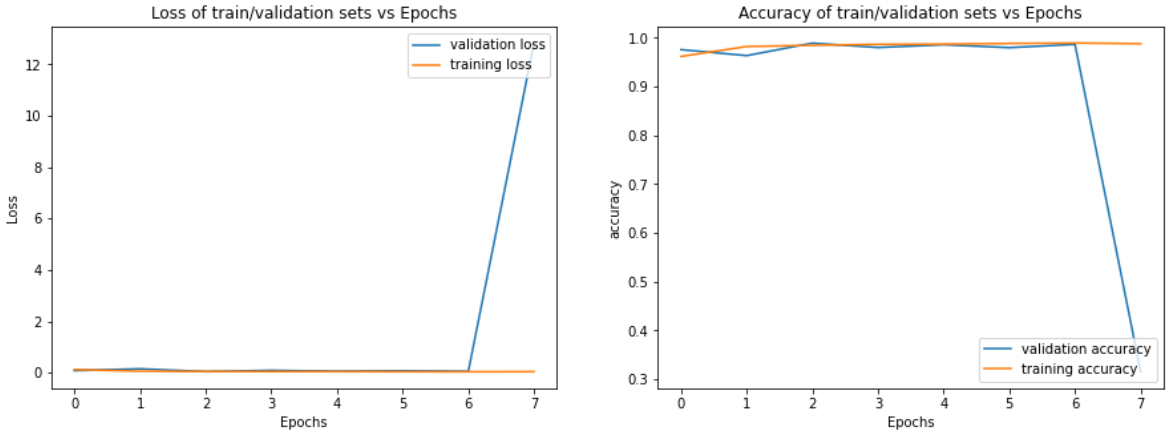

Figure (7): Loss and accuracy of validation set vs training set in model 2

Let's proceed to the third model.

We used the same architecture and configuration, but decreased the learning rate to 0.005 and began the training with 10 epochs. The last epoch had the following results:

loss: 0.2708 - accuracy: 0.9619 - val_loss: 0.3951 - val_accuracy: 0.9217

The results are worse than the prior models. The performance of the model is depicted in the following figure.

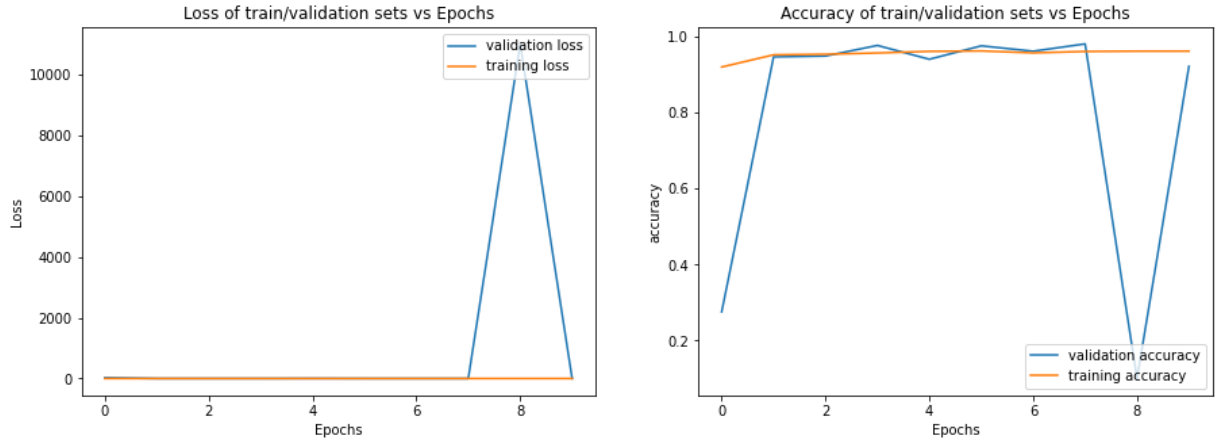

Figure (8): Loss and accuracy of validation set vs training set in model 3

Let's proceed to the fourth model.

The training was 10 epochs long, optimized by SGD with 0.0085 as the learning rate.

The last epoch had the following results:

loss: 0.0271 - accuracy: 0.9914 - val_loss: 0.0290 - val_accuracy: 0.9915

The results of this model are promising. The performance of the model is depicted in the following figure.

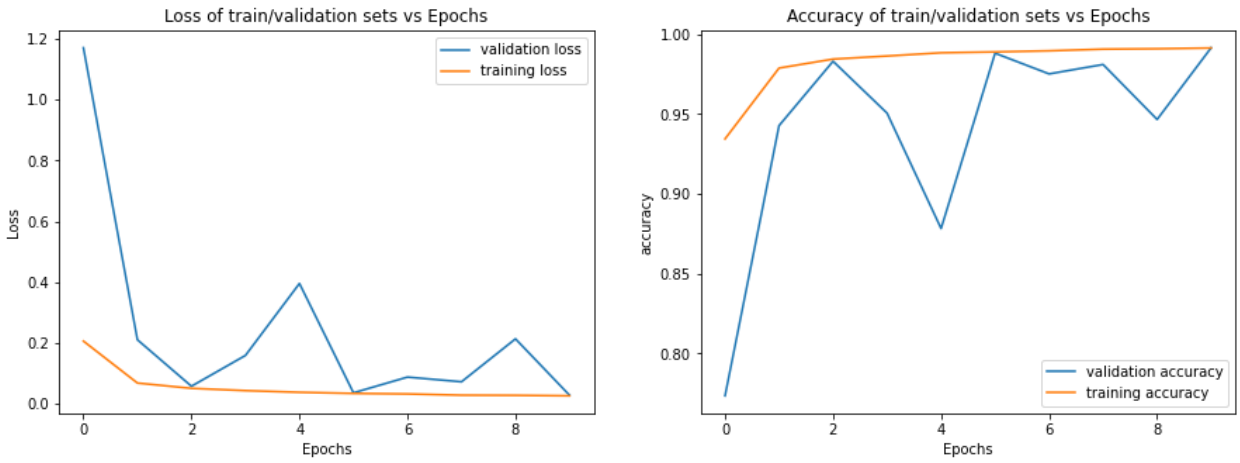

Figure (9): Loss and accuracy of validation set vs training set in model 4

Let's proceed to the fifth model.

We used the same architecture and configuration, but increased the learning rate to 0.01 and began the training with 10 epochs. The last epoch had the following results:

loss: 0.0268 - accuracy: 0.9916 - val_loss: 0.0353 - val_accuracy: 0.9897

Despite that the fourth model had more promising results, the performance of the fifth model on the validation data is more robust.

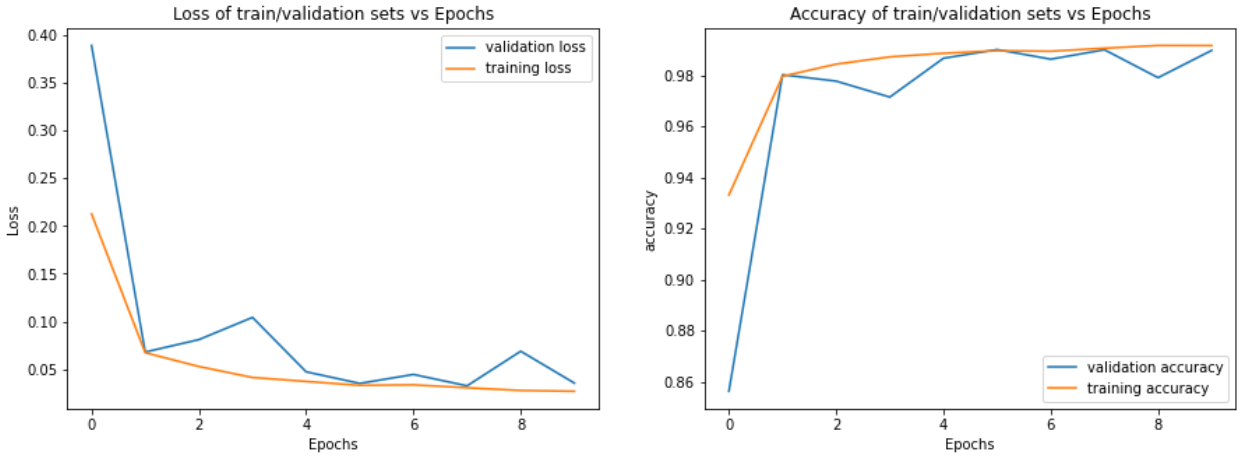

Figure (10): Loss and accuracy of validation set vs training set in model 5

Let's proceed to the sixth model.

We used the same architecture and configuration, but increased the learning rate further to 0.015 and began the training with 15 epochs. The last epoch had the following results:
loss: 0.0264 - accuracy: 0.9920 - val_loss: 0.0309 - val_accuracy: 0.9901

The performance of the model is depicted in the following figure.

