Shani Klein – [redacted]
Shaked Cohen [redacted]

Deep Learning and its applications to Signal and Image Processing and Analysis ASSIGNMENT 1 - BASIC DEEP LEARNING

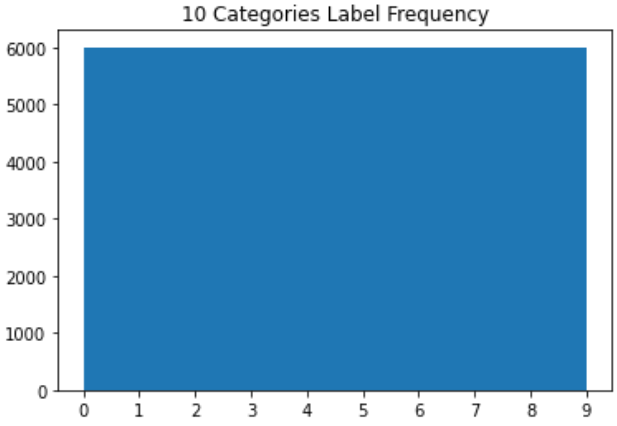
In this report we will present our findings on training a CNN to predict fashion articles from the MNIST- fashion dataset.

First, we explored the MNIST-fashion dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels).

1.1 Dataset Manipulation

1. Change the labels of the images to match the following labels:

As we said above, the original dataset has 10 categories, each label has the same frequency as we can see in the following histogram:



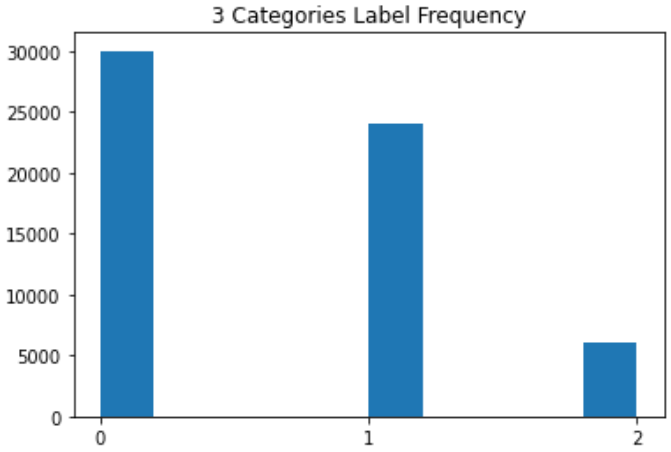
The first task was to change the labels of the images to match the following labels:

(a) Top: T-Shirt/Top (0), Pullover (2), Dress (3), Coat (4), Shirt (6)

(b) Bottom: Trouser (1), Sandal (5), Sneaker (7), Ankle Boot (9)

(c) Accessories: Bag (8)

We map each label to its new label. Now the label histogram is as follow:



We can tell that label 2 has only Bag so we have only 6,000 images for label Accessories while the Top category has 5 articles of clothing, so we get 30,000 images (6,000 per cloth type).

Next, we explored the train dataset and we saw that there were 60,000 images in the training set, with each image represented as 28 x 28 pixels. In addition, the pixel values for each image fall in the range of 0 to 255.
So the first pre-processed step we did was normalize the images to a range of 0 to 1 by divide the values by 255, That ensures that each pixel has a similar data distribution, which will make convergence faster while training the network.

Now let's look at some of the image in our new normalize, 3 categories dataset:

We can tell that there are only 3 categories as we wanted, the images are indeed in grayscale and we notice that all shoes for example are pointing to the left.

Data Augmentation:

Now that we explored the dataset we can start and make data augmentation.
Data augmentation is a technique used to increase the amount of data for training by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It will make our model more robust and will help reduce overfitting when training our model.

