CNN na przykładzie MNIST

Setup

Importujemy potrzebne biblioteki

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

1 Przygotowanie danych

1.0 Pobranie zbioru danych

Pobieramy zbiór danych i sprawdzamy rozmiar 28 x 28 pixeli.

```
(x_train_data, y_train_data), (x_test_data, y_test_data) = tf.keras.datasets.fashion_mnist.load_data()

dataset_labels = ["0",  # index 0
                  "1",  # index 1
                  "2",  # index 2
                  "3",  # index 3
                  "4",  # index 4
                  "5",  # index 5
                  "6",  # index 6
                  "7",  # index 7
                  "8",  # index 8
                  "9"]  # index 9

print("x_train shape:", x_train_data.shape, "y_train shape:", y_train_data.shape)
print("x_test shape:", x_test_data.shape, "y_test shape:", y_test_data.shape)
```

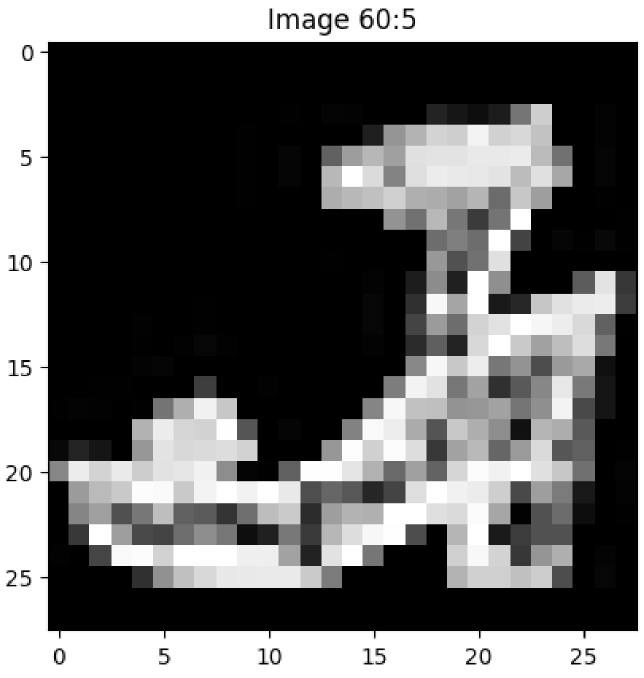
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
```

```
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28) y_train shape: (60000,)
x_test shape: (10000, 28, 28) y_test shape: (10000,)
```

1.1 Wizualizacja danych

Przykładowy obrazek ze bioru danych

```
def plot_image(img_index):
    label_index = y_train_data[img_index]
    plt.imshow(x_train_data[img_index]/255, cmap = 'gray')
    plt.title("Image "+str(img_index)+":"+dataset_labels[label_index])

img_index = 60
plot_image(img_index)
```



1.2 Normalizacja danych

Na początek sprawdzamy jakie są max i min wartości pixeli w obrazkach.

Wartości te powinny być zawarte w przedziale [0,1].

```
print("Wartości min:",np.min(x_train_data)," max:",np.max(x_train_data))

x_train_data = x_train_data.astype('float32') / 255
x_test_data = x_test_data.astype('float32') / 255

print("Wartości po przeskalowaniu min:",np.min(x_train_data)," max:",np.max(x_train_data))
```

```
Wartości min: 0  max: 255
Wartości po przeskalowaniu min: 0.0  max: 1.0
```

1.3 Podział zbioru danych na zbiór treningowy/walidacyjny/testowy

- **Zbiór treningowy** - wykorzystamy go do uczenia.
- **Zbiór walidacyjny** - wykorzystamy go do tuningu hiperparametrów.
- **Zbiór testowy** - wykorzystamy go do ostatecznego sprawdzenia modelu.

Zbiór walidacyjny stworzymy z 10% zbioru treningowego.

```
validation_fraction = .1

total_train_samples = len(x_train_data)
validation_samples = int(total_train_samples * validation_fraction)
train_samples = total_train_samples - validation_samples

(x_train, x_valid) = x_train_data[:train_samples], x_train_data[train_samples:]
(y_train, y_valid) = y_train_data[:train_samples], y_train_data[train_samples:]

x_test, y_test = x_test_data, y_test_data
print(train_samples, validation_samples, len(x_test))
```

```
54000 6000 10000
```

1.4 Dwa dodatkowe kroki

1. Większość zestawów danych obrazu składa się z obrazów rgb. Z tego powodu Keras oczekuje, że każdy obraz będzie miał 3 wymiary: [x_pixels, y_pixels, color_channels]. Ponieważ nasze obrazki są w skali szarości, wymiar koloru jest równy 1. Musimy zatem zmienić kształt obrazków.
2. W procesie uczenia naszego modelu będziemy wykorzystwali tzw. **kategoryczną entropię krzyżową** (https://keras.io/losses/). Musimy przekształcić wektory z etykietami (labelami) do **kodowania one-hot**. Wykorzystamy do tego funkcję tf.keras.utils.to_categorical().

```
# Zmieniamy kształ z (28, 28) na (28, 28, 1)
w, h = 28, 28
x_train = x_train.reshape(x_train.shape[0], w, h, 1)
x_valid = x_valid.reshape(x_valid.shape[0], w, h, 1)
x_test = x_test.reshape(x_test.shape[0], w, h, 1)

# Kodowanie one-hot
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_valid = tf.keras.utils.to_categorical(y_valid, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

print("x_train shape:", x_train.shape, "y_train shape:", y_train.shape)

# Ilość elmentów w zbiorach
print(x_train.shape[0], 'train set')
print(x_valid.shape[0], 'validation set')
print(x_test.shape[0], 'test set')
```

```
x_train shape: (54000, 28, 28, 1) y_train shape: (54000, 10)
54000 train set
6000 validation set
10000 test set
```

2 Stworzenie modelu

Keras oferuje dwa API:

1. Sequential model API
2. Functional API

W naszym modelu wykorzystamy Sequential model API. Będziemy wykorzystwali następujące metody:

- Dense() link text - tworzy **warstwę gęstą**
- Conv2D() link text - tworzy **warstwę konwolucyjną**
- Pooling() link text - tworzy **warstwę pooling**

- Dropout() [link text] - zastowanie **dropout**

2.0 Prosty model liniowy

Zaczniemy od prostego modelu składającego się z jednej transformacji liniowej.

- Model stworzymy za pomocą tf.keras.Sequential() (https://www.tensorflow.org/api_docs/python/tf/keras/models/Sequential). Ponieważ nie zastosujemy jeszcze konwolucji zatem możemy spłaszczyć obrazki do wektorów zawierających 28x28 wartości.
- Następnie dodamy jedną warstwę liniową, która przkształci wejściowe piksele w 10 klas. Poniważ wyniki reprezentują prawdopodobieństwa możemy użyć funkcji aktywacji softmax.
- Szczegóły modelu uzyskamy z pomocą model.summary()

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(10,activation="softmax"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 10)                7850

=================================================================
Total params: 7850 (30.66 KB)
Trainable params: 7850 (30.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Kompilacja modelu

Uwagi:

- Użyjemy **optymizera adam**
- Jako loss function użyjemy '**categorical_crossentropy**'
- Lista parametrów, tutaj zaczniemy od '**precyzji**'

Warto zerknąć: https://keras.io/models/model/

```
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy']) #learnig rate???
```

Uczenie modelu

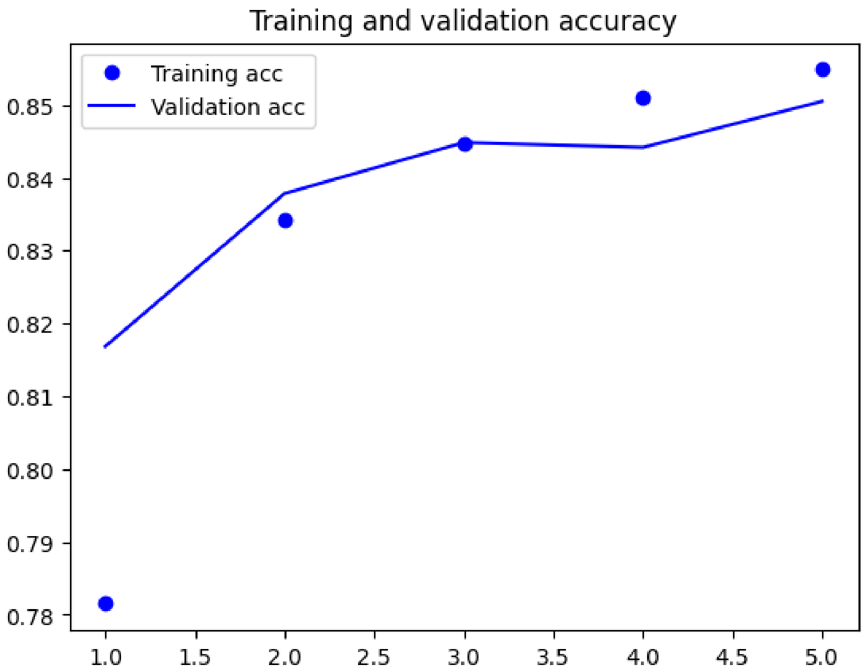
Model uczymy wykorzystując fit().

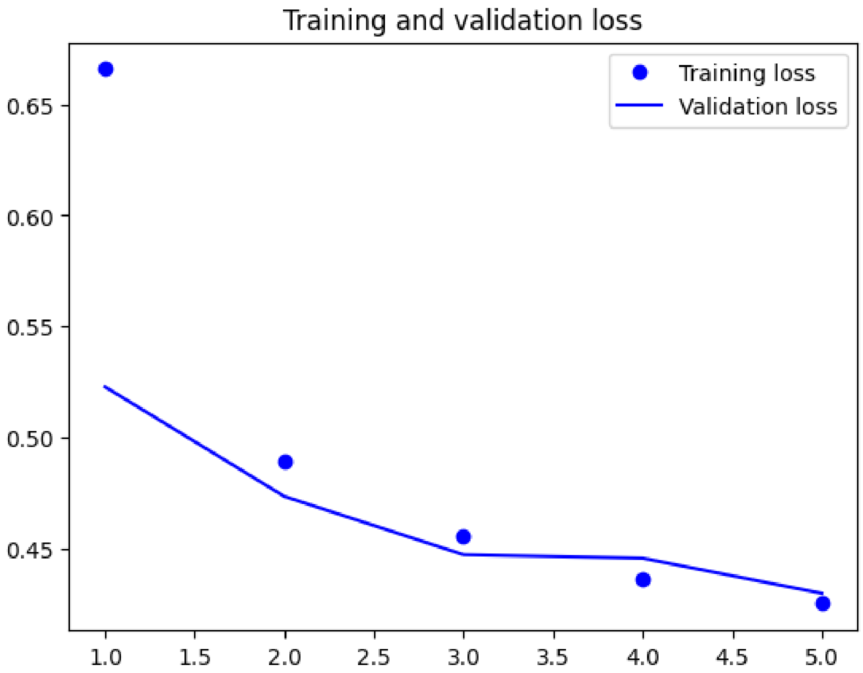
```
history = model.fit(x_train, y_train, batch_size = 64, epochs = 5, validation_data = (x_valid,y_valid))
```

```
Epoch 1/5
844/844 [==============================] - 11s 10ms/step - loss: 0.6659 - accuracy: 0.7815 - val_loss: 0.5227 - val_accuracy: 0.8168
Epoch 2/5
844/844 [==============================] - 5s 6ms/step - loss: 0.4892 - accuracy: 0.8341 - val_loss: 0.4733 - val_accuracy: 0.8378
Epoch 3/5
844/844 [==============================] - 3s 3ms/step - loss: 0.4558 - accuracy: 0.8447 - val_loss: 0.4471 - val_accuracy: 0.8448
Epoch 4/5
844/844 [==============================] - 3s 4ms/step - loss: 0.4362 - accuracy: 0.8511 - val_loss: 0.4455 - val_accuracy: 0.8442
Epoch 5/5
844/844 [==============================] - 2s 3ms/step - loss: 0.4250 - accuracy: 0.8549 - val_loss: 0.4297 - val_accuracy: 0.8505
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0.78
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Training and validation loss
Training loss
Validation loss
0.65
0.60
0.55
0.50
0.45
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Zapisanie i wczytanie modelu

Zapisanie modelu

```
model.save("mnist_simple.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
```

Wczytanie modelu

```
#from keras.models import load_model
#model = load_model("mnist_simple.h5")
```

Precyzja

Wykorzystamy funkcję evaluate()

```
score = model.evaluate(x_test,y_test,verbose=0)
print('Test accuracy:',score[1])
```

