Import biblioteki **TensorFlow** (https://www.tensorflow.org/) z której będziemy korzystali w **uczeniu maszynowym**:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np


import keras
from keras.models import Sequential
from keras.layers import Dense
```

Clothes recognition - dataset **Fasion MNIST**

Download dataset

```
(train_data, train_labels), (test_data, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
```

```
data = np.concatenate([train_data, test_data])
```

```
data.shape
```

```
(70000, 28, 28)
```

```
label = np.concatenate([train_labels,test_labels])
```

```
label.shape
```

```
(70000,)
```

Informations about dataset

```
train_data.shape,train_labels.shape
```

```
((60000, 28, 28), (60000,))
```

```
test_data.shape,test_labels.shape
```

```
((10000, 28, 28), (10000,))
```

```
train_data[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
         69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
        172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
        200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
        229,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
        173,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
        202,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
        219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
        209,  52],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
        244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
        167,  56],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
        236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
```

```
         92,   0],
       [  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
        226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
         77,   0],
       [  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
        207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
        159,   0],
       [  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
        226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
        215,   0],
       [  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
        159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
        246,   0],
       [  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
         80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
        225,   0],
```

```
train_labels[0]
```

```
9
```

One-hot encoding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((60000, 28, 28), (60000, 10))
```

```
test_data.shape,test_labels.shape
```

```
((10000, 28, 28), (10000, 10))
```

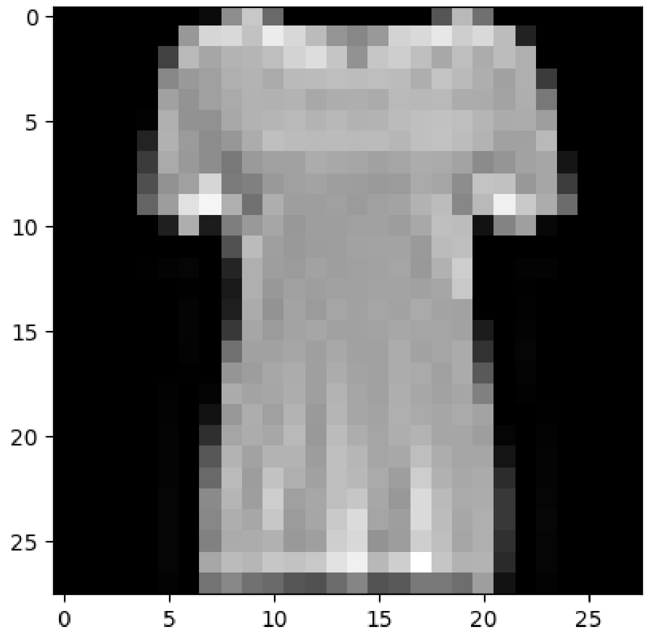
```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

Visulization

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

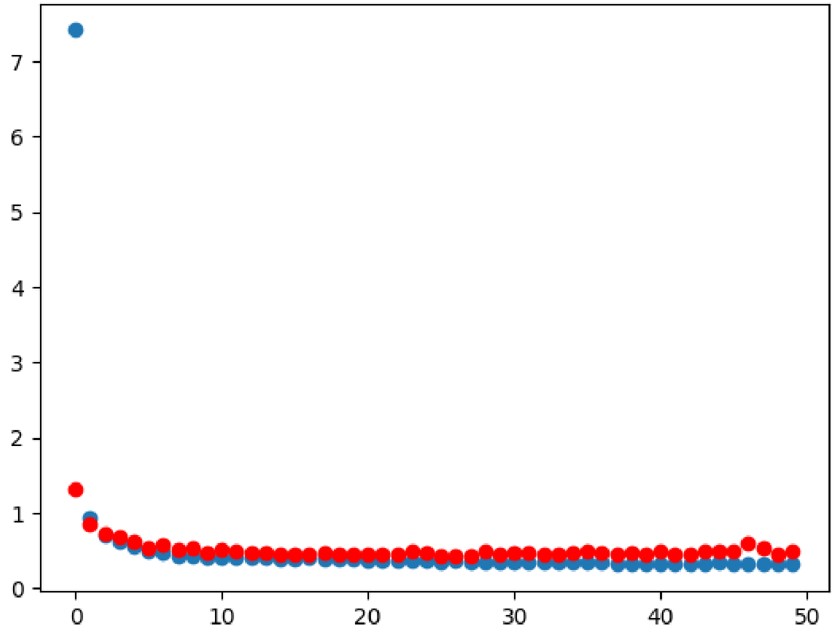
```
model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_14 (Dense)            (None, 128)               100480    
                                                                 
 dense_15 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

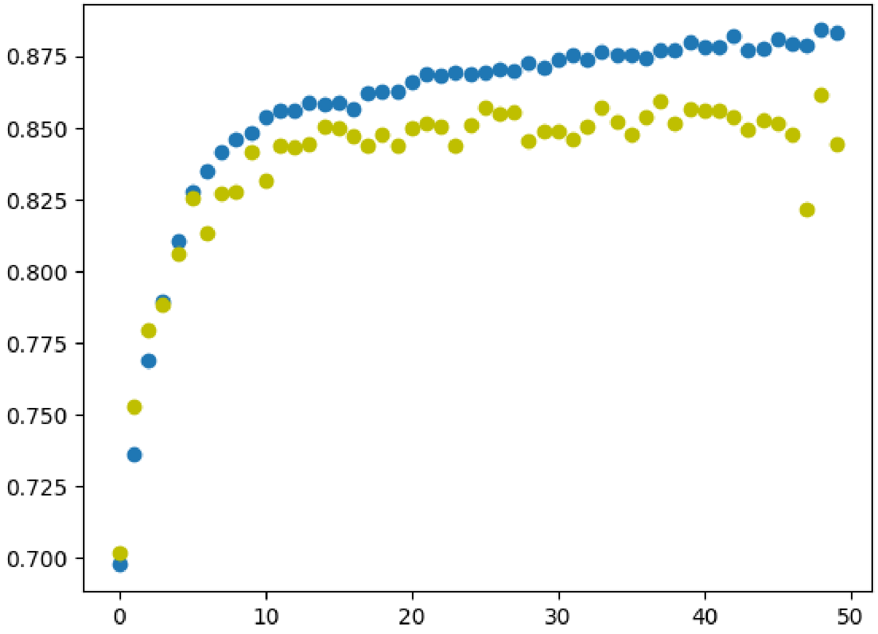
```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs,validation_split=0.2)
```

```
375/375 [==============================] - 2s 6ms/step - loss: 0.3382 - accuracy: 0.8741 - val_loss: 0.4658 - val_accuracy: 0.8535
Epoch 38/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3293 - accuracy: 0.8773 - val_loss: 0.4503 - val_accuracy: 0.8595
Epoch 39/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3297 - accuracy: 0.8773 - val_loss: 0.4704 - val_accuracy: 0.8515
Epoch 40/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3209 - accuracy: 0.8796 - val_loss: 0.4490 - val_accuracy: 0.8563
Epoch 41/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3249 - accuracy: 0.8779 - val_loss: 0.4820 - val_accuracy: 0.8558
Epoch 42/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3276 - accuracy: 0.8782 - val_loss: 0.4509 - val_accuracy: 0.8562
Epoch 43/50
375/375 [==============================] - 3s 7ms/step - loss: 0.3174 - accuracy: 0.8819 - val_loss: 0.4528 - val_accuracy: 0.8539
Epoch 44/50
375/375 [==============================] - 2s 7ms/step - loss: 0.3261 - accuracy: 0.8772 - val_loss: 0.4905 - val_accuracy: 0.8492
Epoch 45/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3328 - accuracy: 0.8777 - val_loss: 0.4974 - val_accuracy: 0.8528
Epoch 46/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3191 - accuracy: 0.8806 - val_loss: 0.4865 - val_accuracy: 0.8515
Epoch 47/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3237 - accuracy: 0.8792 - val_loss: 0.5883 - val_accuracy: 0.8478
Epoch 48/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3242 - accuracy: 0.8788 - val_loss: 0.5254 - val_accuracy: 0.8215
Epoch 49/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3132 - accuracy: 0.8840 - val_loss: 0.4448 - val_accuracy: 0.8618
Epoch 50/50
375/375 [==============================] - 3s 9ms/step - loss: 0.3105 - accuracy: 0.8829 - val_loss: 0.4956 - val_accuracy: 0.8443
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

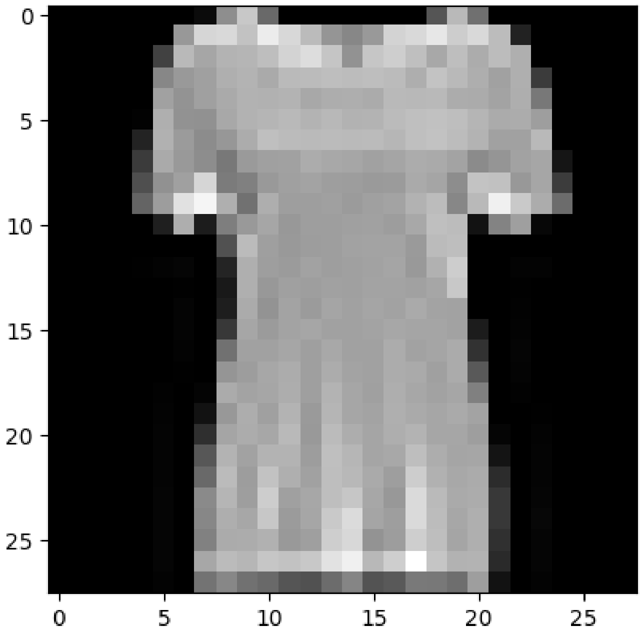



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.5169681906700134
Test accuracy: 0.8378999829292297
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = train_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 51ms/step
array([[5.3179103e-01, 4.9443461e-08, 2.8912007e-07, 3.7933025e-04,
        2.4379283e-08, 5.3894745e-22, 4.6782932e-01, 0.0000000e+00,
        1.1035983e-09, 2.1506313e-38]], dtype=float32)
```



Import biblioteki **TensorFlow** (https://www.tensorflow.org/) z której będziemy korzystali w **uczeniu maszynowym**:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

import keras
from keras.models import Sequential
from keras.layers import Dense
```

Numbers recognition - dataset **MNIST**

Download dataset

```
(train_data, train_labels), (test_data, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
```

```
(train_data, train_labels), (test_data, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
```

```
data = np.concatenate([train_data, test_data])
```

```
data.shape
```

```
(70000, 28, 28)
```

```
label = np.concatenate([train_labels,test_labels])
```

```
label.shape
```

```
(70000,)
```

Informations about dataset

```
train_data.shape,train_labels.shape
```

```
((60000, 28, 28), (60000,))
```

```
test_data.shape,test_labels.shape
```

```
((10000, 28, 28), (10000,))
```

```
train_data[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
```

```
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
           0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
           0,   3],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
           0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
          10,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
          72,  15],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
          69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
         172,  66],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
         200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
         229,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
         173,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
         202,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
         219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
         209,  52],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
         244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
         167,  56],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
         236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
          92,   0],
        [  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
         226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
          77,   0],
        [  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
         207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
         159,   0],
        [  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
         226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
         215,   0],
        [  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
         159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
         246,   0],
        [  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
          80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
         225,   0],
        [ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
```

```
train_labels[0]
```

```
9
```

One-hot encoding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((60000, 28, 28), (60000, 10))
```

```
test_data.shape,test_labels.shape
```

```
((10000, 28, 28), (10000, 10))
```

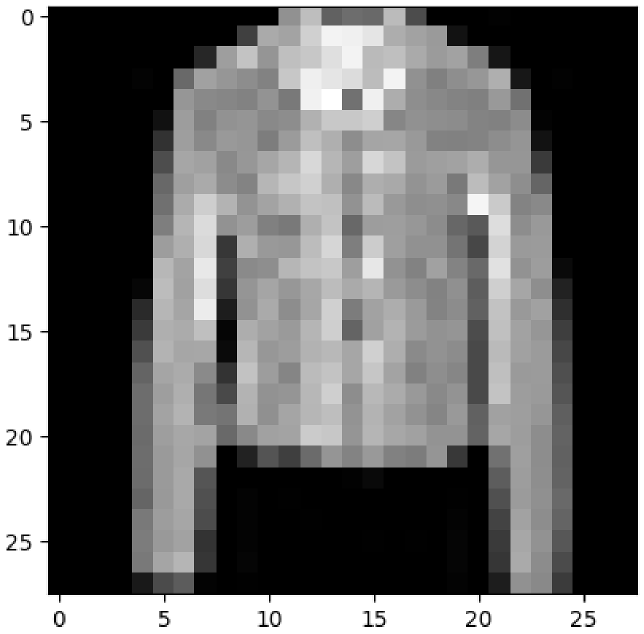
```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

Visulization

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