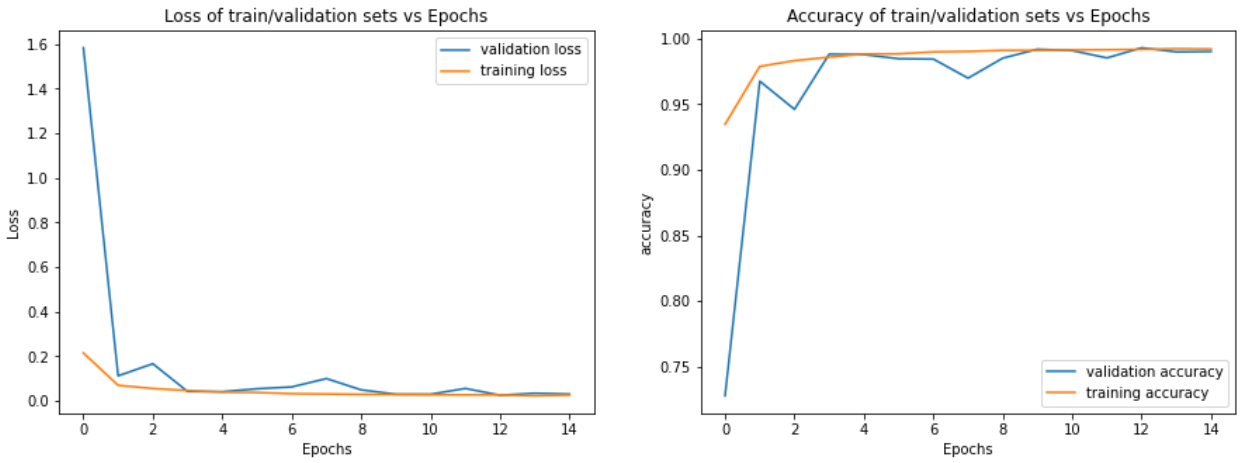

Figure (11): Loss and accuracy of validation set vs training set in model 6

It seems that we found the best model with SGD optimizer. Now let's investigate some models with RMSprop optimizer.

Let's proceed to the seventh and eighth model.

The training was 10 epochs long, optimized by RMSprop with 0.001, and 0.01, respectively as the learning rate. The last epochs had the following results:

7th model: loss: 0.0570 - accuracy: 0.9849 - val_loss: 141.6393 - val_accuracy: 0.1060

8th model: loss: 0.8472 - accuracy: 0.8616 - val_loss: 83.8924 - val_accuracy: 0.2373

As you can guess, both adversely overfitted the data, and from the following figures, we can conclude that with these settings under RMSprop optimizer, models memorize the training data instead of learning from it.

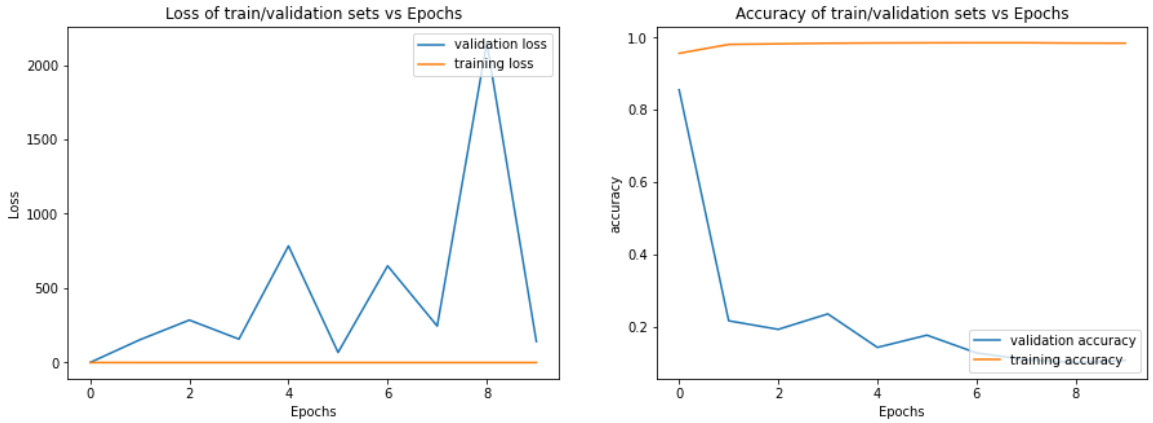

Figure (12): Loss and accuracy of validation set vs training set in model 7

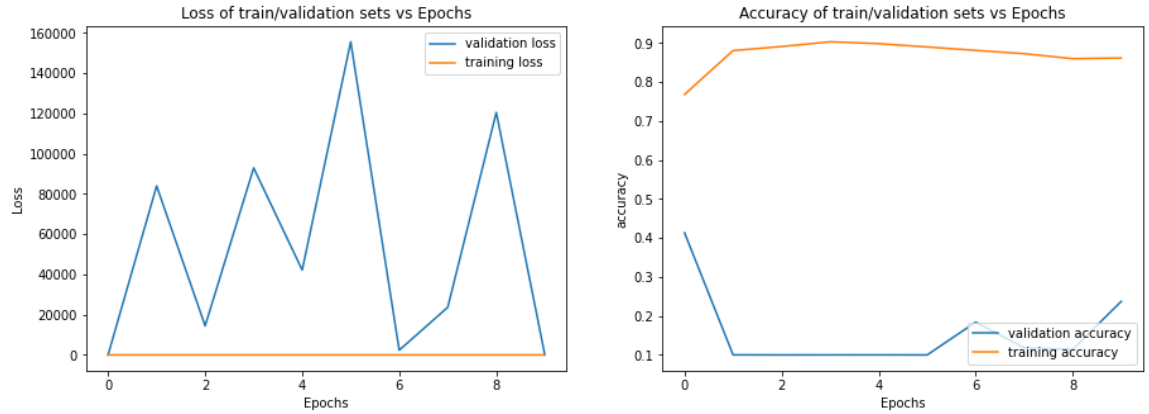

Figure (13): Loss and accuracy of validation set vs training set in model 8

Let's proceed to the ninth model.

The training was 10 epochs long, optimized by Nadam[1] with 0.001. The last two epochs had the following results:

Epoch 9: loss: 0.0416 - accuracy: 0.9888 - val_loss: 0.0911 - val_accuracy: 0.9812

Epoch 10: loss: 0.0361 - accuracy: 0.9904 - val_loss: 11.8712 - val_accuracy: 0.3093

The performance of the model is depicted in the following figure.

[1] Nesterov-accelerated Adaptive Moment Estimation

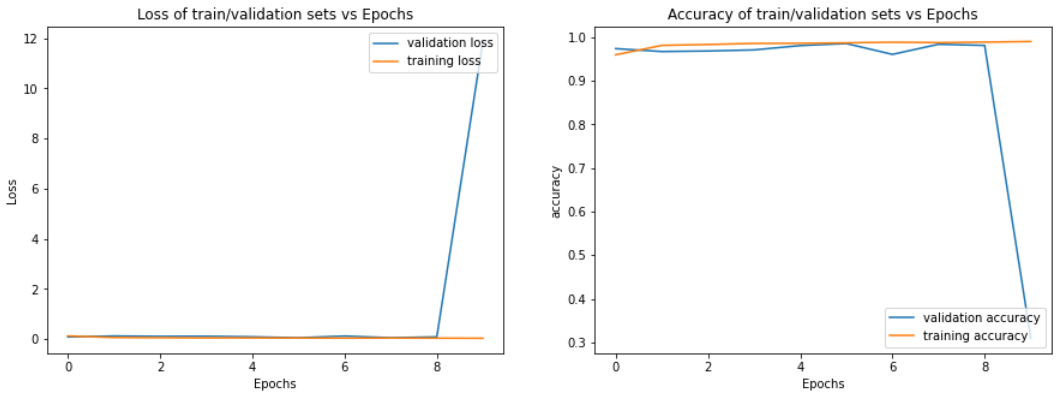

Figure (14): Loss and accuracy of validation set vs training set in model 9

Let's proceed to the tenth model.

We used the same architecture and configuration, but increased the learning rate further to 0.005 and began the training with 10 epochs. The last two epoch had the following results:

Epoch 9: loss: 0.2360 - accuracy: 0.9642 - val_loss: 0.1852 - val_accuracy: 0.9767
Epoch 10: loss: 0.2442 - accuracy: 0.9635 - val_loss: 53.4968 - val_accuracy: 0.2331

The performance of the model is depicted in the following figure.

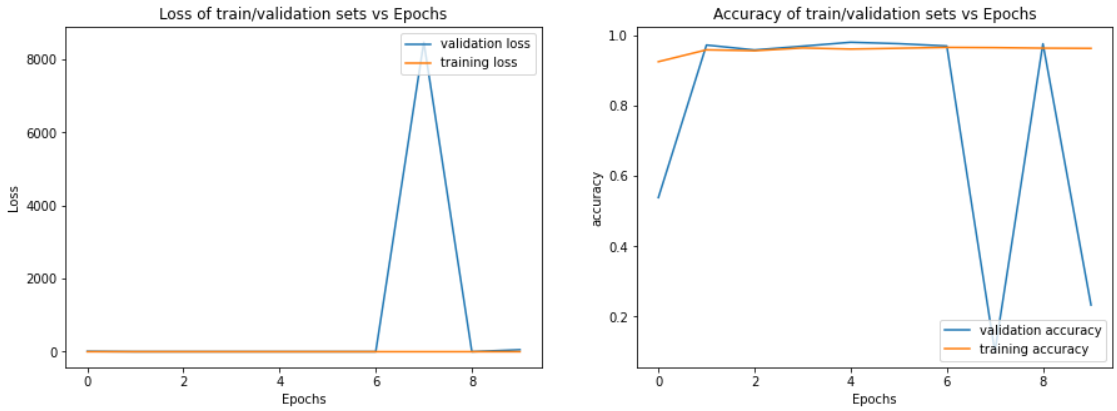

Figure (15): Loss and accuracy of validation set vs training set in model 10

Let's proceed to the eleventh model.

