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.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]
```

```
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   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, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
          195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
         [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
           11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
         [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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,   0,   0,
            0,   0]], dtype=uint8)
```

```
train_labels[0]
```

```
5
```

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))
```

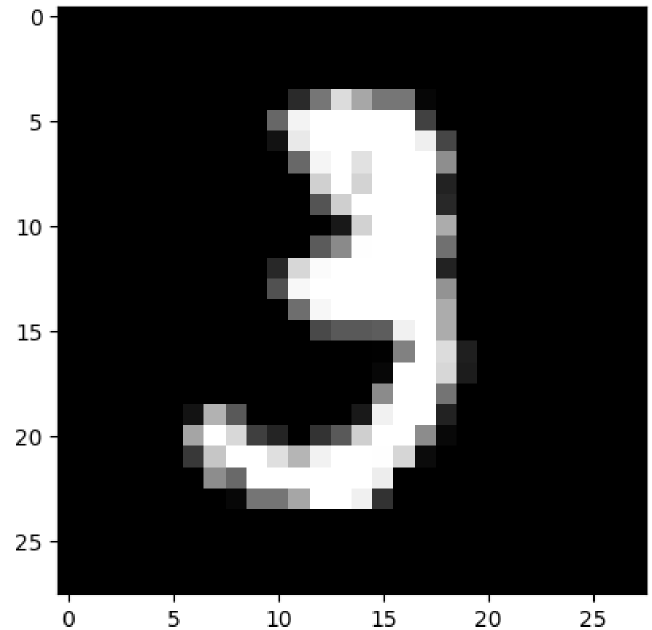
```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], 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)
```

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



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

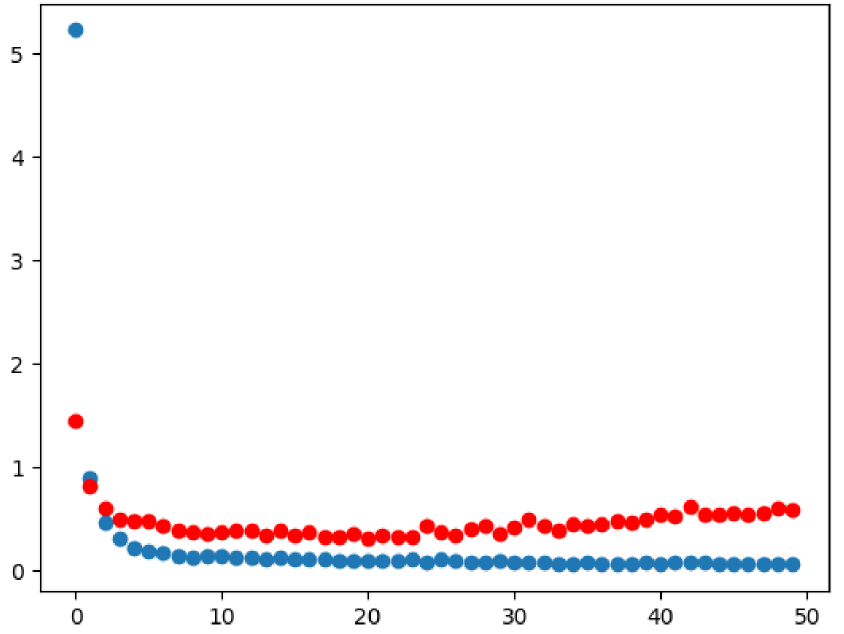
```
model = Sequential()
model.add(Dense(units = 128, use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dense(units = 10, use_bias=True, activation = "softmax"))

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_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_64 (Dense)            (None, 128)               100480    
                                                                 
 dense_65 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

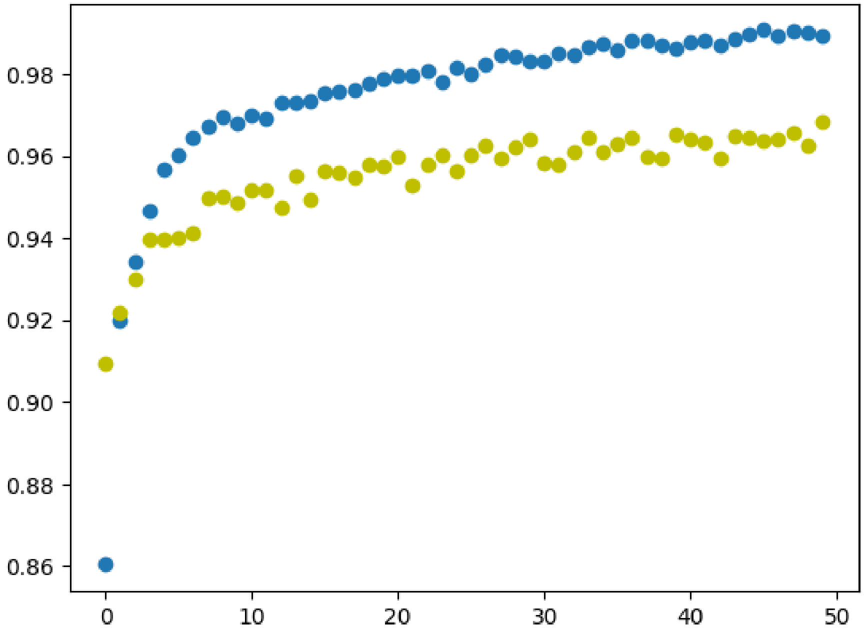
```
batch_size = 128
epochs = 50

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

```
375/375 [==============================] - 1s 3ms/step - loss: 0.0553 - accuracy: 0.9881 - val_loss: 0.4595 - val_accuracy: 0.9645
Epoch 38/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0591 - accuracy: 0.9882 - val_loss: 0.4802 - val_accuracy: 0.9597
Epoch 39/50
375/375 [==============================] - 2s 4ms/step - loss: 0.0621 - accuracy: 0.9870 - val_loss: 0.4626 - val_accuracy: 0.9593
Epoch 40/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0737 - accuracy: 0.9864 - val_loss: 0.4854 - val_accuracy: 0.9652
Epoch 41/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0622 - accuracy: 0.9877 - val_loss: 0.5400 - val_accuracy: 0.9642
Epoch 42/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0653 - accuracy: 0.9882 - val_loss: 0.5235 - val_accuracy: 0.9634
Epoch 43/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0747 - accuracy: 0.9869 - val_loss: 0.6119 - val_accuracy: 0.9593
Epoch 44/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0680 - accuracy: 0.9884 - val_loss: 0.5293 - val_accuracy: 0.9647
Epoch 45/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0534 - accuracy: 0.9896 - val_loss: 0.5393 - val_accuracy: 0.9647
Epoch 46/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0500 - accuracy: 0.9907 - val_loss: 0.5479 - val_accuracy: 0.9638
Epoch 47/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0609 - accuracy: 0.9895 - val_loss: 0.5302 - val_accuracy: 0.9641
Epoch 48/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0524 - accuracy: 0.9903 - val_loss: 0.5504 - val_accuracy: 0.9657
Epoch 49/50
375/375 [==============================] - 2s 5ms/step - loss: 0.0567 - accuracy: 0.9901 - val_loss: 0.5891 - val_accuracy: 0.9628
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0614 - accuracy: 0.9893 - val_loss: 0.5885 - val_accuracy: 0.9682
```

```
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()
```

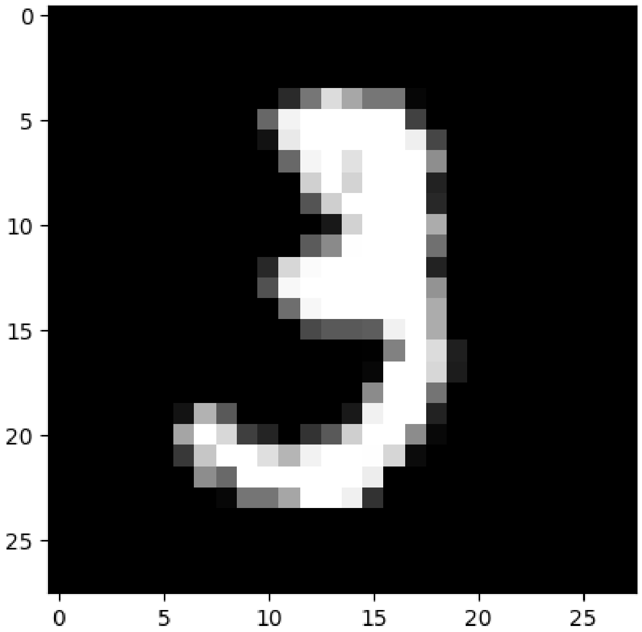



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.6038411259651184
Test accuracy: 0.96670001745224
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 41ms/step
array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
        0.0000000e+00, 2.1509282e-23, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 4.6604197e-13]], 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.mnist.load_data()
```


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

```
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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,   0,   0,
          0,   0]], dtype=uint8)
```

```
train_labels[0]
```

```
5
```

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))
```

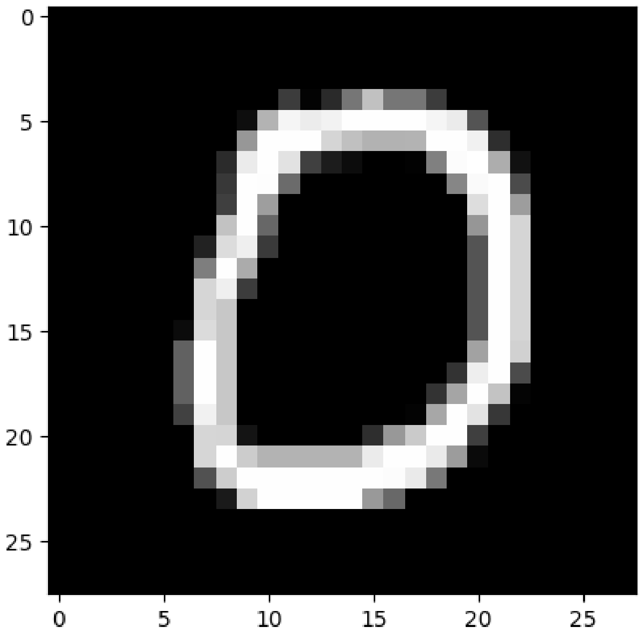
```
train_labels[0]
```

```
array([0., 0., 0., 0., 0., 1., 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()
```