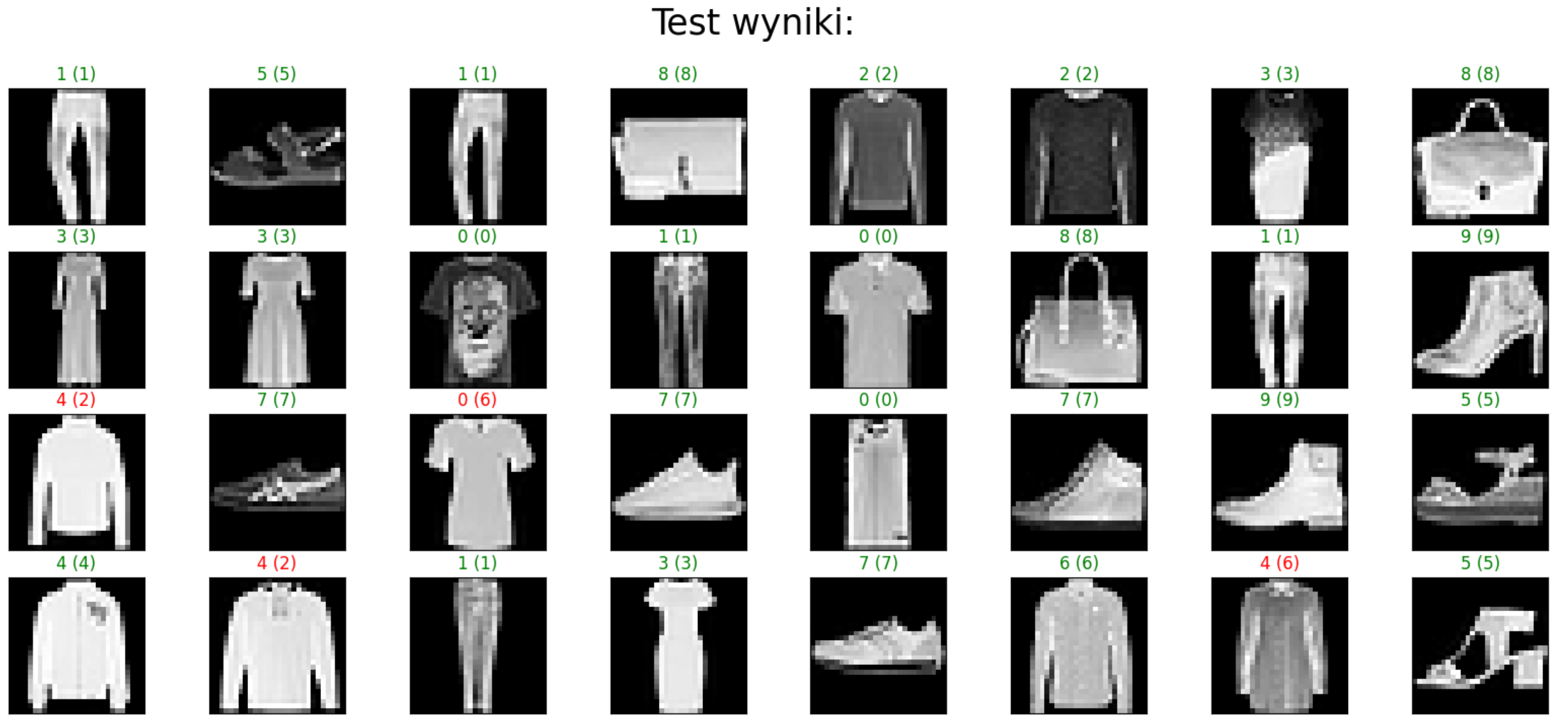
```
Test accuracy: 0.8393999934196472
```

Przewidywania modelu

Przetestujmy przewidywania naszego modelu. Sprawdzimy go na danych testowych. W tym celu wykorzystamy poniższą funkcję 'visualize_model_predictions(model, x, y)'

```
def visualize_model_predictions(model, x_test, y_test, title_string):
    y_hat = model.predict(x_test)

    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=32, replace=False)):
        ax = figure.add_subplot(4, 8, i + 1, xticks=[], yticks=[])

        ax.imshow(np.squeeze(x_test[index]), cmap = 'gray')
        predict_index = np.argmax(y_hat[index])
        true_index = np.argmax(y_test[index])

        ax.set_title("{} ({})".format(dataset_labels[predict_index],
                                      dataset_labels[true_index]),
                                      color=("green" if predict_index == true_index else "red"))
    figure.suptitle("%s wyniki:" %title_string, fontsize=25)

visualize_model_predictions(model, x_test, y_test, 'Test')
```

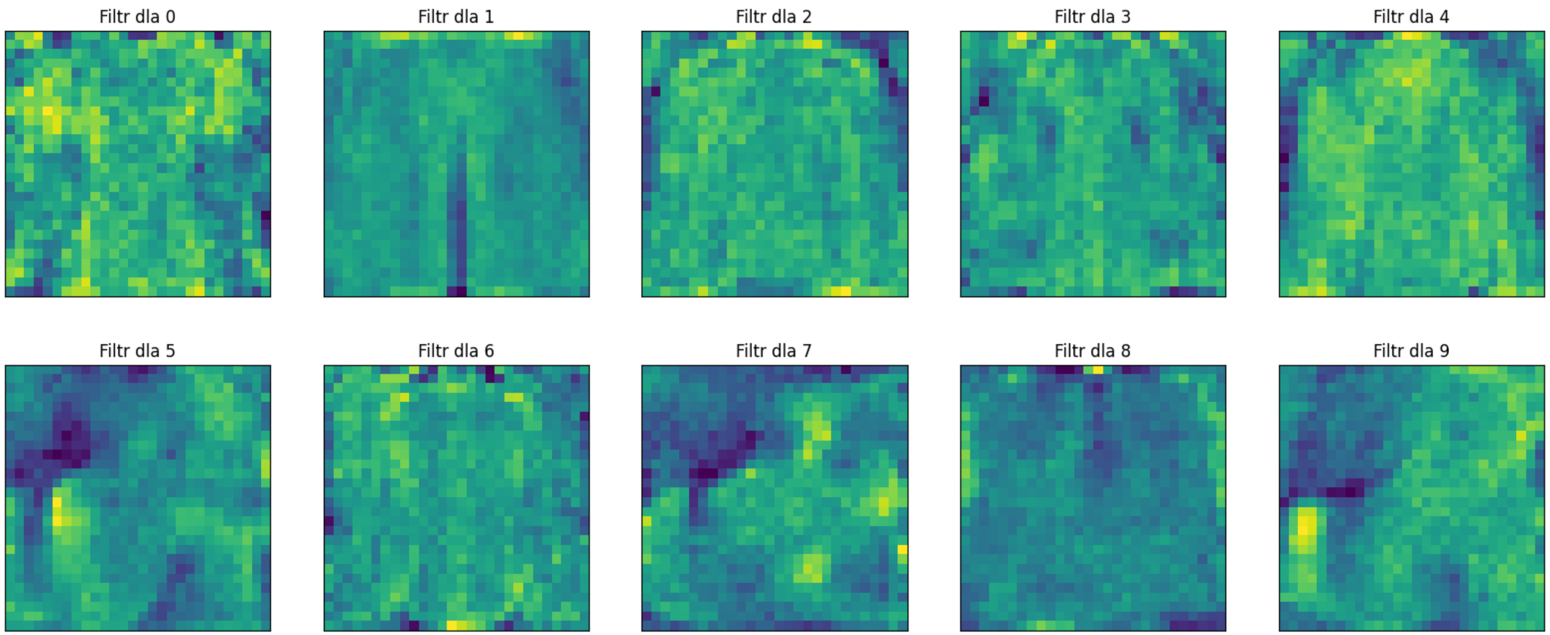
```
313/313 [==============================] - 1s 1ms/step
```



1. Wizualizacja wag dla każdej klasy

Warstwę transformacyjną naszgo modelu można przedstawić za pomocą macierzy wag [28x28, 10]. Spróbujmy narysować każdy z 10 filtrów.
W celu uzyskania wag modelu wykorzystamy funkcje model.layers i get_weights().

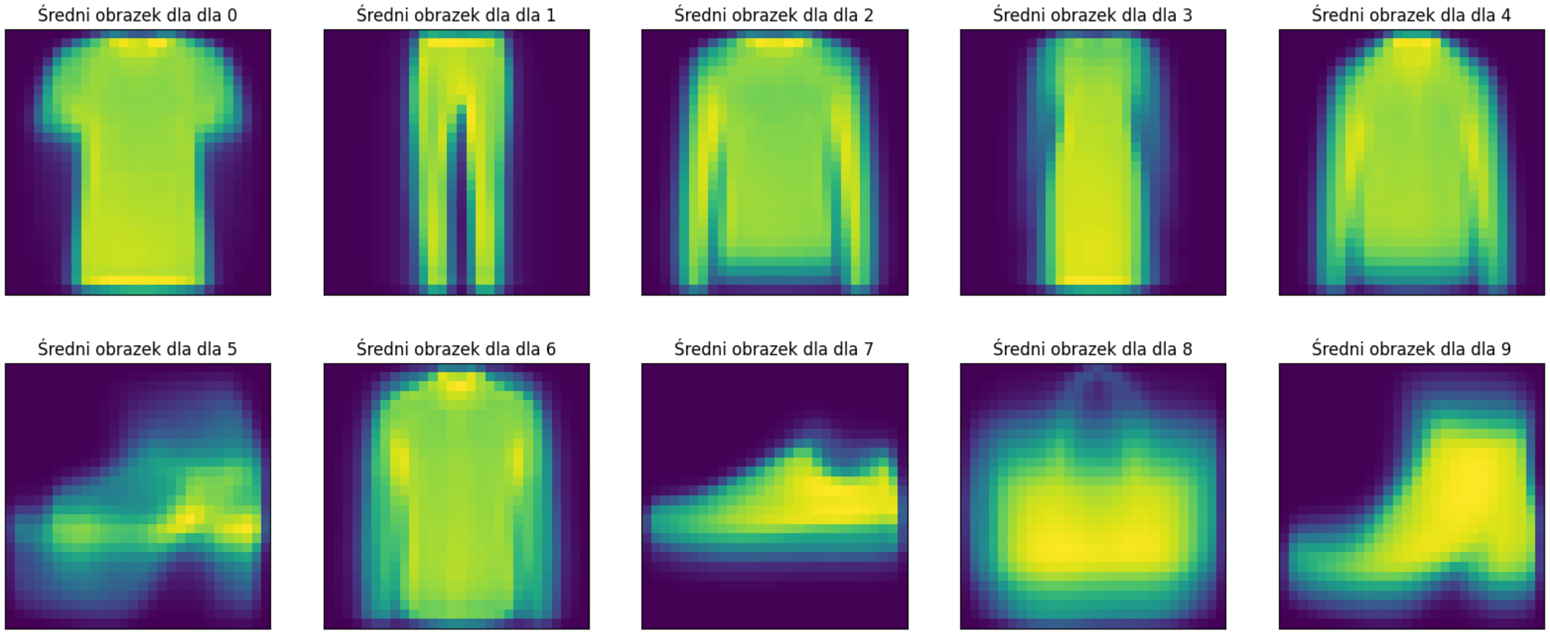
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

def visualize_filters(filters, title_string):
    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=10, replace=False)):
        ax = figure.add_subplot(2, 5, i + 1, xticks=[], yticks=[])
        ax.imshow(filters[:,:,i], cmap = 'viridis')
        ax.set_title("%s dla %s" %(title_string, dataset_labels[i]))

visualize_filters(filters, 'Filtr')
```



2 I jeszcze jedna wizualizacja

Porównajmy powyższe filtry, ze średnim zdjęciem dla każdej klasy.

```
avg_images = np.zeros((28,28,1,10))
class_images = [0]*10

for i in range(len(x_train)):
    img = x_train[i]
    label = np.argmax(y_train[i])

    avg_images[:,:,:,label] += img
    class_images[label] += 1

for i in range(10):
    avg_images[:,:,:,i] = avg_images[:,:,:,i]/class_images[i]

avg_images = np.squeeze(avg_images)
visualize_filters(avg_images, 'Średni obrazek dla')
```



2.0 Prosty model liniowy v2

Zaczniemy od prostego modelu składającego się z jednej transformacji liniowej.

- Model stworzymy za pomocą tf.keras.Sequential() (https://www.tensorflow.org/api_docs/python/tf/keras/models/Sequential). Ponieważ nie zastosujemy jeszcze konwolucji zatem możemy spłaszczyć obrazki do wektorów zawierających 28x28 wartości.
- Następnie dodamy jedną warstwę liniową, która przkształci wejściowe piksele w 10 klas. Poniważ wyniki reprezentują prawdopodobieństwa możemy użyć funkcji aktywacji softmax.
- Szczegóły modelu uzyskamy z pomocą model.summary()

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(10,activation="softmax"))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_1 (Dense)             (None, 10)                7850

=================================================================
Total params: 7850 (30.66 KB)
Trainable params: 7850 (30.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Kompilacja modelu

Uwagi:

- Użyjemy **optymizera adam**
- Jako loss function użyjemy '**categorical_crossentropy**'
- Lista parametrów, tutaj zaczniemy od '**precyzji**'

Warto zerknąć: https://keras.io/models/model/