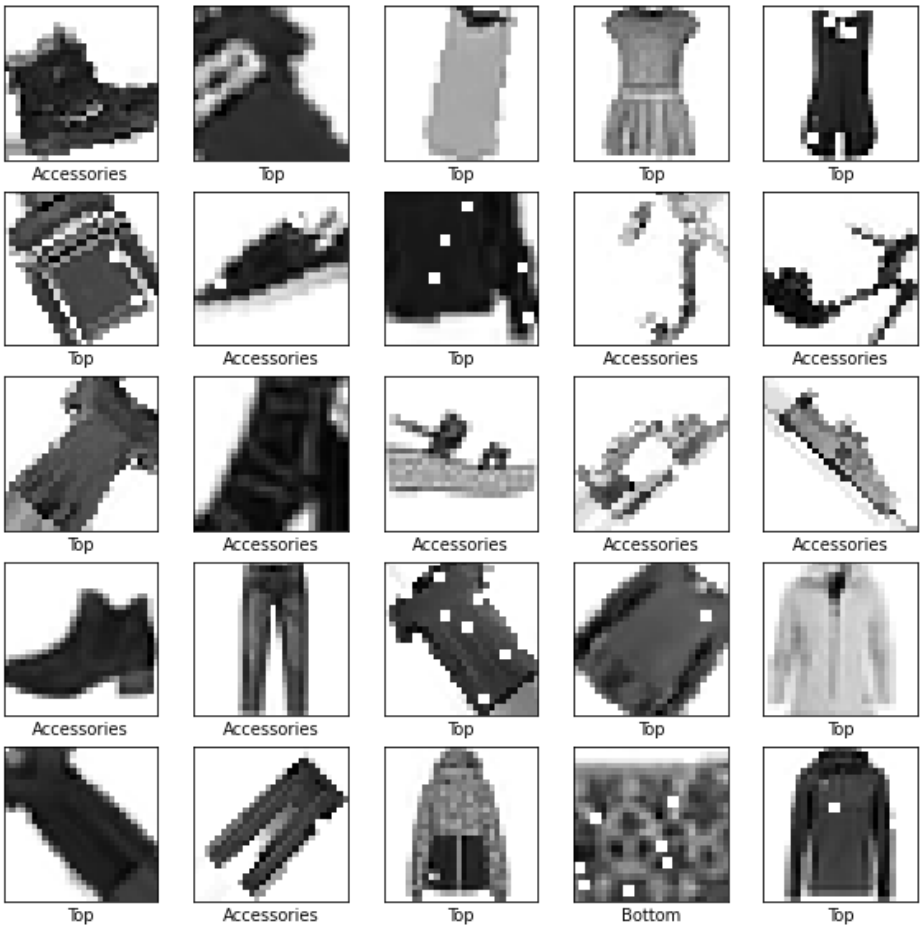
Since the image came originally in a grayscale, augmentation in pixel level such as change saturation, brightness, etc. won't be efficient.

First, we tried several augmentation such as: flip image (left-right and up-down) , rotate the Image in different angels, randomly crop part of the image , and cutout which masks out square regions of input during training.

The main motivation of the rotation and flipping is that the model should recognize and an of a shirt for example even if the image is flipped or rotated , in addition even if we see only part of the shirt we should recognize it is a shirt (hence the cropping) and the motivation of the cutout is to predict our model to recognize shirt even with object occlusion.

All the augmentation was picked randomly per image so there are images that was not augmented or image that got multiple augmentations.

Data after augmentation:



We can see that the augmentation is indeed randomly preformed. In addition we find it hard to recognize with our own eyes some of the image after the augmentation for example when apply random cropping so we may increase the cropping size or will not use this augmentation if the model results won't be as good as we expected.
*we trained our models on data with and without augmentations, since the test data is very similar to the training data and not real world data, we got better performance for the model trained without augmentation (about 1-1.5% better accuracy), but it is less likely to generalize well to unseen data.

1.2 Choosing a Model

We tried different combination of layers:

- Add Conv2d layers to use the spatial features in the images- the main motivation behind using convolution is that computers see images using pixels. Pixels in images are usually related. So, convolution NN is popular when identifying\ classify images.

- Try using regularization layers such as Batch Normalization and Dropout-
  The main motivation to use regularization is to avoid overfitting.

  Let's explain each method and the idea behind it:
  Batch Normalization - as we learnt in class, Batch normalization is a technique to standardize the inputs to a layer, applied to the activations of a prior layer or inputs directly.
  Batch normalization accelerates training, it is used to overcome problems such as:
  - Exploding \vanishing gradients
  - Overfitting - used as a layer regularization method.

Dropout – is a method where you randomly deactivate nodes in the layer during training.
This is done I order to allow the network to learn different features (or network paths) and avoid a case where the network predicts using only one feature.