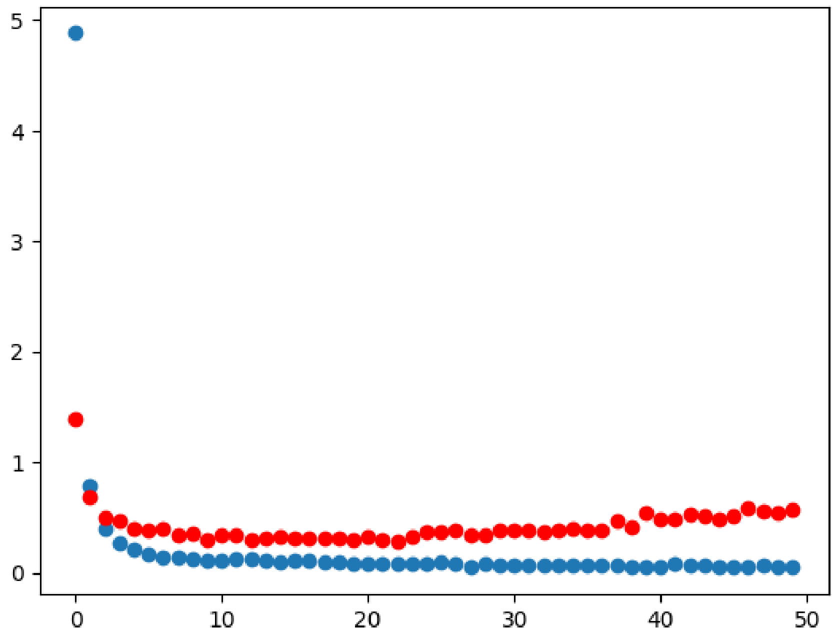
```
Model: "sequential_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_66 (Dense)            (None, 128)               100480

 dense_67 (Dense)            (None, 10)                1290

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

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

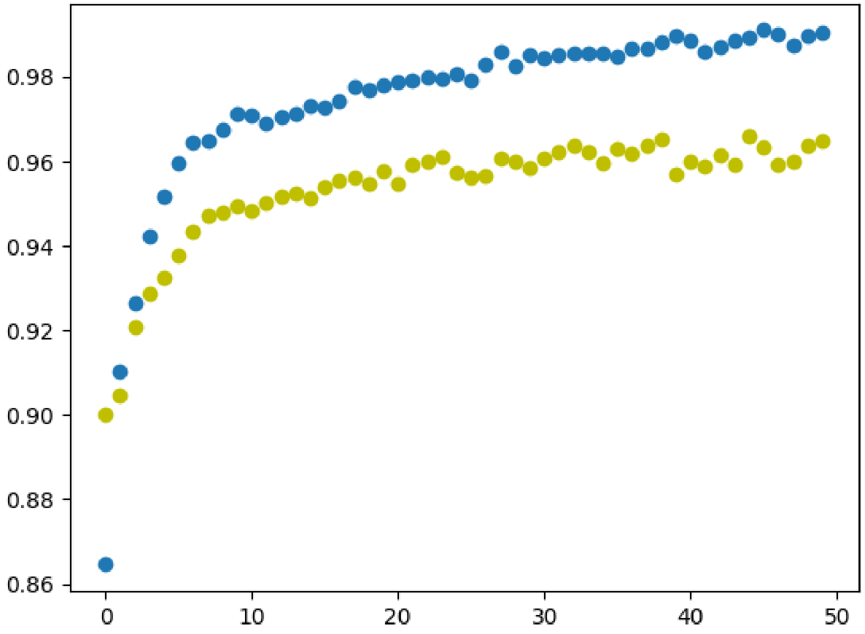
```
batch_size = 128
epochs = 50

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

```
375/375 [==============================] - 1s 3ms/step - loss: 0.0759 - accuracy: 0.9860 - val_loss: 0.4791 - val_accuracy: 0.9587
Epoch 43/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0647 - accuracy: 0.9871 - val_loss: 0.5261 - val_accuracy: 0.9613
Epoch 44/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0638 - accuracy: 0.9885 - val_loss: 0.5135 - val_accuracy: 0.9592
Epoch 45/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0555 - accuracy: 0.9894 - val_loss: 0.4800 - val_accuracy: 0.9658
Epoch 46/50
375/375 [==============================] - 2s 4ms/step - loss: 0.0443 - accuracy: 0.9910 - val_loss: 0.5050 - val_accuracy: 0.9633
Epoch 47/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0451 - accuracy: 0.9901 - val_loss: 0.5835 - val_accuracy: 0.9592
Epoch 48/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0705 - accuracy: 0.9875 - val_loss: 0.5480 - val_accuracy: 0.9598
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0538 - accuracy: 0.9897 - val_loss: 0.5340 - val_accuracy: 0.9636
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0536 - accuracy: 0.9904 - val_loss: 0.5692 - val_accuracy: 0.9649
```

```
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()
```

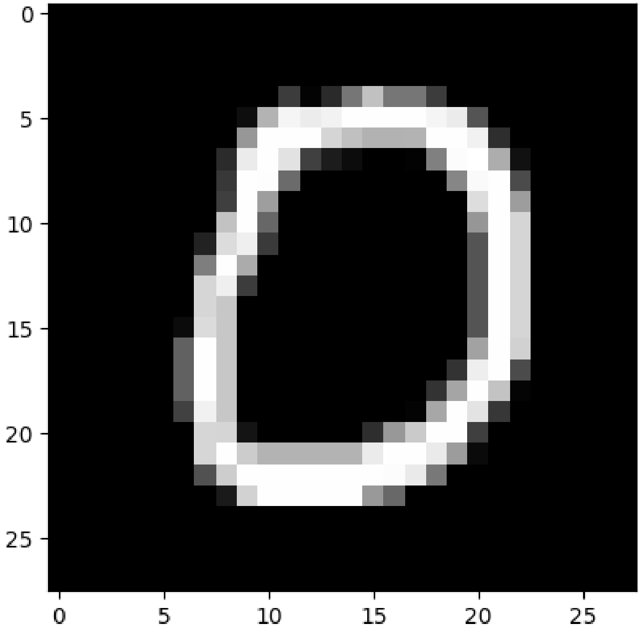



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.506654679775238
Test accuracy: 0.9695000052452087
```

```
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 45ms/step
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], 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))
```

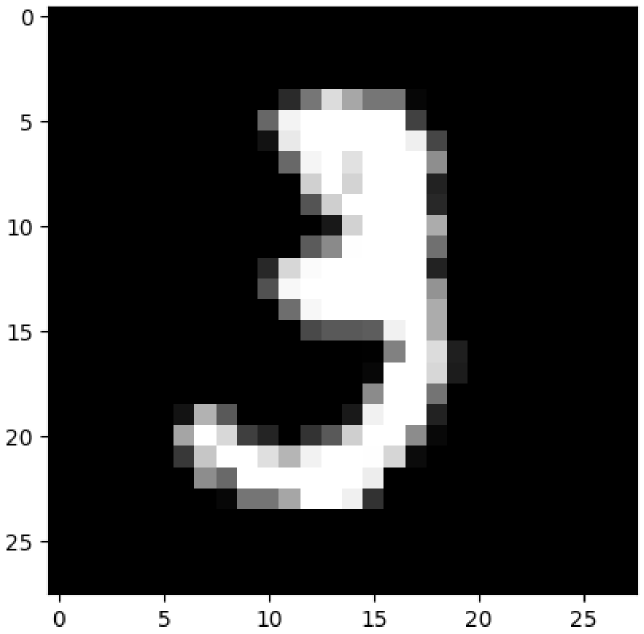
```
train_labels[0]
```

```
array([0., 0., 0., 1., 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)
```

```
[0. 0. 0. 1. 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()
```

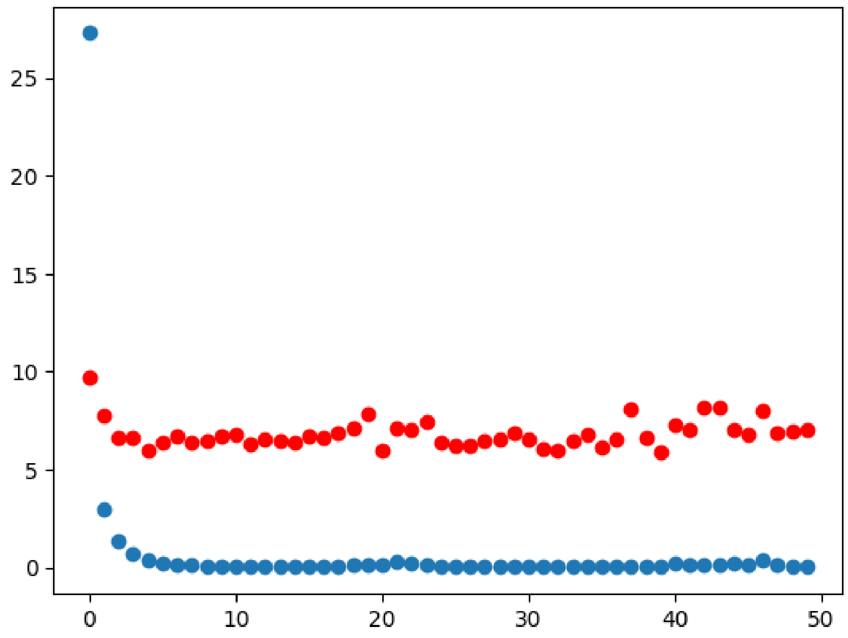
```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_68 (Dense)            (None, 128)               100480

 dense_69 (Dense)            (None, 10)                1290

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

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

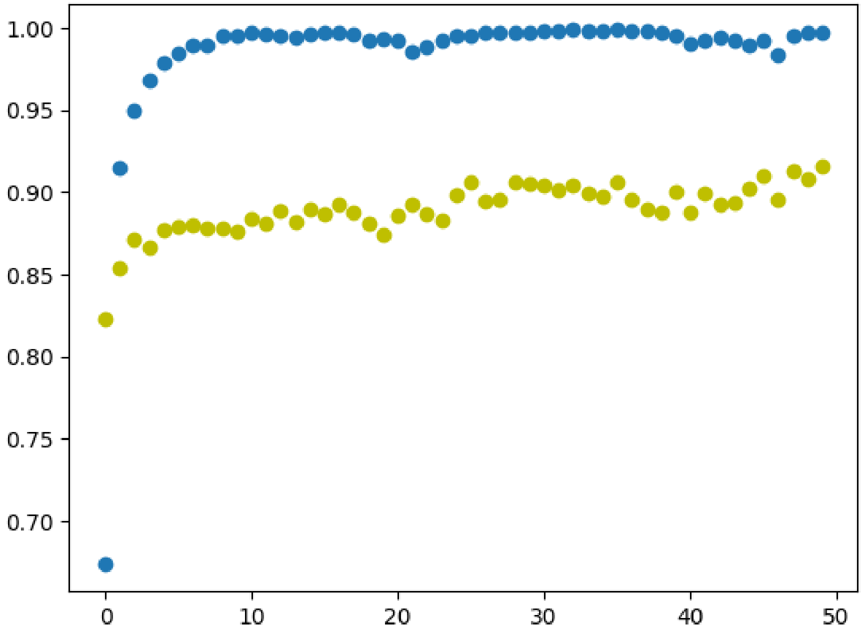
```
batch_size = 128
epochs = 50

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