```
opt = keras.optimizers.Adam(learning_rate=0.002)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy']) #learnig rate???
```

Uczenie modelu

Model uczymy wykorzystując fit().

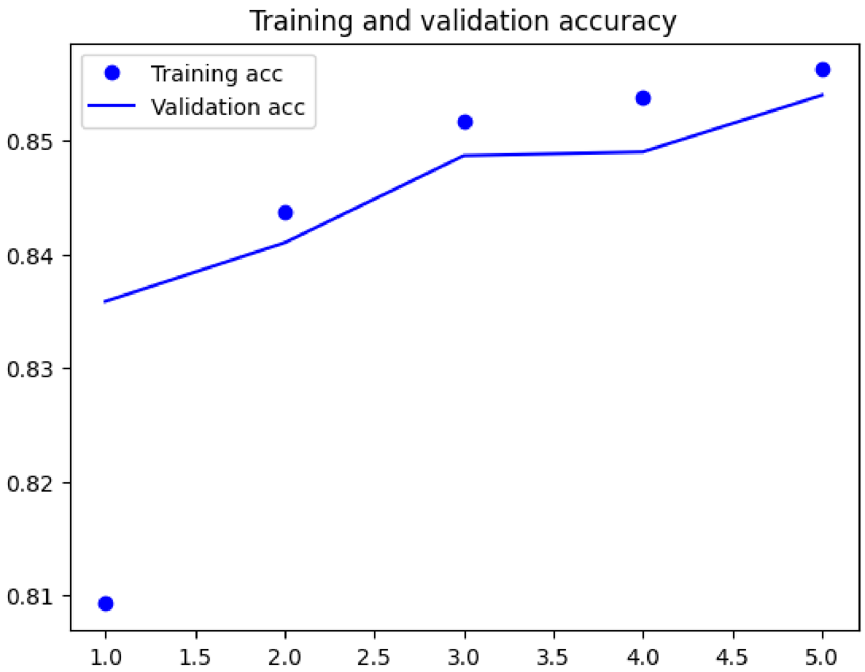
```
history = model.fit(x_train, y_train, batch_size = 32, epochs = 5, validation_data = (x_valid,y_valid))
```

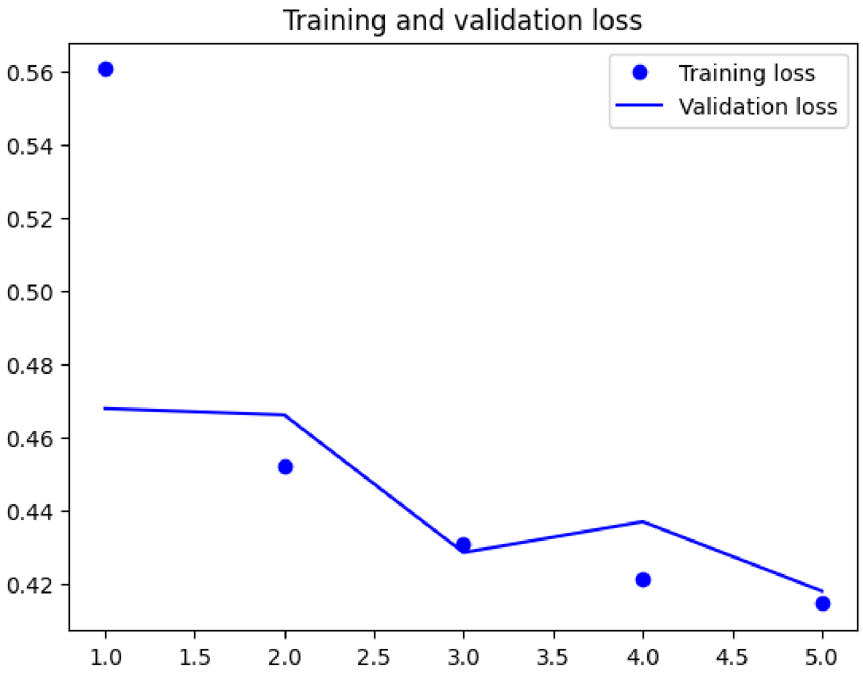
```
Epoch 1/5
1688/1688 [==============================] - 6s 3ms/step - loss: 0.5608 - accuracy: 0.8092 - val_loss: 0.4680 - val_accuracy: 0.8358
Epoch 2/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4523 - accuracy: 0.8437 - val_loss: 0.4662 - val_accuracy: 0.8410
```

```
Epoch 3/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4311 - accuracy: 0.8516 - val_loss: 0.4286 - val_accuracy: 0.8487
Epoch 4/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4212 - accuracy: 0.8538 - val_loss: 0.4370 - val_accuracy: 0.8490
Epoch 5/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.4146 - accuracy: 0.8562 - val_loss: 0.4180 - val_accuracy: 0.8540
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.84
0.83
0.82
0.81
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Training and validation loss
Training loss
Validation loss
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.42
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Zapisanie i wczytanie modelu

Zapisanie modelu

```
model.save("mnist_simple.h5")
```

Wczytanie modelu

```
#from keras.models import load_model
#model = load_model("mnist_simple.h5")
```

Precyzja

Wykorzystamy funkcję evaluate()

```
score = model.evaluate(x_test,y_test,verbose=0)
print('Test accuracy:',score[1])
```

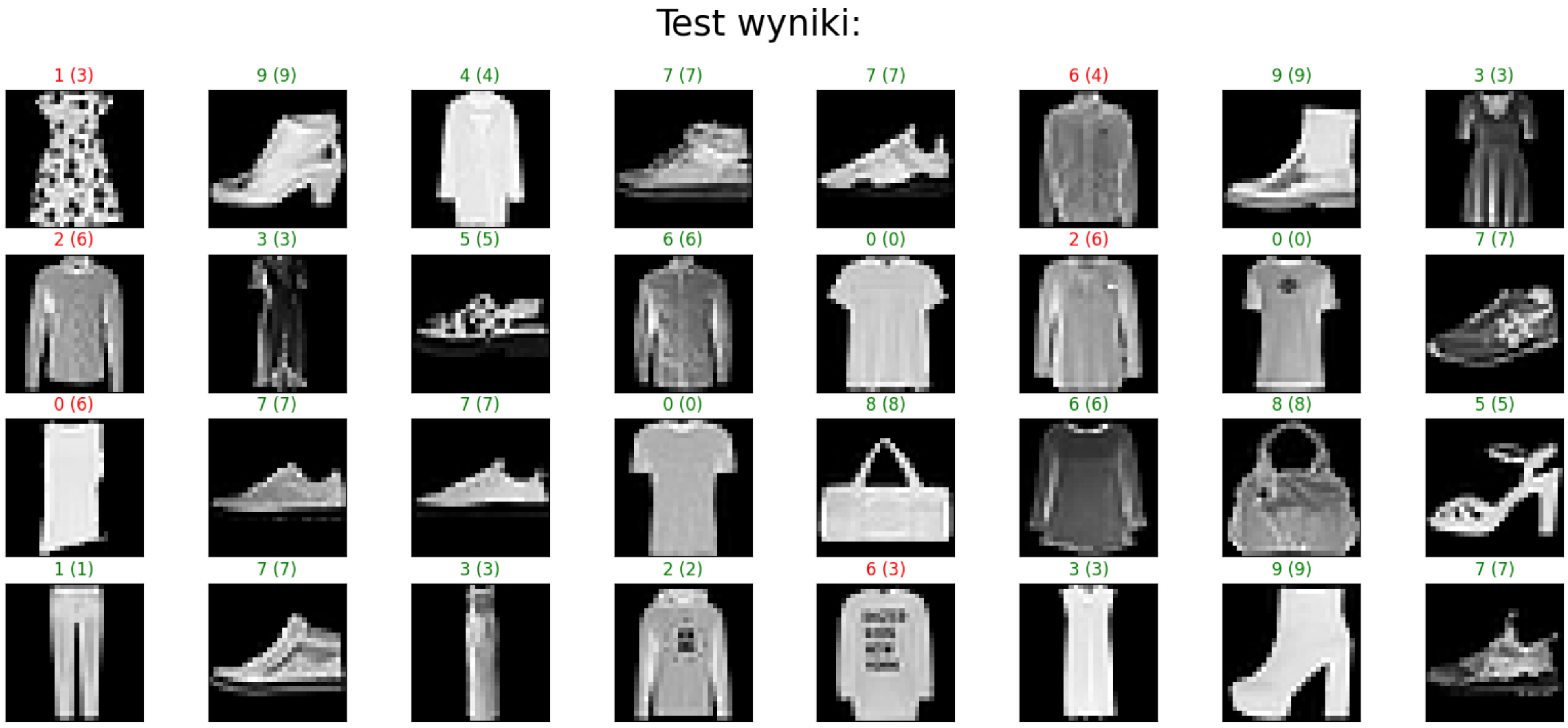
```
Test accuracy: 0.8410999774932861
```

Przewidywania modelu

Przetestujmy przewidywania naszego modelu. Sprawdzimy go na danych testowych. W tym celu wykorzystamy poniższą funkcję 'visualize_model_predictions(model, x, y)'

```
def visualize_model_predictions(model, x_test, y_test, title_string):
    y_hat = model.predict(x_test)

    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=32, replace=False)):
        ax = figure.add_subplot(4, 8, i + 1, xticks=[], yticks=[])

        ax.imshow(np.squeeze(x_test[index]), cmap = 'gray')
        predict_index = np.argmax(y_hat[index])
        true_index = np.argmax(y_test[index])

        ax.set_title("{} ({})".format(dataset_labels[predict_index],
                                      dataset_labels[true_index]),
                                      color=("green" if predict_index == true_index else "red"))
    figure.suptitle("%s wyniki:" %title_string, fontsize=25)

visualize_model_predictions(model, x_test, y_test, 'Test')
```

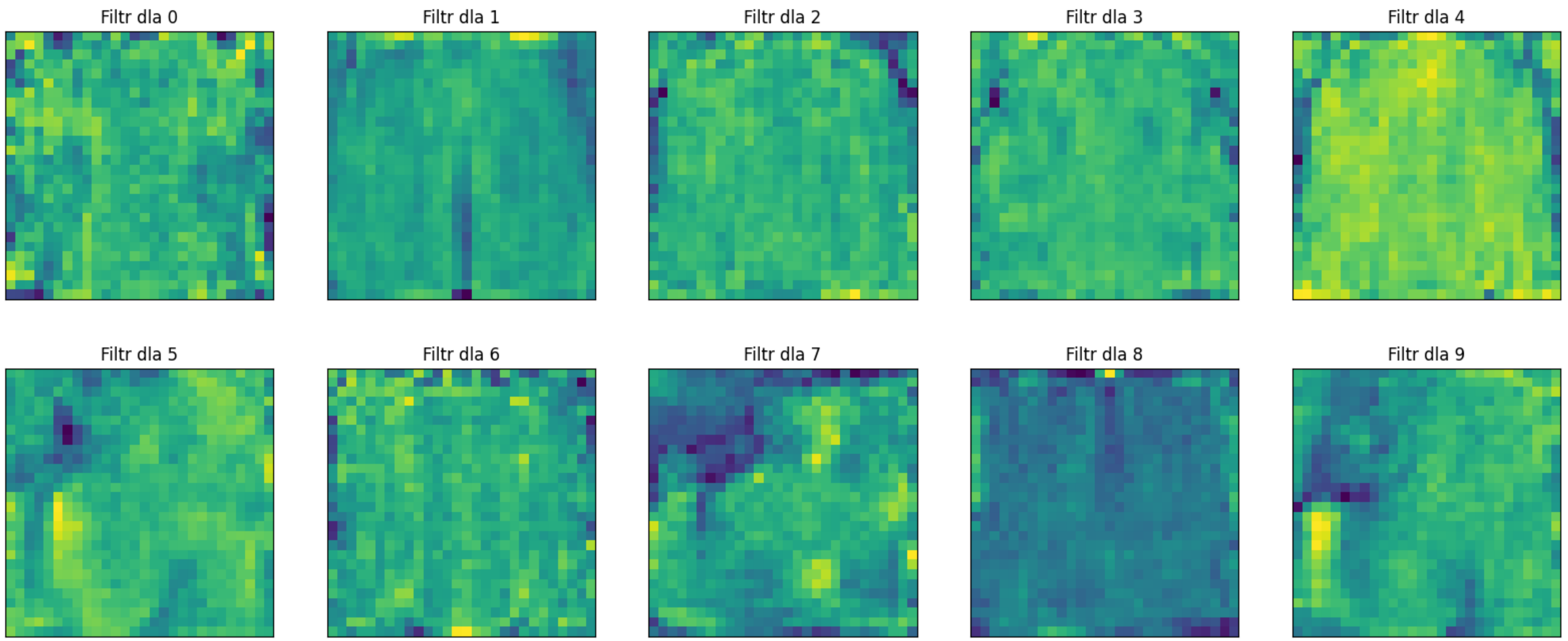
```
313/313 [==============================] - 0s 1ms/step
```



1. Wizualizacja wag dla każdej klasy

Warstwę transformacyjną naszgo modelu można przedstawić za pomocą macierzy wag [28x28, 10]. Spróbujmy narysować każdy z 10 filtrów.
W celu uzyskania wag modelu wykorzystamy funkcje model.layers i get_weights().

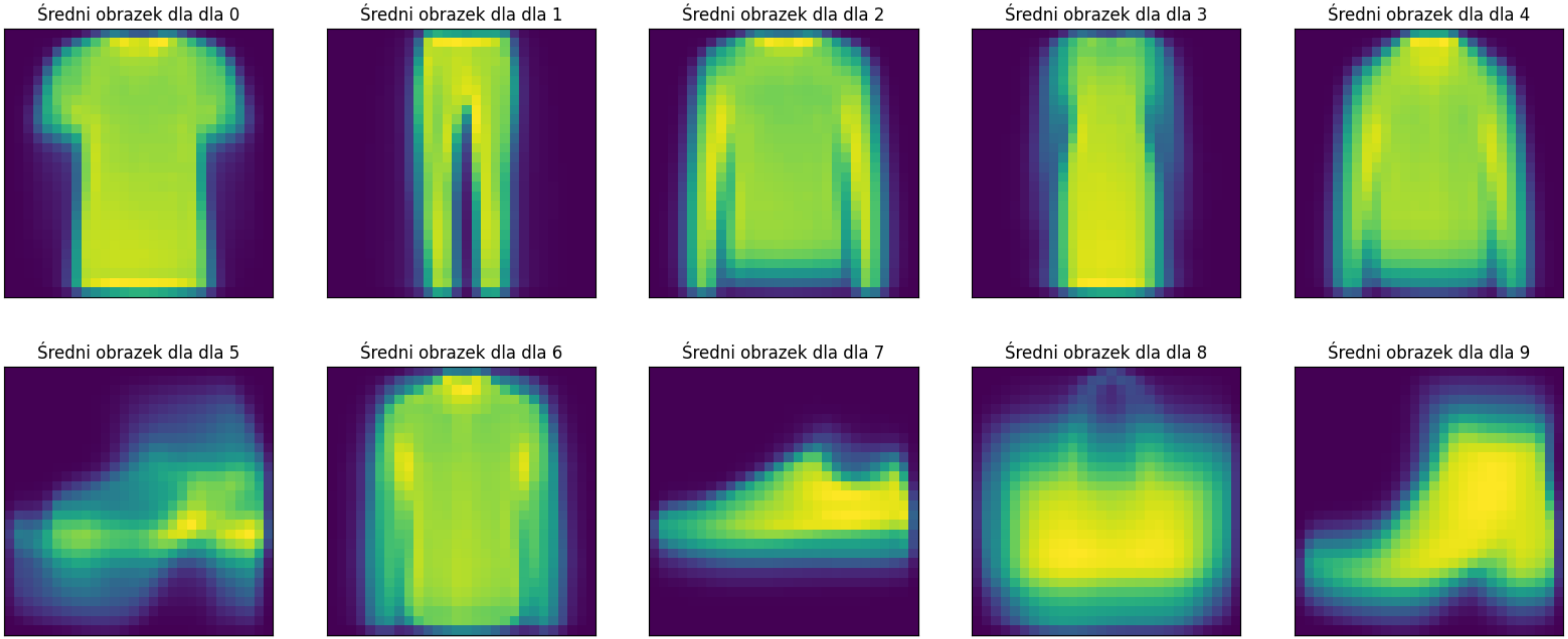
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

def visualize_filters(filters, title_string):
    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=10, replace=False)):
        ax = figure.add_subplot(2, 5, i + 1, xticks=[], yticks=[])
        ax.imshow(filters[:,:,i], cmap = 'viridis')
        ax.set_title("%s dla %s" %(title_string, dataset_labels[i]))

visualize_filters(filters, 'Filtr')
```



2 I jeszcze jedna wizualizacja

Porównajmy powyższe filtry, ze średnim zdjęciem dla każdej klasy.