- Try using loss regularization such as L1/L2,- the main motivation to use loss regularization is to reduce overfitting by feature selection where L1 is usually considered as hard feature selection since it compels some of the features to be zeroed out, and L2 is usually considered as soft feature selection since it compels a combination of the features.
  it is used with a λ parameter that helps control the effect, in a NN it compels the weight to be small and help avoid over complex solutions thus help avoiding overfitting of the model.

The model we experimented with: we add different components to each model to help see their effect on the training and results:

- **Model 1:** this is the basic model of the notebook, for baseline comparing.
  The architecture of the model and the number of trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

- **Model 2:** basic 2 layers of Conv2D and max pooling with pool size of (2,2) and a stride of 2, this makes it a bit harder to calculate the receptive field, gives us a receptive field of 10 pixels in the 2nd Conv2D layer (the max pooling with stride gives a larger respective field faster) and a receptive filed of 16 pixels in the output of the 2nd max pooling layer, hence we get a good receptive field (since the size of the matrix is at this point 7x7). We use this in the following models as well.

The architecture of the model and the number of trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 387
=================================================================
Total params: 420,739
Trainable params: 420,739
Non-trainable params: 0
_________________________________________________________________
```

- **Model 3:** here we add to model 2 l2 regularization to the dense layers of the model

  The architecture of the model and the number of trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 387
=================================================================
Total params: 420,739
Trainable params: 420,739
Non-trainable params: 0
_________________________________________________________________
```

- **Model 4:** here we add to model 2 dropout layers
  The architecture of the model and the number of trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
dropout (Dropout)            (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 387
=================================================================
Total params: 420,739
Trainable params: 420,739
Non-trainable params: 0
_________________________________________________________________
```

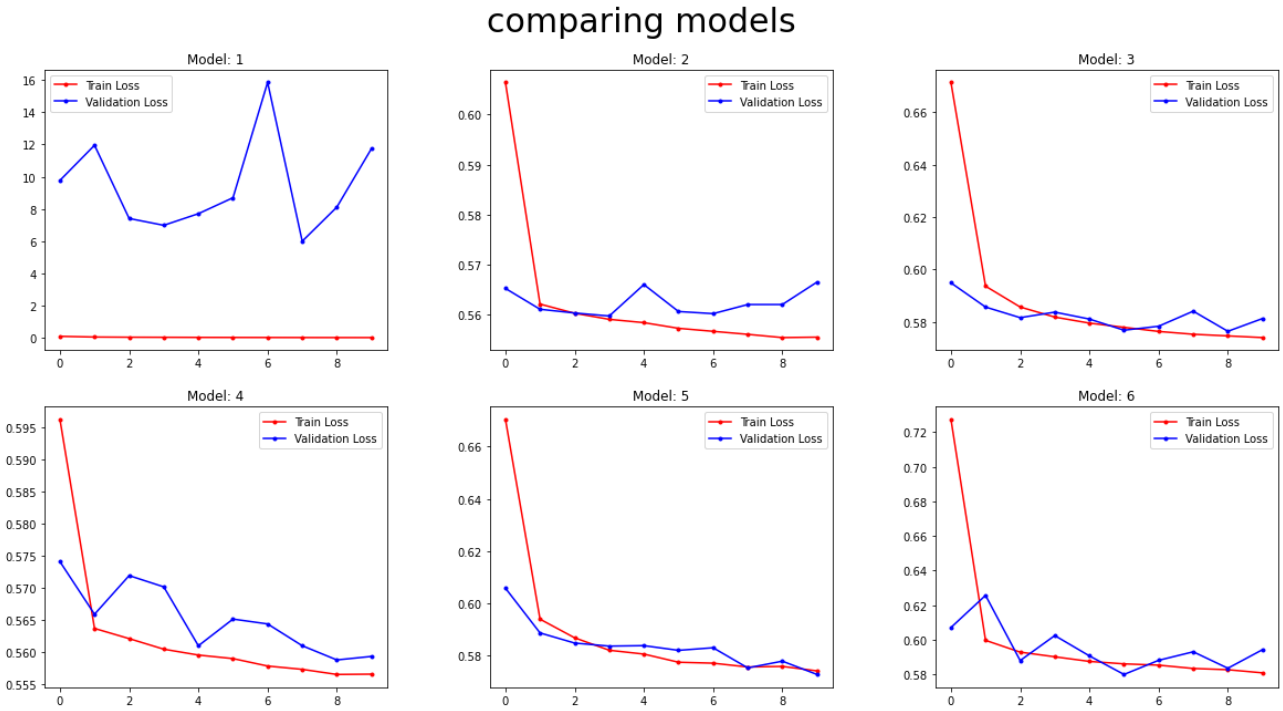
- **Model 5:** here we add to model 4 l2 regularization to the dense layers of the model
  The architecture of the model and the number of trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
dropout_2 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 387
=================================================================
Total params: 420,739
Trainable params: 420,739
Non-trainable params: 0
_________________________________________________________________
```