We used the same architecture and configuration, but increased the learning rate further to 0.00085 and began the training with 10 epochs. The last epoch had the following results:

loss: 0.0331 - accuracy: 0.9909 - val_loss: 65.1485 - val_accuracy: 0.2819

We noticed that after the 7th epoch, the model started overfitting.

The performance of the model is depicted in the following figure.

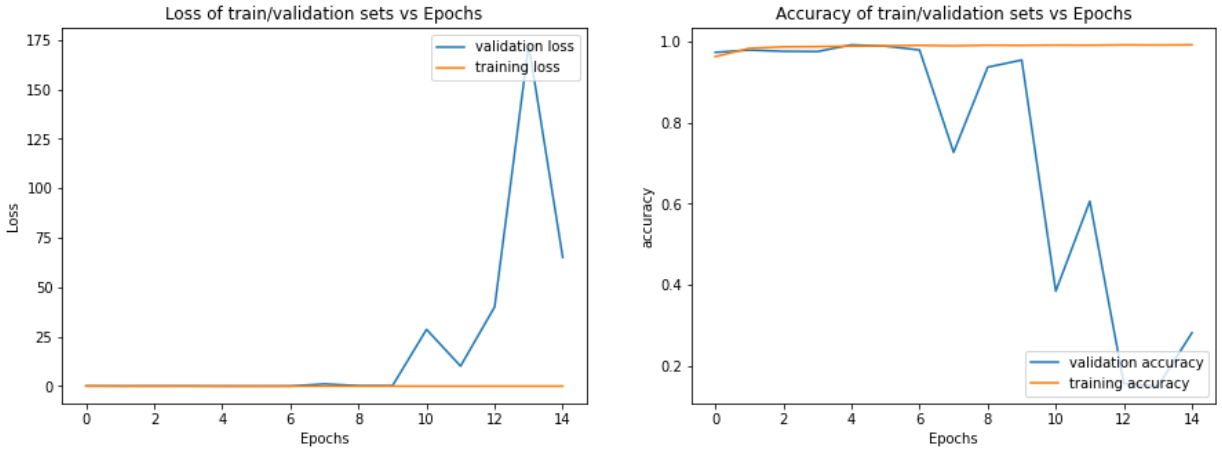

Figure (16): Loss and accuracy of validation set vs training set in model 11

Since the 12th model, we proceeded with the below architecture.

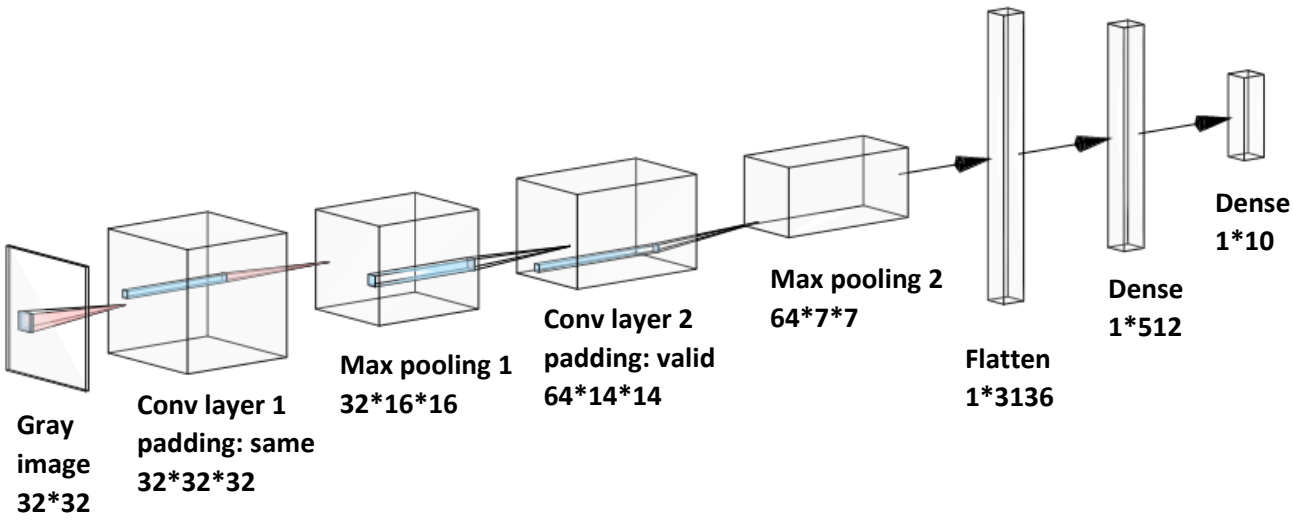

Figure (17): Architecture of the twelfth trained model

The training was 15 epochs long, optimized by Adam with 0.00085 as the learning rate.

The last epoch had the following results:

loss: 0.0897 - accuracy: 0.9723 - val_loss: 0.0648 - val_accuracy: 0.9785

The performance of the model is depicted in the following figure.

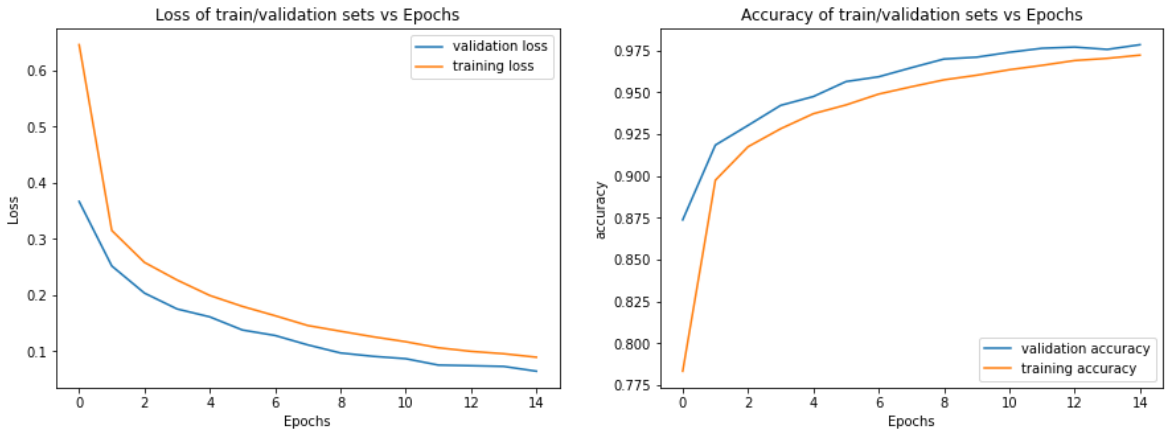

Figure (18): Loss and accuracy of validation set vs training set in model 12

As you can see, the validation performance is better than the training.

Let's proceed to the thirteenth model.

We used the same architecture and configuration, but increased the number of epochs to 25 and began the training. The last epoch had the following results:

loss: 0.0427 - accuracy: 0.9867 - val_loss: 0.0363 - val_accuracy: 0.9884

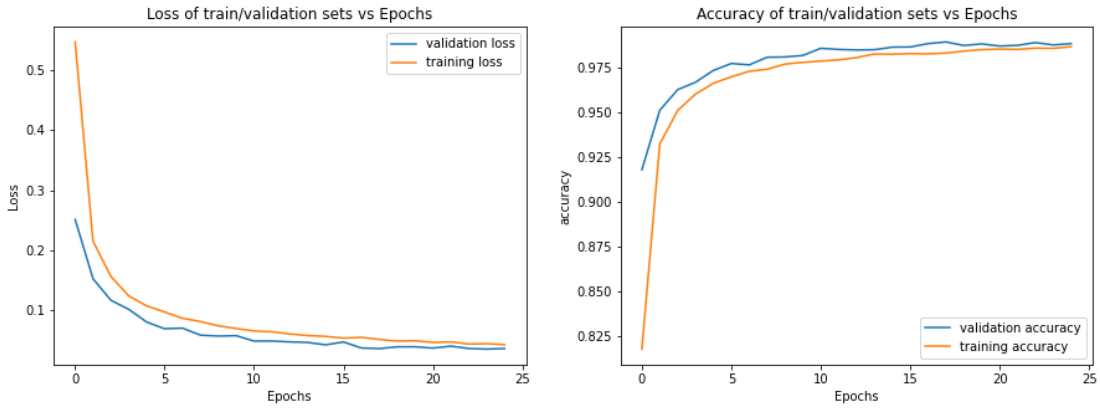

Figure (19): Loss and accuracy of validation set vs training set in model 13

Let's proceed to the fourteenth model.

We used the same architecture and configuration, but increased the learning to 0.001 and began the training with 30 epochs. The last epoch had the following results:

loss: 0.0344 - accuracy: 0.9893 - val_loss: 0.0296 - val_accuracy: 0.9910

The best model so far!

The performance of the model is depicted in the following figure.