```
model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

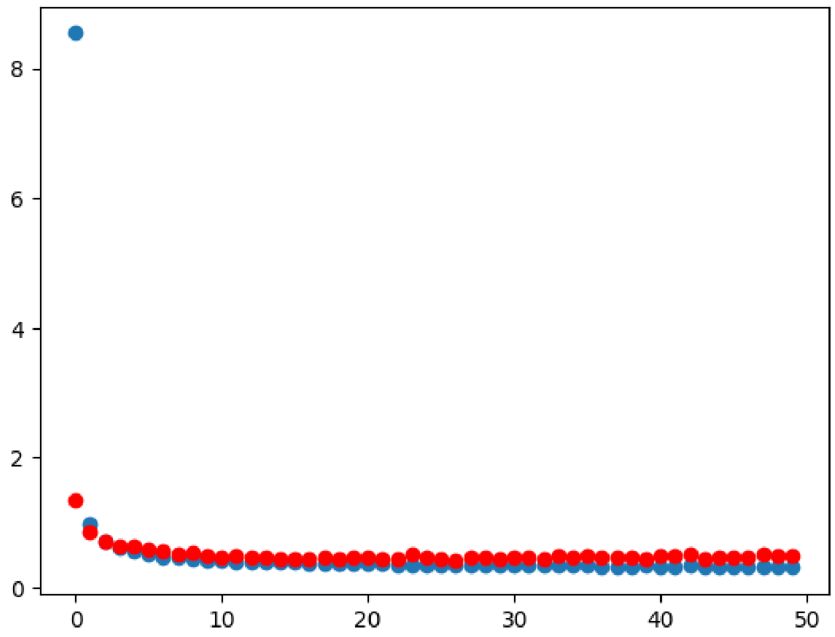
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 128)               100480

 dense_17 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
```

```
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

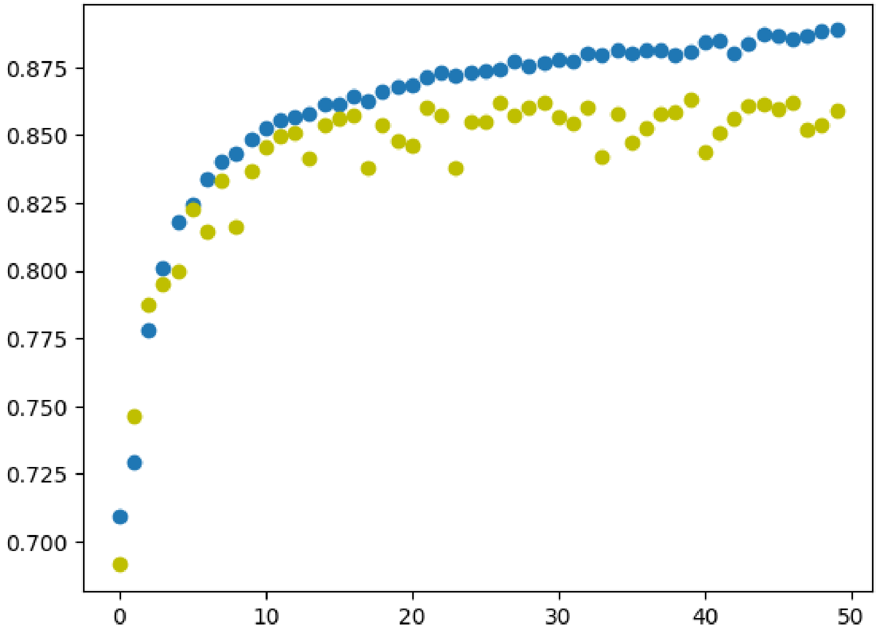
```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
375/375 [==============================] - 2s 5ms/step - loss: 0.3175 - accuracy: 0.8849 - val_loss: 0.4980 - val_accuracy: 0.8506
Epoch 43/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3367 - accuracy: 0.8803 - val_loss: 0.5045 - val_accuracy: 0.8563
Epoch 44/50
375/375 [==============================] - 3s 7ms/step - loss: 0.3258 - accuracy: 0.8835 - val_loss: 0.4459 - val_accuracy: 0.8611
Epoch 45/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3129 - accuracy: 0.8872 - val_loss: 0.4751 - val_accuracy: 0.8616
Epoch 46/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3124 - accuracy: 0.8864 - val_loss: 0.4663 - val_accuracy: 0.8597
Epoch 47/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3129 - accuracy: 0.8855 - val_loss: 0.4625 - val_accuracy: 0.8619
Epoch 48/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3128 - accuracy: 0.8867 - val_loss: 0.5020 - val_accuracy: 0.8518
Epoch 49/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3064 - accuracy: 0.8886 - val_loss: 0.4918 - val_accuracy: 0.8537
Epoch 50/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3074 - accuracy: 0.8887 - val_loss: 0.4881 - val_accuracy: 0.8588
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

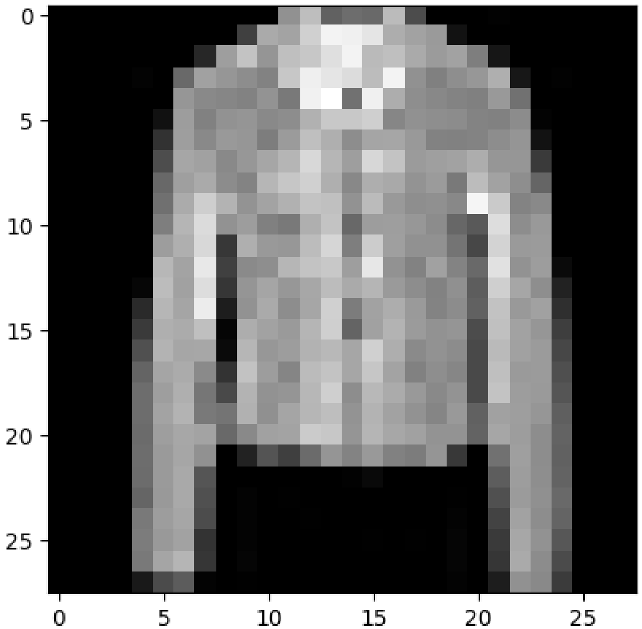



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.509166955947876
Test accuracy: 0.8529000282287598
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 51ms/step
array([[1.7098278e-03, 8.9089546e-05, 1.3766566e-01, 1.4066697e-03,
        7.6899230e-01, 4.5096906e-28, 9.0136059e-02, 0.0000000e+00,
        2.6587446e-07, 8.2854803e-36]], dtype=float32)
```



Regularyzacja - metoda 1

Zwiększamy zbiór treningowy z **60000** do **65000** (20% to zbiór walidacyjny)

```
test_data = data[:65000]
train_data = data[65000:]
test_labels = label[:65000]
train_labels = label[65000:]
```

```
print(test_data.shape,train_data.shape,test_labels.shape,train_labels.shape)
```

```
(65000, 28, 28) (5000, 28, 28) (65000,) (5000,)
```

One-hot coding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((5000, 28, 28), (5000, 10))
```

```
test_data.shape,test_labels.shape
```

```
((65000, 28, 28), (65000, 10))
```

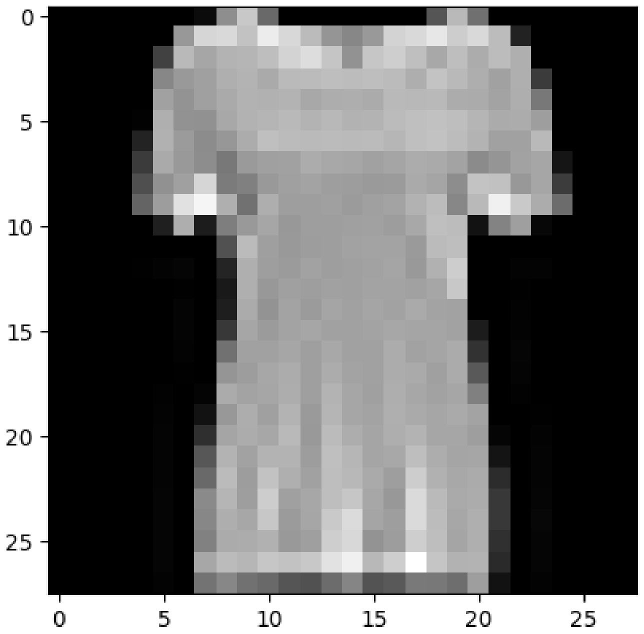
```
train_labels[0]
```

```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Visulization

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```




```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))


model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

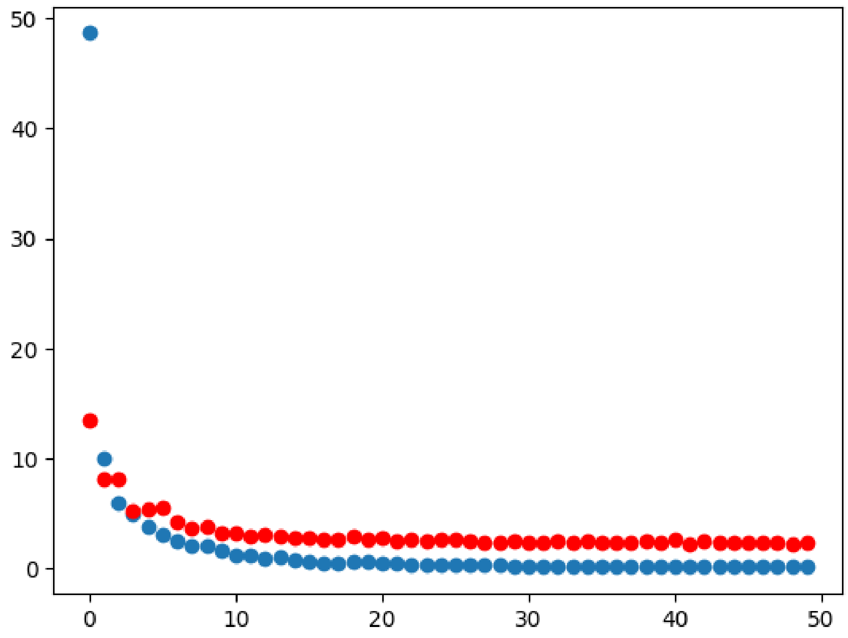
```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 128)               100480

 dense_19 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
```

```
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

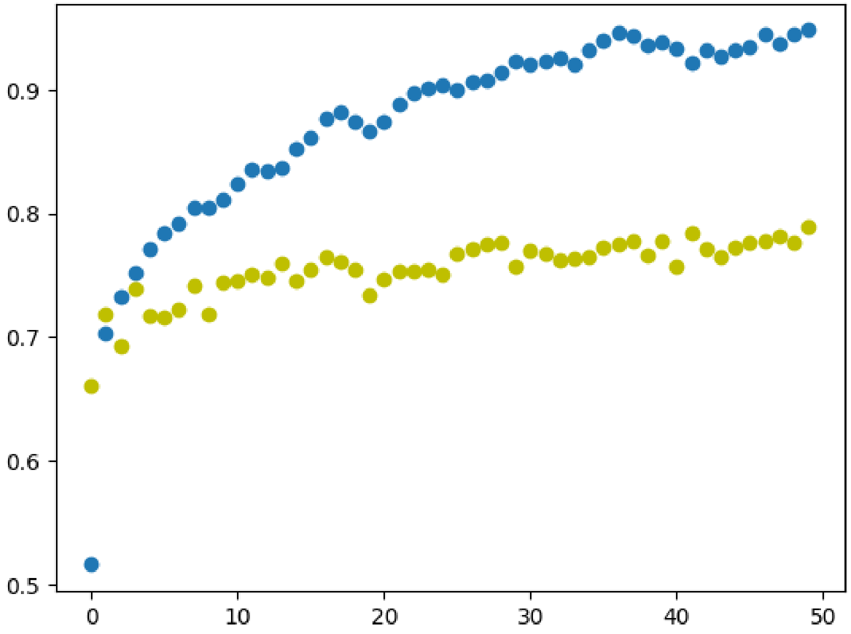
```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
32/32 [==============================] - 0s 5ms/step - loss: 0.2444 - accuracy: 0.9220 - val_loss: 2.2873 - val_accuracy: 0.7840
Epoch 43/50
32/32 [==============================] - 0s 6ms/step - loss: 0.1982 - accuracy: 0.9323 - val_loss: 2.5011 - val_accuracy: 0.7710
Epoch 44/50
32/32 [==============================] - 0s 6ms/step - loss: 0.2081 - accuracy: 0.9268 - val_loss: 2.3810 - val_accuracy: 0.7650
Epoch 45/50
32/32 [==============================] - 0s 6ms/step - loss: 0.1833 - accuracy: 0.9317 - val_loss: 2.3984 - val_accuracy: 0.7720
Epoch 46/50
32/32 [==============================] - 0s 5ms/step - loss: 0.1955 - accuracy: 0.9350 - val_loss: 2.4104 - val_accuracy: 0.7760
Epoch 47/50
32/32 [==============================] - 0s 5ms/step - loss: 0.1492 - accuracy: 0.9445 - val_loss: 2.3421 - val_accuracy: 0.7780
Epoch 48/50
32/32 [==============================] - 0s 6ms/step - loss: 0.1818 - accuracy: 0.9367 - val_loss: 2.3875 - val_accuracy: 0.7820
Epoch 49/50
32/32 [==============================] - 0s 5ms/step - loss: 0.1519 - accuracy: 0.9442 - val_loss: 2.2719 - val_accuracy: 0.7760
Epoch 50/50
32/32 [==============================] - 0s 5ms/step - loss: 0.1452 - accuracy: 0.9480 - val_loss: 2.3460 - val_accuracy: 0.7890
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

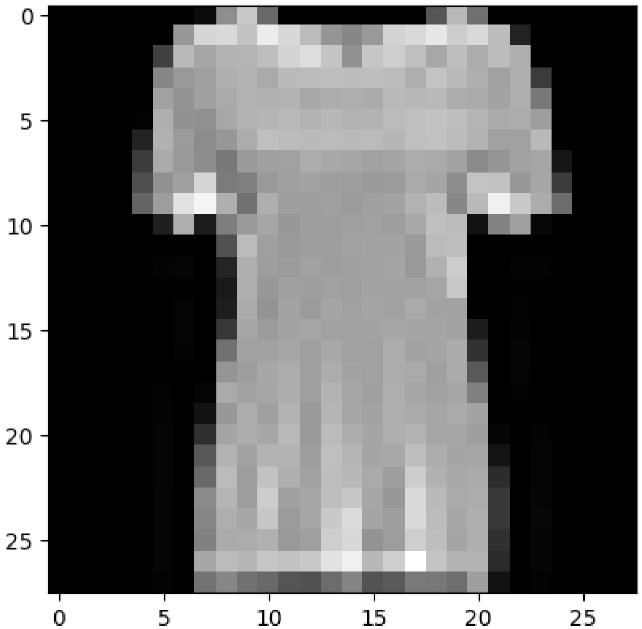



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 2.1894993782043457
Test accuracy: 0.7882000207901001
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 48ms/step
array([[9.9700314e-01, 1.5093416e-11, 1.2314596e-06, 6.6878658e-10,
        0.0000000e+00, 3.9279828e-09, 2.9955881e-03, 2.0454720e-26,
        2.7322981e-15, 4.6357407e-21]], dtype=float32)
```