```
32/32 [==============================] - 0s 6ms/step - loss: 0.1321 - accuracy: 0.9923 - val_loss: 7.0386 - val_accuracy: 0.8990
Epoch 43/50
32/32 [==============================] - 0s 4ms/step - loss: 0.0903 - accuracy: 0.9942 - val_loss: 8.1261 - val_accuracy: 0.8920
Epoch 44/50
32/32 [==============================] - 0s 3ms/step - loss: 0.1175 - accuracy: 0.9923 - val_loss: 8.1469 - val_accuracy: 0.8930
Epoch 45/50
32/32 [==============================] - 0s 3ms/step - loss: 0.2243 - accuracy: 0.9895 - val_loss: 6.9922 - val_accuracy: 0.9020
Epoch 46/50
32/32 [==============================] - 0s 4ms/step - loss: 0.1407 - accuracy: 0.9918 - val_loss: 6.7552 - val_accuracy: 0.9100
Epoch 47/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4111 - accuracy: 0.9835 - val_loss: 7.9961 - val_accuracy: 0.8950
Epoch 48/50
32/32 [==============================] - 0s 4ms/step - loss: 0.1410 - accuracy: 0.9952 - val_loss: 6.8934 - val_accuracy: 0.9130
Epoch 49/50
32/32 [==============================] - 0s 4ms/step - loss: 0.0619 - accuracy: 0.9973 - val_loss: 6.9808 - val_accuracy: 0.9080
Epoch 50/50
32/32 [==============================] - 0s 4ms/step - loss: 0.0243 - accuracy: 0.9970 - val_loss: 6.9950 - val_accuracy: 0.9160
```

```
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()
```

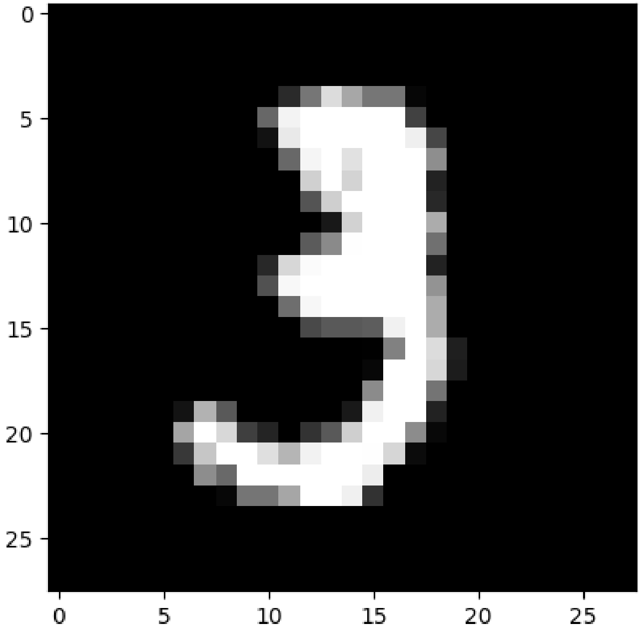



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 8.388226509094238
Test accuracy: 0.883384644985199
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 34ms/step
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]], 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))
```

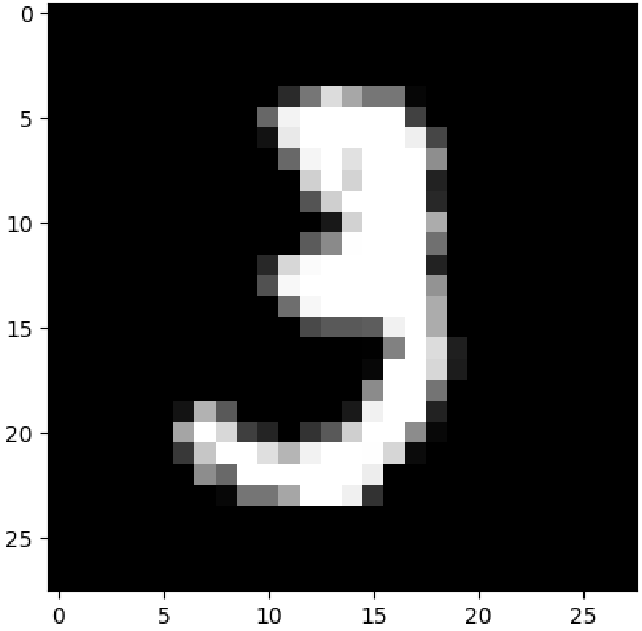
```
train_labels[0]
```

```
array([0., 0., 0., 0., 1., 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)
```

```
[0. 0. 0. 1. 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()
```

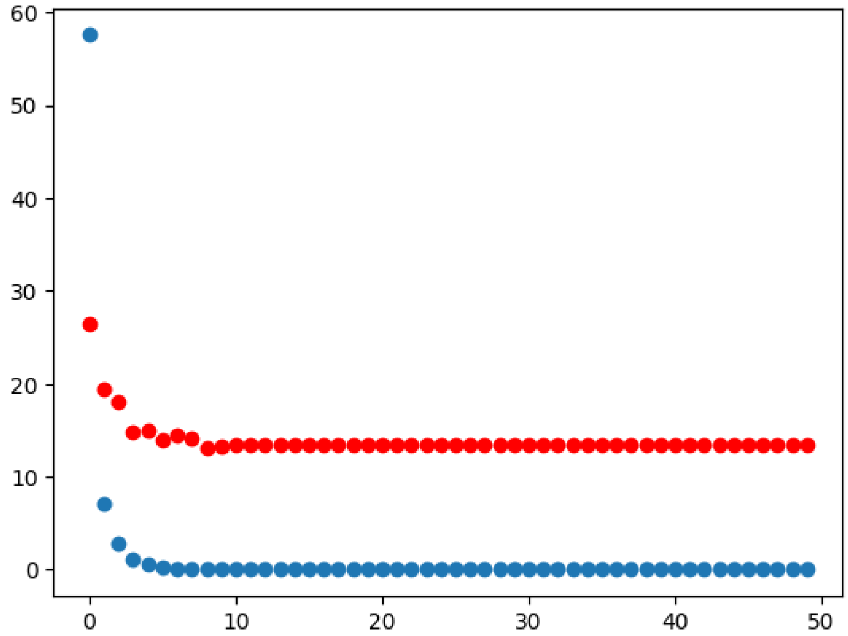
```
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_70 (Dense)            (None, 128)               100480

 dense_71 (Dense)            (None, 10)                1290

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

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

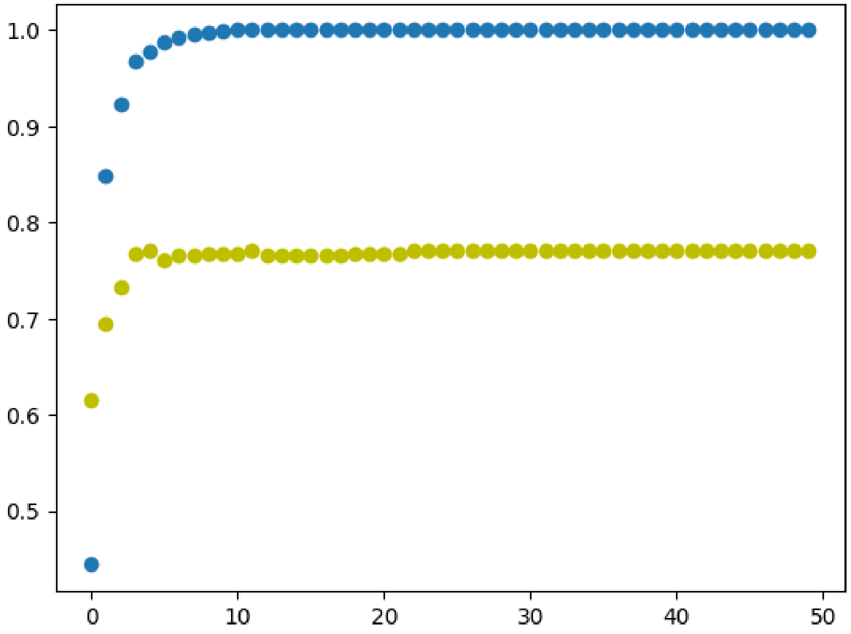
```
batch_size = 128
epochs = 50

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

```
13/13 [==============================] - 0s 5ms/step - loss: 7.0826e-07 - accuracy: 1.0000 - val_loss: 13.4626 - val_accuracy: 0.7700
Epoch 43/50
13/13 [==============================] - 0s 6ms/step - loss: 6.7333e-07 - accuracy: 1.0000 - val_loss: 13.4622 - val_accuracy: 0.7700
Epoch 44/50
13/13 [==============================] - 0s 5ms/step - loss: 6.5470e-07 - accuracy: 1.0000 - val_loss: 13.4616 - val_accuracy: 0.7700
Epoch 45/50
13/13 [==============================] - 0s 6ms/step - loss: 6.3221e-07 - accuracy: 1.0000 - val_loss: 13.4610 - val_accuracy: 0.7700
Epoch 46/50
13/13 [==============================] - 0s 6ms/step - loss: 6.1343e-07 - accuracy: 1.0000 - val_loss: 13.4605 - val_accuracy: 0.7700
Epoch 47/50
13/13 [==============================] - 0s 5ms/step - loss: 5.9690e-07 - accuracy: 1.0000 - val_loss: 13.4600 - val_accuracy: 0.7700
Epoch 48/50
13/13 [==============================] - 0s 5ms/step - loss: 5.7671e-07 - accuracy: 1.0000 - val_loss: 13.4596 - val_accuracy: 0.7700
Epoch 49/50
13/13 [==============================] - 0s 6ms/step - loss: 5.6173e-07 - accuracy: 1.0000 - val_loss: 13.4591 - val_accuracy: 0.7700
Epoch 50/50
13/13 [==============================] - 0s 5ms/step - loss: 5.4482e-07 - accuracy: 1.0000 - val_loss: 13.4586 - val_accuracy: 0.7700
```

```
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()
```

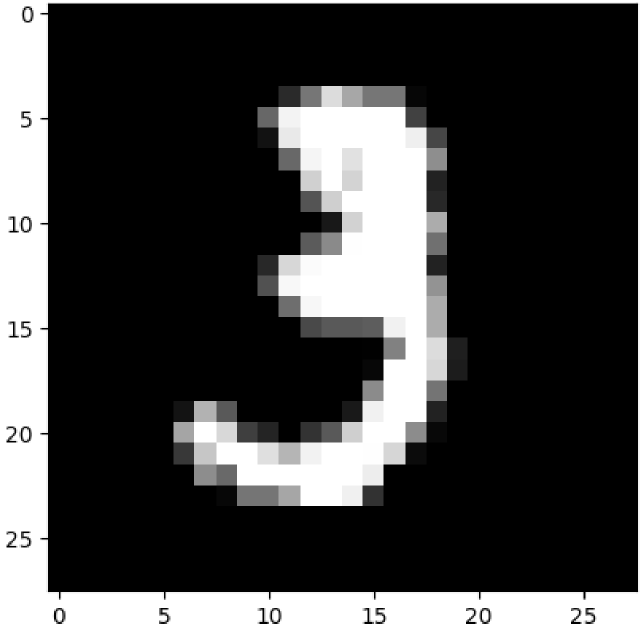



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 10.958120346069336
Test accuracy: 0.8068529367446899
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 38ms/step
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]], 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_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_70 (Dense)            (None, 128)               100480    
                                                                 
 dense_71 (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.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., 1., 0., 0., 0., 0.], dtype=float32)
```

Regularyzacja - metoda 2

Zmniejszamy **wielkość modelu**:

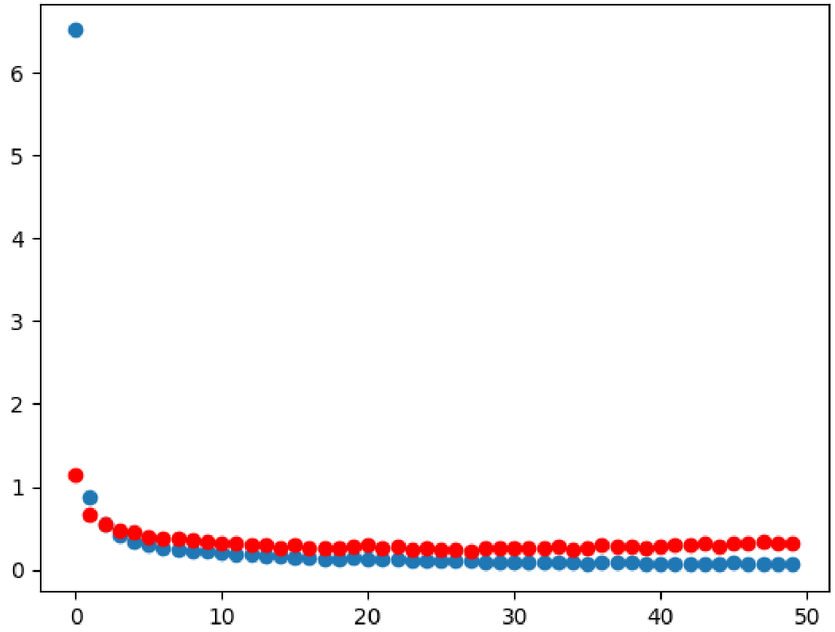
```
train_images = train_data.reshape((-1, 784))
test_images = test_data.reshape((-1, 784))