- **Model 6:** here we add to model 4 a batch normalization layer and l2 regularization to the dense layers of the model and the second Conv2D layer. The architecture of the model and the number of trainable parameters:

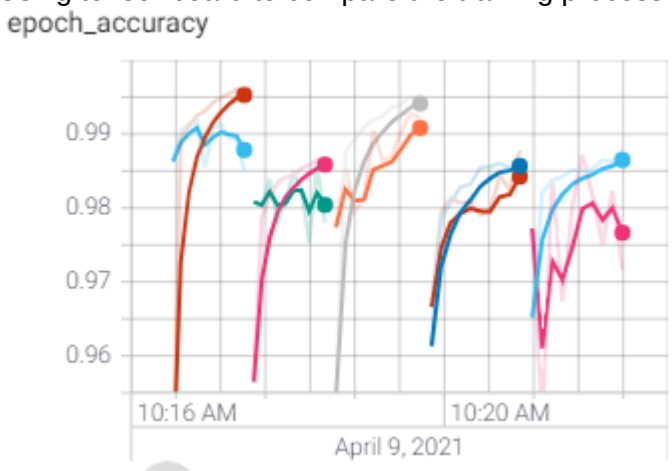
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
dropout_4 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
activation (Activation)      (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
batch_normalization (BatchNo (None, 14, 14, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 14, 14, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               401536
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 387
=================================================================
Total params: 420,995
Trainable params: 420,867
Non-trainable params: 128
_________________________________________________________________
```

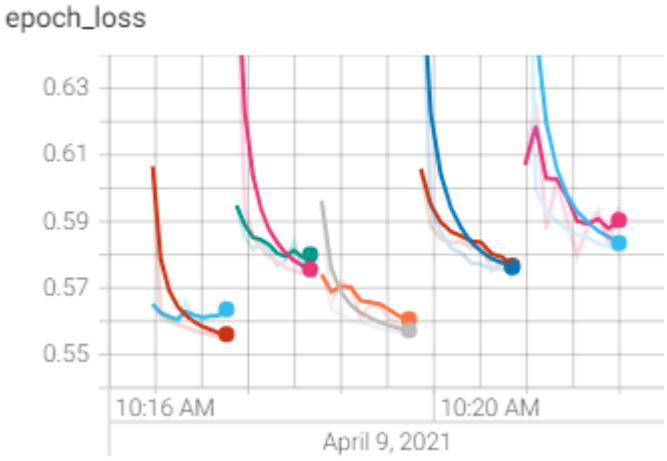
After training the models we compared their loss graphs:



By exploring these graphs, we can learn a bit about the learning process of the network, we can see the big difference from the model without conv2D layer (model 1) and the rest of the models very clearly.
In the graph of the second model we can clearly see when we start the over fitting of the model when the validation loss starts to grow apart from the training loss.
We can see in the graph of model 3 where we add l2 regularization that it helps with training but converges slower.
We can see in the graph of model 3 where we add drop out layers that the validation loss has more fluxion since the random drop of neurons in training can change the way the network learns (as we expected and desired).
In model 5 we see the effect of l2 regularization on the NN (model 4 +l2 regularization).
In model 6 we add a layer of batch normalization, and l2 regularization to the second conv2D layer, it seems to cause fluxion.
It appears that model 5 has the most potential with further epochs of training (and maybe model 6 as well).

Using tensor board to compare the training process:





We left model 1 out of the images since it is too different (it is in a different scale). From left to right we see models 2 to 6.
Here we can see the behavior of the model2 next to each other and in the same time compare their overall accuracy and loss.
It seems that model 2 and 4 achieve a lower loss and higher accuracy in training and in validation compared to the rest.

