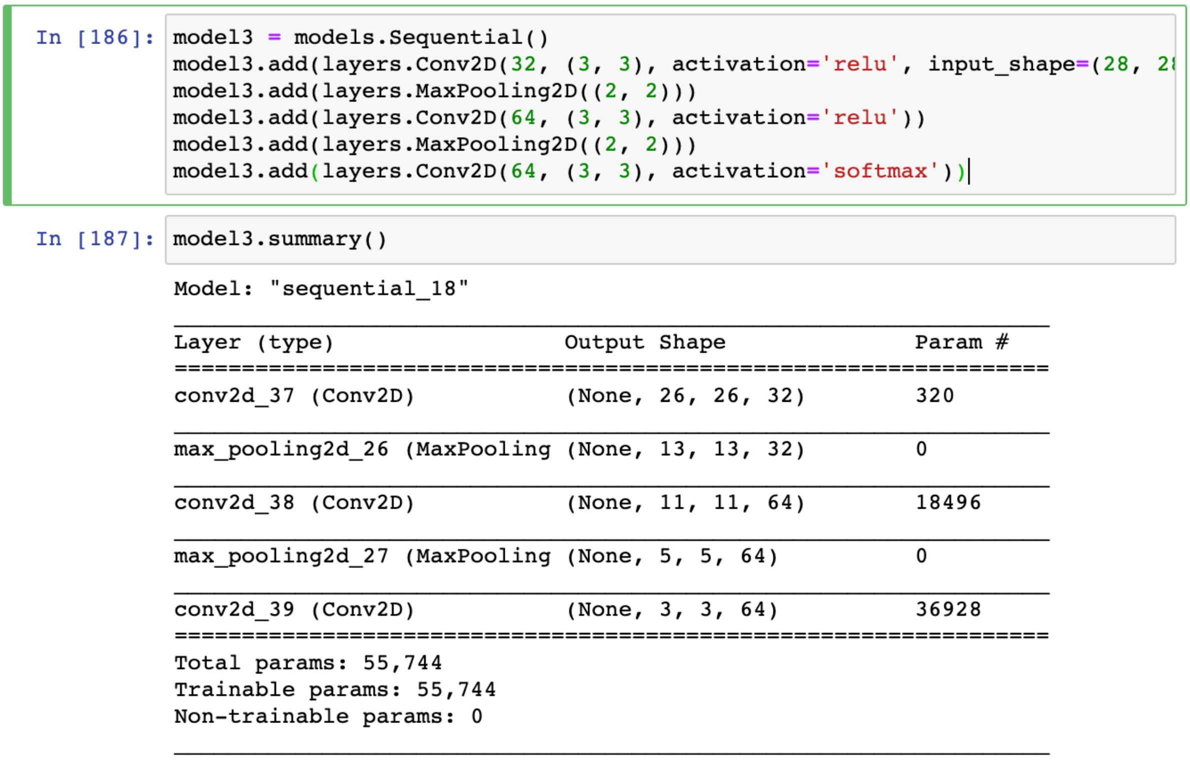
# REPORT NO. 4

Jakub Otręba 394131

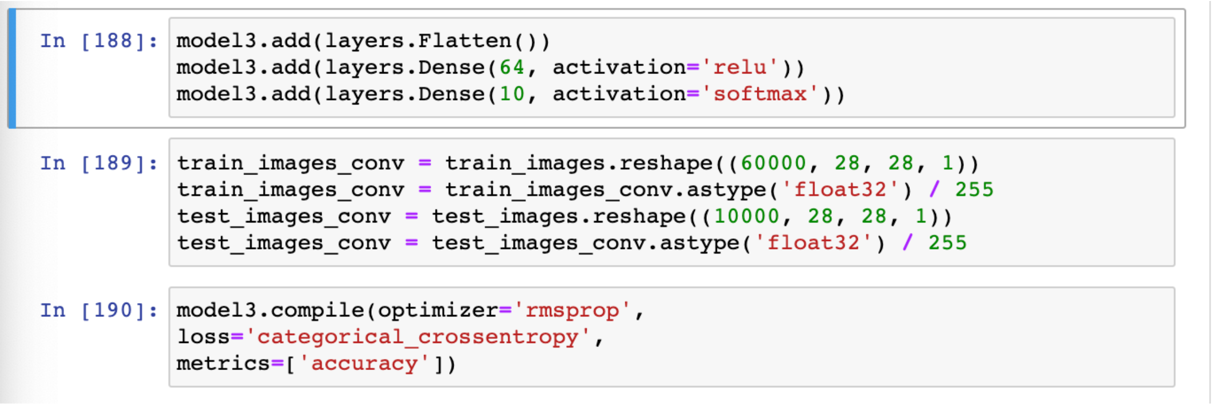


```
In [186]: model3 = models.Sequential()
          model3.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 2
          model3.add(layers.MaxPooling2D((2, 2)))
          model3.add(layers.Conv2D(64, (3, 3), activation='relu'))
          model3.add(layers.MaxPooling2D((2, 2)))
          model3.add(layers.Conv2D(64, (3, 3), activation='softmax'))