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_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_72 (Dense)            (None, 64)                50240

 dense_73 (Dense)            (None, 10)                650

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

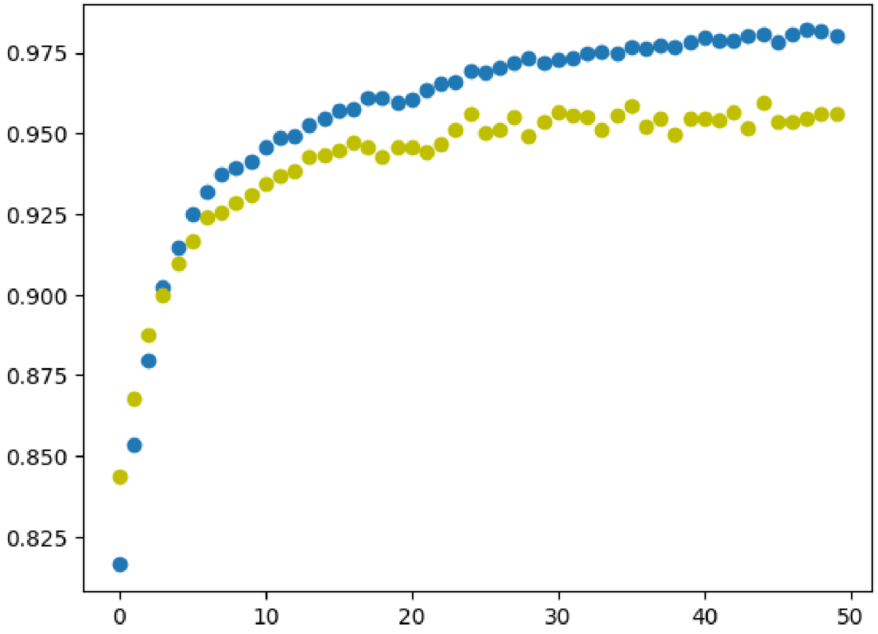
```
batch_size = 128
epochs = 50

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

```
Epoch 29/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0920 - accuracy: 0.9735 - val_loss: 0.2624 - val_accuracy: 0.9492
Epoch 30/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0940 - accuracy: 0.9720 - val_loss: 0.2652 - val_accuracy: 0.9536
Epoch 31/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0891 - accuracy: 0.9730 - val_loss: 0.2602 - val_accuracy: 0.9567
Epoch 32/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0921 - accuracy: 0.9731 - val_loss: 0.2584 - val_accuracy: 0.9556
Epoch 33/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0888 - accuracy: 0.9746 - val_loss: 0.2608 - val_accuracy: 0.9550
Epoch 34/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0856 - accuracy: 0.9751 - val_loss: 0.2730 - val_accuracy: 0.9512
Epoch 35/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0818 - accuracy: 0.9746 - val_loss: 0.2396 - val_accuracy: 0.9555
Epoch 36/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0767 - accuracy: 0.9767 - val_loss: 0.2671 - val_accuracy: 0.9588
Epoch 37/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0809 - accuracy: 0.9761 - val_loss: 0.2979 - val_accuracy: 0.9519
Epoch 38/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0790 - accuracy: 0.9773 - val_loss: 0.2740 - val_accuracy: 0.9547
Epoch 39/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0811 - accuracy: 0.9769 - val_loss: 0.2847 - val_accuracy: 0.9498
Epoch 40/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0733 - accuracy: 0.9783 - val_loss: 0.2700 - val_accuracy: 0.9544
Epoch 41/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0670 - accuracy: 0.9797 - val_loss: 0.2873 - val_accuracy: 0.9545
Epoch 42/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0727 - accuracy: 0.9787 - val_loss: 0.3045 - val_accuracy: 0.9543
Epoch 43/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0731 - accuracy: 0.9787 - val_loss: 0.2954 - val_accuracy: 0.9567
Epoch 44/50
375/375 [==============================] - 1s 3ms/step - loss: 0.0662 - accuracy: 0.9802 - val_loss: 0.3271 - val_accuracy: 0.9515
Epoch 45/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0674 - accuracy: 0.9806 - val_loss: 0.2850 - val_accuracy: 0.9597
Epoch 46/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0802 - accuracy: 0.9785 - val_loss: 0.3203 - val_accuracy: 0.9538
Epoch 47/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0671 - accuracy: 0.9808 - val_loss: 0.3123 - val_accuracy: 0.9538
Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0599 - accuracy: 0.9820 - val_loss: 0.3301 - val_accuracy: 0.9546
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0634 - accuracy: 0.9816 - val_loss: 0.3227 - val_accuracy: 0.9563
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0724 - accuracy: 0.9804 - val_loss: 0.3091 - val_accuracy: 0.9558
```

```
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()
```

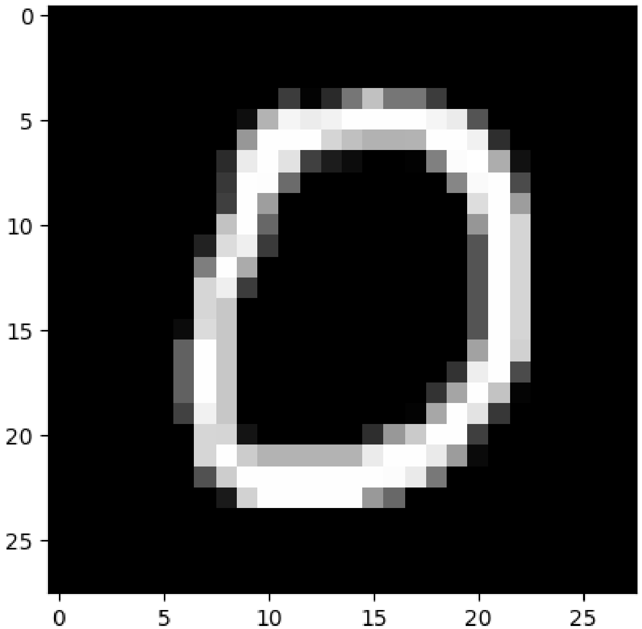



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.3401225507259369
Test accuracy: 0.9570000171661377
```

```
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 53ms/step
array([[1.0000000e+00, 0.0000000e+00, 2.7850517e-09, 3.2065512e-29,
        8.2517981e-16, 9.5437488e-22, 5.3893983e-21, 5.8958713e-18,
        1.5102388e-26, 2.4283206e-14]], 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_31"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 dense_72 (Dense)            (None, 64)                50240

 dense_73 (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:

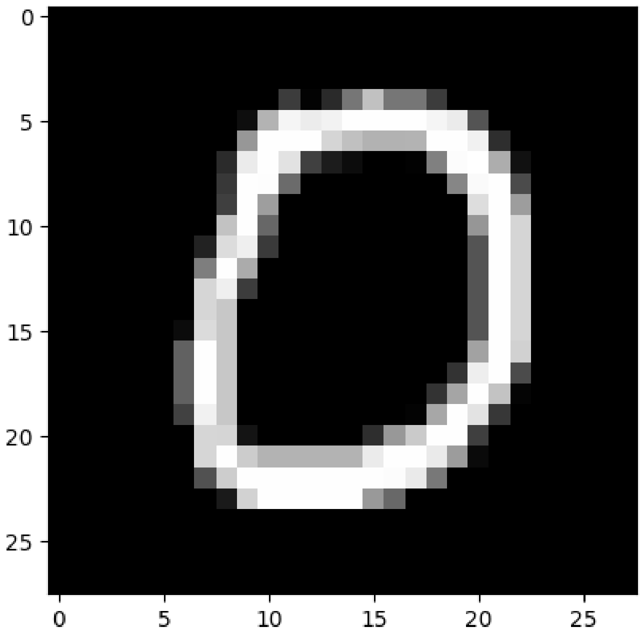
```
from keras.regularizers import l2
```

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)
```

```
[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,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()
```