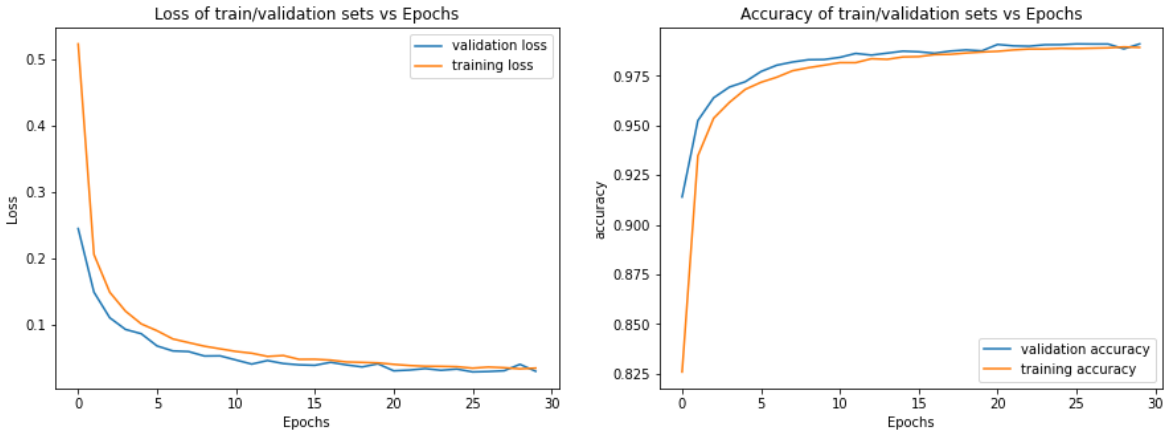

Figure (20): Loss and accuracy of validation set vs training set in model 14

The validation performance is robust as well.

Let's proceed to the fifteenth and sixteenth model.

The training was 10 epochs long, optimized by SGD with 0.001, and 0.1, respectively as the learning rate. The last epochs had the following results:

15th loss: 2.3016 - accuracy: 0.1108 - val_loss: 2.3013 - val_accuracy: 0.1001

16th model: loss: 2.3096 - accuracy: 0.0988 - val_loss: 2.3130 - val_accuracy: 0.0999

As you can guess, none of the models learned anything!

So, we will proceed with model 14, as it had the performance.

These are the first 9 images in the test set:

Figure (21): First 9 digits in the training set alongside their labels

We reached to the accuracy of 0.994 of the test set.

In the below figure you can see the visualized results.

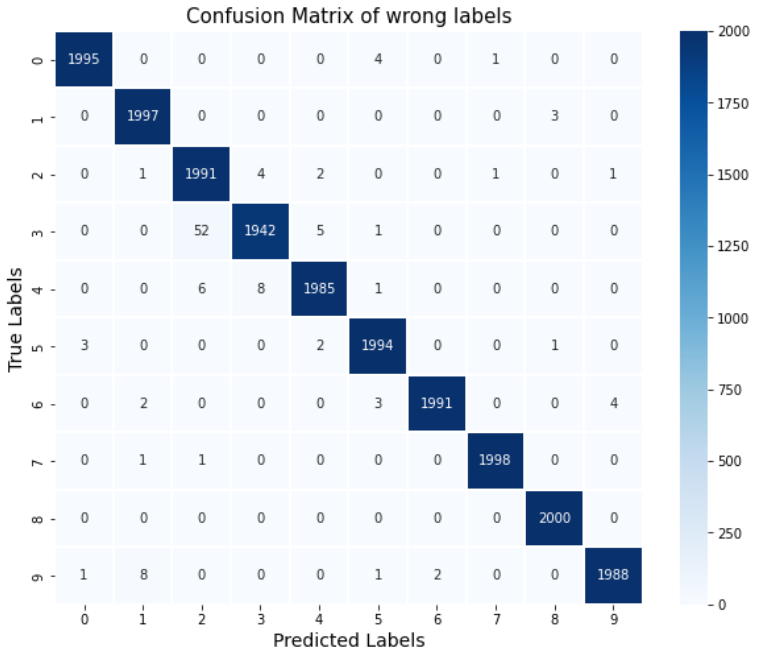

Figure (22): Confusion Matrix of the 14th model on the test set

Only digit 3 is a little hard for the model. Th ere were 52 images of digit 3 that was labeled as 2.

At last, we visualized some of the images that the model wrongly labeled. Let's take a look!

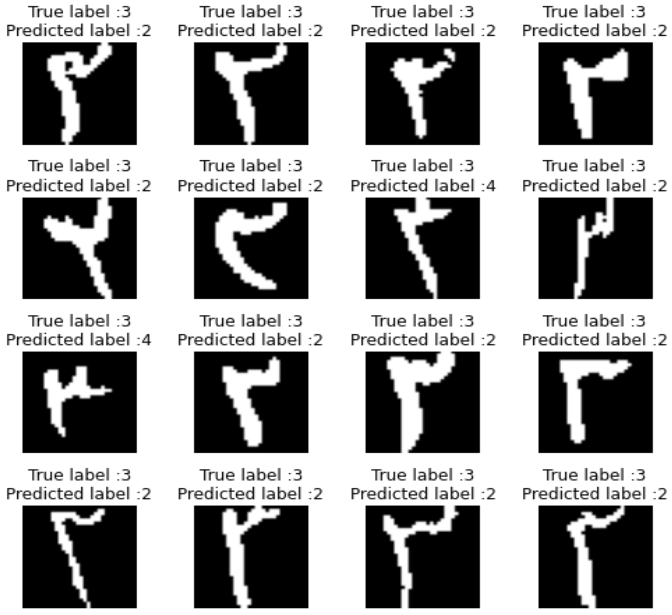

Figure (23): 16 Images of wrong predictions made by 14th model

To be fair, most of the wrongly labeled images of digit 3, looked similar to digit 2.

3. The Fashion MNIST dataset

3.1. Getting to know the Fashion MNIST dataset

Th e training set includes 60,000 28*28 images, and the test set includes 10,000 28*28 images. Both data sets are provided with the labels. In the below table, we you can see that each label represents a type of cloth.

LABEL	NAME
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneakers
8	Bag
9	Ankle boot

Table (1): Labels' corresponding cloths

In below figures, we demonstrated the number of images in each class.

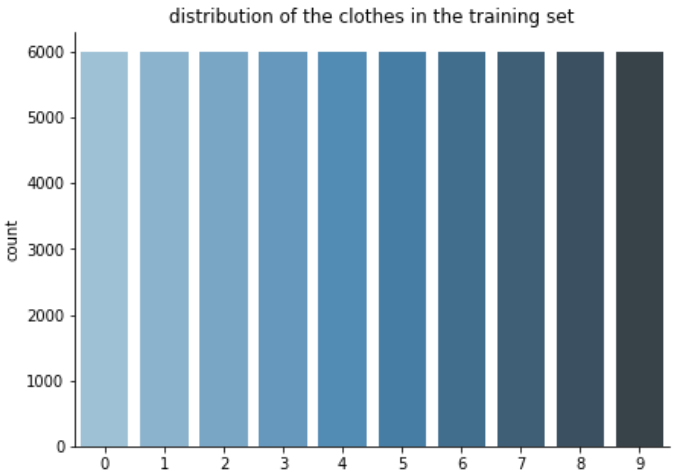

Figure (24): Bar Plot of cloth labels in the training set

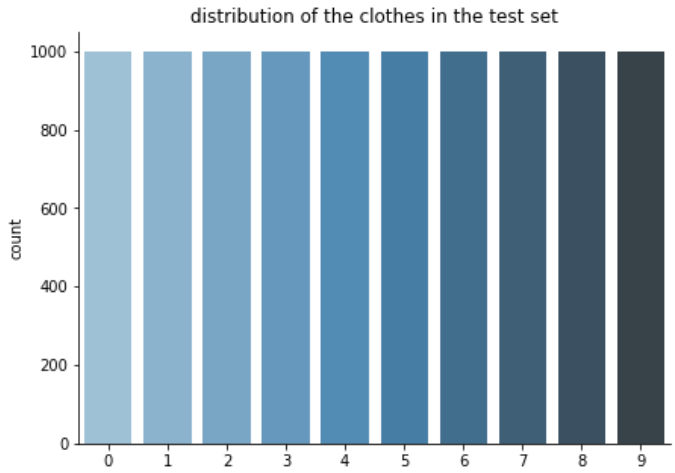

Figure (25): Bar Plot of cloth labels in the test set

Let's take a look at the first 25 images in the training dataset.

Figure (26): First 25 images in the training set

We extracted a validation set with 10,000 images out of the training set, with 1,000 data within each class. Notice than the training set now has 50,000 images with 5,000 images in each class. The following figure is the corresponding distribution for validation set.

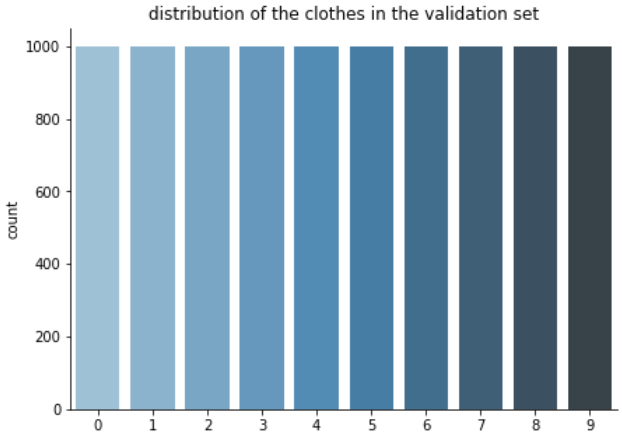

Figure (27): Bar Plot of cloth labels in the validation set

3.2. Getting to know the Fashion MNIST dataset

At the first step, we normalized all the images in training, and test sets by dividing the image matrices by 255. Following by that, the created validation set of the training images were initially normal. Regarding that the validation data was in order, we shuffled the mentioned dataset. We converted the labels to one-hot vectors. Providing that there were no missing pixels or images, we went straight forward to build image generators for the training and validation data.