```
avg_images = np.zeros((28,28,1,10))
class_images = [0]*10

for i in range(len(x_train)):
    img = x_train[i]
    label = np.argmax(y_train[i])

    avg_images[:,:,:,label] += img
    class_images[label] += 1

for i in range(10):
    avg_images[:,:,:,i] = avg_images[:,:,:,i]/class_images[i]

avg_images = np.squeeze(avg_images)
visualize_filters(avg_images, 'Średni obrazek dla')
```



2.0 Prosty model liniowy v3

Zaczniemy od prostego modelu składającego się z jednej transformacji liniowej.

- Model stworzymy za pomocą tf.keras.Sequential() (https://www.tensorflow.org/api_docs/python/tf/keras/models/Sequential). Ponieważ nie zastosujemy jeszcze konwolucji zatem możemy spłaszczyć obrazki do wektorów zawierających 28x28 wartości.
- Następnie dodamy jedną warstwę liniową, która przkształci wejściowe piksele w 10 klas. Poniważ wyniki reprezentują prawdopodobieństwa możemy użyć funkcji aktywacji softmax.
- Szczegóły modelu uzyskamy z pomocą model.summary()

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(10,activation="softmax"))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 10)                7850

=================================================================
Total params: 7850 (30.66 KB)
Trainable params: 7850 (30.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Kompilacja modelu

Uwagi:

- Użyjemy **optymizera sgd**
- Jako loss function użyjemy '**categorical_crossentropy**'
- Lista parametrów, tutaj zaczniemy od '**precyzji**'

Warto zerknąć: https://keras.io/models/model/

```
opt = keras.optimizers.SGD(learning_rate=0.04)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

Uczenie modelu

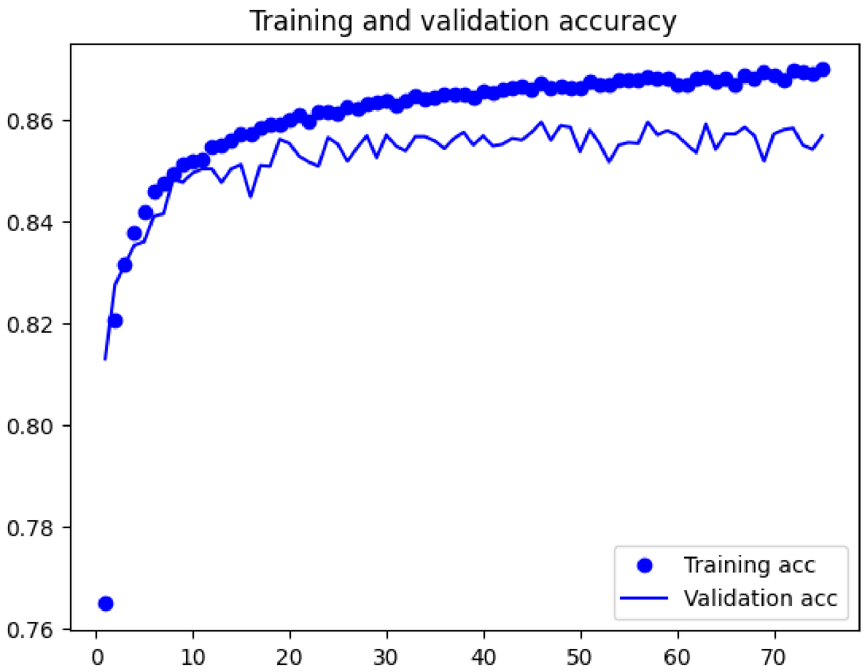
Model uczymy wykorzystując fit().

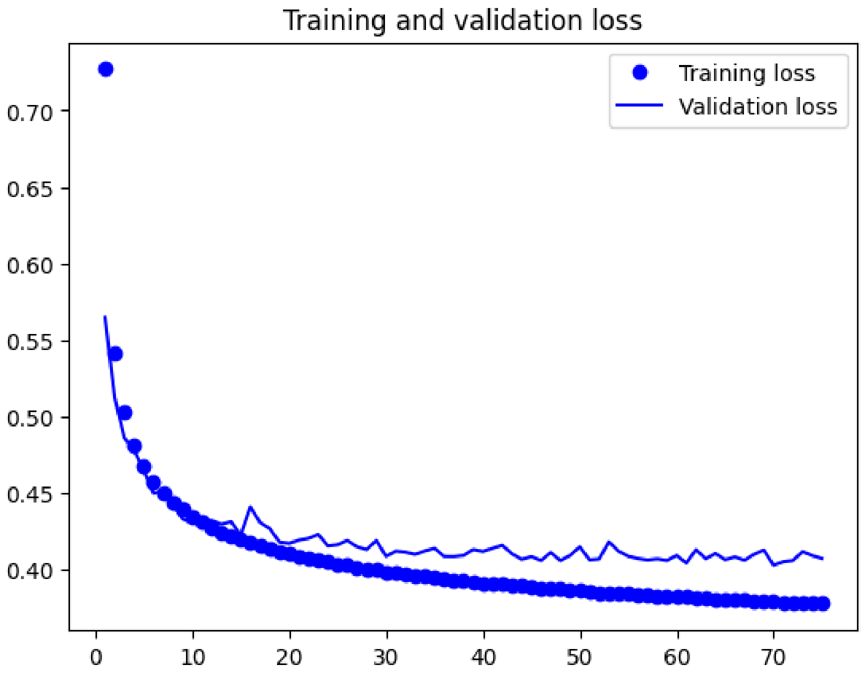
```
history = model.fit(x_train, y_train, batch_size = 64, epochs = 75, validation_data = (x_valid,y_valid))
```

```
844/844 [==============================] - 3s 3ms/step - loss: 0.3781 - accuracy: 0.8698 - val_loss: 0.4075 - val_accuracy: 0.8568
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
0.86
0.84
0.82
0.80
0.78
0.76
0
10
20
30
40
50
60
70
Training acc
Validation acc


Training and validation loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
10
20
30
40
50
60
70
Training loss
Validation loss

Zapisanie i wczytanie modelu

Zapisanie modelu

```
model.save("mnist_simple.h5")
```

Wczytanie modelu

```
#from keras.models import load_model
#model = load_model("mnist_simple.h5")
```

Precyzja

Wykorzystamy funkcję evaluate()

```
score = model.evaluate(x_test,y_test,verbose=0)
print('Test accuracy:',score[1])
```

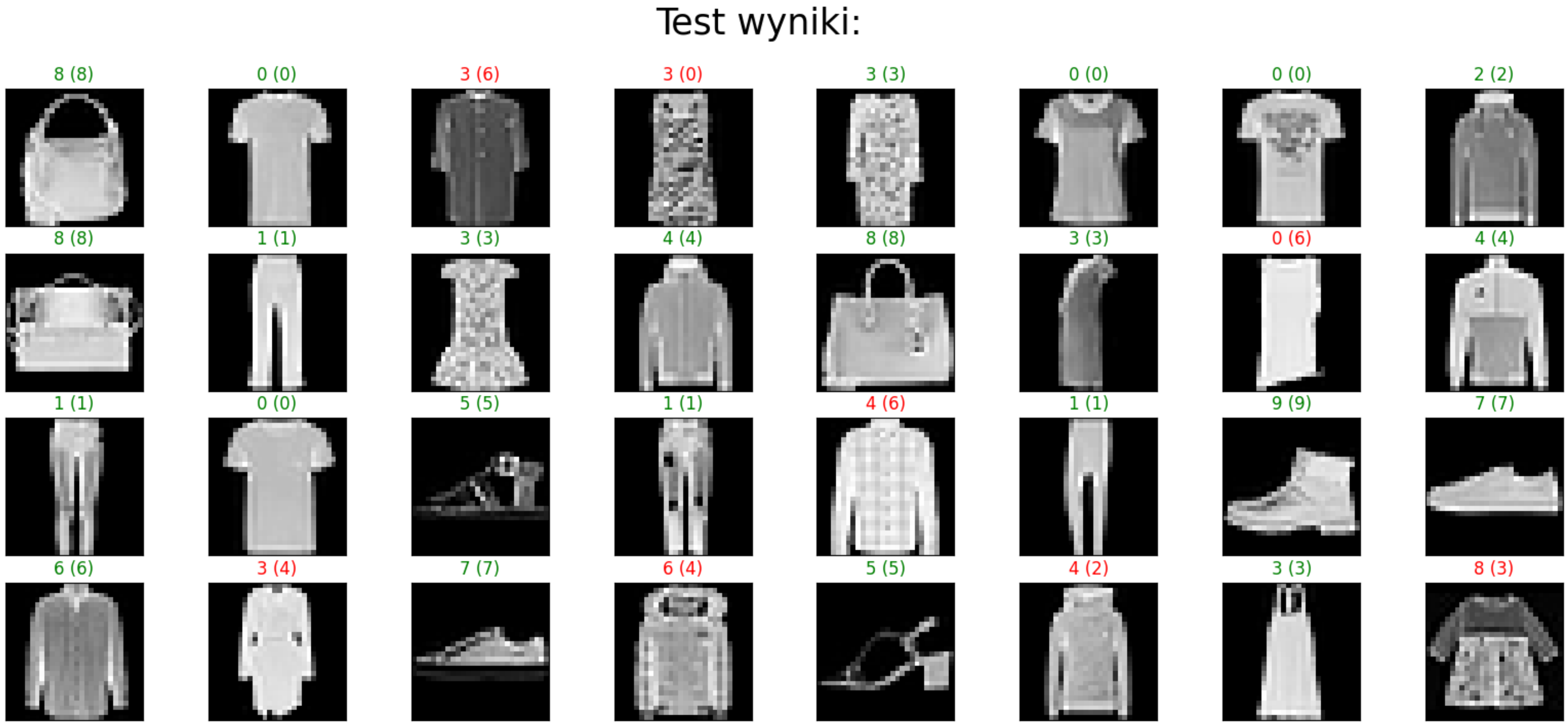
```
    Test accuracy: 0.8434000015258789
```

Przewidywania modelu

Przetestujmy przewidywania naszego modelu. Sprawdzimy go na danych testowych. W tym celu wykorzystamy poniższą funkcję 'visualize_model_predictions(model, x, y)'

```
def visualize_model_predictions(model, x_test, y_test, title_string):
    y_hat = model.predict(x_test)

    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=32, replace=False)):
        ax = figure.add_subplot(4, 8, i + 1, xticks=[], yticks=[])

        ax.imshow(np.squeeze(x_test[index]), cmap = 'gray')
        predict_index = np.argmax(y_hat[index])
        true_index = np.argmax(y_test[index])

        ax.set_title("{} ({})".format(dataset_labels[predict_index],
                                      dataset_labels[true_index]),
                                      color=("green" if predict_index == true_index else "red"))
    figure.suptitle("%s wyniki:" %title_string, fontsize=25)

