

Lab 4 Deep Learning 1

Elżbieta Karpińska

Homework 1

I built a new model and added 3 convolutional layers, two of them with relu activation function and the other one with softmax activation function. I flattened the layers and add one dense layer (size 64) and relu activation function and one dense layer (size 10) and softmax activation function. Rest of the parameters were unchanged and model was fitted with 15 epochs and with batch size of 128.

```
model_1 = models.Sequential()
model_1.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), activation='softmax'))
model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model_1.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
=====		
conv2d_9 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d_6 (MaxPooling 2D)	(None, 13, 13, 64)	0
conv2d_10 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_7 (MaxPooling 2D)	(None, 5, 5, 64)	0
conv2d_11 (Conv2D)	(None, 3, 3, 64)	36928
=====		
Total params: 74,496		
Trainable params: 74,496		
Non-trainable params: 0		

```
model_1.add(layers.Flatten())
model_1.add(layers.Dense(64, activation='relu'))
model_1.add(layers.Dense(10, activation='softmax'))
```

```
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
```

```
model_1.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

Below I present training accuracy.



```
model_1.fit(train_images_conv, train_labels, epochs=15, batch_size=128)
```

```
Train on 60000 samples
```

```
Epoch 1/15
```

```
60000/60000 [=====] - 7s 121us/sample - loss: 0.1973 - accuracy: 0.9263
```

```
Epoch 2/15
```

```
60000/60000 [=====] - 7s 116us/sample - loss: 0.1882 - accuracy: 0.9306
```

```
Epoch 3/15
```

```
60000/60000 [=====] - 7s 114us/sample - loss: 0.1813 - accuracy: 0.9322
```

```
Epoch 4/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1741 - accuracy: 0.9352
```

```
Epoch 5/15
```

```
60000/60000 [=====] - 7s 117us/sample - loss: 0.1674 - accuracy: 0.9380
```

```
Epoch 6/15
```

```
60000/60000 [=====] - 7s 119us/sample - loss: 0.1616 - accuracy: 0.9401
```

```
Epoch 7/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1556 - accuracy: 0.9426
```

```
Epoch 8/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1492 - accuracy: 0.9450
```

```
Epoch 9/15
```

```
60000/60000 [=====] - 7s 116us/sample - loss: 0.1446 - accuracy: 0.9452
```

```
Epoch 10/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1386 - accuracy: 0.9491
```

```
Epoch 11/15
```

```
60000/60000 [=====] - 7s 116us/sample - loss: 0.1338 - accuracy: 0.9502
```

```
Epoch 12/15
```

```
60000/60000 [=====] - 7s 116us/sample - loss: 0.1293 - accuracy: 0.9515
```

```
Epoch 13/15
```

```
60000/60000 [=====] - 7s 116us/sample - loss: 0.1246 - accuracy: 0.9539
```

```
Epoch 14/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1183 - accuracy: 0.9552
```

```
Epoch 15/15
```

```
60000/60000 [=====] - 7s 115us/sample - loss: 0.1142 - accuracy: 0.9578
```

```
<keras.callbacks.History at 0x7fef7b1c1e90>
```

Below one can see test accuracy of 0.907, which is better than original result 0.905.

```
[44] test_loss, test_acc = model_1.evaluate(test_images_conv, test_labels)
     print(test_loss, test_acc)
```

```
0.33180018522739413 0.9071
```

Homework 2

Below I present a function that takes as arguments the name of the layer and filter index and outputs the displayable filter response.

```

#homework2 function
def function(arg1, arg2):
    layer_name = arg1
    filter_index = arg2
    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))])
    #print(grads)
    #print(grads_value)

    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

    #print(grads_value)
    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)

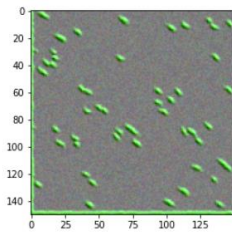
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x

```

Filter response examples

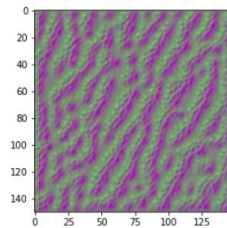
plt.imshow(function("block1_conv1",11))

<matplotlib.image.AxesImage at 0x7fc6feb1bf90>



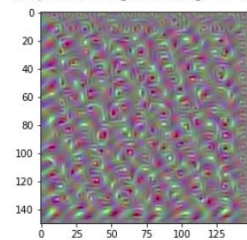
plt.imshow(function('block2_conv2', 20))

<matplotlib.image.AxesImage at 0x7fc6ffe52550>



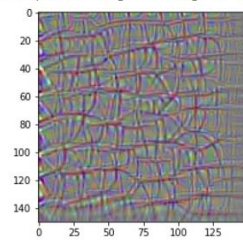
[] plt.imshow(function('block3_conv2', 20))

<matplotlib.image.AxesImage at 0x7fc6ffd700d0>



plt.imshow(function('block4_conv2', 37))

<matplotlib.image.AxesImage at 0x7fc6fec1f250>



plt.imshow(function('block2_conv2', 2))

<matplotlib.image.AxesImage at 0x7fc6ff88a890>