We created new images by randomly rotate images through any degree between 0 and 5.
We created new images by randomly shift images 0.1 horizontally or vertically or both.
We created new images by randomly zoom in the image by 0.1 magnitude.
We created new images by randomly shearing the image by 10%.

3.3. Designing the model and start learning

The first model that we trained had the below architecture.

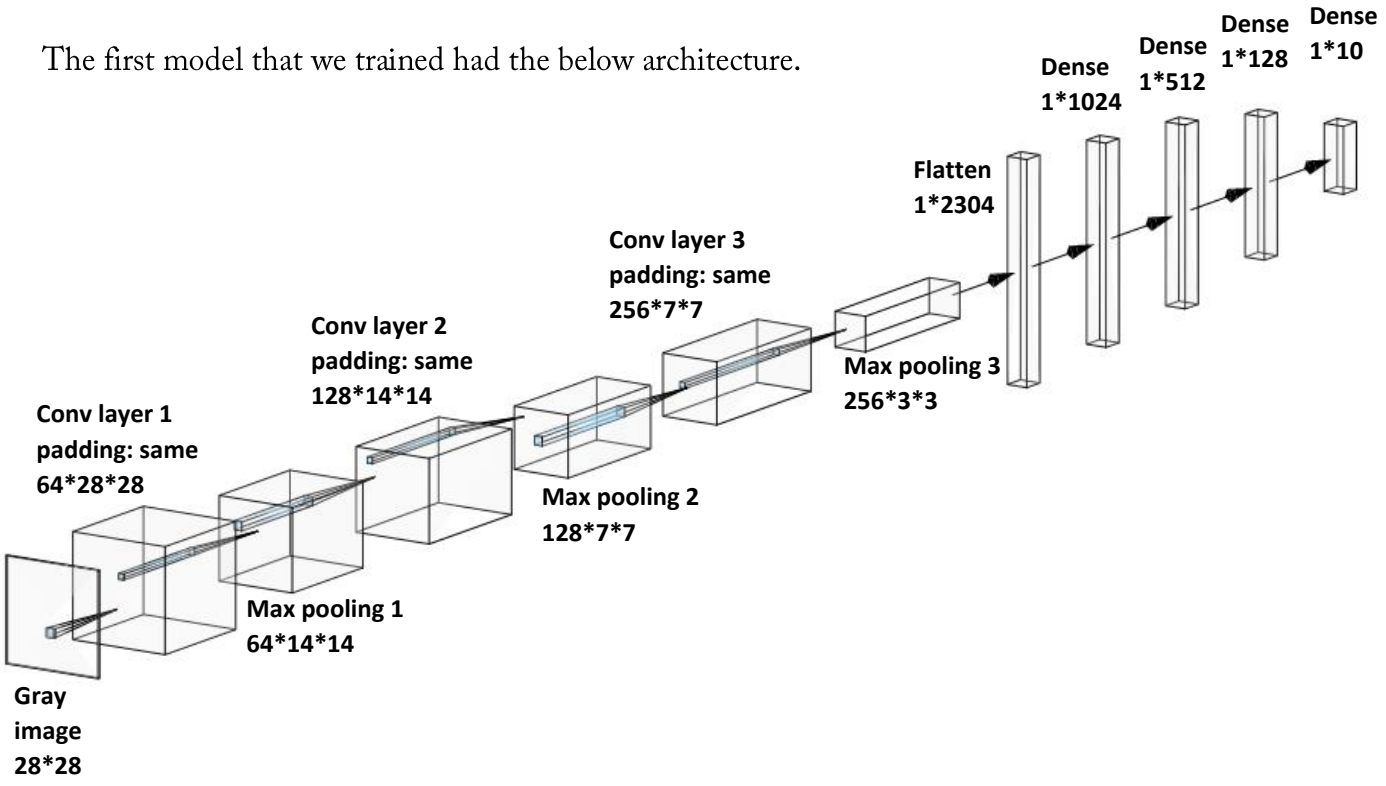

Figure (28): Architecture of the first trained model

The training was 20 epochs long, optimized by Adam with 0.001 as the learning rate.

The last epoch had the following results:

loss: 0.3557 - accuracy: 0.8746 - val_loss: 0.3203 - val_accuracy: 0.8741

The performance of the model is depicted in the following figure.

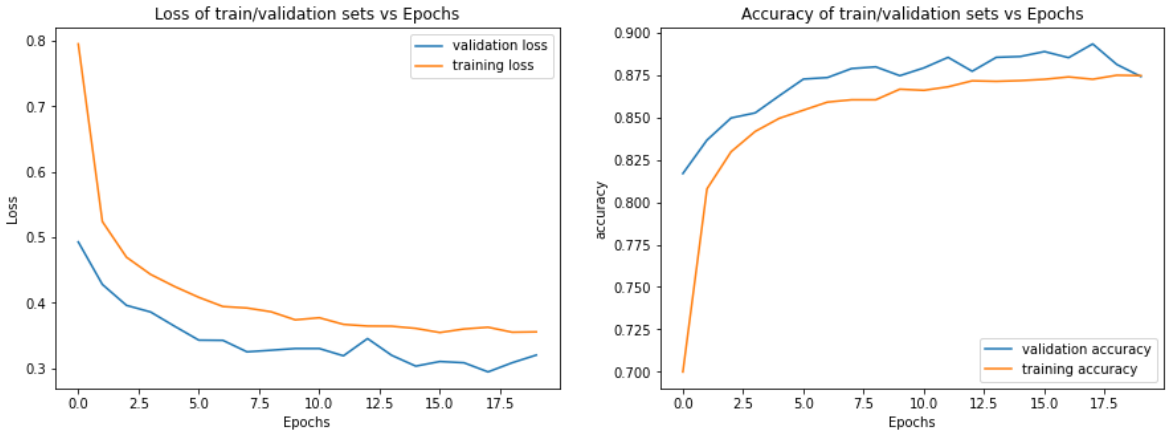

Figure (29): Loss and accuracy of validation set vs training set in model 1

As you can see, the first model is performing better on the validation set than on the training set.

Let's proceed to the second model.

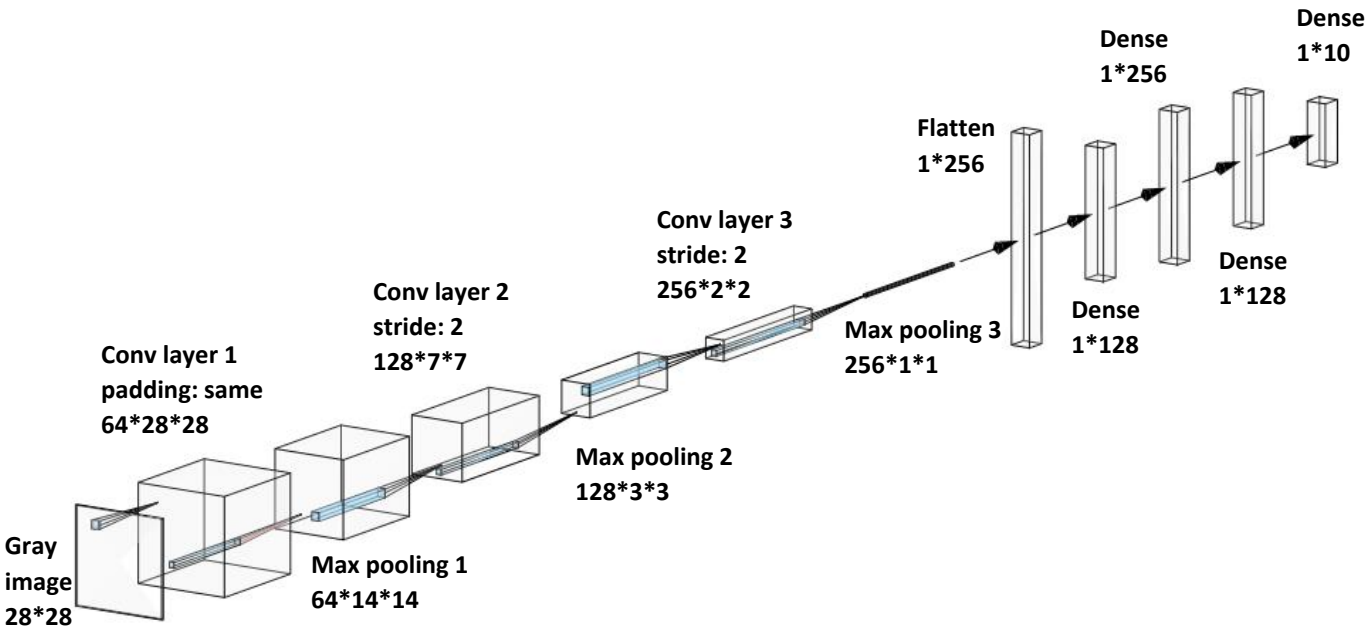

Figure (30): Architecture of the second trained model

The second model that we trained had the above architecture.

The training was 20 epochs long, optimized by Adam with 0.001 as the learning rate.

The last epoch had the following results:

loss: 0.3711 - accuracy: 0.8668 - val_loss: 0.3381 - val_accuracy: 0.8793

The performance of the model is depicted in the following figure.