```

```
In [187]: model3.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_37 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 3, 3, 64)          36928     
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

I built a new model and add 3 convolutional layers 3D (2 times with relu activation function and one time with softmax activation function) interlaced with 2 MaxPooling2D layers

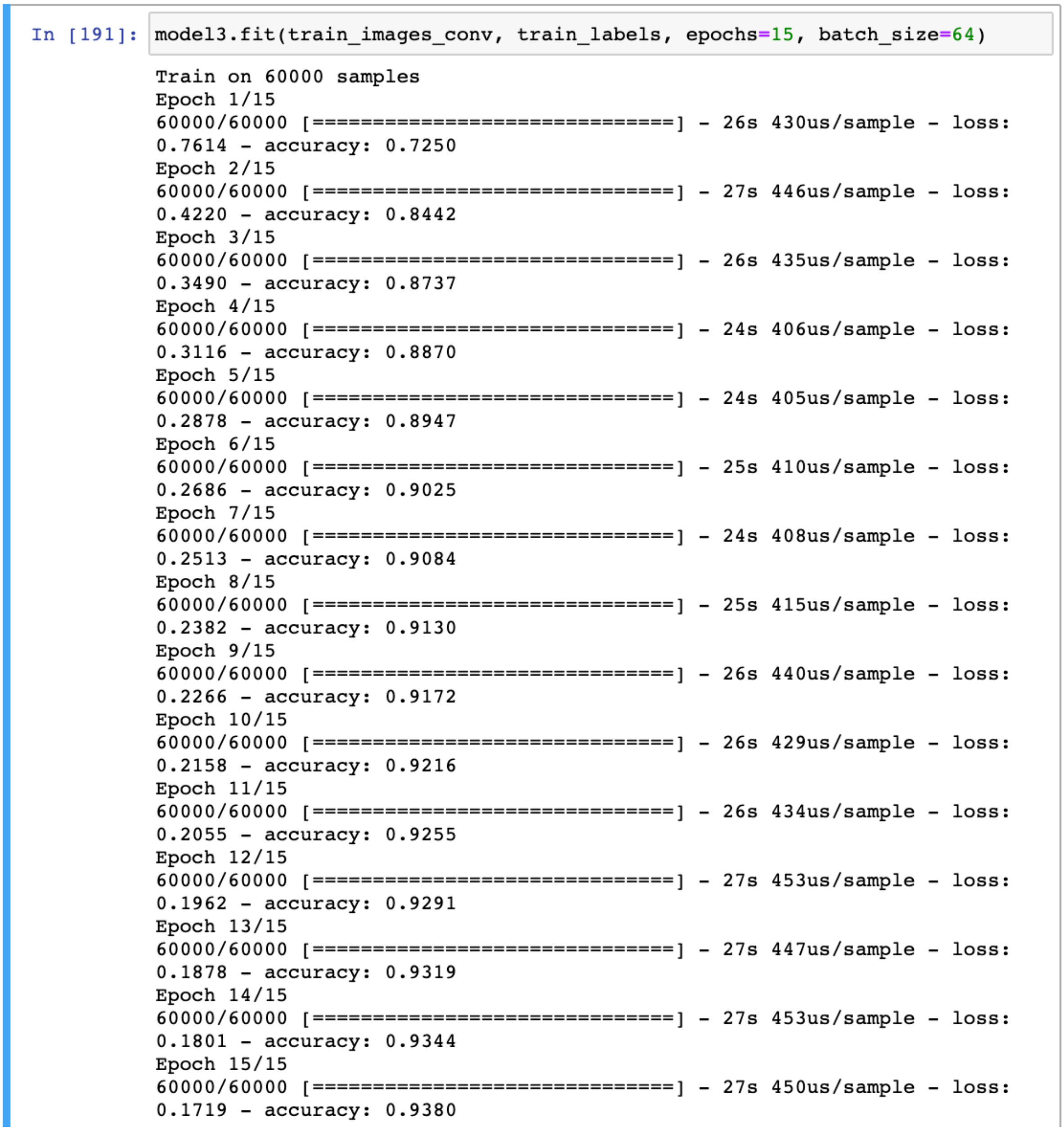