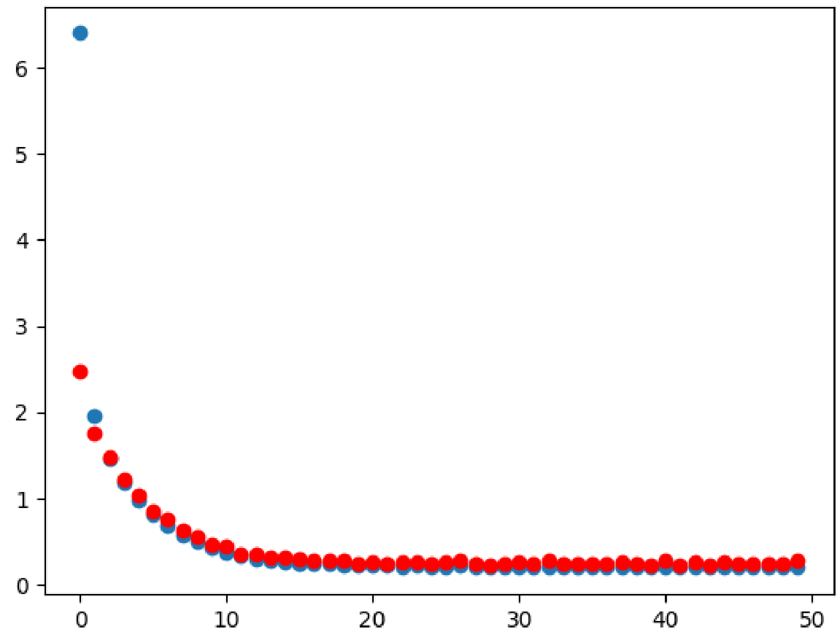
```
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_74 (Dense)            (None, 128)               100480

 dense_75 (Dense)            (None, 10)                1290

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

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

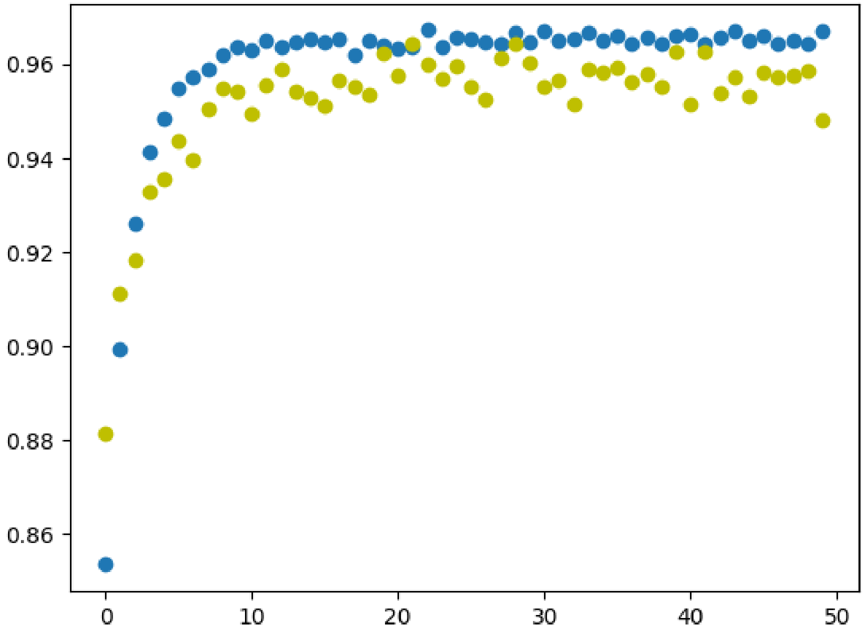
```
batch_size = 128
epochs = 50

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

```
375/375 [==============================] - 1s 3ms/step - loss: 0.2044 - accuracy: 0.9643 - val_loss: 0.2244 - val_accuracy: 0.9627
Epoch 43/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1962 - accuracy: 0.9655 - val_loss: 0.2591 - val_accuracy: 0.9538
Epoch 44/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1911 - accuracy: 0.9671 - val_loss: 0.2264 - val_accuracy: 0.9573
Epoch 45/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1958 - accuracy: 0.9650 - val_loss: 0.2562 - val_accuracy: 0.9530
Epoch 46/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1937 - accuracy: 0.9660 - val_loss: 0.2417 - val_accuracy: 0.9583
Epoch 47/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1994 - accuracy: 0.9643 - val_loss: 0.2376 - val_accuracy: 0.9571
Epoch 48/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1977 - accuracy: 0.9649 - val_loss: 0.2392 - val_accuracy: 0.9575
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1977 - accuracy: 0.9644 - val_loss: 0.2344 - val_accuracy: 0.9587
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1907 - accuracy: 0.9669 - val_loss: 0.2727 - val_accuracy: 0.9481
```

```
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()
```

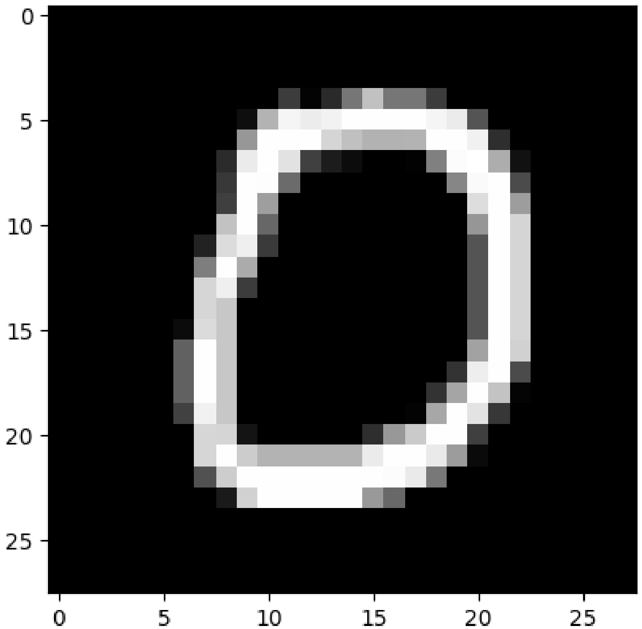



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.2521417737007141
Test accuracy: 0.9491999745368958
```

```
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 34ms/step
array([[9.9994195e-01, 1.3246866e-09, 5.5768312e-05, 1.8196541e-09,
        4.0701824e-11, 4.8784417e-09, 6.4530008e-07, 4.2877174e-11,
        1.2116991e-11, 1.7269718e-06]], 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_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_74 (Dense)            (None, 128)               100480

 dense_75 (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.

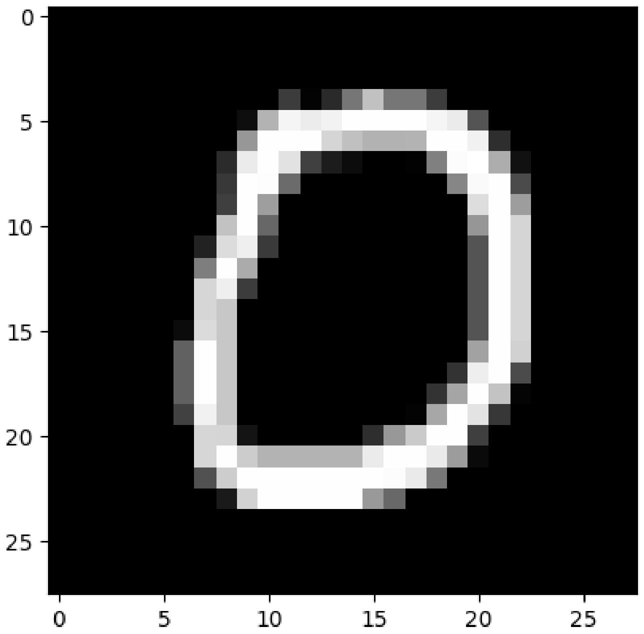
Regularyzacja - metoda 4

```
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)
```

```
[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(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()
```

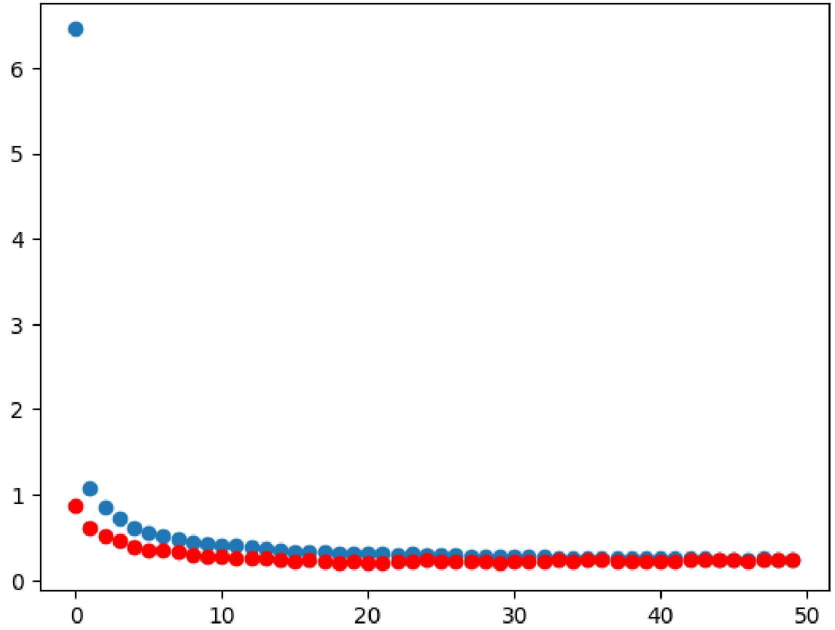
```
Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_76 (Dense)            (None, 128)               100480

 dropout_26 (Dropout)        (None, 128)               0

 dense_77 (Dense)            (None, 10)                1290
```

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

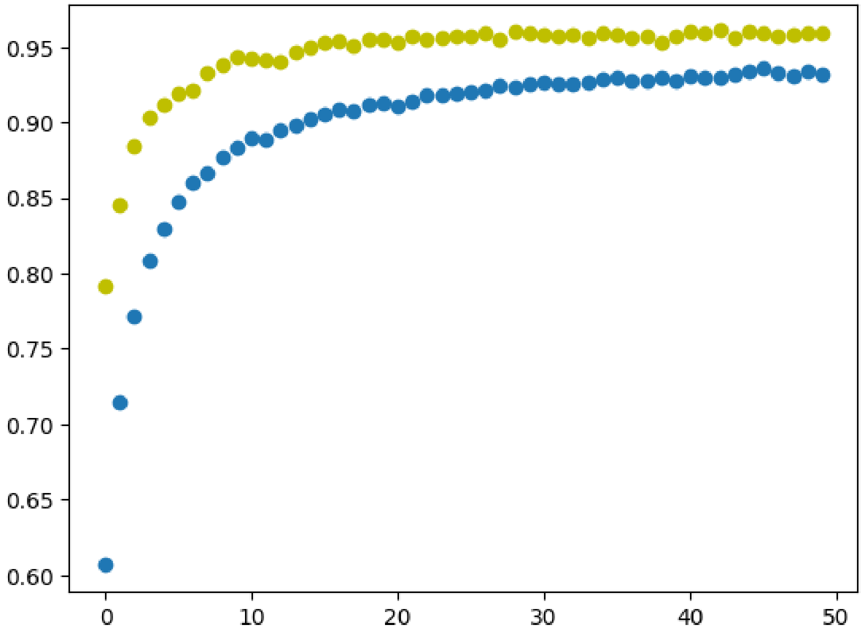
```
batch_size = 128
epochs = 50

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

```
Epoch 41/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2535 - accuracy: 0.9305 - val_loss: 0.2204 - val_accuracy: 0.9604
Epoch 42/50
375/375 [==============================] - 1s 4ms/step - loss: 0.2472 - accuracy: 0.9301 - val_loss: 0.2253 - val_accuracy: 0.9596
Epoch 43/50
375/375 [==============================] - 2s 4ms/step - loss: 0.2596 - accuracy: 0.9293 - val_loss: 0.2277 - val_accuracy: 0.9609
Epoch 44/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2465 - accuracy: 0.9319 - val_loss: 0.2385 - val_accuracy: 0.9567
Epoch 45/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2457 - accuracy: 0.9338 - val_loss: 0.2307 - val_accuracy: 0.9604
Epoch 46/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2329 - accuracy: 0.9363 - val_loss: 0.2398 - val_accuracy: 0.9590
Epoch 47/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2442 - accuracy: 0.9329 - val_loss: 0.2246 - val_accuracy: 0.9577
Epoch 48/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2467 - accuracy: 0.9313 - val_loss: 0.2394 - val_accuracy: 0.9580
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2416 - accuracy: 0.9339 - val_loss: 0.2363 - val_accuracy: 0.9593
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.2412 - accuracy: 0.9322 - val_loss: 0.2334 - val_accuracy: 0.9592
```