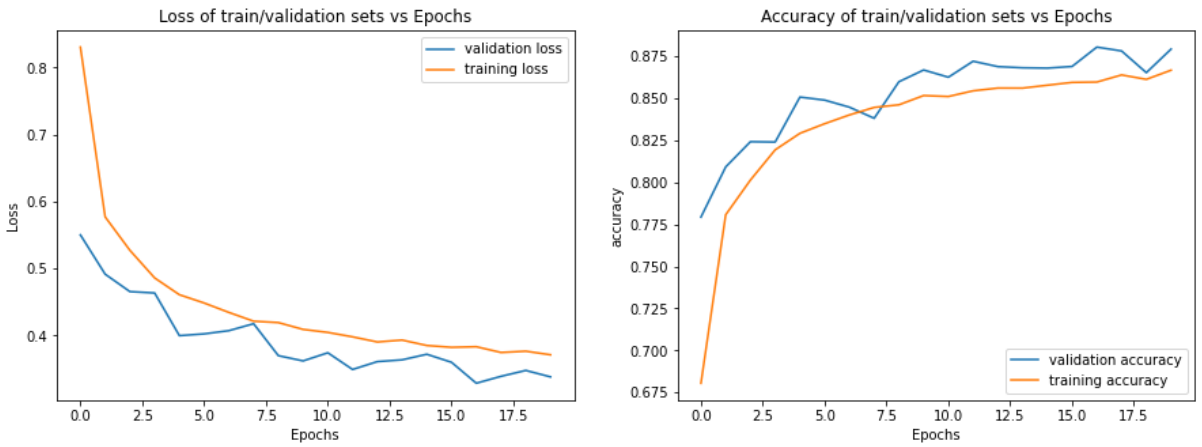

Figure (31): Loss and accuracy of validation set vs training set in model 2

Let's proceed to the third and fourth model.

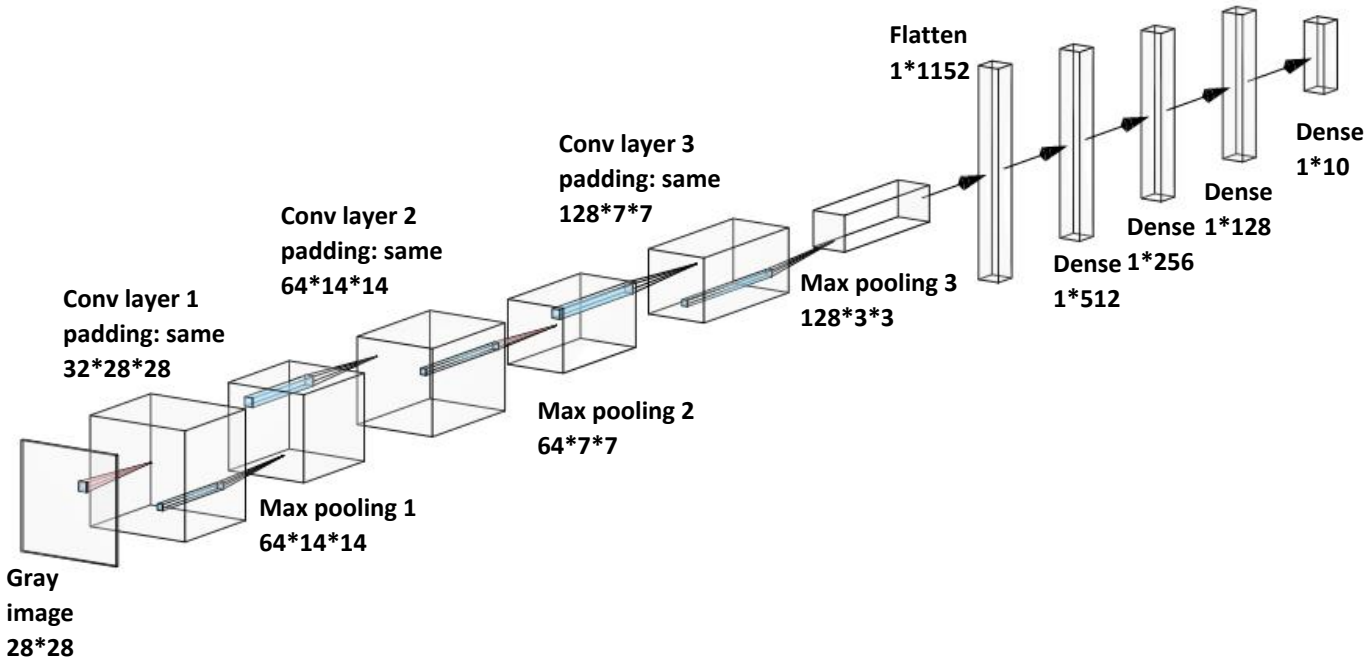

Figure (32): Architecture of the third and fourth trained models

The third and fourth models that we trained had the above architecture.

The third model's training was 20 epochs long, optimized by Adam with 0.001 as the learning rate. The last epoch had the following results:

loss: 0.2957 - accuracy: 0.8926 - val_loss: 0.2757 - val_accuracy: 0.8993

The performance of the model is depicted in the following figure.

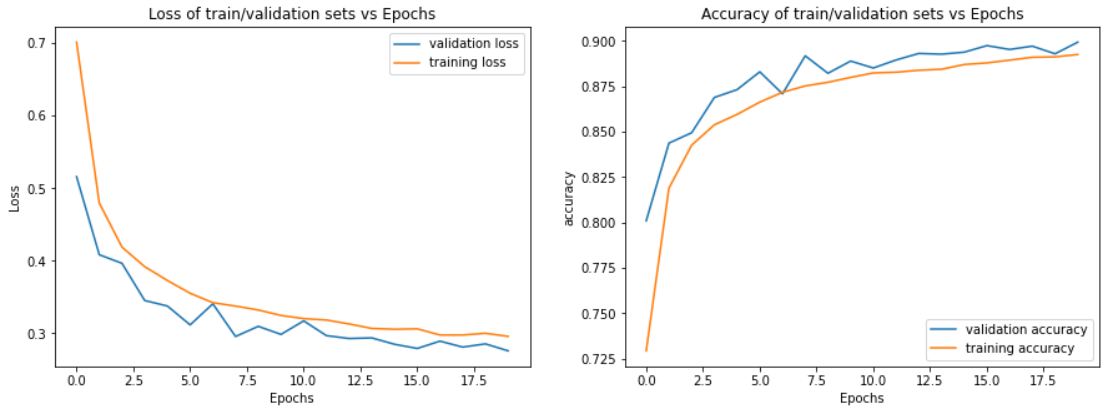

Figure (33): Loss and accuracy of validation set vs training set in model 3

We used the same architecture and configuration, but increased the number of epochs to 50. The last epoch had the following results:

loss: 0.2674 - accuracy: 0.9019 - val_loss: 0.2511 - val_accuracy: 0.9121

The performance of the model is depicted in the following figure.

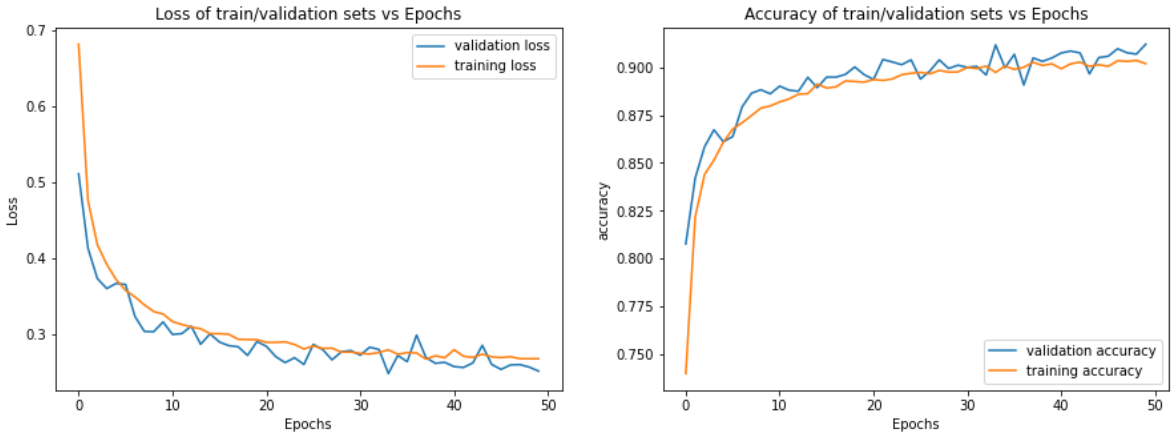

Figure (34): Loss and accuracy of validation set vs training set in model 4

As you can see, It's performance is better than the prior models.

Let's proceed to the fifth model.

From the previous architecture, we omitted 2 of the dropout layers and trained 2 models with it. The fifth model's training was 30 epochs long, optimized by SGD with 0.001 as the learning rate. The last epoch had the following results:

loss: 0.3575 - accuracy: 0.8637 - val_loss: 0.3329 - val_accuracy: 0.8772

The performance of the model is depicted in the following figure.