```
In [188]: model3.add(layers.Flatten())
          model3.add(layers.Dense(64, activation='relu'))
          model3.add(layers.Dense(10, activation='softmax'))
```

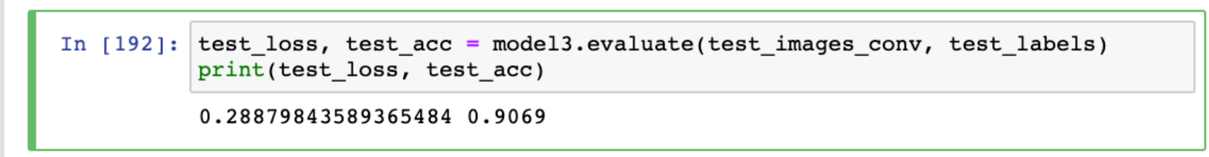
```
In [189]: train_images_conv = train_images.reshape((60000, 28, 28, 1))
          train_images_conv = train_images_conv.astype('float32') / 255
          test_images_conv = test_images.reshape((10000, 28, 28, 1))
          test_images_conv = test_images_conv.astype('float32') / 255
```

```
In [190]: model3.compile(optimizer='rmsprop',
          loss='categorical_crossentropy',
          metrics=['accuracy'])
```

Then I flattened the layers and add one Dense layer with size 64 and relu activation function and one Dense layer with size 10 and softmax activation function. I left all other parameters the same and fitted the model with 15 epochs and with batch size of 64.

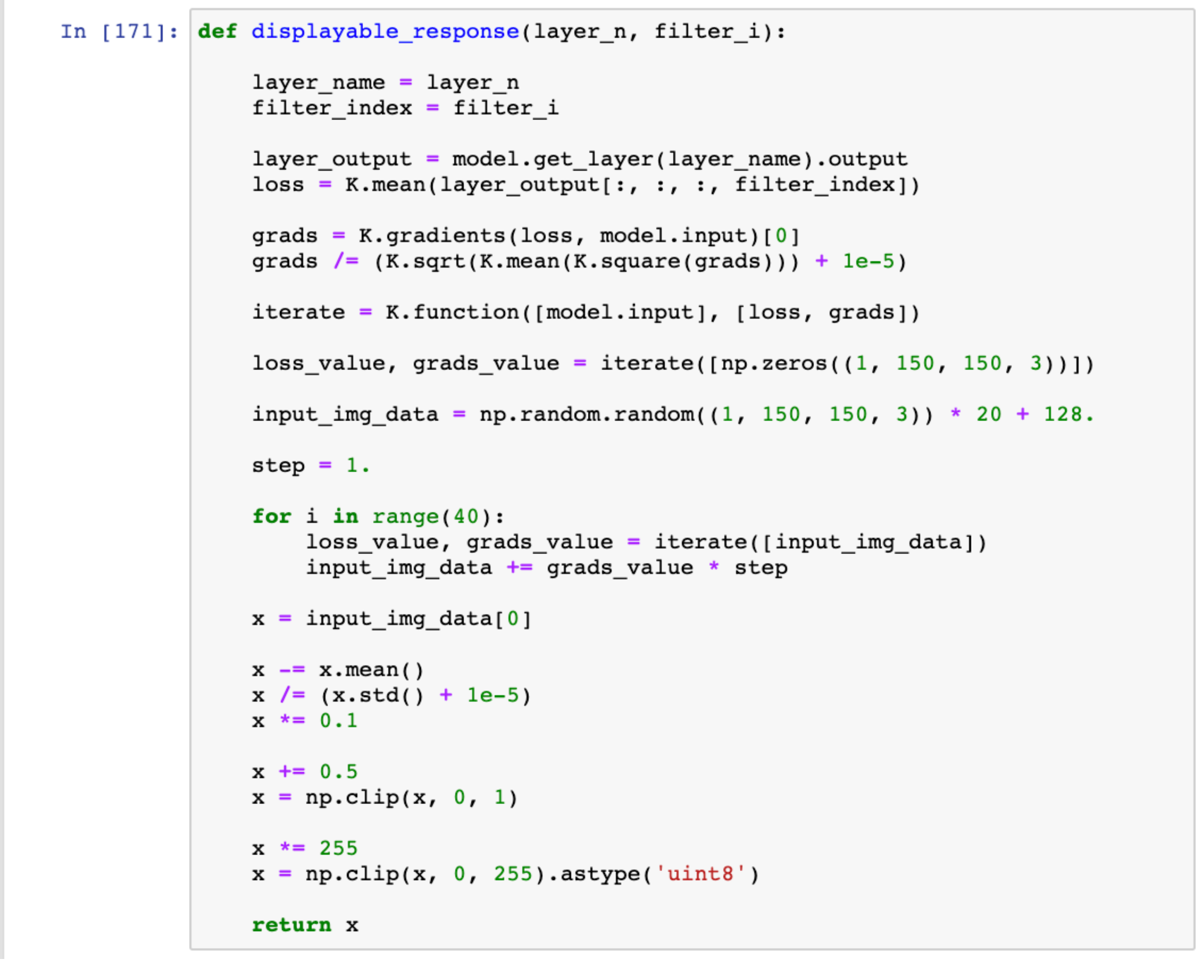