visualize_model_predictions(model, x_test, y_test, 'Test')
```

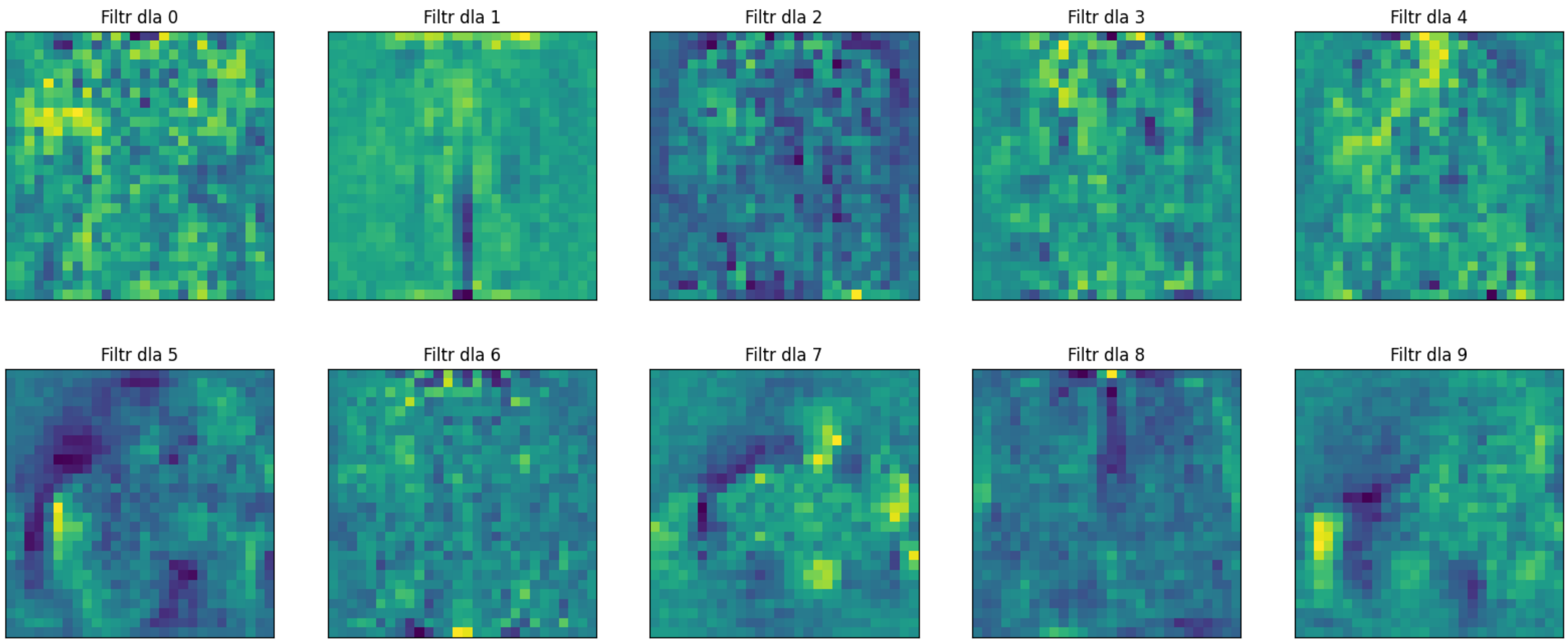
```
313/313 [==============================] - 0s 1ms/step
```



1. Wizualizacja wag dla każdej klasy

Warstwę transformacyjną naszgo modelu można przedstawić za pomocą macierzy wag [28x28, 10]. Spróbujmy narysować każdy z 10 filtrów.
W celu uzyskania wag modelu wykorzystamy funkcje model.layers i get_weights().

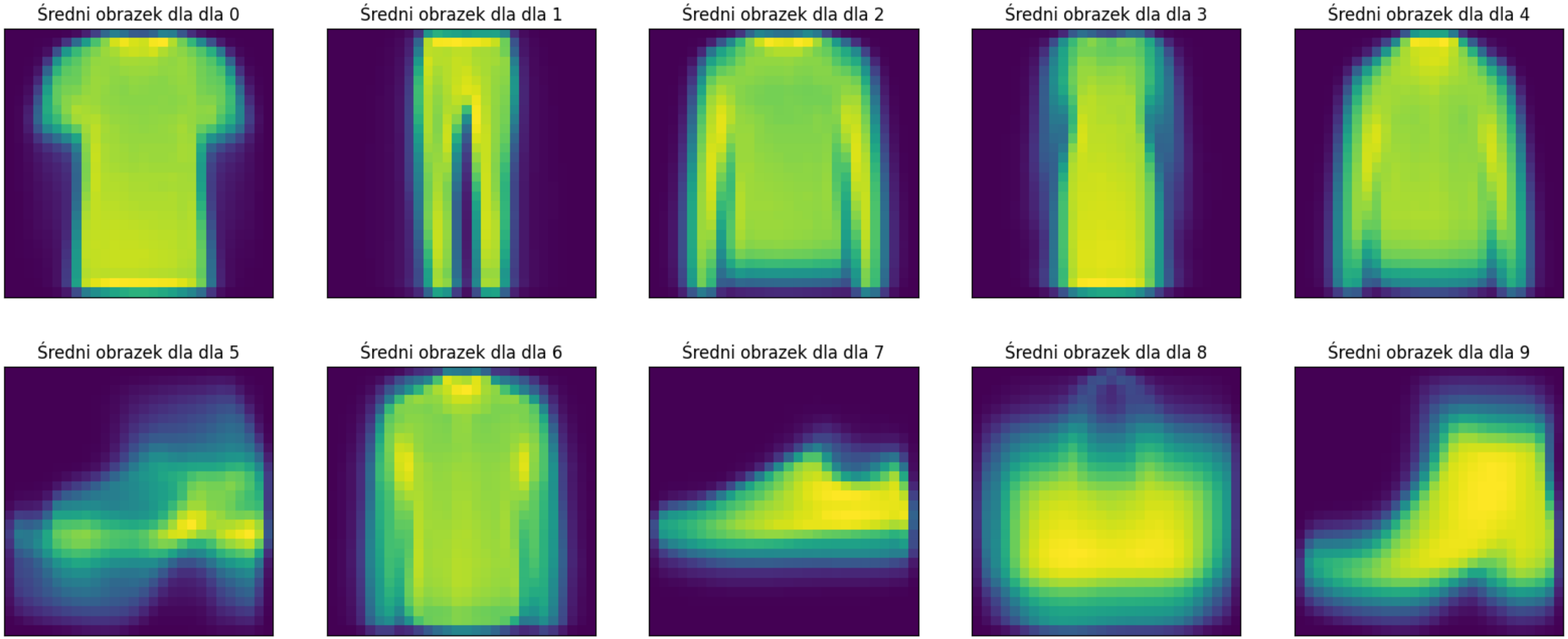
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

def visualize_filters(filters, title_string):
    figure = plt.figure(figsize=(20, 8))
    for i, index in enumerate(np.random.choice(x_test.shape[0], size=10, replace=False)):
        ax = figure.add_subplot(2, 5, i + 1, xticks=[], yticks=[])
        ax.imshow(filters[:,:,i], cmap = 'viridis')
        ax.set_title("%s dla %s" %(title_string, dataset_labels[i]))

visualize_filters(filters, 'Filtr')
```



2 I jeszcze jedna wizualizacja

Porównajmy powyższe filtry, ze średnim zdjęciem dla każdej klasy.

```
avg_images = np.zeros((28,28,1,10))
class_images = [0]*10

for i in range(len(x_train)):
    img = x_train[i]
    label = np.argmax(y_train[i])

    avg_images[:,:,:,label] += img
    class_images[label] += 1

for i in range(10):
    avg_images[:,:,:,i] = avg_images[:,:,:,i]/class_images[i]

avg_images = np.squeeze(avg_images)
visualize_filters(avg_images, 'Średni obrazek dla')
```



2.1 A teraz sieć neuronowa

Dodajmy teraz warstwy wewnętrzne w naszej sieci. Funkcja aktywacji w takich warstwach to zwykle relu.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(60,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

model.summary()

model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy']) #learnig rate???

history = model.fit(x_train, y_train, batch_size = 64, epochs = 10, validation_data = (x_valid,y_valid))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_3 (Flatten)         (None, 784)               0         
                                                                 
 dense_3 (Dense)             (None, 60)                47100     
                                                                 
 dense_4 (Dense)             (None, 10)                610       
                                                                 
=================================================================
Total params: 47710 (186.37 KB)
Trainable params: 47710 (186.37 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
844/844 [==============================] - 4s 3ms/step - loss: 0.5653 - accuracy: 0.8077 - val_loss: 0.4265 - val_accuracy: 0.8442
Epoch 2/10
844/844 [==============================] - 4s 4ms/step - loss: 0.4121 - accuracy: 0.8532 - val_loss: 0.4095 - val_accuracy: 0.8583
Epoch 3/10
844/844 [==============================] - 3s 3ms/step - loss: 0.3755 - accuracy: 0.8665 - val_loss: 0.3804 - val_accuracy: 0.8623
Epoch 4/10
844/844 [==============================] - 3s 3ms/step - loss: 0.3510 - accuracy: 0.8751 - val_loss: 0.3873 - val_accuracy: 0.8627
Epoch 5/10
844/844 [==============================] - 3s 3ms/step - loss: 0.3331 - accuracy: 0.8804 - val_loss: 0.3767 - val_accuracy: 0.8678
Epoch 6/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3205 - accuracy: 0.8843 - val_loss: 0.3475 - val_accuracy: 0.8717
Epoch 7/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3064 - accuracy: 0.8886 - val_loss: 0.3397 - val_accuracy: 0.8783
Epoch 8/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2953 - accuracy: 0.8921 - val_loss: 0.3408 - val_accuracy: 0.8763
Epoch 9/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2856 - accuracy: 0.8970 - val_loss: 0.3514 - val_accuracy: 0.8767
Epoch 10/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2788 - accuracy: 0.8984 - val_loss: 0.3277 - val_accuracy: 0.8805
```

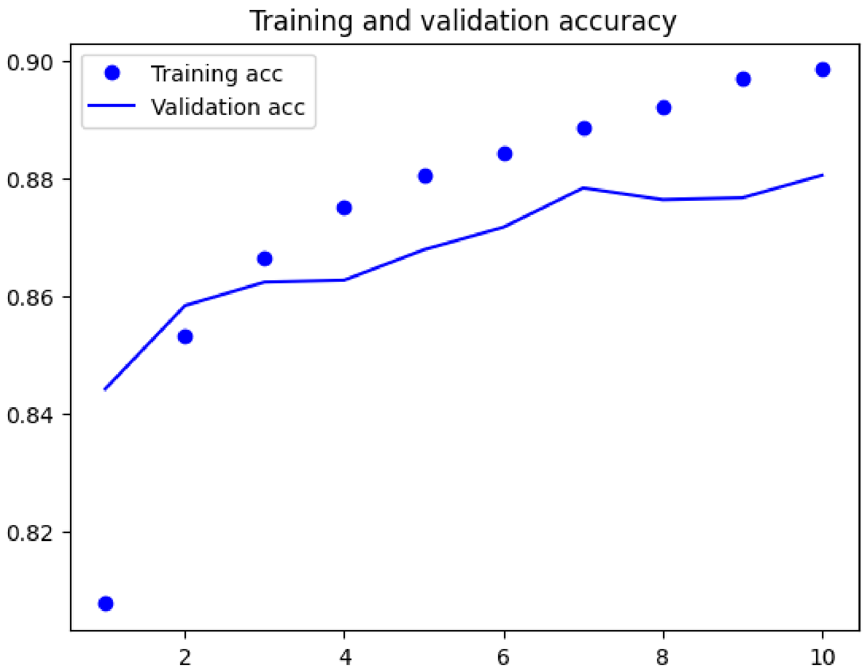
Precyzja

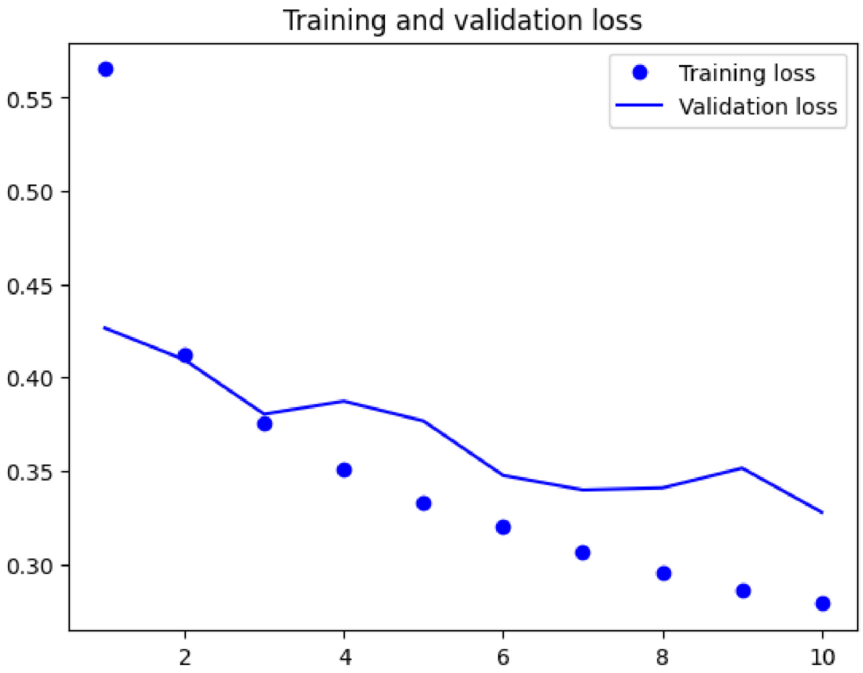
```
score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.8726999759674072
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.88
0.86
0.84
0.82
2
4
6
8
10

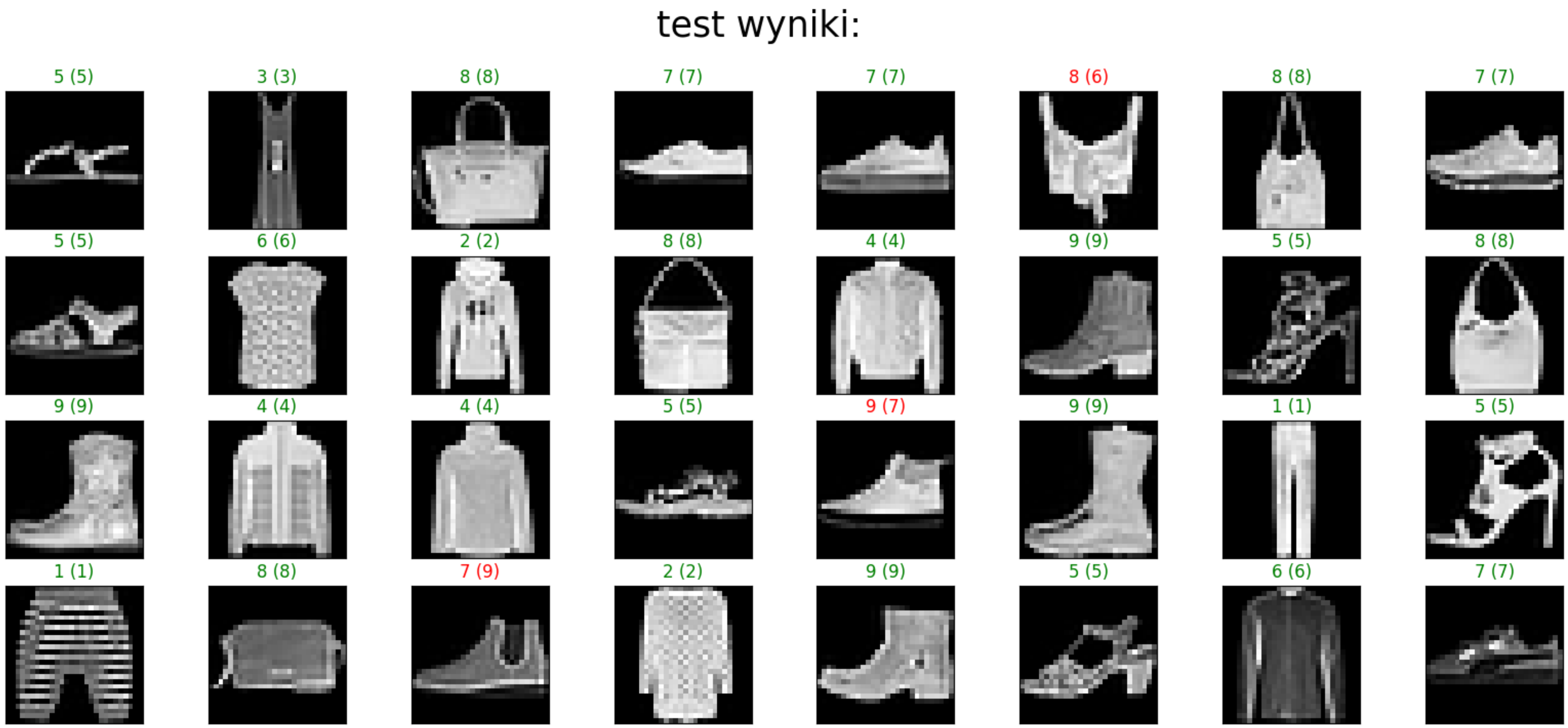

Training and validation loss
Training loss
Validation loss
0.55
0.50
0.45
0.40
0.35
0.30
2
4
6
8
10