Zwiększamy zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

```
test_data = data[:68000]
train_data = data[68000:]
test_labels = label[:68000]
train_labels = label[68000:]
```

```
print(test_data.shape,train_data.shape,test_labels.shape,train_labels.shape)
```

```
(68000, 28, 28) (2000, 28, 28) (68000,) (2000,)
```

One-hot coding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((2000, 28, 28), (2000, 10))
```

```
test_data.shape,test_labels.shape
```

```
((68000, 28, 28), (68000, 10))
```

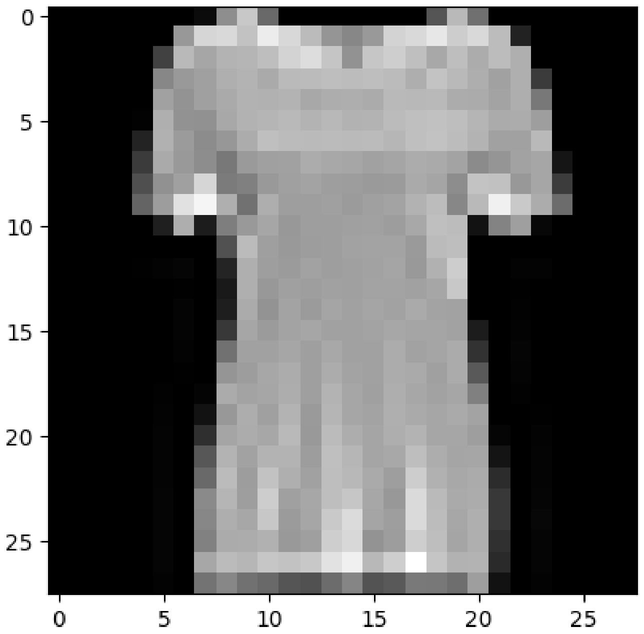
```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

Visulization

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```




```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))


model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 128)               100480

 dense_21 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
```

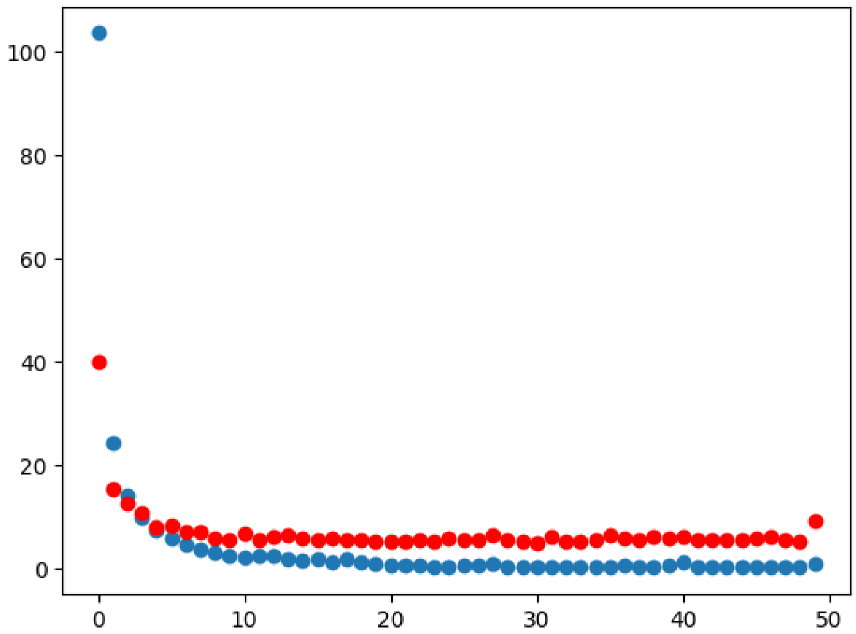
```
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

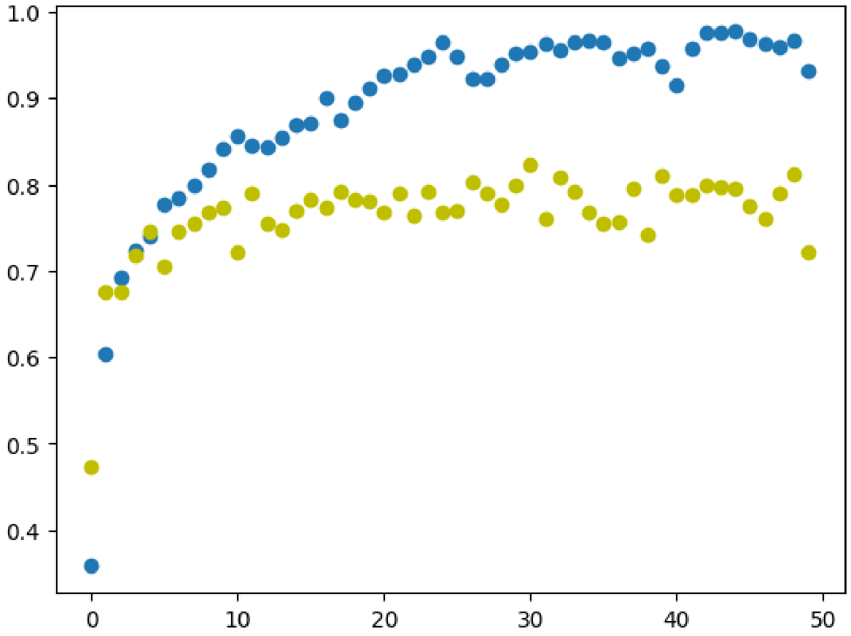
```
Epoch 1/50
13/13 [==============================] - 1s 31ms/step - loss: 103.5510 - accuracy: 0.3581 - val_loss: 40.0667 - val_accuracy: 0.4725
Epoch 2/50
13/13 [==============================] - 0s 11ms/step - loss: 24.1392 - accuracy: 0.6037 - val_loss: 15.3471 - val_accuracy: 0.6750
Epoch 3/50
13/13 [==============================] - 0s 12ms/step - loss: 14.0421 - accuracy: 0.6925 - val_loss: 12.6863 - val_accuracy: 0.6750
Epoch 4/50
13/13 [==============================] - 0s 11ms/step - loss: 9.6410 - accuracy: 0.7244 - val_loss: 10.5986 - val_accuracy: 0.7175
Epoch 5/50
13/13 [==============================] - 0s 12ms/step - loss: 7.3955 - accuracy: 0.7406 - val_loss: 8.0434 - val_accuracy: 0.7450
Epoch 6/50
13/13 [==============================] - 0s 11ms/step - loss: 5.8013 - accuracy: 0.7769 - val_loss: 8.1976 - val_accuracy: 0.7050
Epoch 7/50
13/13 [==============================] - 0s 12ms/step - loss: 4.5044 - accuracy: 0.7850 - val_loss: 6.9670 - val_accuracy: 0.7450
Epoch 8/50
13/13 [==============================] - 0s 11ms/step - loss: 3.7469 - accuracy: 0.8000 - val_loss: 7.0383 - val_accuracy: 0.7550
Epoch 9/50
13/13 [==============================] - 0s 12ms/step - loss: 3.0596 - accuracy: 0.8181 - val_loss: 5.7809 - val_accuracy: 0.7675
Epoch 10/50
13/13 [==============================] - 0s 13ms/step - loss: 2.3605 - accuracy: 0.8413 - val_loss: 5.3989 - val_accuracy: 0.7725
Epoch 11/50
13/13 [==============================] - 0s 11ms/step - loss: 2.1807 - accuracy: 0.8569 - val_loss: 6.6049 - val_accuracy: 0.7225
Epoch 12/50
13/13 [==============================] - 0s 11ms/step - loss: 2.3857 - accuracy: 0.8444 - val_loss: 5.5992 - val_accuracy: 0.7900
Epoch 13/50
13/13 [==============================] - 0s 12ms/step - loss: 2.4752 - accuracy: 0.8425 - val_loss: 5.9807 - val_accuracy: 0.7550
Epoch 14/50
13/13 [==============================] - 0s 12ms/step - loss: 1.8086 - accuracy: 0.8537 - val_loss: 6.3599 - val_accuracy: 0.7475
Epoch 15/50
13/13 [==============================] - 0s 8ms/step - loss: 1.5521 - accuracy: 0.8687 - val_loss: 5.7410 - val_accuracy: 0.7700
Epoch 16/50
13/13 [==============================] - 0s 8ms/step - loss: 1.6466 - accuracy: 0.8706 - val_loss: 5.3697 - val_accuracy: 0.7825
Epoch 17/50
13/13 [==============================] - 0s 9ms/step - loss: 1.0975 - accuracy: 0.9013 - val_loss: 5.7618 - val_accuracy: 0.7725
Epoch 18/50
13/13 [==============================] - 0s 8ms/step - loss: 1.6555 - accuracy: 0.8750 - val_loss: 5.3824 - val_accuracy: 0.7925
Epoch 19/50
13/13 [==============================] - 0s 8ms/step - loss: 1.1808 - accuracy: 0.8944 - val_loss: 5.3693 - val_accuracy: 0.7825
Epoch 20/50
13/13 [==============================] - 0s 6ms/step - loss: 0.9592 - accuracy: 0.9119 - val_loss: 5.2706 - val_accuracy: 0.7800
Epoch 21/50
```

```
13/13 [==============================] - 0s 8ms/step - loss: 0.6497 - accuracy: 0.9269 - val_loss: 5.3130 - val_accuracy: 0.7675
Epoch 22/50
13/13 [==============================] - 0s 7ms/step - loss: 0.5670 - accuracy: 0.9281 - val_loss: 5.1757 - val_accuracy: 0.7900
Epoch 23/50
13/13 [==============================] - 0s 8ms/step - loss: 0.5039 - accuracy: 0.9400 - val_loss: 5.4355 - val_accuracy: 0.7650
Epoch 24/50
13/13 [==============================] - 0s 8ms/step - loss: 0.3513 - accuracy: 0.9475 - val_loss: 5.1484 - val_accuracy: 0.7925
Epoch 25/50
13/13 [==============================] - 0s 7ms/step - loss: 0.2953 - accuracy: 0.9650 - val_loss: 5.6867 - val_accuracy: 0.7675
Epoch 26/50
13/13 [==============================] - 0s 8ms/step - loss: 0.4298 - accuracy: 0.9481 - val_loss: 5.5839 - val_accuracy: 0.7700
Epoch 27/50
13/13 [==============================] - 0s 8ms/step - loss: 0.6587 - accuracy: 0.9225 - val_loss: 5.6219 - val_accuracy: 0.8025
Epoch 28/50
13/13 [==============================] - 0s 8ms/step - loss: 0.7573 - accuracy: 0.9225 - val_loss: 6.3727 - val_accuracy: 0.7900
Epoch 29/50
```

```python
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```python
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

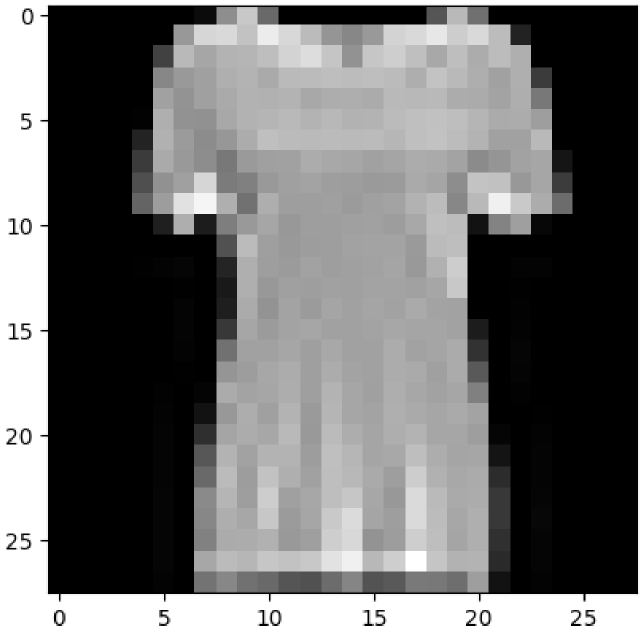



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 9.17452621459961
Test accuracy: 0.7237794399261475
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 70ms/step
array([[1.0000000e+00, 0.0000000e+00, 0.0000000e+00, 2.8871810e-21,
        0.0000000e+00, 0.0000000e+00, 1.4660929e-26, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 50

Zwiększony zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 128)               100480
```

```
 dense_21 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się do około 4 epoki, potem praktycznie się nie uczy, wykres błędu (treningowego i walidacyjnego) jest praktycznie na stałym poziomie.

```
(train_data, train_labels), (test_data, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
```

One-hot encoding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((60000, 28, 28), (60000, 10))
```

```
test_data.shape,test_labels.shape
```

```
((10000, 28, 28), (10000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