```
In [191]: model3.fit(train_images_conv, train_labels, epochs=15, batch_size=64)

          Train on 60000 samples
          Epoch 1/15
          60000/60000 [==============================] - 26s 430us/sample - loss:
          0.7614 - accuracy: 0.7250
          Epoch 2/15
          60000/60000 [==============================] - 27s 446us/sample - loss:
          0.4220 - accuracy: 0.8442
          Epoch 3/15
          60000/60000 [==============================] - 26s 435us/sample - loss:
          0.3490 - accuracy: 0.8737
          Epoch 4/15
          60000/60000 [==============================] - 24s 406us/sample - loss:
          0.3116 - accuracy: 0.8870
          Epoch 5/15
          60000/60000 [==============================] - 24s 405us/sample - loss:
          0.2878 - accuracy: 0.8947
          Epoch 6/15
          60000/60000 [==============================] - 25s 410us/sample - loss:
          0.2686 - accuracy: 0.9025
          Epoch 7/15
          60000/60000 [==============================] - 24s 408us/sample - loss:
          0.2513 - accuracy: 0.9084
          Epoch 8/15
          60000/60000 [==============================] - 25s 415us/sample - loss:
          0.2382 - accuracy: 0.9130
          Epoch 9/15
          60000/60000 [==============================] - 26s 440us/sample - loss:
          0.2266 - accuracy: 0.9172
          Epoch 10/15
          60000/60000 [==============================] - 26s 429us/sample - loss:
          0.2158 - accuracy: 0.9216
          Epoch 11/15
          60000/60000 [==============================] - 26s 434us/sample - loss:
          0.2055 - accuracy: 0.9255
          Epoch 12/15
          60000/60000 [==============================] - 27s 453us/sample - loss:
          0.1962 - accuracy: 0.9291
          Epoch 13/15
          60000/60000 [==============================] - 27s 447us/sample - loss:
          0.1878 - accuracy: 0.9319
          Epoch 14/15
          60000/60000 [==============================] - 27s 453us/sample - loss:
          0.1801 - accuracy: 0.9344
          Epoch 15/15
          60000/60000 [==============================] - 27s 450us/sample - loss:
          0.1719 - accuracy: 0.9380
```



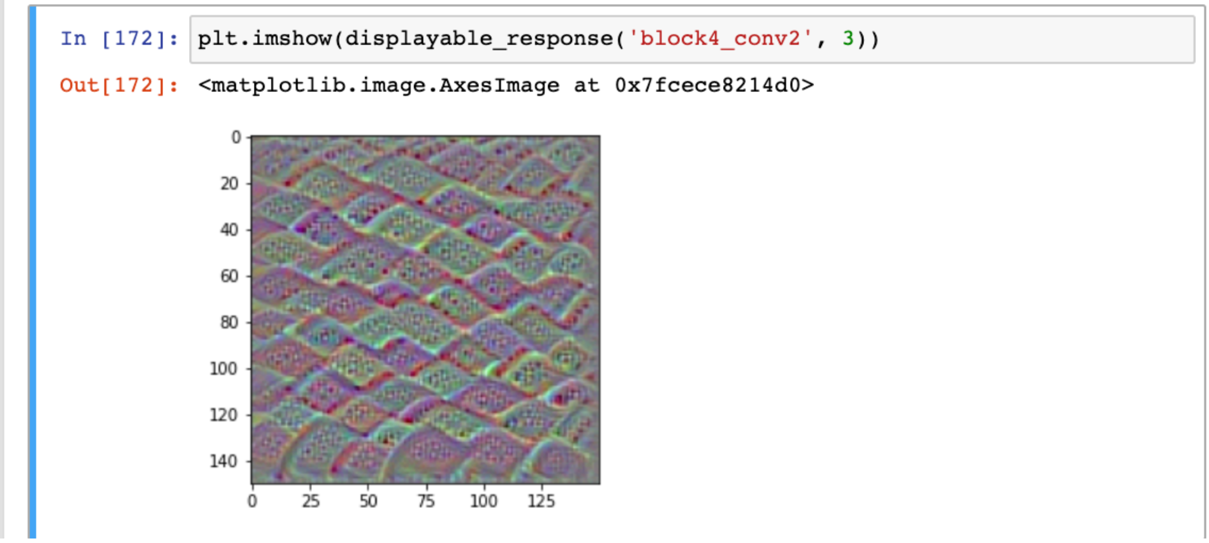
```
In [192]: test_loss, test_acc = model3.evaluate(test_images_conv, test_labels)
          print(test_loss, test_acc)