Przewidywania modelu

Sprawdźmy jakie są przewidywania naszego modelu

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 1s 2ms/step
```



2.1 Sieć neuronowa v2

Dodajmy teraz warstwy wewnętrzne w naszej sieci. Funkcja aktywacji w takich warstwach to zwykle relu.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(64,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

model.summary()
opt = keras.optimizers.Adam(learning_rate=0.004)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy']) #learnig rate???

history = model.fit(x_train, y_train, batch_size = 64, epochs = 13, validation_data = (x_valid,y_valid))
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 784)               0

 dense_5 (Dense)             (None, 64)                50240

 dense_6 (Dense)             (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/13
844/844 [==============================] - 4s 4ms/step - loss: 0.4945 - accuracy: 0.8241 - val_loss: 0.3975 - val_accuracy: 0.8548
Epoch 2/13
844/844 [==============================] - 3s 3ms/step - loss: 0.3838 - accuracy: 0.8597 - val_loss: 0.3851 - val_accuracy: 0.8600
Epoch 3/13
844/844 [==============================] - 4s 4ms/step - loss: 0.3518 - accuracy: 0.8710 - val_loss: 0.3633 - val_accuracy: 0.8670
Epoch 4/13
844/844 [==============================] - 3s 3ms/step - loss: 0.3352 - accuracy: 0.8764 - val_loss: 0.3787 - val_accuracy: 0.8648
Epoch 5/13
844/844 [==============================] - 3s 3ms/step - loss: 0.3216 - accuracy: 0.8816 - val_loss: 0.3957 - val_accuracy: 0.8547
Epoch 6/13
844/844 [==============================] - 3s 3ms/step - loss: 0.3127 - accuracy: 0.8836 - val_loss: 0.3613 - val_accuracy: 0.8735
Epoch 7/13
844/844 [==============================] - 3s 3ms/step - loss: 0.2995 - accuracy: 0.8890 - val_loss: 0.3703 - val_accuracy: 0.8655
Epoch 8/13
844/844 [==============================] - 3s 4ms/step - loss: 0.2933 - accuracy: 0.8914 - val_loss: 0.3580 - val_accuracy: 0.8722
Epoch 9/13
844/844 [==============================] - 3s 3ms/step - loss: 0.2811 - accuracy: 0.8966 - val_loss: 0.3680 - val_accuracy: 0.8728
```

```
Epoch 10/13
844/844 [==============================] - 3s 3ms/step - loss: 0.2784 - accuracy: 0.8960 - val_loss: 0.3735 - val_accuracy: 0.8690
Epoch 11/13
844/844 [==============================] - 3s 3ms/step - loss: 0.2719 - accuracy: 0.8977 - val_loss: 0.3533 - val_accuracy: 0.8778
Epoch 12/13
844/844 [==============================] - 4s 4ms/step - loss: 0.2643 - accuracy: 0.9018 - val_loss: 0.3645 - val_accuracy: 0.8797
Epoch 13/13
844/844 [==============================] - 3s 3ms/step - loss: 0.2613 - accuracy: 0.9029 - val_loss: 0.3778 - val_accuracy: 0.8732
```

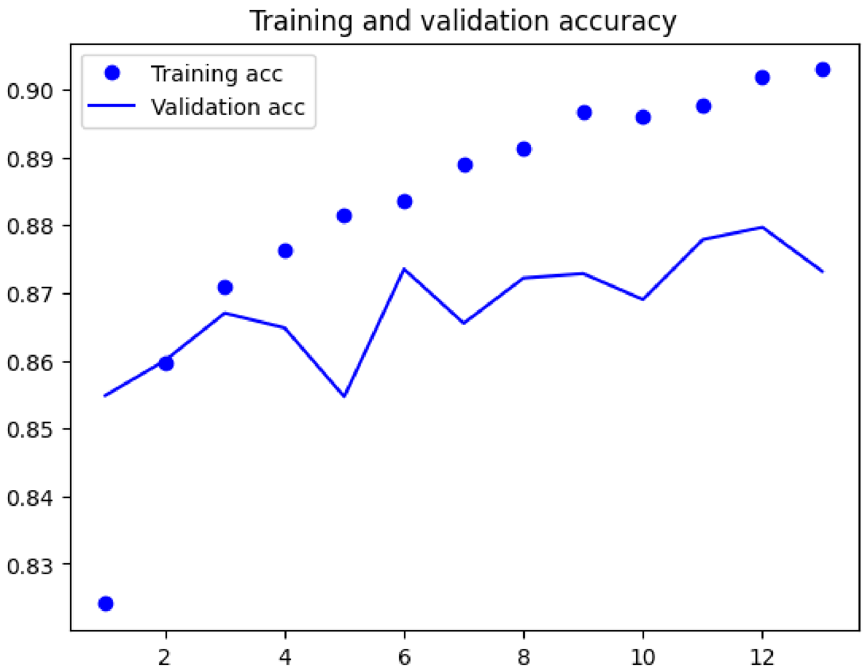
Precyzja

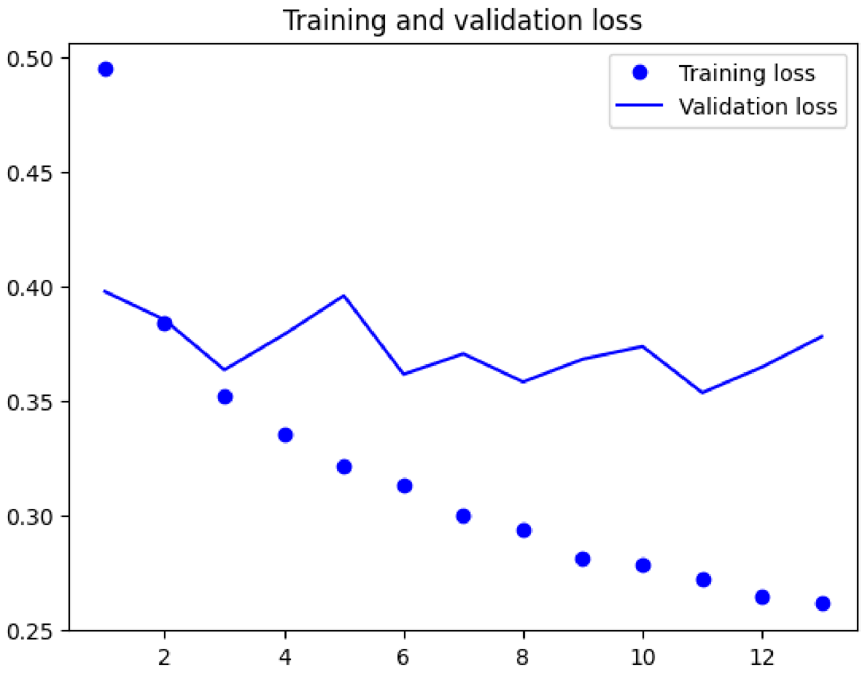
```
score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.8708999752998352
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0.83
2
4
6
8
10
12

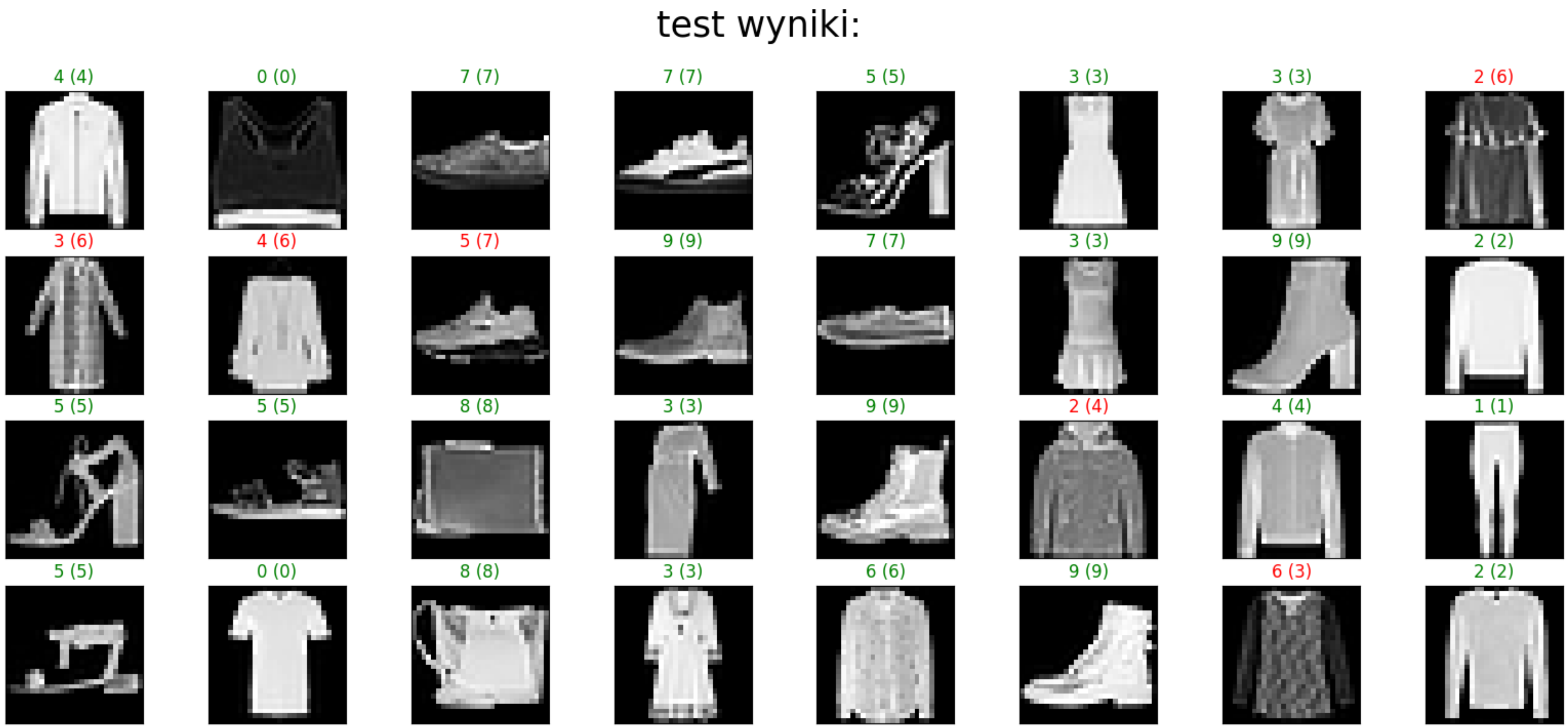

Training and validation loss
Training loss
Validation loss
0.50
0.45
0.40
0.35
0.30
0.25
2
4
6
8
10
12

Przewidywania modelu

Sprawdźmy jakie są przewidywania naszego modelu

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 1s 1ms/step
```



2.1 Sieć neuronowa v3

Dodajmy teraz warstwy wewnętrzne w naszej sieci. Funkcja aktywacji w takich warstwach to zwykle relu.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(64,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

model.summary()
opt = keras.optimizers.SGD(learning_rate=0.08)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, y_train, batch_size = 128, epochs = 50, validation_data = (x_valid,y_valid))
```