Regularyzacja - metoda 2

Zmniejszamy **wielkość modelu**:

```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

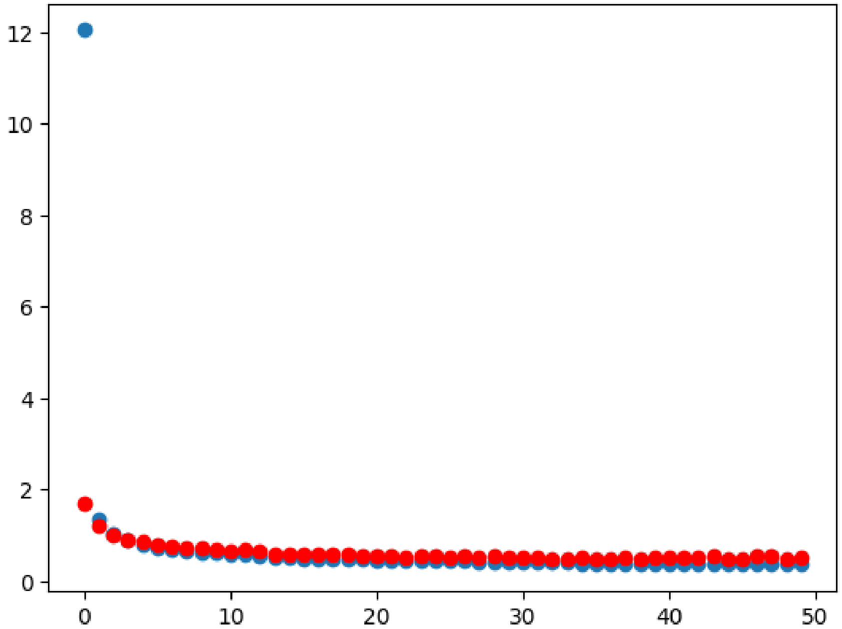
```
model = Sequential()
model.add(Dense(units = 64, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 64)                50240

 dense_23 (Dense)            (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

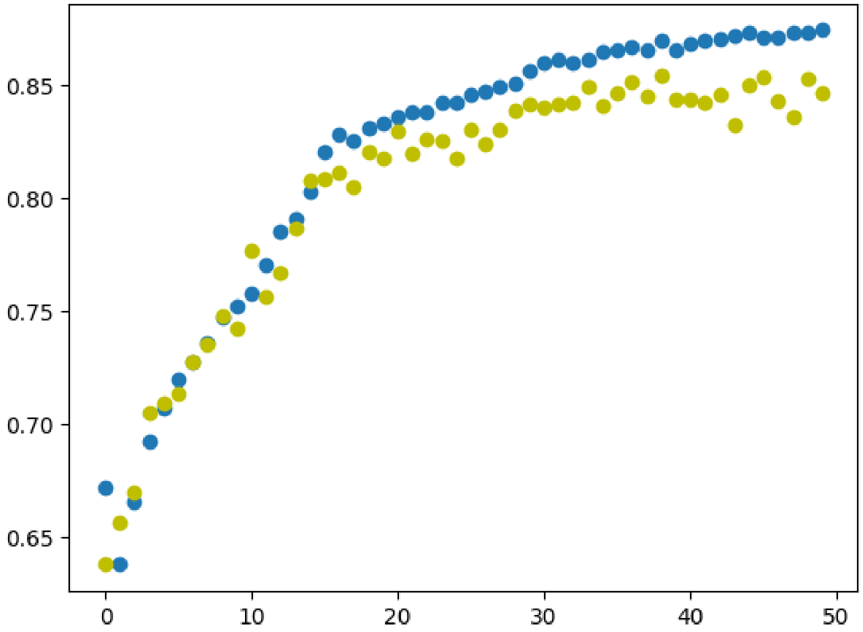
```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
Epoch 28/50
375/375 [==============================] - 1s 3ms/step - loss: 0.4255 - accuracy: 0.8493 - val_loss: 0.5208 - val_accuracy: 0.8300
Epoch 29/50
375/375 [==============================] - 1s 3ms/step - loss: 0.4233 - accuracy: 0.8508 - val_loss: 0.5558 - val_accuracy: 0.8391
Epoch 30/50
375/375 [==============================] - 1s 3ms/step - loss: 0.4103 - accuracy: 0.8564 - val_loss: 0.5049 - val_accuracy: 0.8413
Epoch 31/50
375/375 [==============================] - 1s 4ms/step - loss: 0.4038 - accuracy: 0.8599 - val_loss: 0.5021 - val_accuracy: 0.8402
Epoch 32/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3976 - accuracy: 0.8613 - val_loss: 0.5069 - val_accuracy: 0.8415
Epoch 33/50
375/375 [==============================] - 1s 3ms/step - loss: 0.4029 - accuracy: 0.8596 - val_loss: 0.4775 - val_accuracy: 0.8420
Epoch 34/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3998 - accuracy: 0.8609 - val_loss: 0.4897 - val_accuracy: 0.8493
Epoch 35/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3902 - accuracy: 0.8650 - val_loss: 0.4996 - val_accuracy: 0.8410
Epoch 36/50
375/375 [==============================] - 2s 6ms/step - loss: 0.3864 - accuracy: 0.8658 - val_loss: 0.4982 - val_accuracy: 0.8462
Epoch 37/50
375/375 [==============================] - 2s 4ms/step - loss: 0.3866 - accuracy: 0.8668 - val_loss: 0.4815 - val_accuracy: 0.8511
Epoch 38/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3874 - accuracy: 0.8655 - val_loss: 0.5077 - val_accuracy: 0.8453
Epoch 39/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3743 - accuracy: 0.8698 - val_loss: 0.4713 - val_accuracy: 0.8539
Epoch 40/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3857 - accuracy: 0.8657 - val_loss: 0.5095 - val_accuracy: 0.8436
Epoch 41/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3790 - accuracy: 0.8685 - val_loss: 0.5231 - val_accuracy: 0.8436
Epoch 42/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3715 - accuracy: 0.8696 - val_loss: 0.5009 - val_accuracy: 0.8424
Epoch 43/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3732 - accuracy: 0.8707 - val_loss: 0.4991 - val_accuracy: 0.8459
Epoch 44/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3704 - accuracy: 0.8721 - val_loss: 0.5404 - val_accuracy: 0.8324
Epoch 45/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3643 - accuracy: 0.8733 - val_loss: 0.4862 - val_accuracy: 0.8500
Epoch 46/50
375/375 [==============================] - 2s 5ms/step - loss: 0.3749 - accuracy: 0.8709 - val_loss: 0.4871 - val_accuracy: 0.8533
Epoch 47/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3727 - accuracy: 0.8709 - val_loss: 0.5386 - val_accuracy: 0.8432
Epoch 48/50
375/375 [==============================] - 1s 4ms/step - loss: 0.3636 - accuracy: 0.8734 - val_loss: 0.5403 - val_accuracy: 0.8362
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3609 - accuracy: 0.8734 - val_loss: 0.4892 - val_accuracy: 0.8527
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.3600 - accuracy: 0.8743 - val_loss: 0.5066 - val_accuracy: 0.8462
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

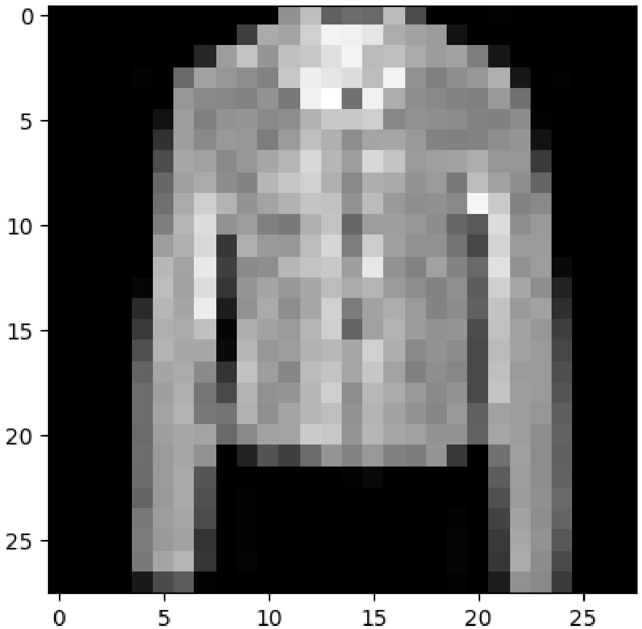



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.5330622792243958
Test accuracy: 0.8331999778747559
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 54ms/step
array([[3.4073522e-04, 6.0094747e-04, 2.4575439e-01, 7.5479900e-03,
        6.1756253e-01, 4.2736414e-10, 1.2804353e-01, 4.0521580e-32,
        1.4999736e-04, 4.7665488e-24]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 50

Zbiór treningowy **60000** (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_22 (Dense)            (None, 64)                50240     
                                                                 
 dense_23 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się, mniewięcej do 35 epoki, poźniej następuje lekkie niewielkie przeuczeniem wykresy błędu (treningowego i walidacyjnego) się obijają

Regularyzacja - metoda 3

Import normy L2:

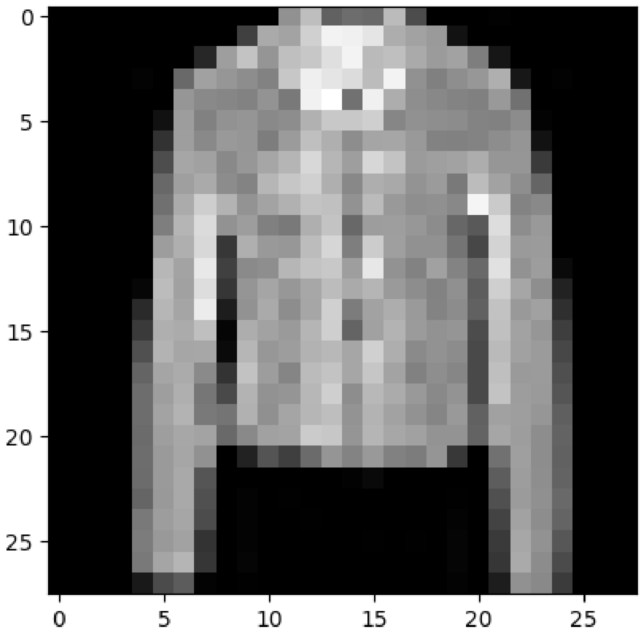
```
from keras.regularizers import l2
```

Danie regularyzacji L2 do warstw:

Visulization

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

```
model = Sequential()
model.add(Dense(units = 128,kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10,kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_24 (Dense)            (None, 128)               100480    
                                                                 
 dense_25 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
```

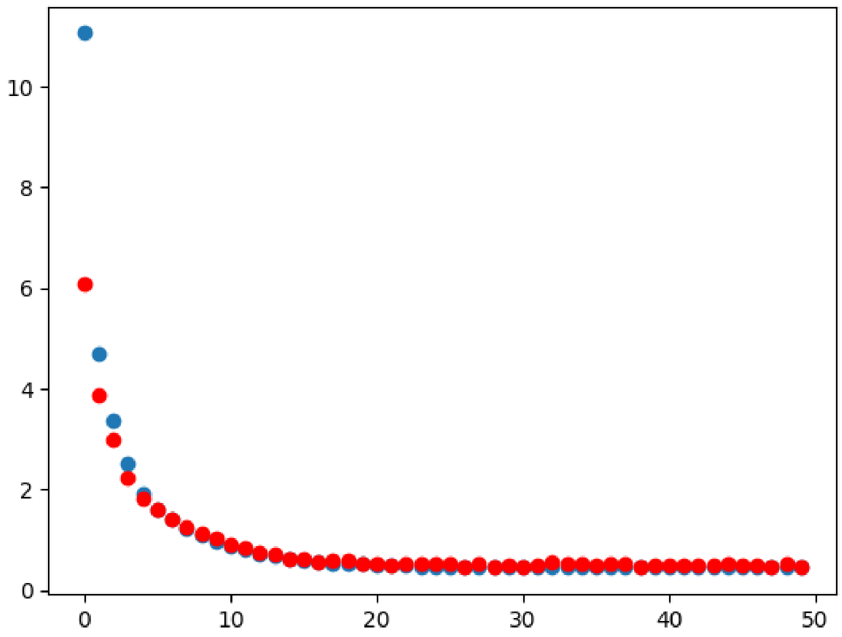
```
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