```
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()
```

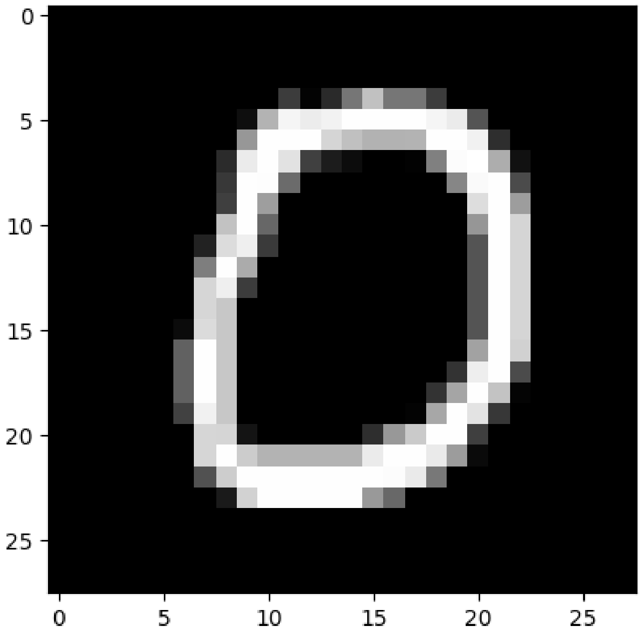


```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.2699497938156128
Test accuracy: 0.9580000042915344
```

```
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 37ms/step
array([[9.9996579e-01, 1.7283757e-34, 2.5498052e-06, 1.1869583e-16,
        3.2256109e-10, 3.5288206e-11, 3.0627398e-05, 8.4430391e-16,
        1.1219787e-14, 1.0727665e-06]], 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_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 dense_76 (Dense)            (None, 128)               100480

 dropout_26 (Dropout)        (None, 128)               0

 dense_77 (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
```

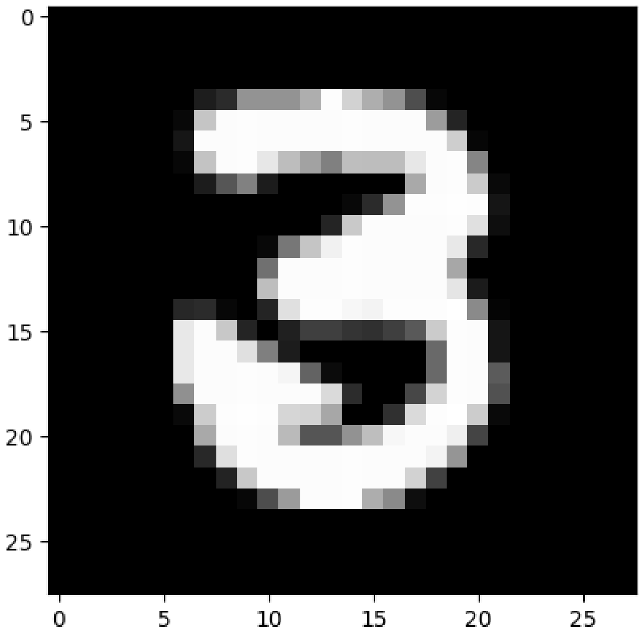
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 1., 0., 0., 0., 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. 1. 0. 0. 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

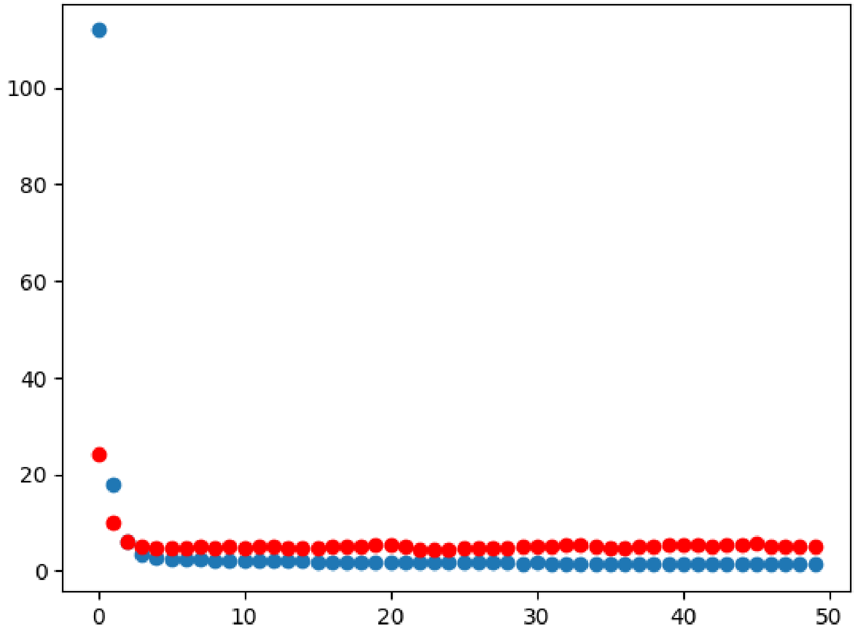
```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

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_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_78 (Dense)            (None, 64)                50240

 dropout_27 (Dropout)        (None, 64)                0

 dense_79 (Dense)            (None, 10)                650

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

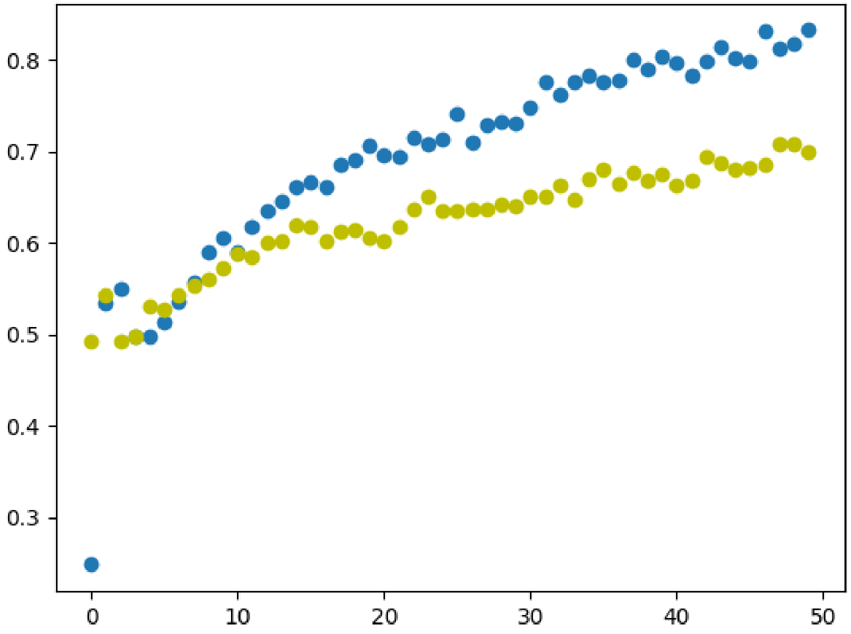
```
batch_size = 128
epochs = 50

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

```
13/13 [==============================] - 0s 5ms/step - loss: 1.6080 - accuracy: 0.7312 - val_loss: 4.9985 - val_accuracy: 0.6400
Epoch 31/50
13/13 [==============================] - 0s 5ms/step - loss: 1.6160 - accuracy: 0.7475 - val_loss: 5.1645 - val_accuracy: 0.6500
Epoch 32/50
13/13 [==============================] - 0s 5ms/step - loss: 1.5158 - accuracy: 0.7763 - val_loss: 5.2210 - val_accuracy: 0.6500
Epoch 33/50
13/13 [==============================] - 0s 5ms/step - loss: 1.5161 - accuracy: 0.7625 - val_loss: 5.2337 - val_accuracy: 0.6625
Epoch 34/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4906 - accuracy: 0.7763 - val_loss: 5.3286 - val_accuracy: 0.6475
Epoch 35/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4734 - accuracy: 0.7831 - val_loss: 5.1145 - val_accuracy: 0.6700
Epoch 36/50
13/13 [==============================] - 0s 4ms/step - loss: 1.4990 - accuracy: 0.7756 - val_loss: 4.7616 - val_accuracy: 0.6800
Epoch 37/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4810 - accuracy: 0.7775 - val_loss: 4.8401 - val_accuracy: 0.6650
Epoch 38/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4053 - accuracy: 0.8006 - val_loss: 4.9413 - val_accuracy: 0.6775
Epoch 39/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4415 - accuracy: 0.7906 - val_loss: 5.1680 - val_accuracy: 0.6675
Epoch 40/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4229 - accuracy: 0.8044 - val_loss: 5.4715 - val_accuracy: 0.6750
Epoch 41/50
13/13 [==============================] - 0s 5ms/step - loss: 1.4507 - accuracy: 0.7962 - val_loss: 5.2357 - val_accuracy: 0.6625
Epoch 42/50
13/13 [==============================] - 0s 4ms/step - loss: 1.4447 - accuracy: 0.7837 - val_loss: 5.3088 - val_accuracy: 0.6675
Epoch 43/50
13/13 [==============================] - 0s 5ms/step - loss: 1.3620 - accuracy: 0.7987 - val_loss: 5.1936 - val_accuracy: 0.6950
Epoch 44/50
13/13 [==============================] - 0s 5ms/step - loss: 1.3751 - accuracy: 0.8144 - val_loss: 5.2944 - val_accuracy: 0.6875
Epoch 45/50
13/13 [==============================] - 0s 5ms/step - loss: 1.3511 - accuracy: 0.8019 - val_loss: 5.5340 - val_accuracy: 0.6800
Epoch 46/50
13/13 [==============================] - 0s 4ms/step - loss: 1.3905 - accuracy: 0.7994 - val_loss: 5.6892 - val_accuracy: 0.6825
Epoch 47/50
13/13 [==============================] - 0s 4ms/step - loss: 1.3428 - accuracy: 0.8319 - val_loss: 5.1355 - val_accuracy: 0.6850
Epoch 48/50
13/13 [==============================] - 0s 4ms/step - loss: 1.3342 - accuracy: 0.8119 - val_loss: 5.1278 - val_accuracy: 0.7075
Epoch 49/50
13/13 [==============================] - 0s 4ms/step - loss: 1.3597 - accuracy: 0.8181 - val_loss: 4.9230 - val_accuracy: 0.7075
Epoch 50/50
13/13 [==============================] - 0s 4ms/step - loss: 1.2901 - accuracy: 0.8325 - val_loss: 4.9970 - val_accuracy: 0.7000
```