```
422/422 [==============================] - 1s 4ms/step - loss: 0.2412 - accuracy: 0.9128 - val_loss: 0.3240 - val_accuracy: 0.8840
Epoch 39/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2390 - accuracy: 0.9137 - val_loss: 0.3658 - val_accuracy: 0.8697
Epoch 40/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2376 - accuracy: 0.9130 - val_loss: 0.3291 - val_accuracy: 0.8818
Epoch 41/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2337 - accuracy: 0.9150 - val_loss: 0.3156 - val_accuracy: 0.8872
Epoch 42/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2321 - accuracy: 0.9164 - val_loss: 0.3318 - val_accuracy: 0.8818
Epoch 43/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2320 - accuracy: 0.9160 - val_loss: 0.3172 - val_accuracy: 0.8838
Epoch 44/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2293 - accuracy: 0.9178 - val_loss: 0.3276 - val_accuracy: 0.8843
Epoch 45/50
422/422 [==============================] - 2s 4ms/step - loss: 0.2277 - accuracy: 0.9174 - val_loss: 0.3395 - val_accuracy: 0.8842
Epoch 46/50
422/422 [==============================] - 2s 5ms/step - loss: 0.2249 - accuracy: 0.9197 - val_loss: 0.3272 - val_accuracy: 0.8855
Epoch 47/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2251 - accuracy: 0.9189 - val_loss: 0.3270 - val_accuracy: 0.8860
Epoch 48/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2217 - accuracy: 0.9207 - val_loss: 0.3282 - val_accuracy: 0.8855
Epoch 49/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2199 - accuracy: 0.9207 - val_loss: 0.3722 - val_accuracy: 0.8703
Epoch 50/50
422/422 [==============================] - 1s 3ms/step - loss: 0.2191 - accuracy: 0.9205 - val_loss: 0.3217 - val_accuracy: 0.8875
```

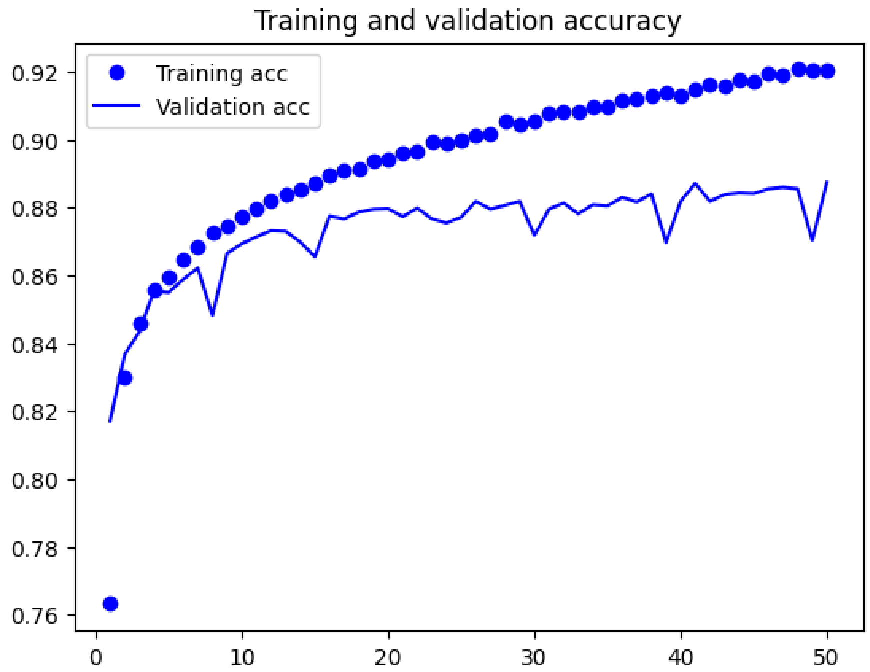
Precyzja

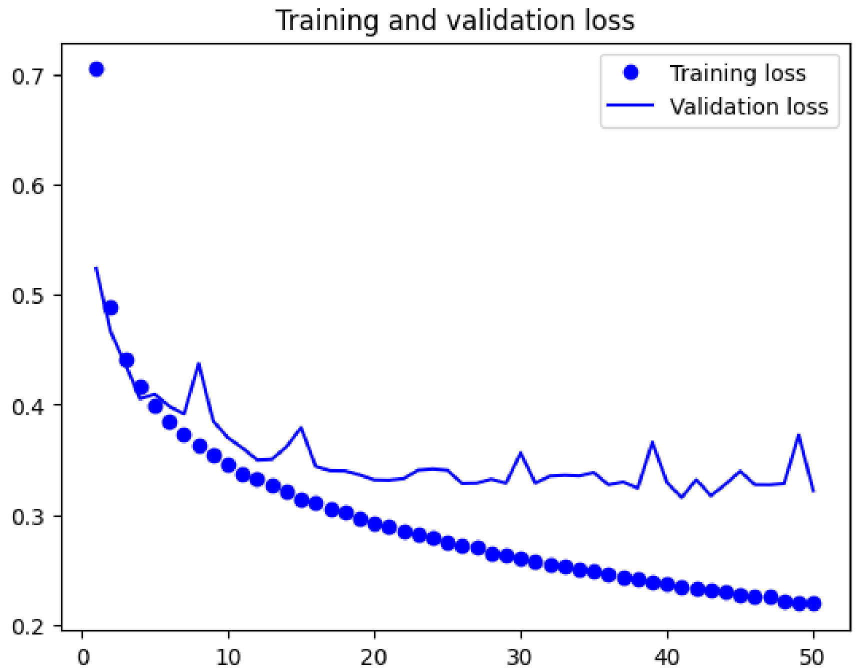
```
score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.8774999976158142
```

Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
10
20
30
40
50

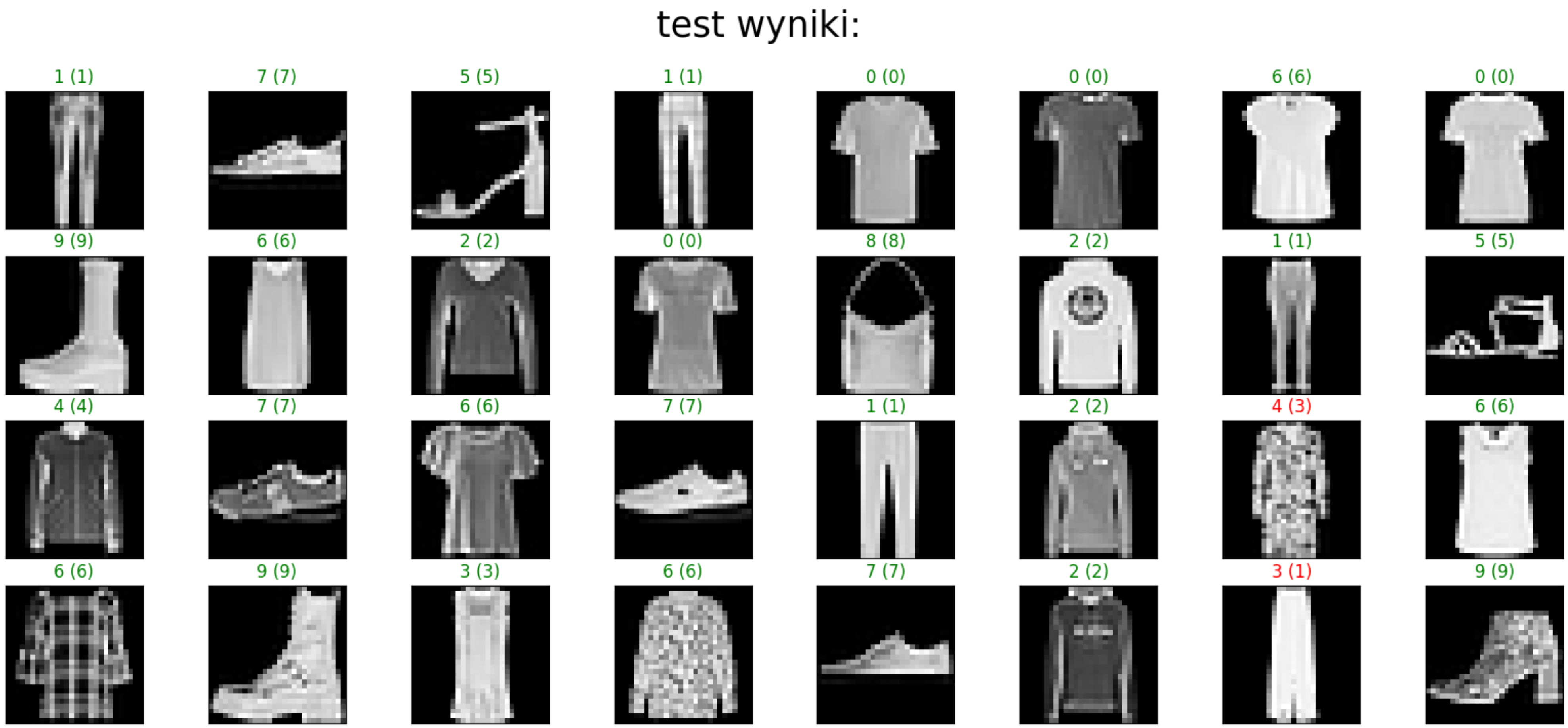

Training and validation loss
Training loss
Validation loss
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

Przewidywania modelu

Sprawdźmy jakie są przewidywania naszego modelu

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 1s 2ms/step
```



2.2 Spróbujmy pogłębić nasz model!

Dodajmy 3 warstwy gęste.

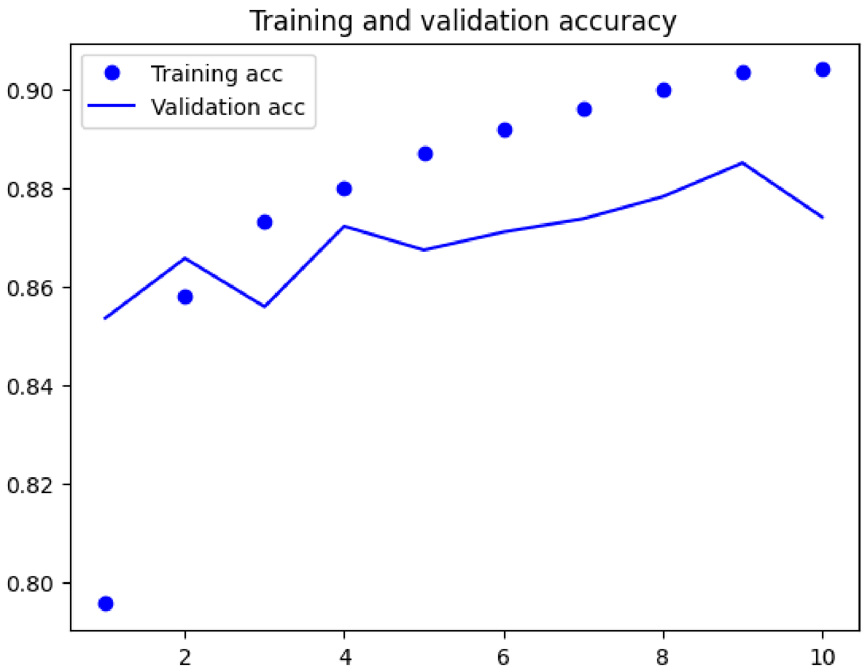
```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

history = model.fit(x_train, y_train, batch_size = 64, epochs = 10, validation_data = (x_valid,y_valid))

score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

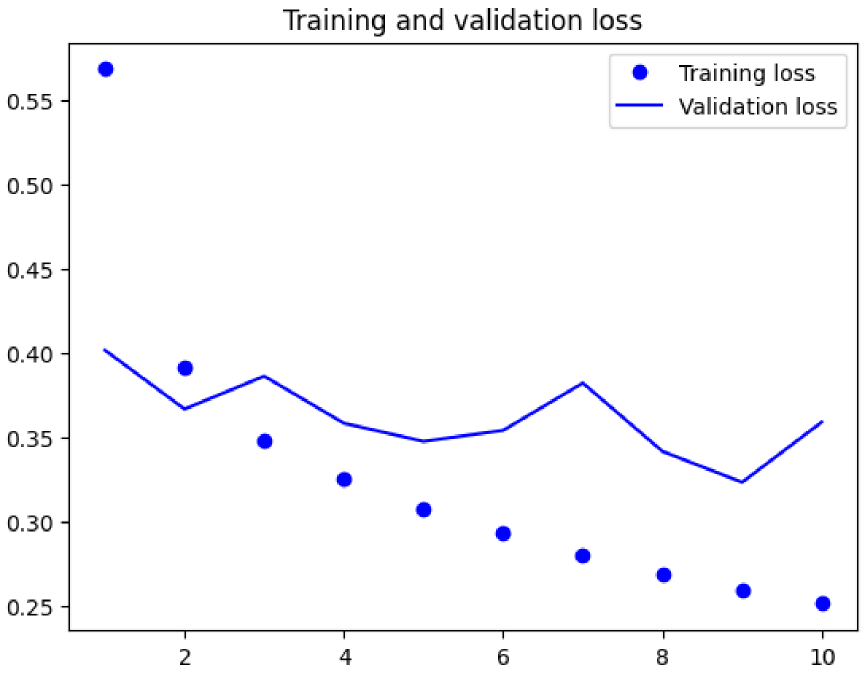
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_6 (Flatten)         (None, 784)               0         
                                                                 
 dense_9 (Dense)             (None, 128)               100480    
                                                                 
 dense_10 (Dense)            (None, 64)                8256      
                                                                 
 dense_11 (Dense)            (None, 64)                4160      
                                                                 
 dense_12 (Dense)            (None, 32)                2080      
                                                                 
 dense_13 (Dense)            (None, 10)                330       
                                                                 
=================================================================
Total params: 115306 (450.41 KB)
Trainable params: 115306 (450.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
844/844 [==============================] - 6s 5ms/step - loss: 0.5682 - accuracy: 0.7956 - val_loss: 0.4015 - val_accuracy: 0.8535
Epoch 2/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3907 - accuracy: 0.8580 - val_loss: 0.3665 - val_accuracy: 0.8657
Epoch 3/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3477 - accuracy: 0.8731 - val_loss: 0.3860 - val_accuracy: 0.8558
Epoch 4/10
844/844 [==============================] - 4s 5ms/step - loss: 0.3252 - accuracy: 0.8798 - val_loss: 0.3581 - val_accuracy: 0.8722
```

```
Epoch 5/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3068 - accuracy: 0.8869 - val_loss: 0.3473 - val_accuracy: 0.8673
Epoch 6/10
844/844 [==============================] - 3s 4ms/step - loss: 0.2932 - accuracy: 0.8919 - val_loss: 0.3538 - val_accuracy: 0.8710
Epoch 7/10
844/844 [==============================] - 4s 5ms/step - loss: 0.2794 - accuracy: 0.8959 - val_loss: 0.3819 - val_accuracy: 0.8737
Epoch 8/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2680 - accuracy: 0.9000 - val_loss: 0.3414 - val_accuracy: 0.8782
Epoch 9/10
844/844 [==============================] - 3s 4ms/step - loss: 0.2589 - accuracy: 0.9034 - val_loss: 0.3230 - val_accuracy: 0.8850
Epoch 10/10
844/844 [==============================] - 3s 4ms/step - loss: 0.2510 - accuracy: 0.9039 - val_loss: 0.3588 - val_accuracy: 0.8740
Precyzja:  0.8686000108718872
```