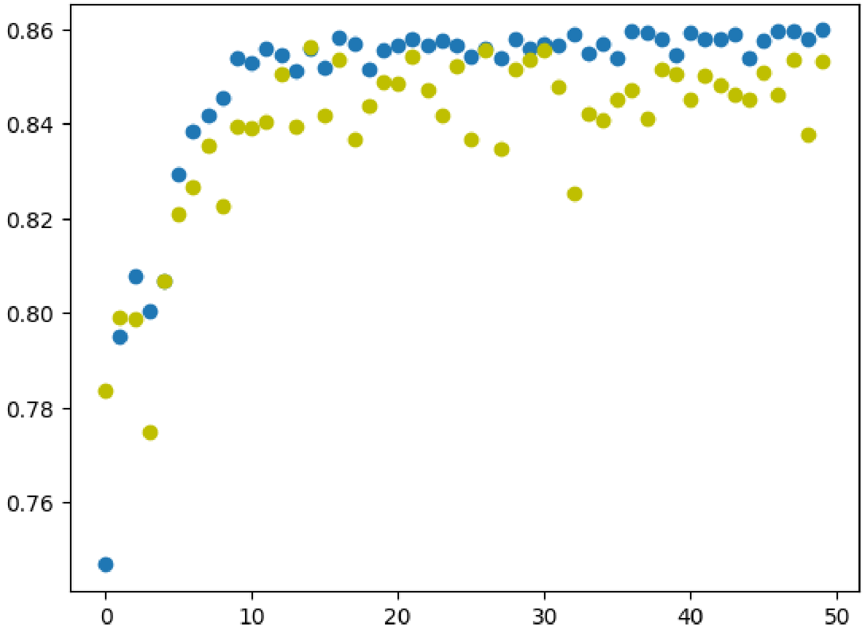
```
Epoch 1/50
375/375 [==============================] - 3s 5ms/step - loss: 11.0544 - accuracy: 0.7468 - val_loss: 6.0950 - val_accuracy: 0.7835
Epoch 2/50
375/375 [==============================] - 2s 5ms/step - loss: 4.6912 - accuracy: 0.7949 - val_loss: 3.8637 - val_accuracy: 0.7991
Epoch 3/50
375/375 [==============================] - 2s 5ms/step - loss: 3.3657 - accuracy: 0.8078 - val_loss: 2.9910 - val_accuracy: 0.7987
Epoch 4/50
375/375 [==============================] - 3s 9ms/step - loss: 2.5110 - accuracy: 0.8005 - val_loss: 2.2197 - val_accuracy: 0.7750
Epoch 5/50
375/375 [==============================] - 2s 5ms/step - loss: 1.8971 - accuracy: 0.8067 - val_loss: 1.8154 - val_accuracy: 0.8069
Epoch 6/50
375/375 [==============================] - 2s 5ms/step - loss: 1.6013 - accuracy: 0.8295 - val_loss: 1.5867 - val_accuracy: 0.8207
Epoch 7/50
375/375 [==============================] - 2s 5ms/step - loss: 1.3967 - accuracy: 0.8384 - val_loss: 1.3991 - val_accuracy: 0.8266
Epoch 8/50
375/375 [==============================] - 2s 5ms/step - loss: 1.2305 - accuracy: 0.8419 - val_loss: 1.2346 - val_accuracy: 0.8353
Epoch 9/50
375/375 [==============================] - 2s 5ms/step - loss: 1.1005 - accuracy: 0.8456 - val_loss: 1.1323 - val_accuracy: 0.8225
Epoch 10/50
375/375 [==============================] - 3s 7ms/step - loss: 0.9770 - accuracy: 0.8540 - val_loss: 1.0251 - val_accuracy: 0.8395
Epoch 11/50
375/375 [==============================] - 3s 7ms/step - loss: 0.8823 - accuracy: 0.8530 - val_loss: 0.8950 - val_accuracy: 0.8389
Epoch 12/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7951 - accuracy: 0.8558 - val_loss: 0.8399 - val_accuracy: 0.8405
Epoch 13/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7266 - accuracy: 0.8546 - val_loss: 0.7284 - val_accuracy: 0.8503
Epoch 14/50
375/375 [==============================] - 2s 5ms/step - loss: 0.6783 - accuracy: 0.8513 - val_loss: 0.7233 - val_accuracy: 0.8393
Epoch 15/50
375/375 [==============================] - 2s 5ms/step - loss: 0.6217 - accuracy: 0.8557 - val_loss: 0.6101 - val_accuracy: 0.8563
Epoch 16/50
375/375 [==============================] - 3s 7ms/step - loss: 0.5949 - accuracy: 0.8518 - val_loss: 0.6331 - val_accuracy: 0.8418
Epoch 17/50
375/375 [==============================] - 3s 8ms/step - loss: 0.5475 - accuracy: 0.8581 - val_loss: 0.5642 - val_accuracy: 0.8534
Epoch 18/50
375/375 [==============================] - 2s 5ms/step - loss: 0.5316 - accuracy: 0.8568 - val_loss: 0.5948 - val_accuracy: 0.8366
Epoch 19/50
375/375 [==============================] - 2s 5ms/step - loss: 0.5276 - accuracy: 0.8515 - val_loss: 0.5728 - val_accuracy: 0.8438
Epoch 20/50
375/375 [==============================] - 2s 5ms/step - loss: 0.5101 - accuracy: 0.8555 - val_loss: 0.5264 - val_accuracy: 0.8487
Epoch 21/50
```

```
375/375 [==============================] - 2s 5ms/step - loss: 0.4904 - accuracy: 0.8566 - val_loss: 0.5181 - val_accuracy: 0.8483
Epoch 22/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4793 - accuracy: 0.8578 - val_loss: 0.5007 - val_accuracy: 0.8543
Epoch 23/50
375/375 [==============================] - 3s 8ms/step - loss: 0.4798 - accuracy: 0.8565 - val_loss: 0.5157 - val_accuracy: 0.8472
Epoch 24/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4686 - accuracy: 0.8576 - val_loss: 0.5129 - val_accuracy: 0.8418
Epoch 25/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4700 - accuracy: 0.8567 - val_loss: 0.5083 - val_accuracy: 0.8522
Epoch 26/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4683 - accuracy: 0.8541 - val_loss: 0.5270 - val_accuracy: 0.8366
Epoch 27/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4629 - accuracy: 0.8559 - val_loss: 0.4728 - val_accuracy: 0.8557
Epoch 28/50
375/375 [==============================] - 2s 5ms/step - loss: 0.4678 - accuracy: 0.8537 - val_loss: 0.5197 - val_accuracy: 0.8347
Epoch 29/50
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

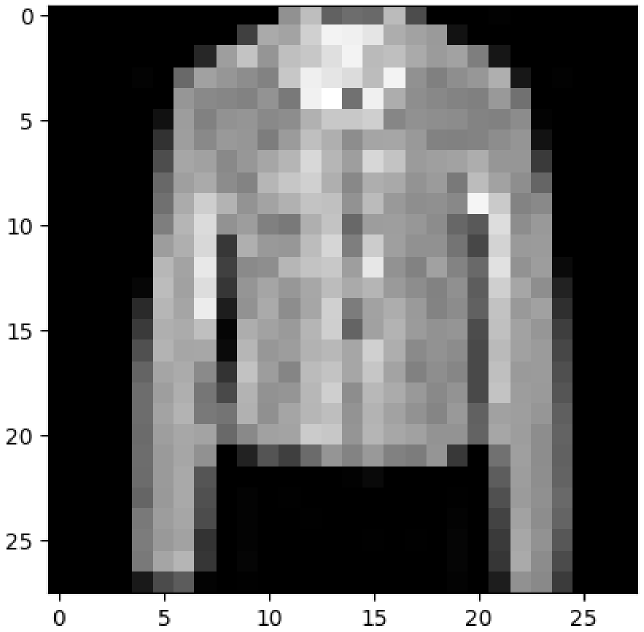



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
    Test loss: 0.5001400709152222
    Test accuracy: 0.8482000231742859
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 53ms/step
array([[7.0416898e-04, 2.7944816e-07, 2.2078680e-01, 1.8817747e-03,
        7.0002413e-01, 1.7965411e-06, 7.4053541e-02, 1.0402940e-17,
        2.5475516e-03, 8.2536505e-16]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 50

Zwiększony zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

kernel_regularizer=l2(0.01) (we wszystkich warstwach)

```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_24 (Dense)            (None, 128)               100480    
                                                                 
 dense_25 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się do około 30 epoki potem się praktycznie nie uczy, ale się nie przeucza, wykresy błędu (treningowego i walidacyjnego)są podobne przy czym błędy cały czas spadają, ale od około 30 epoki spada bardzo wolno wręcz są stałe. Dokładność modelu dla danych treningowych i walidacyjnych praktycznie cały czas rośnie, ale po 30 epoce mniej.

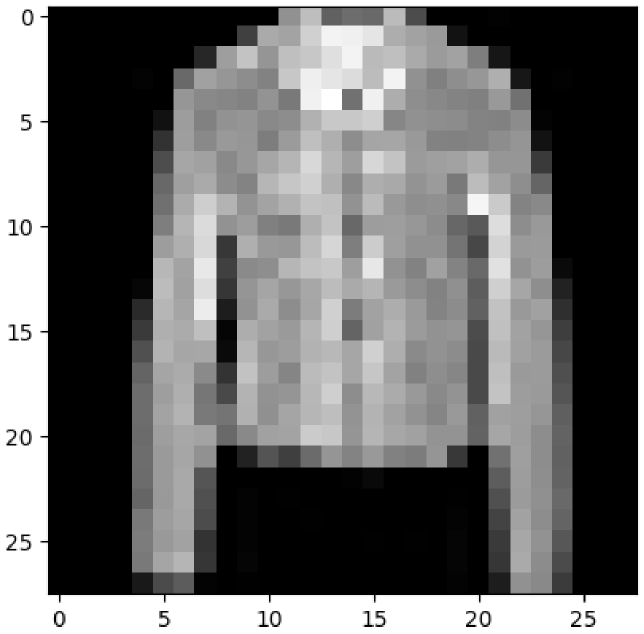
Regularyzacja - metoda 4

```
from keras.layers import Dropout
```

Visulization

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

```
model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_26 (Dense)            (None, 128)               100480    
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_27 (Dense)            (None, 10)                1290      
```

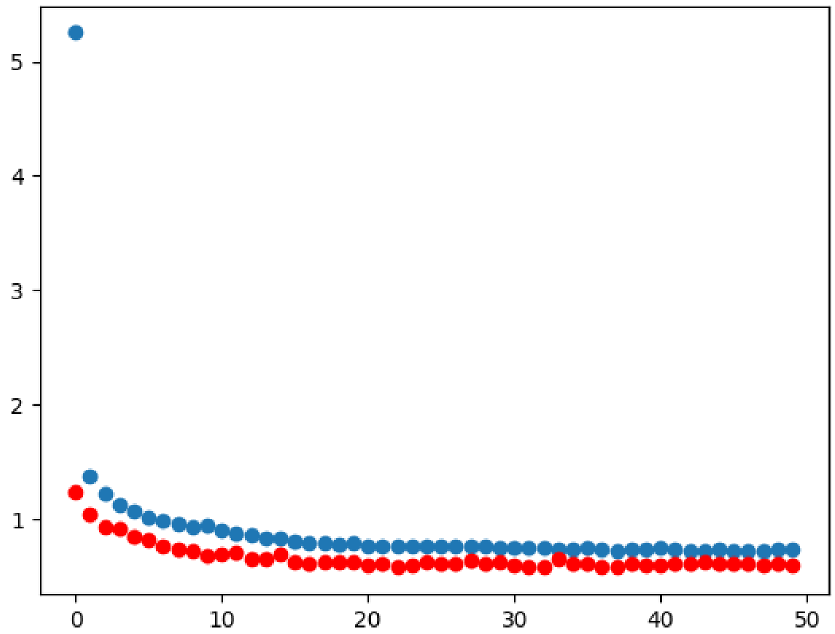
```
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

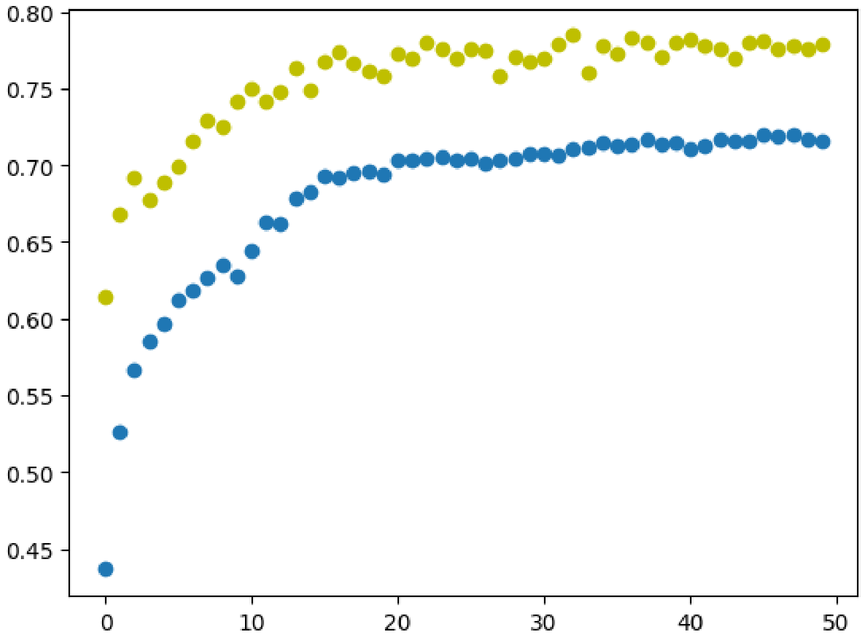
```
Epoch 1/50
375/375 [==============================] - 3s 6ms/step - loss: 5.2511 - accuracy: 0.4366 - val_loss: 1.2329 - val_accuracy: 0.6142
Epoch 2/50
375/375 [==============================] - 2s 5ms/step - loss: 1.3757 - accuracy: 0.5263 - val_loss: 1.0405 - val_accuracy: 0.6683
Epoch 3/50
375/375 [==============================] - 2s 5ms/step - loss: 1.2157 - accuracy: 0.5670 - val_loss: 0.9289 - val_accuracy: 0.6915
Epoch 4/50
375/375 [==============================] - 3s 8ms/step - loss: 1.1210 - accuracy: 0.5855 - val_loss: 0.9082 - val_accuracy: 0.6775
Epoch 5/50
375/375 [==============================] - 3s 7ms/step - loss: 1.0657 - accuracy: 0.5962 - val_loss: 0.8424 - val_accuracy: 0.6892
Epoch 6/50
375/375 [==============================] - 2s 5ms/step - loss: 1.0139 - accuracy: 0.6119 - val_loss: 0.8199 - val_accuracy: 0.6992
Epoch 7/50
375/375 [==============================] - 2s 5ms/step - loss: 0.9866 - accuracy: 0.6183 - val_loss: 0.7584 - val_accuracy: 0.7153
Epoch 8/50
375/375 [==============================] - 2s 5ms/step - loss: 0.9610 - accuracy: 0.6270 - val_loss: 0.7285 - val_accuracy: 0.7288
Epoch 9/50
375/375 [==============================] - 2s 5ms/step - loss: 0.9285 - accuracy: 0.6354 - val_loss: 0.7244 - val_accuracy: 0.7256
Epoch 10/50
375/375 [==============================] - 2s 6ms/step - loss: 0.9409 - accuracy: 0.6277 - val_loss: 0.6728 - val_accuracy: 0.7419
Epoch 11/50
375/375 [==============================] - 3s 8ms/step - loss: 0.8991 - accuracy: 0.6443 - val_loss: 0.6968 - val_accuracy: 0.7498
Epoch 12/50
375/375 [==============================] - 2s 5ms/step - loss: 0.8659 - accuracy: 0.6627 - val_loss: 0.7039 - val_accuracy: 0.7422
Epoch 13/50
375/375 [==============================] - 2s 5ms/step - loss: 0.8632 - accuracy: 0.6622 - val_loss: 0.6496 - val_accuracy: 0.7482
Epoch 14/50
375/375 [==============================] - 2s 5ms/step - loss: 0.8308 - accuracy: 0.6789 - val_loss: 0.6421 - val_accuracy: 0.7638
Epoch 15/50
375/375 [==============================] - 2s 5ms/step - loss: 0.8283 - accuracy: 0.6824 - val_loss: 0.6882 - val_accuracy: 0.7494
Epoch 16/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7966 - accuracy: 0.6934 - val_loss: 0.6198 - val_accuracy: 0.7679
Epoch 17/50
375/375 [==============================] - 3s 7ms/step - loss: 0.7883 - accuracy: 0.6917 - val_loss: 0.6105 - val_accuracy: 0.7738
Epoch 18/50
375/375 [==============================] - 3s 7ms/step - loss: 0.7845 - accuracy: 0.6954 - val_loss: 0.6216 - val_accuracy: 0.7670
Epoch 19/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7805 - accuracy: 0.6962 - val_loss: 0.6183 - val_accuracy: 0.7613
```