1.3 Evaluation:

Before we start review our evaluation methods, we explain some terms:

- **True Positive (TP):** an outcome where the model correctly predicts the positive class. For example, the model predicts a shirt as a shirt.
- **False Positive (FP):** an outcome where the model incorrectly predicts the positive class. For example, predicts a shirt as a shoe when the class is shoe.
- **False Negative (FN):** an outcome where the model incorrectly predicts the negative class. For example, the model didn't predict a shirt as a shirt.
- **True Negative (TN):** an outcome where the model correctly predicts the negative class. For example, the model didn't predict shoes as a shirt.

Now let's discuss our evaluation of the models. We tried different evaluations for the models:

Overall accuracy: The Overall accuracy is the probability that an individual will be correctly classified by a test. Accuracy is defined as follows (for the binary case):

$$Acc = \frac{TP + TN}{P + N}$$

In our case since we have multi class classification the overall accuracy is:

$$Acc = \frac{\sum_{i \in class} TP_i}{P + N}$$

Our models accuracies:

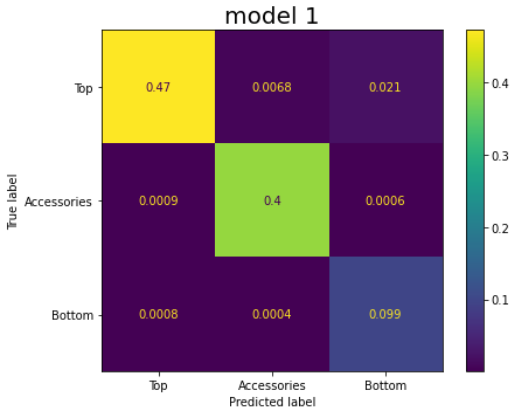
	test_loss	test_acc
model_1	11.742665	0.9698
model_2	0.566435	0.9850
model_3	0.581192	0.9780
model_4	0.559280	0.9922
model_5	0.572829	0.9878
model_6	0.594231	0.9717

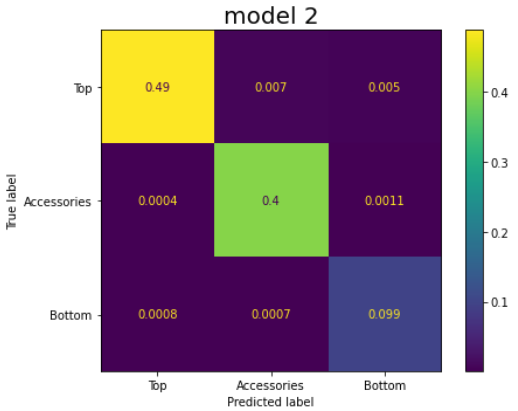
We can tell that for model 1 the accuracy might be good, but the loss is pretty high. it means that the model is very good when predicting, but there are some outliers of a huge impact on the loss.
It seems that model 4 gives us the best results after 10 epochs.

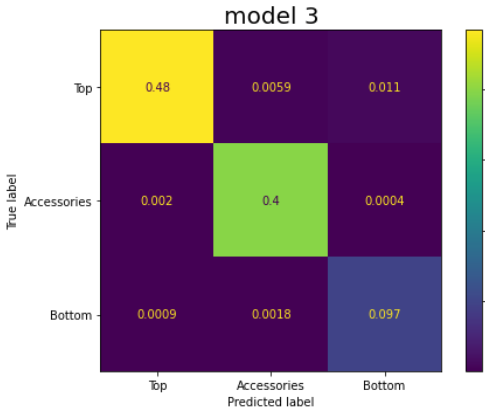
Per class accuracy: the per class accuracy is defined as the accuracy of the class where it a binary classification problem, meaning we define the TN of the class to be the prediction of an individual that is not in the class to be predicted outside of the class (even if the prediction inside the other class was wrong).
Our models per class accuracy:

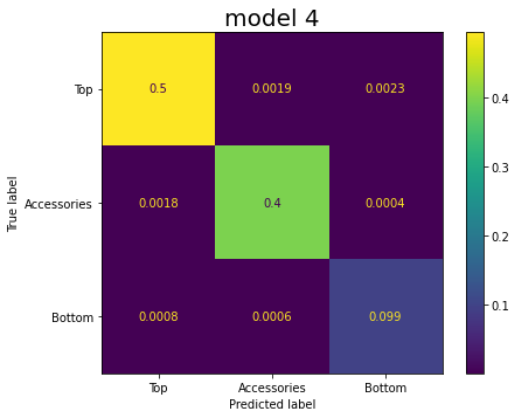
	Top_acc	Accessories_acc	Bottom_acc
model_1	0.9867	0.9924	0.9913
model_2	0.9918	0.9937	0.9959
model_3	0.9873	0.9920	0.9927
model_4	0.9934	0.9958	0.9958
model_5	0.9860	0.9914	0.9908
model_6	0.9801	0.9859	0.9840

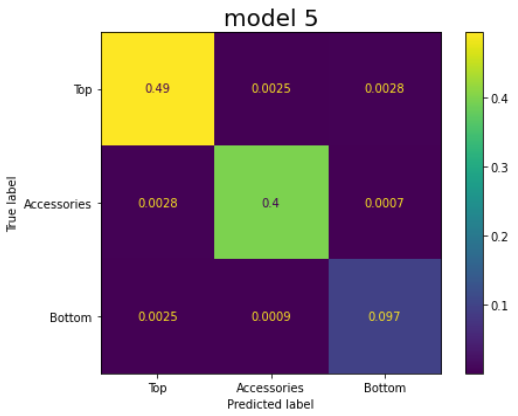