```
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()
```

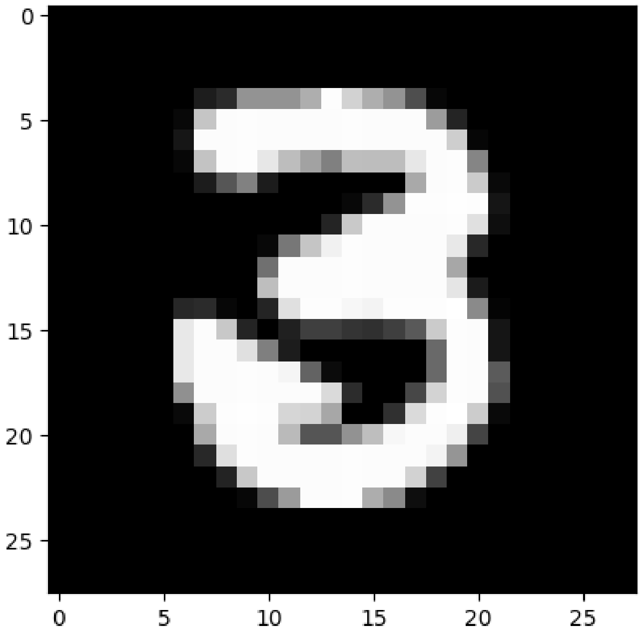



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 4.359646797180176
Test accuracy: 0.747735321521759
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 43ms/step
array([[1.13327535e-10, 2.14035153e-05, 3.56254607e-01, 2.41581634e-01,
        4.04927307e-11, 8.10949225e-03, 1.22830599e-07, 4.47372006e-09,
        3.94032657e-01, 9.93330289e-08]], 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_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_78 (Dense)            (None, 64)                50240

 dropout_27 (Dropout)        (None, 64)                0

 dense_79 (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
```

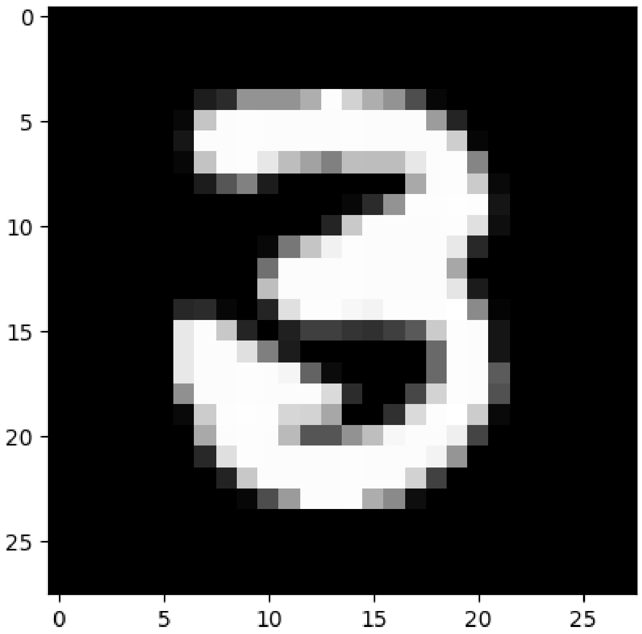
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 1., 0., 0., 0., 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. 1. 0. 0. 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

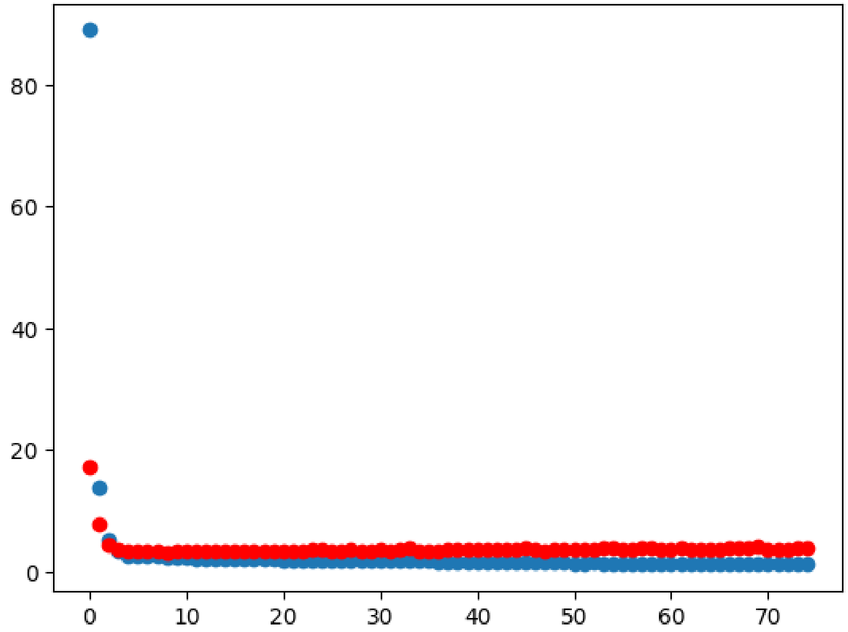
```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

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_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_80 (Dense)            (None, 64)                50240

 dropout_28 (Dropout)        (None, 64)                0

 dense_81 (Dense)            (None, 10)                650

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

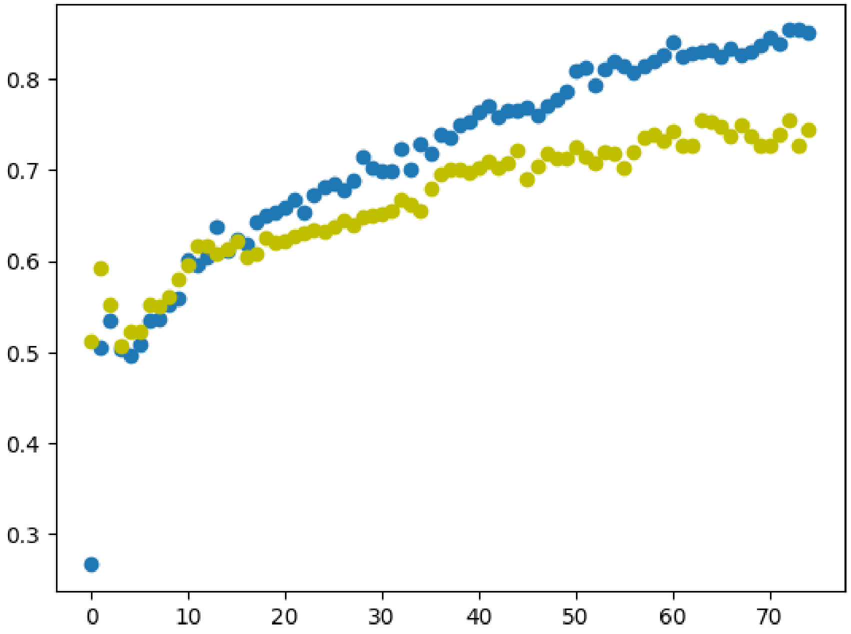
```
batch_size = 128
epochs = 75

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

```
13/13 [==============================] - 0s 6ms/step - loss: 1.3213 - accuracy: 0.8188 - val_loss: 3.8455 - val_accuracy: 0.7175
Epoch 56/75
13/13 [==============================] - 0s 5ms/step - loss: 1.3052 - accuracy: 0.8138 - val_loss: 3.6509 - val_accuracy: 0.7025
Epoch 57/75
13/13 [==============================] - 0s 4ms/step - loss: 1.3274 - accuracy: 0.8075 - val_loss: 3.7006 - val_accuracy: 0.7200
Epoch 58/75
13/13 [==============================] - 0s 5ms/step - loss: 1.3045 - accuracy: 0.8150 - val_loss: 3.9538 - val_accuracy: 0.7350
Epoch 59/75
13/13 [==============================] - 0s 4ms/step - loss: 1.2735 - accuracy: 0.8200 - val_loss: 3.8605 - val_accuracy: 0.7400
Epoch 60/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2634 - accuracy: 0.8269 - val_loss: 3.6781 - val_accuracy: 0.7325
Epoch 61/75
13/13 [==============================] - 0s 4ms/step - loss: 1.2399 - accuracy: 0.8413 - val_loss: 3.6363 - val_accuracy: 0.7425
Epoch 62/75
13/13 [==============================] - 0s 4ms/step - loss: 1.2620 - accuracy: 0.8256 - val_loss: 3.8278 - val_accuracy: 0.7275
Epoch 63/75
13/13 [==============================] - 0s 4ms/step - loss: 1.2392 - accuracy: 0.8281 - val_loss: 3.6231 - val_accuracy: 0.7275
Epoch 64/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2658 - accuracy: 0.8294 - val_loss: 3.5911 - val_accuracy: 0.7550
Epoch 65/75
13/13 [==============================] - 0s 6ms/step - loss: 1.2315 - accuracy: 0.8325 - val_loss: 3.5282 - val_accuracy: 0.7525
Epoch 66/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2311 - accuracy: 0.8244 - val_loss: 3.5099 - val_accuracy: 0.7475
Epoch 67/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2244 - accuracy: 0.8331 - val_loss: 3.8716 - val_accuracy: 0.7375
Epoch 68/75
13/13 [==============================] - 0s 4ms/step - loss: 1.2266 - accuracy: 0.8263 - val_loss: 3.7557 - val_accuracy: 0.7500
Epoch 69/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2315 - accuracy: 0.8300 - val_loss: 3.9692 - val_accuracy: 0.7375
Epoch 70/75
13/13 [==============================] - 0s 5ms/step - loss: 1.1914 - accuracy: 0.8369 - val_loss: 4.0792 - val_accuracy: 0.7275
Epoch 71/75
13/13 [==============================] - 0s 6ms/step - loss: 1.1636 - accuracy: 0.8450 - val_loss: 3.6859 - val_accuracy: 0.7275
Epoch 72/75
13/13 [==============================] - 0s 5ms/step - loss: 1.2058 - accuracy: 0.8381 - val_loss: 3.5828 - val_accuracy: 0.7400
Epoch 73/75
13/13 [==============================] - 0s 5ms/step - loss: 1.1353 - accuracy: 0.8537 - val_loss: 3.5551 - val_accuracy: 0.7550
Epoch 74/75
13/13 [==============================] - 0s 5ms/step - loss: 1.1154 - accuracy: 0.8537 - val_loss: 3.7857 - val_accuracy: 0.7275
Epoch 75/75
13/13 [==============================] - 0s 5ms/step - loss: 1.1103 - accuracy: 0.8512 - val_loss: 3.9424 - val_accuracy: 0.7450
```

```
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()
```