```
Epoch 20/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7897 - accuracy: 0.6938 - val_loss: 0.6240 - val_accuracy: 0.7587
Epoch 21/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7619 - accuracy: 0.7035 - val_loss: 0.5924 - val_accuracy: 0.7722
Epoch 22/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7592 - accuracy: 0.7029 - val_loss: 0.6085 - val_accuracy: 0.7696
Epoch 23/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7579 - accuracy: 0.7048 - val_loss: 0.5814 - val_accuracy: 0.7799
Epoch 24/50
375/375 [==============================] - 3s 8ms/step - loss: 0.7625 - accuracy: 0.7054 - val_loss: 0.5865 - val_accuracy: 0.7754
Epoch 25/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7598 - accuracy: 0.7033 - val_loss: 0.6239 - val_accuracy: 0.7692
Epoch 26/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7569 - accuracy: 0.7044 - val_loss: 0.6074 - val_accuracy: 0.7760
Epoch 27/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7614 - accuracy: 0.7015 - val_loss: 0.6012 - val_accuracy: 0.7753
Epoch 28/50
375/375 [==============================] - 2s 5ms/step - loss: 0.7578 - accuracy: 0.7029 - val_loss: 0.6362 - val_accuracy: 0.7584
Epoch 29/50
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```



```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

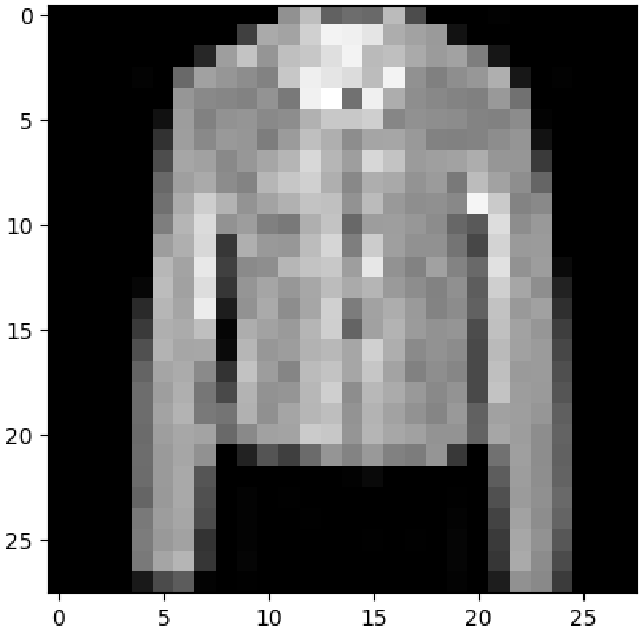



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.6161739826202393
Test accuracy: 0.7731000185012817
```

```
def plot_image(img_index):
    label_index = test_labels[img_index]
    plt.imshow(test_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = test_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 50ms/step
array([[0.04613129, 0.00544799, 0.3449152 , 0.05428627, 0.32339495,
        0.00322585, 0.17669947, 0.00753833, 0.0344306 , 0.00392997]],
      dtype=float32)
```



Opis:

batch_size = 128

epochs = 50

Zbiór treningowy **60000** (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

model.add(Dropout(0.4))

```
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 dense_26 (Dense)            (None, 128)               100480

 dropout_1 (Dropout)         (None, 128)               0

 dense_27 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się do samego końca i się nie przeucza, wykresy błędu (treningowego i walidacyjnego)są podobne przy czym błędy cały czas spadają. Dokładność modelu dla danych treningowych i walidacyjnych praktycznie cały czas rośnie.

Regularyzacja all in one #1

Zwiększamy zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

```
test_data = data[:68000]
train_data = data[68000:]
test_labels = label[:68000]
train_labels = label[68000:]
```

```
print(test_data.shape,train_data.shape,test_labels.shape,train_labels.shape)
```

```
(68000, 28, 28) (2000, 28, 28) (68000,) (2000,)
```

One-hot coding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((2000, 28, 28), (2000, 10))
```

```
test_data.shape,test_labels.shape
```

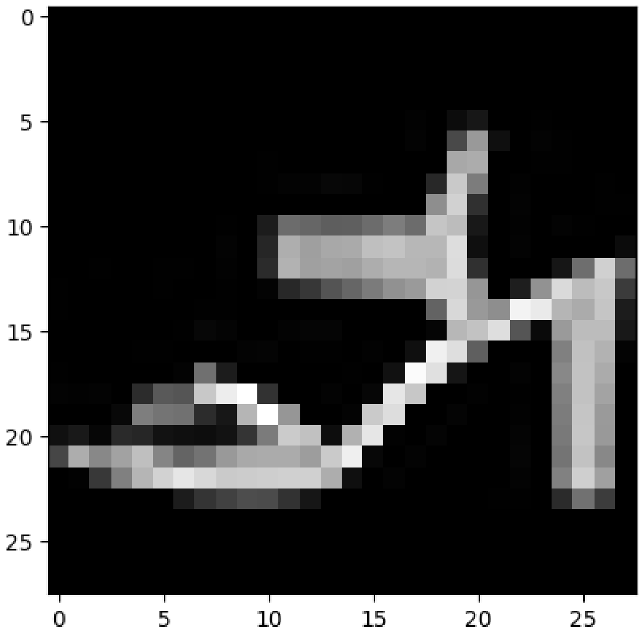
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

Danie **regularyzacji L2** do warstw:

Adding dropout layer

Resizing model

```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

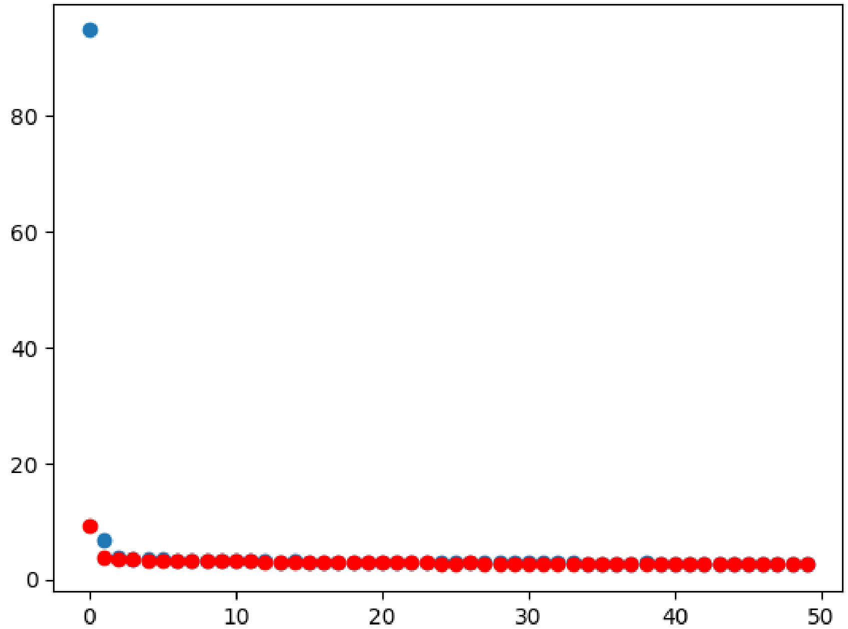
```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_28 (Dense)            (None, 64)                50240     
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_29 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
batch_size = 128
epochs = 50

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

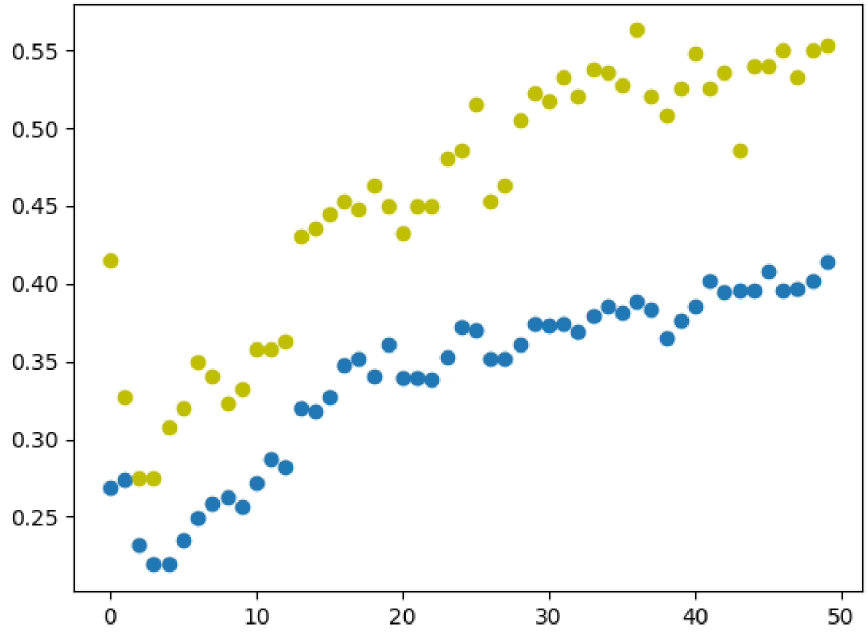
```
Epoch 1/50
13/13 [==============================] - 1s 21ms/step - loss: 94.5970 - accuracy: 0.2688 - val_loss: 9.1260 - val_accuracy: 0.4150
Epoch 2/50
13/13 [==============================] - 0s 7ms/step - loss: 6.7608 - accuracy: 0.2744 - val_loss: 3.7434 - val_accuracy: 0.3275
Epoch 3/50
13/13 [==============================] - 0s 6ms/step - loss: 3.8201 - accuracy: 0.2325 - val_loss: 3.5108 - val_accuracy: 0.2750
Epoch 4/50
13/13 [==============================] - 0s 7ms/step - loss: 3.5522 - accuracy: 0.2200 - val_loss: 3.4397 - val_accuracy: 0.2750
Epoch 5/50
13/13 [==============================] - 0s 6ms/step - loss: 3.4397 - accuracy: 0.2194 - val_loss: 3.3179 - val_accuracy: 0.3075
Epoch 6/50
13/13 [==============================] - 0s 6ms/step - loss: 3.3405 - accuracy: 0.2350 - val_loss: 3.2609 - val_accuracy: 0.3200
Epoch 7/50
13/13 [==============================] - 0s 6ms/step - loss: 3.2884 - accuracy: 0.2494 - val_loss: 3.1970 - val_accuracy: 0.3500
Epoch 8/50
13/13 [==============================] - 0s 6ms/step - loss: 3.3240 - accuracy: 0.2587 - val_loss: 3.1951 - val_accuracy: 0.3400
Epoch 9/50
```

```
13/13 [==============================] - 0s 6ms/step - loss: 3.2225 - accuracy: 0.2631 - val_loss: 3.2238 - val_accuracy: 0.3225
Epoch 10/50
13/13 [==============================] - 0s 7ms/step - loss: 3.2560 - accuracy: 0.2562 - val_loss: 3.1718 - val_accuracy: 0.3325
Epoch 11/50
13/13 [==============================] - 0s 6ms/step - loss: 3.1887 - accuracy: 0.2719 - val_loss: 3.0836 - val_accuracy: 0.3575
Epoch 12/50
13/13 [==============================] - 0s 6ms/step - loss: 3.1100 - accuracy: 0.2869 - val_loss: 3.0682 - val_accuracy: 0.3575
Epoch 13/50
13/13 [==============================] - 0s 6ms/step - loss: 3.0696 - accuracy: 0.2825 - val_loss: 2.9911 - val_accuracy: 0.3625
Epoch 14/50
13/13 [==============================] - 0s 6ms/step - loss: 3.0460 - accuracy: 0.3200 - val_loss: 2.9773 - val_accuracy: 0.4300
Epoch 15/50
13/13 [==============================] - 0s 6ms/step - loss: 3.0686 - accuracy: 0.3181 - val_loss: 2.9308 - val_accuracy: 0.4350
Epoch 16/50
13/13 [==============================] - 0s 7ms/step - loss: 3.0269 - accuracy: 0.3275 - val_loss: 2.8891 - val_accuracy: 0.4450
Epoch 17/50
13/13 [==============================] - 0s 7ms/step - loss: 2.9543 - accuracy: 0.3475 - val_loss: 2.8804 - val_accuracy: 0.4525
Epoch 18/50
13/13 [==============================] - 0s 6ms/step - loss: 2.9568 - accuracy: 0.3519 - val_loss: 2.8498 - val_accuracy: 0.4475
Epoch 19/50
13/13 [==============================] - 0s 7ms/step - loss: 2.9638 - accuracy: 0.3400 - val_loss: 2.8417 - val_accuracy: 0.4625
Epoch 20/50
13/13 [==============================] - 0s 6ms/step - loss: 2.9032 - accuracy: 0.3613 - val_loss: 2.8663 - val_accuracy: 0.4500
Epoch 21/50
13/13 [==============================] - 0s 7ms/step - loss: 2.9466 - accuracy: 0.3394 - val_loss: 2.8828 - val_accuracy: 0.4325
Epoch 22/50
13/13 [==============================] - 0s 7ms/step - loss: 2.9363 - accuracy: 0.3394 - val_loss: 2.8518 - val_accuracy: 0.4500
Epoch 23/50
13/13 [==============================] - 0s 6ms/step - loss: 2.9294 - accuracy: 0.3381 - val_loss: 2.8270 - val_accuracy: 0.4500
Epoch 24/50
13/13 [==============================] - 0s 8ms/step - loss: 2.8886 - accuracy: 0.3531 - val_loss: 2.8281 - val_accuracy: 0.4800
Epoch 25/50
13/13 [==============================] - 0s 7ms/step - loss: 2.8545 - accuracy: 0.3725 - val_loss: 2.7652 - val_accuracy: 0.4850
Epoch 26/50
13/13 [==============================] - 0s 6ms/step - loss: 2.8305 - accuracy: 0.3700 - val_loss: 2.7827 - val_accuracy: 0.5150
Epoch 27/50
13/13 [==============================] - 0s 6ms/step - loss: 2.9159 - accuracy: 0.3512 - val_loss: 2.7959 - val_accuracy: 0.4525
Epoch 28/50
13/13 [==============================] - 0s 6ms/step - loss: 2.8792 - accuracy: 0.3519 - val_loss: 2.7486 - val_accuracy: 0.4625
Epoch 29/50
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

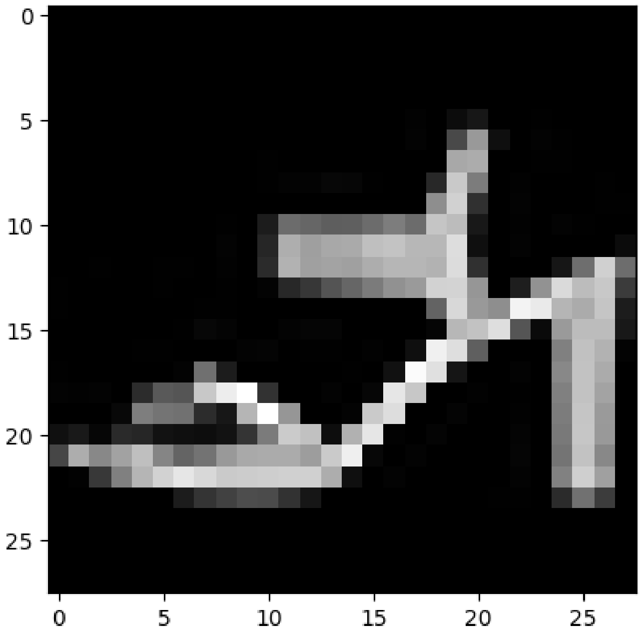



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 2.6748206615448
Test accuracy: 0.50688236951828
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = train_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
1/1 [==============================] - 0s 54ms/step
array([[1.66832937e-13, 1.38013626e-14, 2.98805759e-15, 1.05429781e-16,
        5.17348916e-22, 1.00000000e+00, 2.45318936e-14, 5.63692697e-16,
        7.48609855e-12, 3.32365185e-13]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 50

Zbiór treningowy zwiększony z ***60000*** do **68000* (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

kernel_regularizer=l2(0.01) we wszystkich warstwach

model.add(Dropout(0.4))