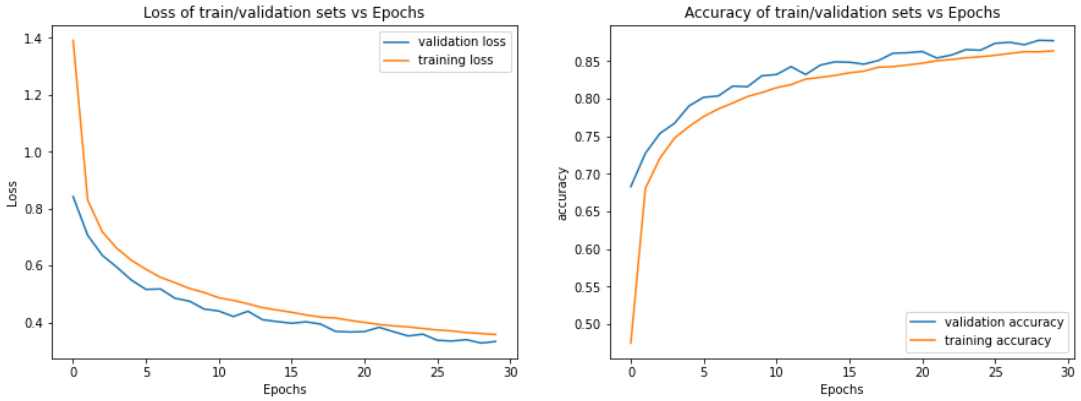

Figure (35): Loss and accuracy of validation set vs training set in model 5

Let's proceed to the sixth model.

We used the same architecture and configuration, but changed the optimizer to Adam with 0.001 learning rate. The last epoch had the following results:

loss: 0.2531 - accuracy: 0.9076 - val_loss: 0.2593 - val_accuracy: 0.9106

The performance of the model is depicted in the following figure.

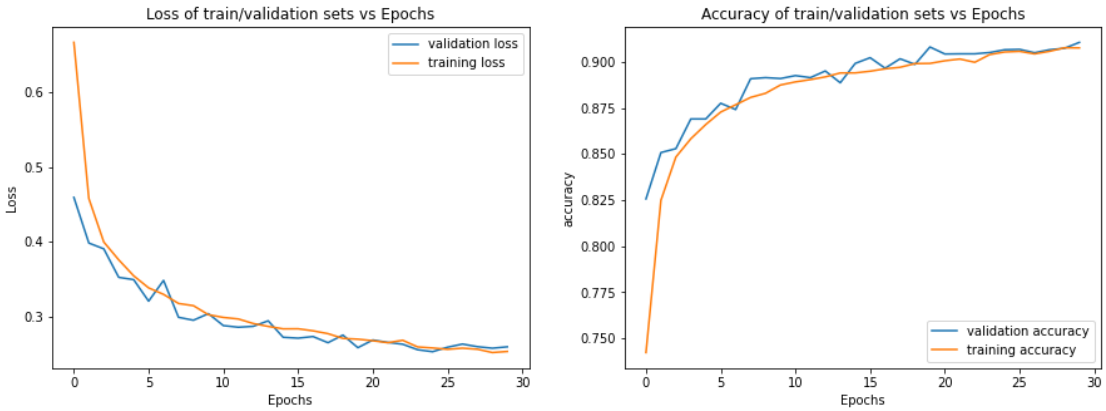

Figure (36): Loss and accuracy of validation set vs training set in model 6

So far, model 4 and 6 had the best performance. Let's proceed to the next model with a new architecture.

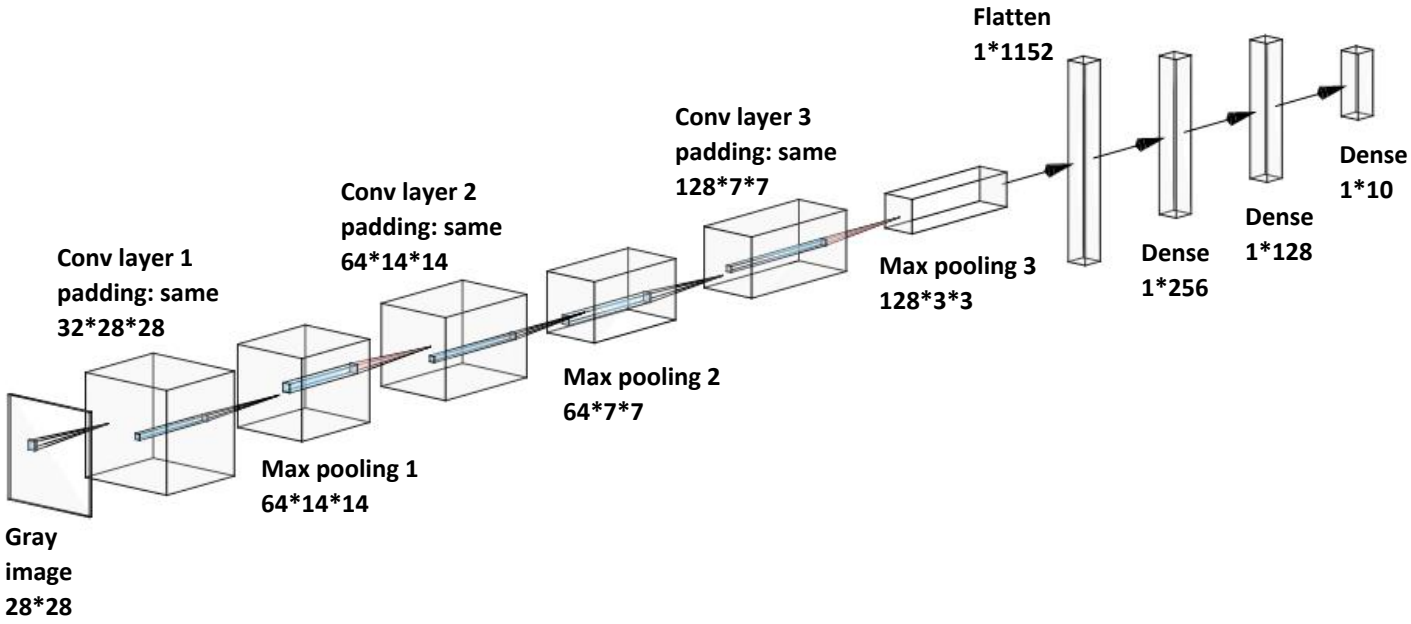

Figure (37): Architecture of the seventh trained model

The seventh model's training was 50 epochs long, optimized by Adam with 0.001 as the learning rate. The last epoch had the following results:

loss: 0.2561 - accuracy: 0.9049 - val_loss: 0.2634 - val_accuracy: 0.8994

The performance of the model is depicted in the following figure.

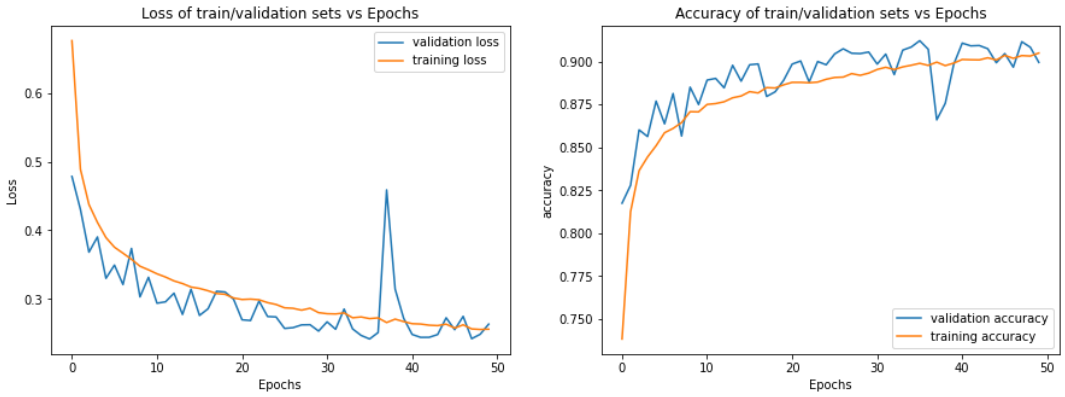

Figure (37): Loss and accuracy of validation set vs training set in model 7

Let's proceed to the eighth model. This model has the below architecture.

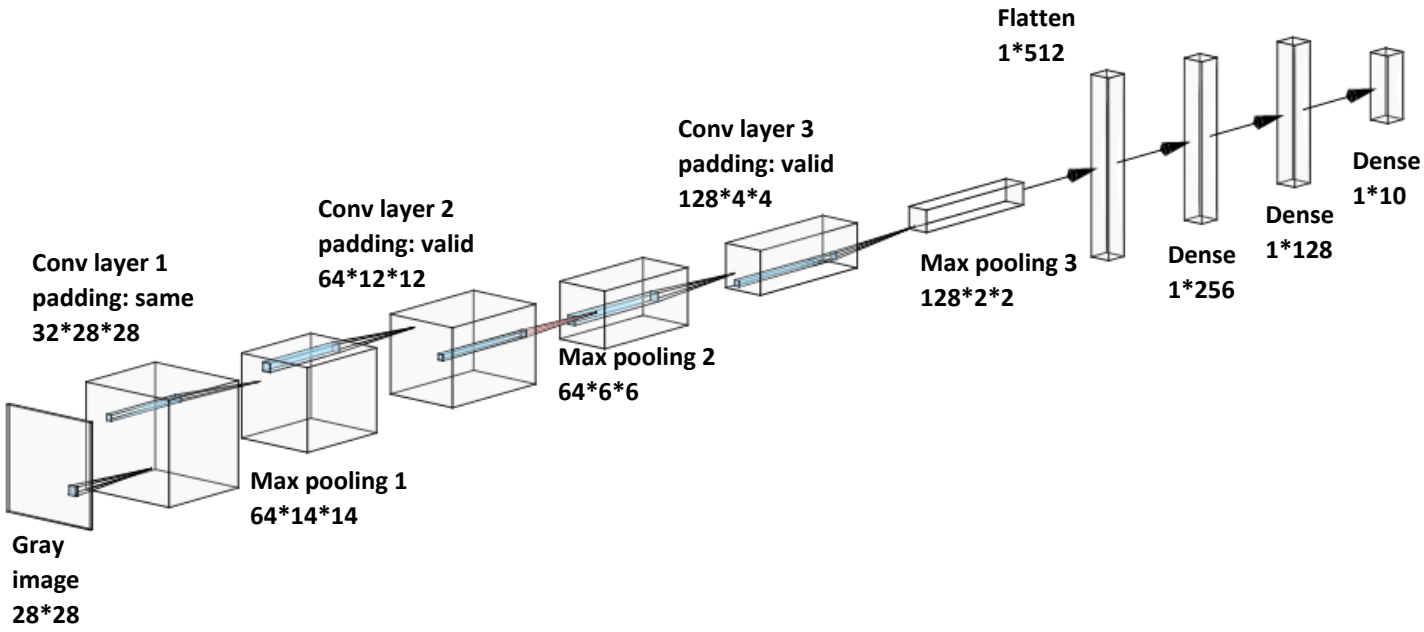

Figure (38): Architecture of the eighth trained model

The eighth model's training was 30 epochs long, optimized by Adam with 0.001 as the learning rate. The last epoch had the following results:

loss: 0.2737 - accuracy: 0.8971 - val_loss: 0.2611 - val_accuracy: 0.9049

The performance of the model is depicted in the following figure.

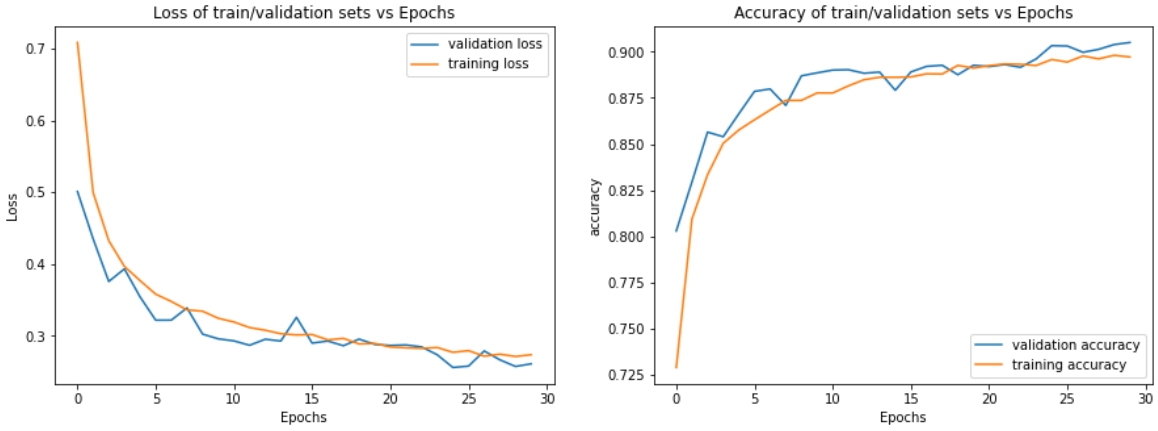

Figure (39): Loss and accuracy of validation set vs training set in model 8

This model has a satisfying performance as well.

We should pick a model between model 4, 6, and 8.

We will choose a model regarding their performance on the test set.

	LOSS	ACCURACY
MODEL 4	0.253	91.32%
MODEL 6	0.245	91.31%
MODEL 8	0.240	91.21%

We will go with model 8 which has the lowest loss.

Figure (40): First 25 images in the test set along the true labels and the predictions

In the above figure, the model mistakenly labeled the ankle book as a sandal.

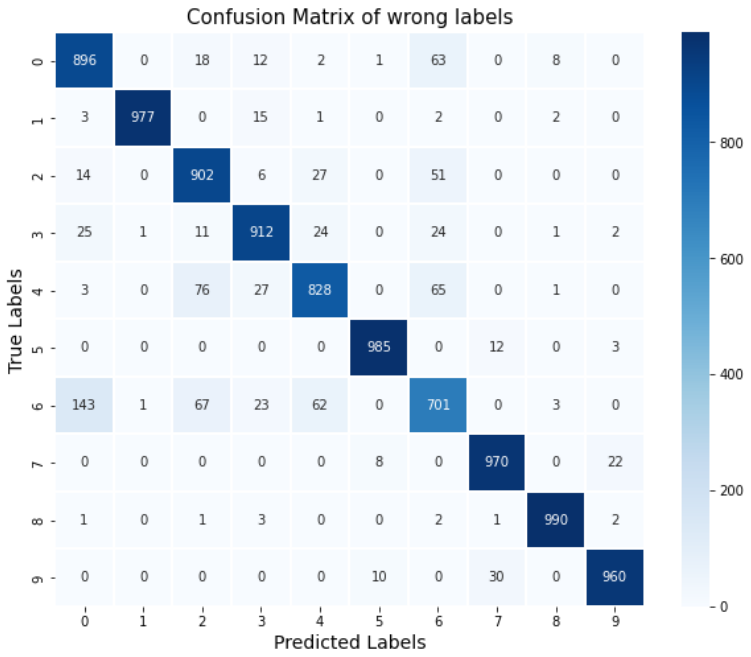

Figure (41): Confusion Matrix of the 8th model on the test set

As you can see, the model confuses similar cloths. For example, the model confuses 143 shirt images with t-shirt/top.

3.4. ResNet on Fashion MNIST

The ResNet model's training was 50 epochs long, optimized by Adam with 0.001 as the learning rate. The last epoch had the following results:

loss: 0.2267 - accuracy: 0.9160 - val_loss: 0.4077 - val_accuracy: 0.8642

The performance of the model is depicted in the following figure.

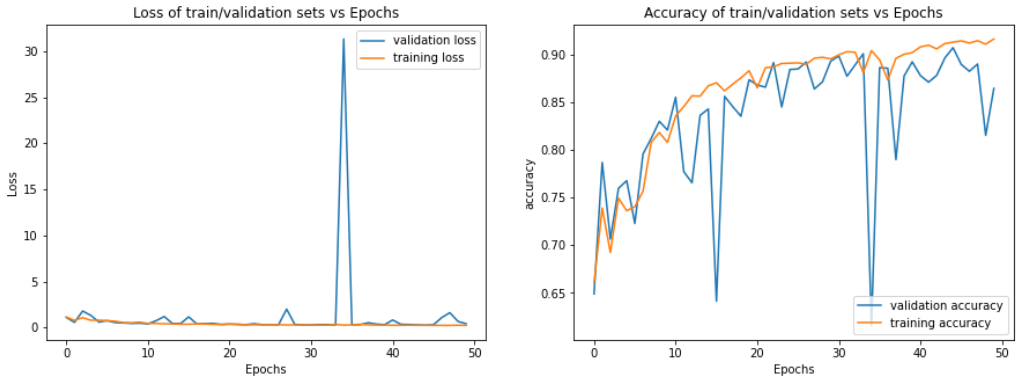

Figure (42): Loss and accuracy of validation set vs training set in the ResNet

Below we put the performance of the ResNet on the test set:

ResNet Test loss: 0.329

ResNet Test accuracy: 0.89

By comparing our selected model with the output of the ResNet, we figured the our model is better.

For the last part, let's take a look at the confusion matrix of ResNet on the test set.

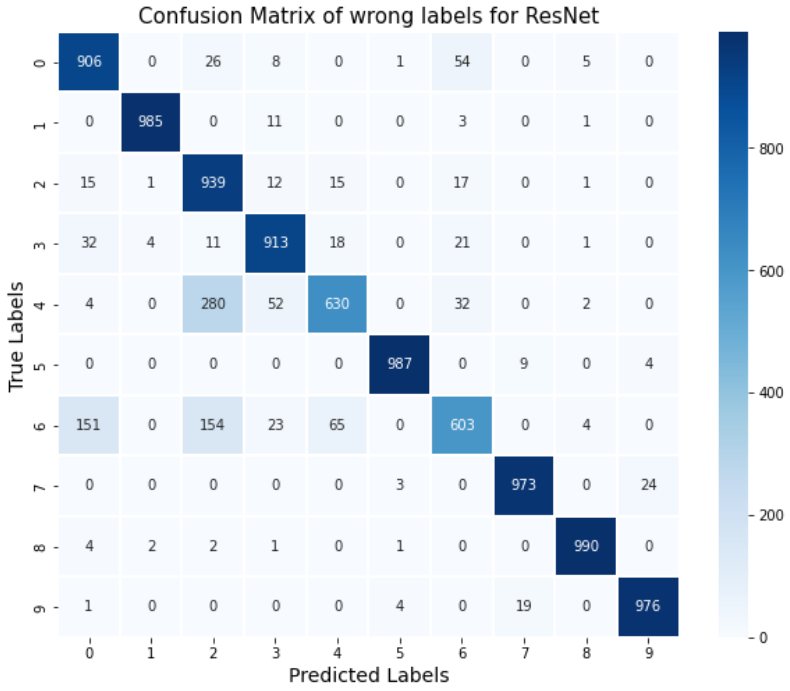

Figure (43): Confusion Matrix of the ResNet model on the test set

As you can see, the model confuses similar cloths. For example, the model confuses 280 coat images with Pullover, or 151 shirt images with t-shirt/top.