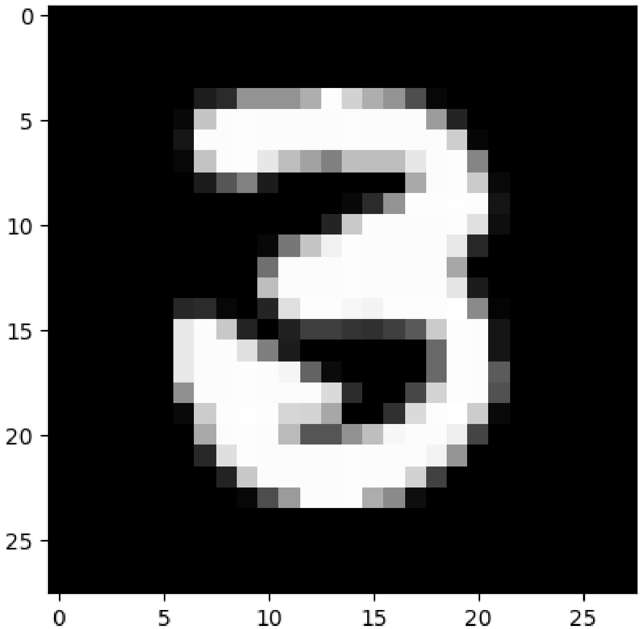



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 3.6267483234405518
Test accuracy: 0.7762647271156311
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 44ms/step
array([[6.36910846e-22, 9.86549845e-14, 1.69708755e-07, 9.99999762e-01,
        1.13557604e-07, 4.34623069e-18, 4.24941226e-23, 1.04775387e-17,
        4.43427519e-23, 3.87955533e-22]], 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_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_80 (Dense)            (None, 64)                50240

 dropout_28 (Dropout)        (None, 64)                0

 dense_81 (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 walidacyjny 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
```

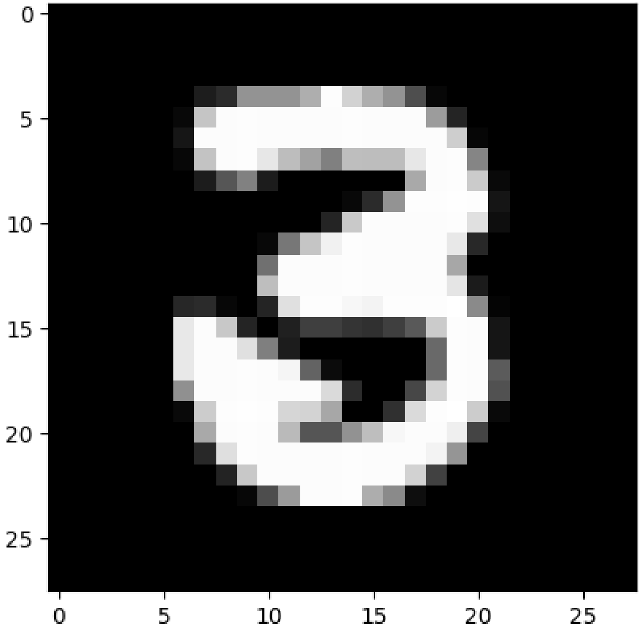
```
((68000, 28, 28), (68000, 10))
```

```
train_labels[0]
```

```
array([0., 0., 0., 0., 1., 0., 0., 0., 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. 1. 0. 0. 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

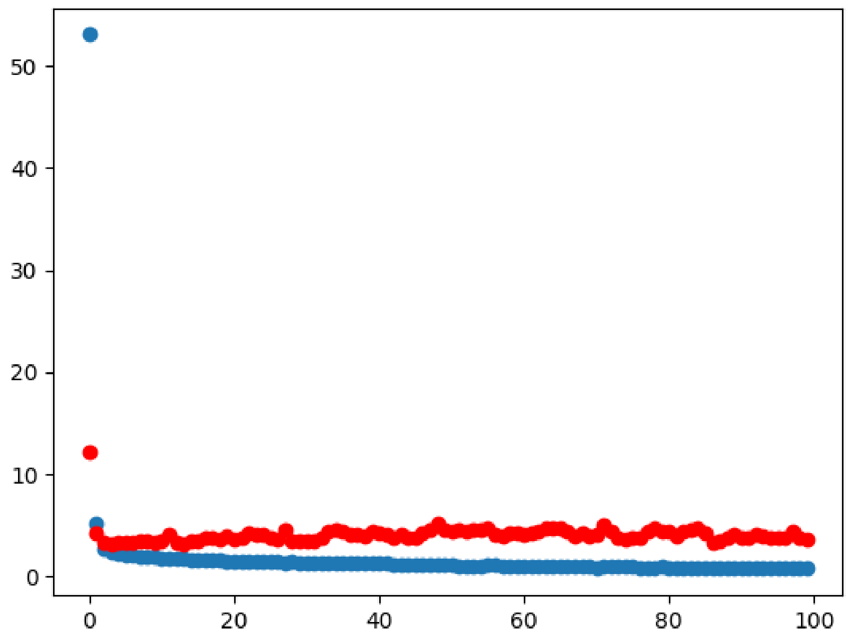
```
model = Sequential()
model.add(Dense(units = 64, kernel_regularizer=l2(0.01), use_bias=True, input_shape=(784,), activation = "relu"))
model.add(Dropout(0.3))
model.add(Dense(units = 10, kernel_regularizer=l2(0.01), use_bias=True, activation = "softmax"))

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_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_82 (Dense)            (None, 64)                50240

 dropout_29 (Dropout)        (None, 64)                0

 dense_83 (Dense)            (None, 10)                650

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

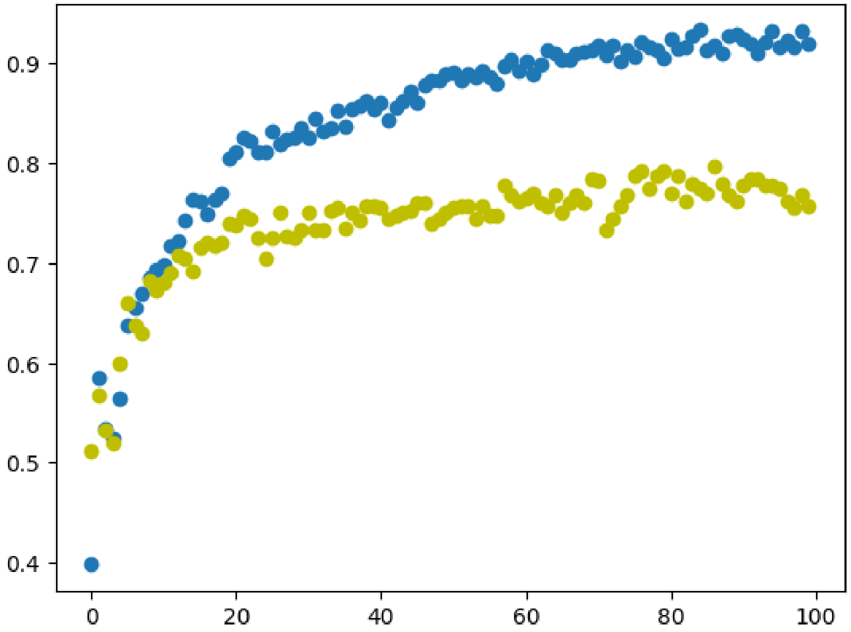
```
batch_size = 128
epochs = 100

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

```
13/13 [==============================] - 0s 4ms/step - loss: 0.8520 - accuracy: 0.9056 - val_loss: 4.3334 - val_accuracy: 0.7925
Epoch 81/100
13/13 [==============================] - 0s 5ms/step - loss: 0.7797 - accuracy: 0.9244 - val_loss: 4.2802 - val_accuracy: 0.7700
Epoch 82/100
13/13 [==============================] - 0s 5ms/step - loss: 0.8135 - accuracy: 0.9144 - val_loss: 3.9075 - val_accuracy: 0.7875
Epoch 83/100
13/13 [==============================] - 0s 6ms/step - loss: 0.7766 - accuracy: 0.9169 - val_loss: 4.2677 - val_accuracy: 0.7625
Epoch 84/100
13/13 [==============================] - 0s 6ms/step - loss: 0.7616 - accuracy: 0.9269 - val_loss: 4.4429 - val_accuracy: 0.7800
Epoch 85/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7286 - accuracy: 0.9331 - val_loss: 4.6314 - val_accuracy: 0.7750
Epoch 86/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7900 - accuracy: 0.9131 - val_loss: 4.1504 - val_accuracy: 0.7700
Epoch 87/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7971 - accuracy: 0.9175 - val_loss: 3.2181 - val_accuracy: 0.7975
Epoch 88/100
13/13 [==============================] - 0s 5ms/step - loss: 0.7633 - accuracy: 0.9106 - val_loss: 3.3430 - val_accuracy: 0.7800
Epoch 89/100
13/13 [==============================] - 0s 5ms/step - loss: 0.7370 - accuracy: 0.9275 - val_loss: 3.7560 - val_accuracy: 0.7675
Epoch 90/100
13/13 [==============================] - 0s 5ms/step - loss: 0.7152 - accuracy: 0.9294 - val_loss: 4.0196 - val_accuracy: 0.7625
Epoch 91/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7396 - accuracy: 0.9237 - val_loss: 3.6820 - val_accuracy: 0.7775
Epoch 92/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7302 - accuracy: 0.9200 - val_loss: 3.6781 - val_accuracy: 0.7850
Epoch 93/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7435 - accuracy: 0.9106 - val_loss: 3.9936 - val_accuracy: 0.7850
Epoch 94/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7386 - accuracy: 0.9212 - val_loss: 3.9131 - val_accuracy: 0.7775
Epoch 95/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7052 - accuracy: 0.9325 - val_loss: 3.7564 - val_accuracy: 0.7775
Epoch 96/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7170 - accuracy: 0.9156 - val_loss: 3.6580 - val_accuracy: 0.7750
Epoch 97/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7417 - accuracy: 0.9231 - val_loss: 3.7504 - val_accuracy: 0.7625
Epoch 98/100
13/13 [==============================] - 0s 4ms/step - loss: 0.7512 - accuracy: 0.9162 - val_loss: 4.3234 - val_accuracy: 0.7550
Epoch 99/100
13/13 [==============================] - 0s 5ms/step - loss: 0.6779 - accuracy: 0.9319 - val_loss: 3.7824 - val_accuracy: 0.7675
Epoch 100/100
13/13 [==============================] - 0s 5ms/step - loss: 0.7058 - accuracy: 0.9200 - val_loss: 3.4924 - val_accuracy: 0.7575
```

```
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()
```

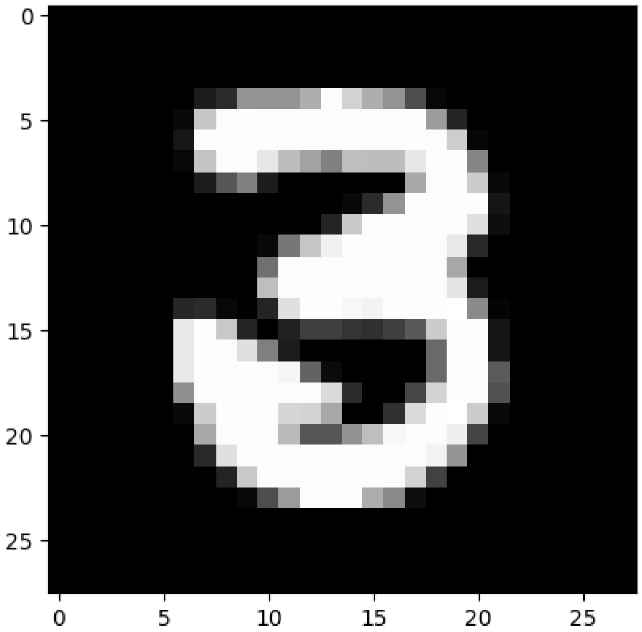



```
score = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 2.8622798919677734
Test accuracy: 0.8228235244750977
```

```
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. 1. 0. 0. 0. 0. 0. 0.]
1/1 [==============================] - 0s 40ms/step
array([[3.0179252e-09, 2.0148222e-15, 5.3240196e-10, 1.0000000e+00,
        5.6465964e-14, 4.2878010e-08, 4.2194460e-17, 1.0682056e-20,
        2.1817123e-10, 1.0109484e-19]], 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_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_82 (Dense)            (None, 64)                50240     
                                                                 
 dropout_29 (Dropout)        (None, 64)                0         
                                                                 
 dense_83 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 50890 (198.79 KB)
```