```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 64)                50240

 dropout_2 (Dropout)         (None, 64)                0

 dense_29 (Dense)            (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się do samego końca i się nie przeucza, wykresy błędu (treningowego i walidacyjnego)są podobne przy czym błędy cały czas, ale od 5 epoki dużo wolniej. Dokładność modelu dla danych treningowych i walidacyjnych praktycznie cały czas rośnie.

Regularyzacja all in one #2

Zwiększamy zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

```
test_data = data[:68000]
train_data = data[68000:]
test_labels = label[:68000]
train_labels = label[68000:]
```

```
print(test_data.shape,train_data.shape,test_labels.shape,train_labels.shape)
```

```
(68000, 28, 28) (2000, 28, 28) (68000,) (2000,)
```

One-hot coding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((2000, 28, 28), (2000, 10))
```

```
test_data.shape,test_labels.shape
```

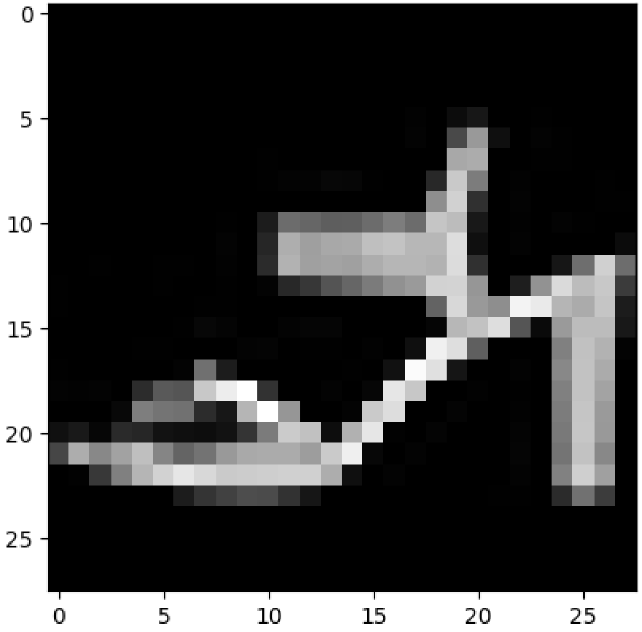
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```




```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

Danie **regularyzacji L2** do warstw:

Adding dropout layer

Resizing model

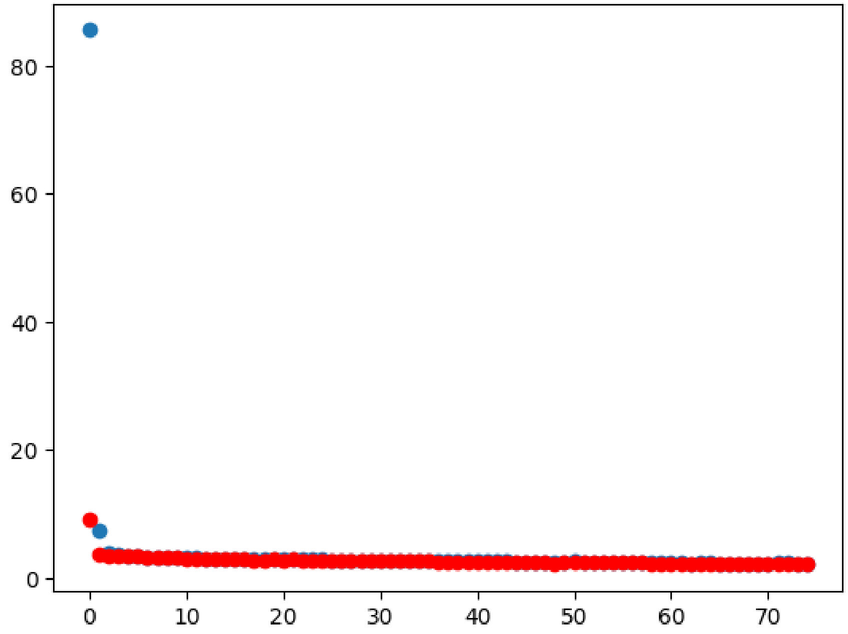
```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.001)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_30 (Dense)            (None, 64)                50240     
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_31 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

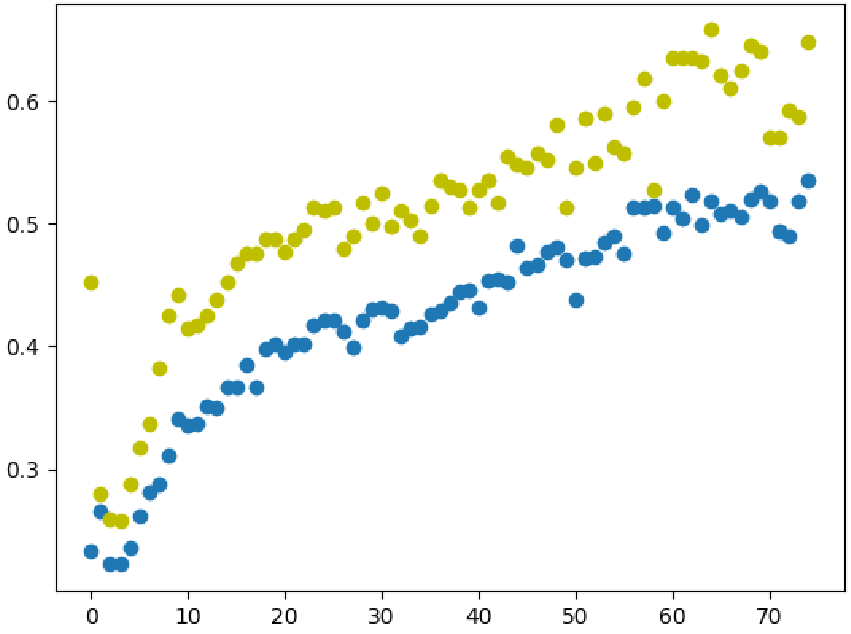
```
batch_size = 128
epochs = 75

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
13/13 [==============================] - 0s 7ms/step - loss: 2.4358 - accuracy: 0.4894 - val_loss: 2.4337 - val_accuracy: 0.5625
Epoch 56/75
13/13 [==============================] - 0s 5ms/step - loss: 2.4122 - accuracy: 0.4750 - val_loss: 2.3062 - val_accuracy: 0.5575
Epoch 57/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3680 - accuracy: 0.5138 - val_loss: 2.4059 - val_accuracy: 0.5950
Epoch 58/75
13/13 [==============================] - 0s 7ms/step - loss: 2.4393 - accuracy: 0.5138 - val_loss: 2.3628 - val_accuracy: 0.6175
Epoch 59/75
13/13 [==============================] - 0s 7ms/step - loss: 2.4037 - accuracy: 0.5144 - val_loss: 2.2812 - val_accuracy: 0.5275
Epoch 60/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3773 - accuracy: 0.4919 - val_loss: 2.1593 - val_accuracy: 0.6000
Epoch 61/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3666 - accuracy: 0.5125 - val_loss: 2.0989 - val_accuracy: 0.6350
Epoch 62/75
13/13 [==============================] - 0s 7ms/step - loss: 2.3574 - accuracy: 0.5044 - val_loss: 2.0466 - val_accuracy: 0.6350
Epoch 63/75
13/13 [==============================] - 0s 7ms/step - loss: 2.2886 - accuracy: 0.5231 - val_loss: 2.0769 - val_accuracy: 0.6350
Epoch 64/75
13/13 [==============================] - 0s 7ms/step - loss: 2.3254 - accuracy: 0.4988 - val_loss: 2.1444 - val_accuracy: 0.6325
Epoch 65/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3137 - accuracy: 0.5181 - val_loss: 2.0541 - val_accuracy: 0.6575
Epoch 66/75
13/13 [==============================] - 0s 6ms/step - loss: 2.2741 - accuracy: 0.5081 - val_loss: 2.1783 - val_accuracy: 0.6200
Epoch 67/75
13/13 [==============================] - 0s 7ms/step - loss: 2.2529 - accuracy: 0.5106 - val_loss: 2.2188 - val_accuracy: 0.6100
Epoch 68/75
13/13 [==============================] - 0s 6ms/step - loss: 2.2756 - accuracy: 0.5056 - val_loss: 2.0831 - val_accuracy: 0.6250
Epoch 69/75
13/13 [==============================] - 0s 6ms/step - loss: 2.2358 - accuracy: 0.5200 - val_loss: 2.1722 - val_accuracy: 0.6450
Epoch 70/75
13/13 [==============================] - 0s 5ms/step - loss: 2.2524 - accuracy: 0.5256 - val_loss: 2.1875 - val_accuracy: 0.6400
Epoch 71/75
13/13 [==============================] - 0s 6ms/step - loss: 2.2713 - accuracy: 0.5188 - val_loss: 2.2721 - val_accuracy: 0.5700
Epoch 72/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3126 - accuracy: 0.4931 - val_loss: 2.2008 - val_accuracy: 0.5700
Epoch 73/75
13/13 [==============================] - 0s 6ms/step - loss: 2.3094 - accuracy: 0.4900 - val_loss: 2.1761 - val_accuracy: 0.5925
Epoch 74/75
13/13 [==============================] - 0s 7ms/step - loss: 2.2076 - accuracy: 0.5188 - val_loss: 2.1266 - val_accuracy: 0.5875
Epoch 75/75
13/13 [==============================] - 0s 7ms/step - loss: 2.1532 - accuracy: 0.5356 - val_loss: 2.2686 - val_accuracy: 0.6475
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