          0.28879843589365484 0.9069
```

I managed to obtain a 0.9069 test accuracy, which is better than the original 0.905 test accuracy obtained on the same dataset (MNIST fashion).



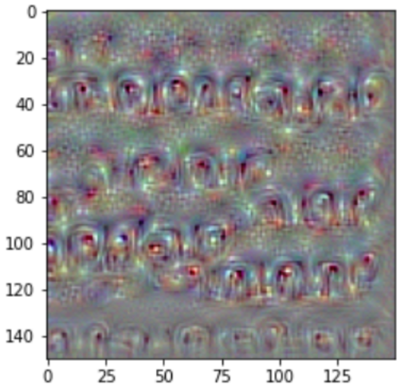
```
In [171]: def displayable_response(layer_n, filter_i):

              layer_name = layer_n
              filter_index = filter_i

              layer_output = model.get_layer(layer_name).output
              loss = K.mean(layer_output[:, :, :, filter_index])

              grads = K.gradients(loss, model.input)[0]
              grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

              iterate = K.function([model.input], [loss, grads])

              loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])

              input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

              step = 1.

              for i in range(40):
                  loss_value, grads_value = iterate([input_img_data])
                  input_img_data += grads_value * step

              x = input_img_data[0]

              x -= x.mean()
              x /= (x.std() + 1e-5)
              x *= 0.1

              x += 0.5
              x = np.clip(x, 0, 1)

              x *= 255
              x = np.clip(x, 0, 255).astype('uint8')

              return x
```

Above, there is my function which takes layer number and filter index as inputs and outputs the displayable filter response. Below, I attached four examples of the function outputs.

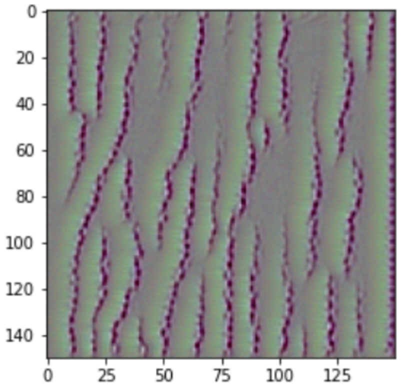


```
In [172]: plt.imshow(displayable_response('block4_conv2', 3))

Out[172]: <matplotlib.image.AxesImage at 0x7fcece8214d0>
```

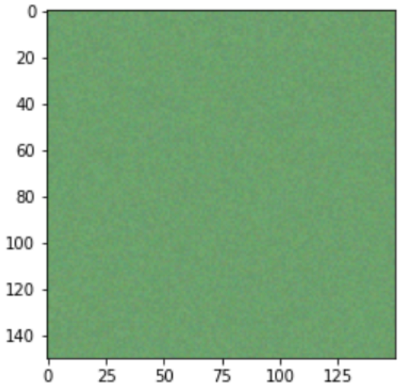
```
In [173]: plt.imshow(displayable_response('block4_pool', 20))
Out[173]: <matplotlib.image.AxesImage at 0x7fcece42ae10>
```



```
In [174]: plt.imshow(displayable_response('block2_conv2', 1))
Out[174]: <matplotlib.image.AxesImage at 0x7fcece71c510>
```



```
In [175]: plt.imshow(displayable_response('input_1', 1))
Out[175]: <matplotlib.image.AxesImage at 0x7fceceb7d6d0>
```


0
20
40
60
80
100
120
140
0
25
50
75
100
125