Confusion matrix: the confusion matrix allows us to visualize the data predictions and extract different evaluations on it such as accuracy, precision, recall etc.
the method is simple, it count how many times something happened, for example how many times an item from the Top class was predicted to be Top, Bottom or Accessories.

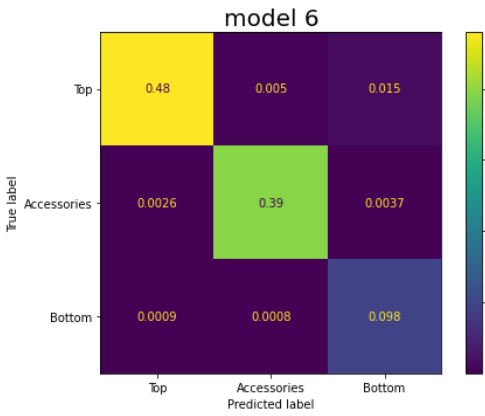












*The confusion matrix here shows the results of the predictions of the test set after normalizing them by the number of all the test samples.

And for fun here are a few more of the evaluations gained from the confusion matrix:

Model 1 :

	precision	recall	f1-score	support
Top	0.99	0.98	0.99	5000
Accessories	0.99	0.99	0.99	4000
Bottom	0.93	0.98	0.96	1000
accuracy			0.99	10000
macro avg	0.97	0.99	0.98	10000
weighted avg	0.99	0.99	0.99	10000

Model 2 :

	precision	recall	f1-score	support
Top	1.00	0.99	0.99	5000
Accessories	0.99	1.00	0.99	4000
Bottom	0.97	0.98	0.98	1000
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Model 3 :

	precision	recall	f1-score	support
Top	0.99	0.99	0.99	5000
Accessories	0.99	0.99	0.99	4000
Bottom	0.97	0.96	0.96	1000
accuracy			0.99	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.99	0.99	0.99	10000

Model 4 :

	precision	recall	f1-score	support
Top	0.99	1.00	0.99	5000
Accessories	1.00	0.99	0.99	4000
Bottom	0.98	0.98	0.98	1000
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

```
--------------------------------
Model 5 :
              precision    recall  f1-score   support

         Top       0.99      0.99      0.99      5000
 Accessories       0.99      0.98      0.99      4000
      Bottom       0.93      0.98      0.95      1000

    accuracy                           0.98     10000
   macro avg       0.97      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

--------------------------------
Model 6 :
              precision    recall  f1-score   support

         Top       0.99      0.97      0.98      5000
 Accessories       0.98      0.98      0.98      4000
      Bottom       0.87      0.99      0.93      1000

    accuracy                           0.97     10000
   macro avg       0.95      0.98      0.96     10000
weighted avg       0.98      0.97      0.98     10000
```

Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

Recall is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

F1-score is defined as follows:

$$F1 = \frac{2TP}{2TP + FP + FN}$$

*the support is the size of that class