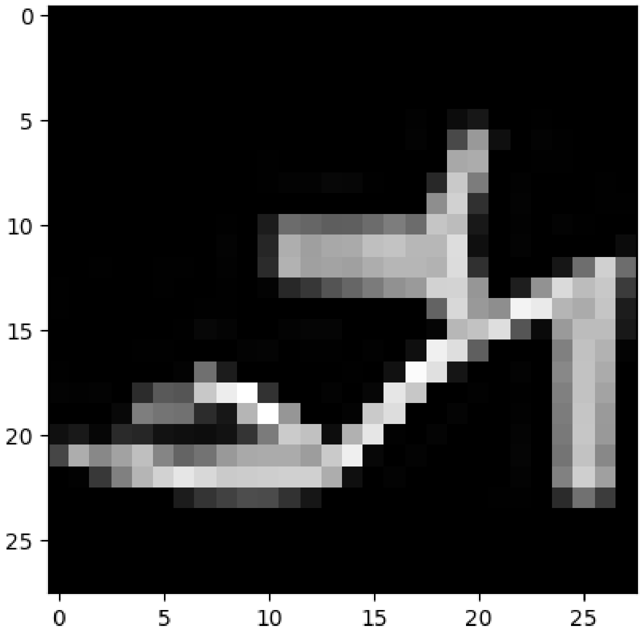



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 2.452636241912842
Test accuracy: 0.6238088011741638
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = train_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
1/1 [==============================] - 0s 51ms/step
array([[1.47238107e-11, 1.82209268e-25, 1.53627853e-23, 1.86553946e-15,
        1.00571526e-30, 9.99999881e-01, 2.51367218e-16, 5.36095533e-22,
        1.00579308e-13, 1.15462115e-07]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 75

Zbiór treningowy zwiększony z ***60000*** do **68000* (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.001)

kernel_regularizer=l2(0.01) we wszystkich warstwach

model.add(Dropout(0.4))

```
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_30 (Dense)            (None, 64)                50240     
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_31 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz

Model uczy się do około 30 epoki potem deliktanie się przeucza, wykresy błędu (treningowego i walidacyjnego) są podobne przy czym po 30 epoce błąd validacyjny rośnie dalej, a błąd treningowy spada. Dokładność modelu dla danych treningowych i walidacyjnych praktycznie cały czas rośnie.

Regularyzacja all in one #3

Zwiększamy zbiór treningowy z **60000** do **68000** (20% to zbiór walidacyjny)

```
test_data = data[:68000]
train_data = data[68000:]
test_labels = label[:68000]
train_labels = label[68000:]
```

```
print(test_data.shape,train_data.shape,test_labels.shape,train_labels.shape)
```

```
(68000, 28, 28) (2000, 28, 28) (68000,) (2000,)
```

One-hot coding

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
train_data.shape,train_labels.shape
```

```
((2000, 28, 28), (2000, 10))
```

```
test_data.shape,test_labels.shape
```

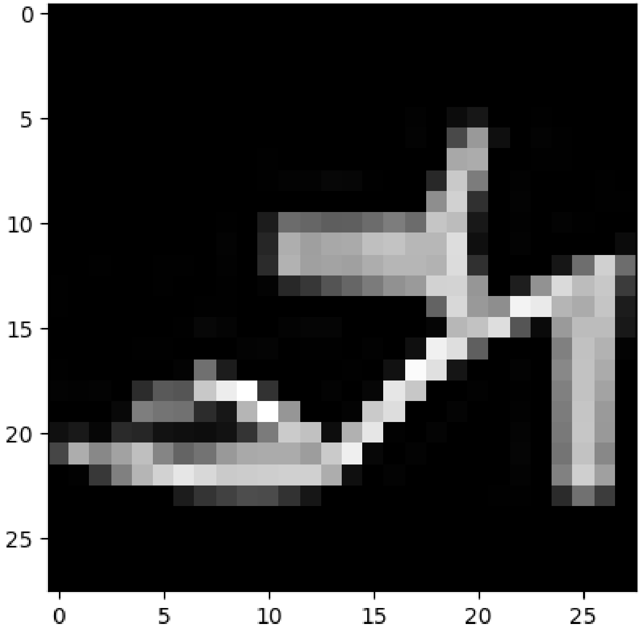
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```



```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))
```

Danie **regularyzacji L2** do warstw:

Adding dropout layer

Resizing model

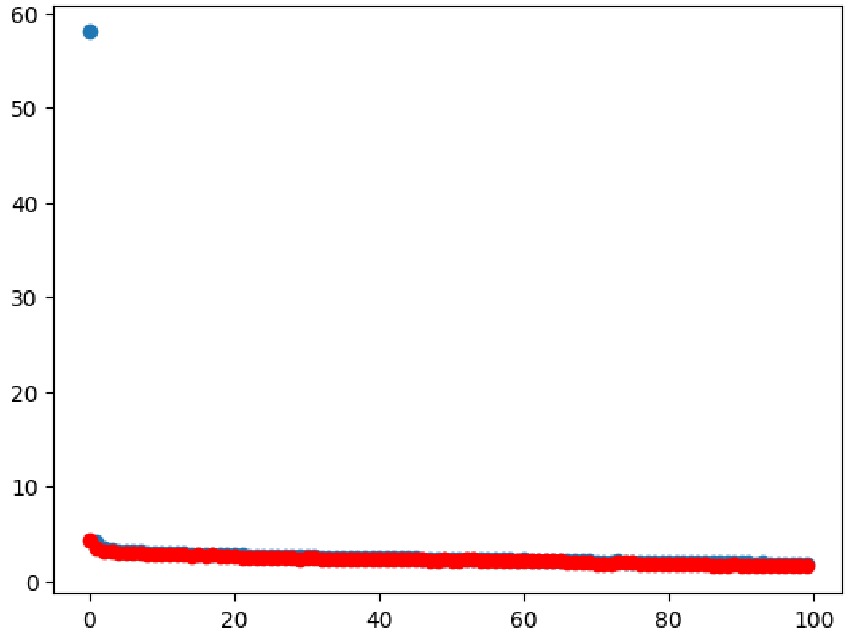
```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.3))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

opt = keras.optimizers.Adam(learning_rate=0.002)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 64)                50240

 dropout_4 (Dropout)         (None, 64)                0

 dense_33 (Dense)            (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

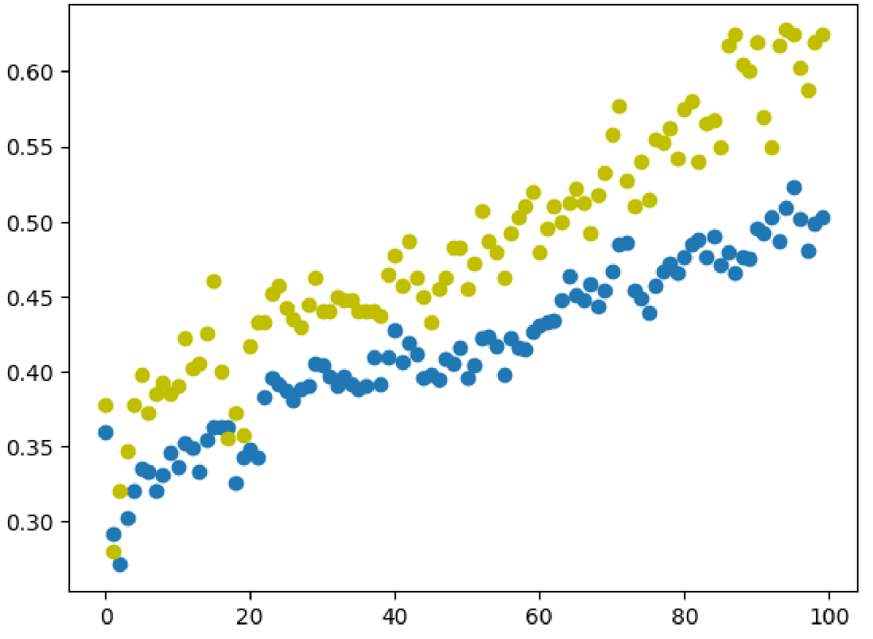
```
batch_size = 128
epochs = 100

h = model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
13/13 [==============================] - 0s 6ms/step - loss: 1.9532 - accuracy: 0.4656 - val_loss: 1.7691 - val_accuracy: 0.5425
Epoch 81/100
13/13 [==============================] - 0s 7ms/step - loss: 1.9512 - accuracy: 0.4769 - val_loss: 1.7186 - val_accuracy: 0.5750
Epoch 82/100
13/13 [==============================] - 0s 7ms/step - loss: 1.9237 - accuracy: 0.4850 - val_loss: 1.7482 - val_accuracy: 0.5800
Epoch 83/100
13/13 [==============================] - 0s 6ms/step - loss: 1.8639 - accuracy: 0.4881 - val_loss: 1.7825 - val_accuracy: 0.5400
Epoch 84/100
13/13 [==============================] - 0s 7ms/step - loss: 1.9181 - accuracy: 0.4769 - val_loss: 1.7186 - val_accuracy: 0.5650
Epoch 85/100
13/13 [==============================] - 0s 7ms/step - loss: 1.9303 - accuracy: 0.4906 - val_loss: 1.7062 - val_accuracy: 0.5675
Epoch 86/100
13/13 [==============================] - 0s 6ms/step - loss: 1.9321 - accuracy: 0.4712 - val_loss: 1.7237 - val_accuracy: 0.5500
Epoch 87/100
13/13 [==============================] - 0s 7ms/step - loss: 1.9058 - accuracy: 0.4800 - val_loss: 1.6659 - val_accuracy: 0.6175
Epoch 88/100
13/13 [==============================] - 0s 6ms/step - loss: 1.9044 - accuracy: 0.4663 - val_loss: 1.5687 - val_accuracy: 0.6250
Epoch 89/100
13/13 [==============================] - 0s 6ms/step - loss: 1.9442 - accuracy: 0.4769 - val_loss: 1.6303 - val_accuracy: 0.6050
Epoch 90/100
13/13 [==============================] - 0s 6ms/step - loss: 1.9094 - accuracy: 0.4756 - val_loss: 1.6937 - val_accuracy: 0.6000
Epoch 91/100
13/13 [==============================] - 0s 7ms/step - loss: 1.8696 - accuracy: 0.4950 - val_loss: 1.6352 - val_accuracy: 0.6200
Epoch 92/100
13/13 [==============================] - 0s 7ms/step - loss: 1.8633 - accuracy: 0.4925 - val_loss: 1.6215 - val_accuracy: 0.5700
Epoch 93/100
13/13 [==============================] - 0s 6ms/step - loss: 1.8501 - accuracy: 0.5031 - val_loss: 1.6575 - val_accuracy: 0.5500
Epoch 94/100
13/13 [==============================] - 0s 8ms/step - loss: 1.8602 - accuracy: 0.4875 - val_loss: 1.6051 - val_accuracy: 0.6175
Epoch 95/100
13/13 [==============================] - 0s 6ms/step - loss: 1.8319 - accuracy: 0.5088 - val_loss: 1.5763 - val_accuracy: 0.6275
Epoch 96/100
13/13 [==============================] - 0s 6ms/step - loss: 1.7818 - accuracy: 0.5231 - val_loss: 1.5249 - val_accuracy: 0.6250
Epoch 97/100
13/13 [==============================] - 0s 6ms/step - loss: 1.7886 - accuracy: 0.5019 - val_loss: 1.5737 - val_accuracy: 0.6025
Epoch 98/100
13/13 [==============================] - 0s 6ms/step - loss: 1.8208 - accuracy: 0.4806 - val_loss: 1.5991 - val_accuracy: 0.5875
Epoch 99/100
13/13 [==============================] - 0s 6ms/step - loss: 1.7905 - accuracy: 0.4988 - val_loss: 1.5389 - val_accuracy: 0.6200
Epoch 100/100
13/13 [==============================] - 0s 6ms/step - loss: 1.7847 - accuracy: 0.5031 - val_loss: 1.6213 - val_accuracy: 0.6250
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
plt.scatter(np.arange(epochs),h.history['accuracy'])
plt.scatter(np.arange(epochs),h.history['val_accuracy'],c='y')
plt.show()
```

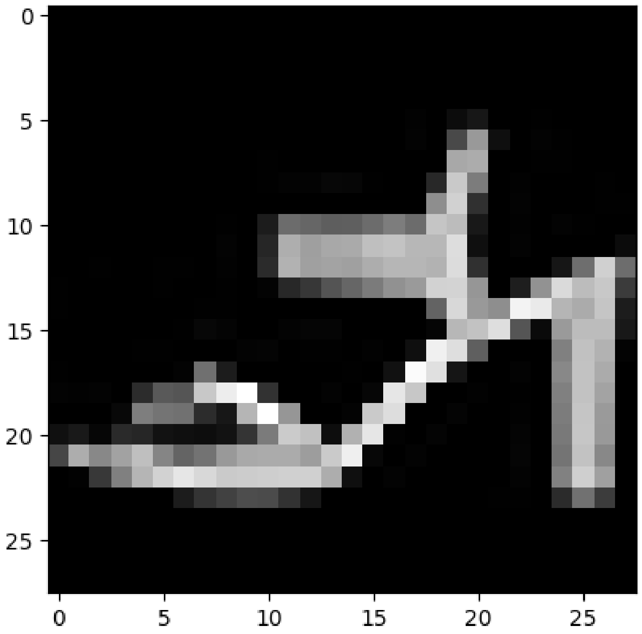



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 1.7682009935379028
Test accuracy: 0.5903382301330566
```

```
def plot_image(img_index):
    label_index = train_labels[img_index]
    plt.imshow(train_data[img_index]/255, cmap = 'gray')
    print(label_index)

img_index = 10
plot_image(img_index)

picture = train_data[img_index].reshape(-1,784)

model.predict(picture)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
1/1 [==============================] - 0s 50ms/step
array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 7.4471058e-27,
        8.8909479e-34, 3.1758628e-21]], dtype=float32)
```



Opis:

batch_size = 128

epochs = 100

Zbiór treningowy zwiększony z ***60000*** do **68000* (20% to zbiór walidacyjny)

opt = keras.optimizers.Adam(learning_rate=0.002)

kernel_regularizer=l2(0.01) we wszystkich warstwach

model.add(Dropout(0.3))

```
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 64)                50240

 dropout_4 (Dropout)         (None, 64)                0

 dense_33 (Dense)            (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Wnioski i komentarz