Wykresy precyzji i błędu

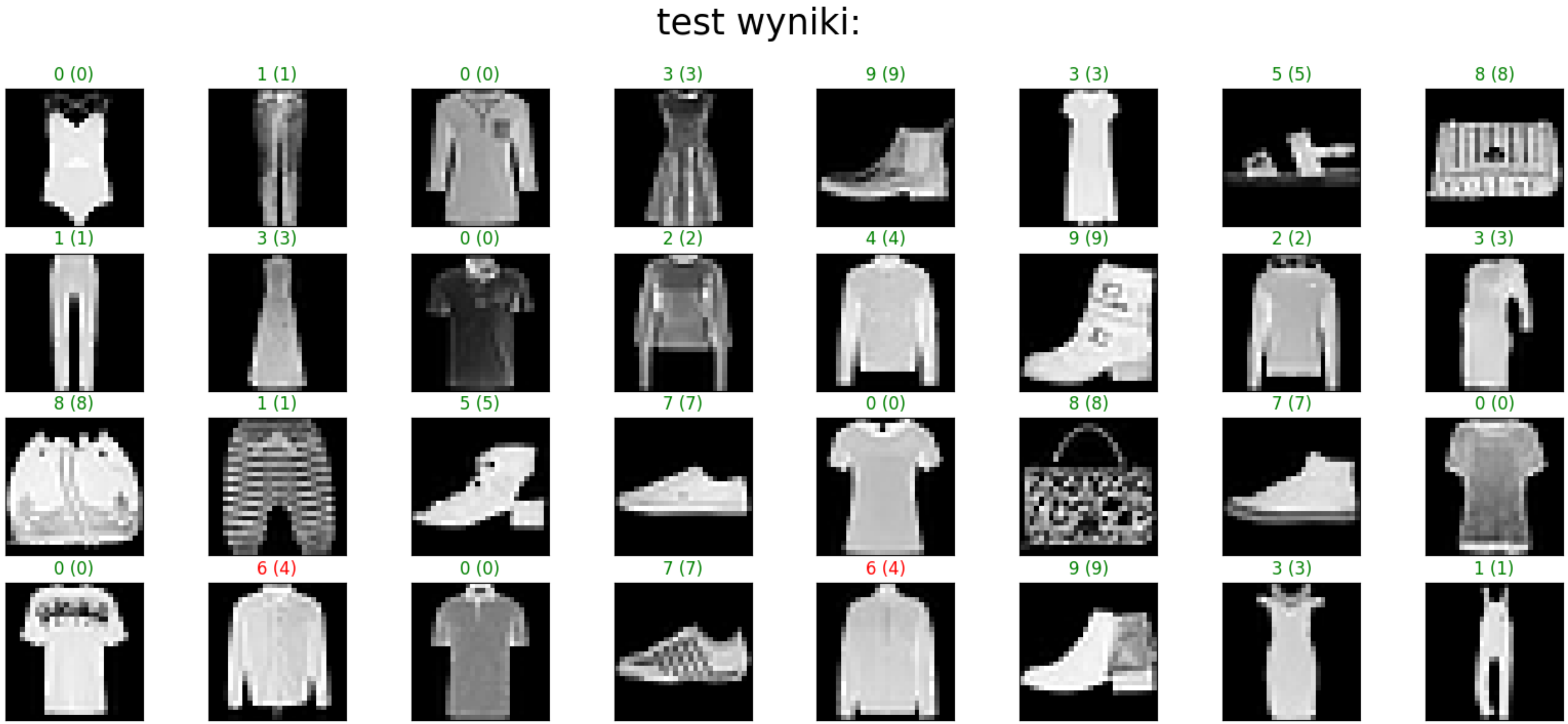
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.88
0.86
0.84
0.82
0.80
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.55
0.50
0.45
0.40
0.35
0.30
0.25
2
4
6
8
10

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 1s 2ms/step
```



2.2 Pogłębinienie modelu v2

Dodajmy 2 warstwy gęste.

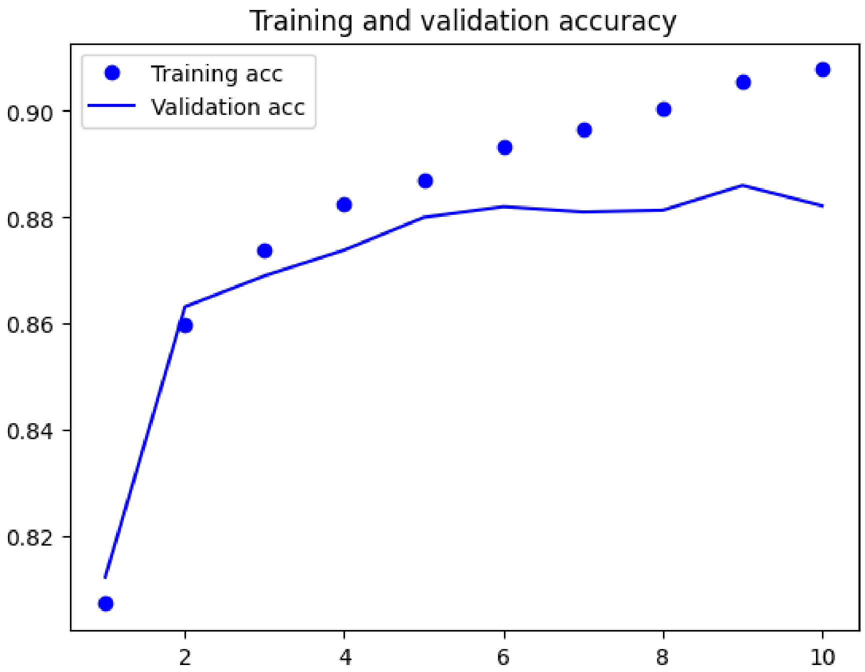
```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

history = model.fit(x_train, y_train, batch_size = 64, epochs = 10, validation_data = (x_valid,y_valid))

score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

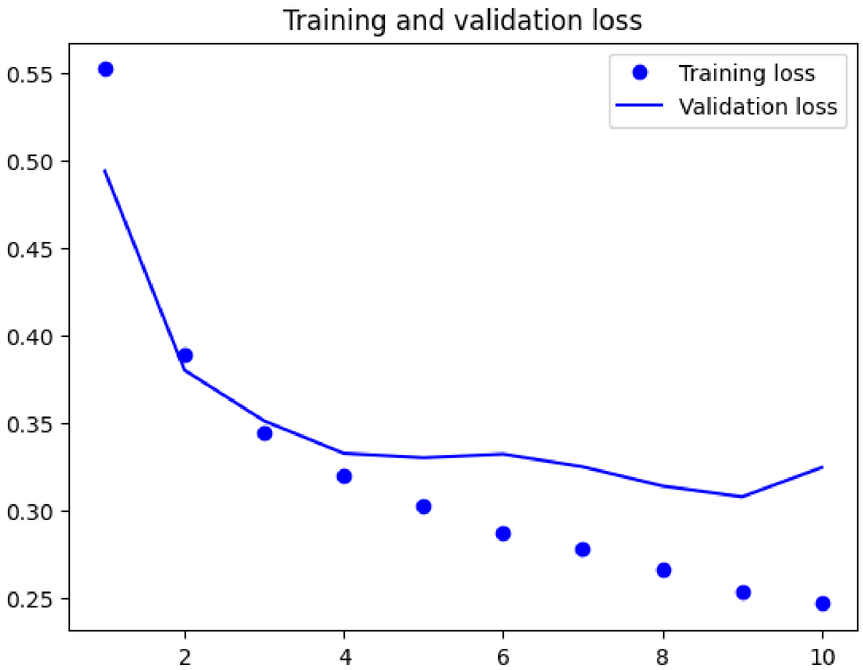
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_7 (Flatten)         (None, 784)               0

 dense_14 (Dense)            (None, 128)               100480

 dense_15 (Dense)            (None, 64)                8256

 dense_16 (Dense)            (None, 32)                2080

 dense_17 (Dense)            (None, 10)                330

=================================================================
Total params: 111146 (434.16 KB)
Trainable params: 111146 (434.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
844/844 [==============================] - 5s 4ms/step - loss: 0.5522 - accuracy: 0.8073 - val_loss: 0.4940 - val_accuracy: 0.8123
Epoch 2/10
844/844 [==============================] - 4s 5ms/step - loss: 0.3888 - accuracy: 0.8599 - val_loss: 0.3804 - val_accuracy: 0.8632
Epoch 3/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3449 - accuracy: 0.8739 - val_loss: 0.3514 - val_accuracy: 0.8690
Epoch 4/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3200 - accuracy: 0.8824 - val_loss: 0.3328 - val_accuracy: 0.8738
Epoch 5/10
844/844 [==============================] - 4s 4ms/step - loss: 0.3033 - accuracy: 0.8869 - val_loss: 0.3304 - val_accuracy: 0.8800
Epoch 6/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2872 - accuracy: 0.8934 - val_loss: 0.3324 - val_accuracy: 0.8820
Epoch 7/10
```

```
844/844 [==============================] - 3s 4ms/step - loss: 0.2787 - accuracy: 0.8966 - val_loss: 0.3252 - val_accuracy: 0.8810
Epoch 8/10
844/844 [==============================] - 3s 4ms/step - loss: 0.2666 - accuracy: 0.9005 - val_loss: 0.3142 - val_accuracy: 0.8813
Epoch 9/10
844/844 [==============================] - 4s 5ms/step - loss: 0.2534 - accuracy: 0.9055 - val_loss: 0.3081 - val_accuracy: 0.8860
Epoch 10/10
844/844 [==============================] - 4s 5ms/step - loss: 0.2471 - accuracy: 0.9078 - val_loss: 0.3248 - val_accuracy: 0.8822
Precyzja:  0.8751999735832214
```

Wykresy precyzji i błędu

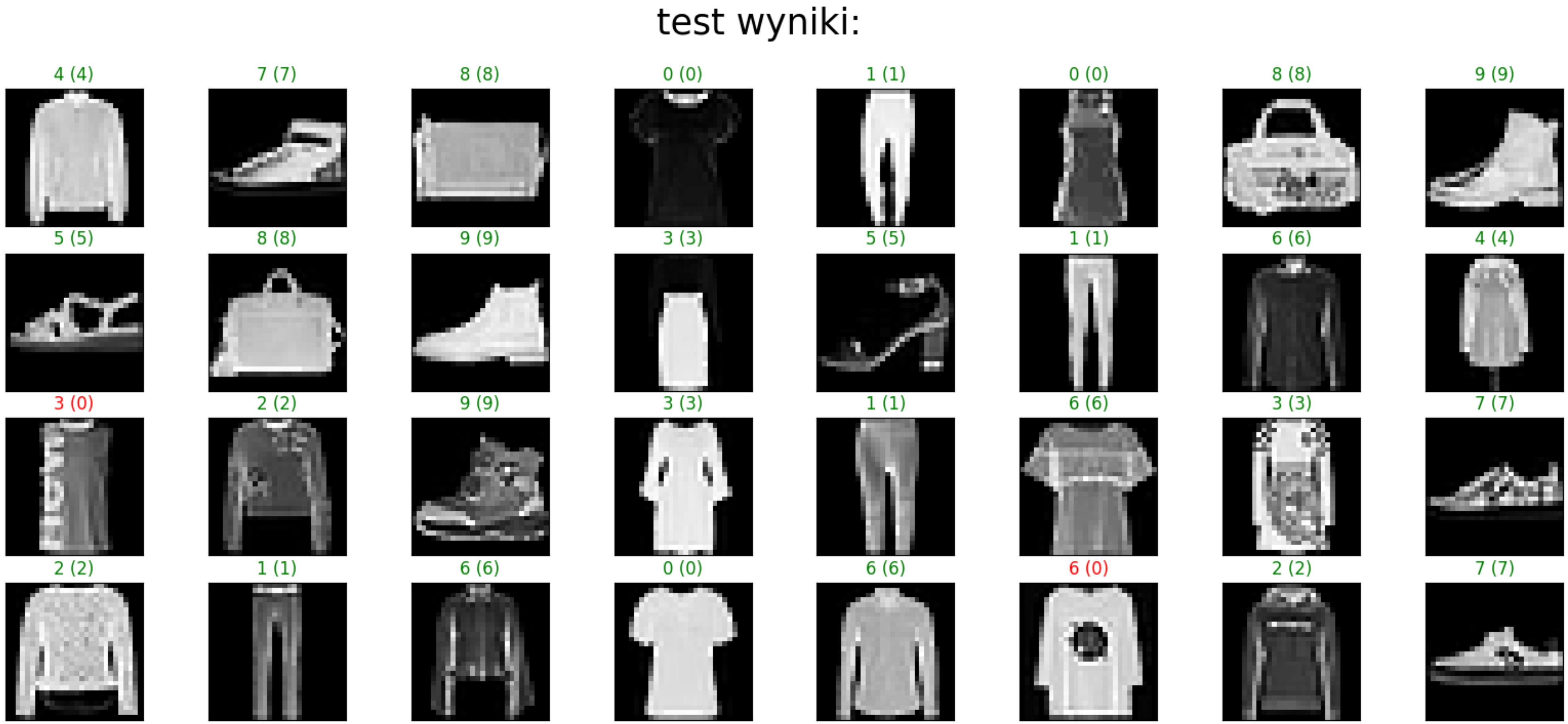
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.88
0.86
0.84
0.82
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.55
0.50
0.45
0.40
0.35
0.30
0.25
2
4
6
8
10

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 2s 4ms/step
```



2.2 Pogłębiamy model v3

Dodajmy 4 warstwy gęste.

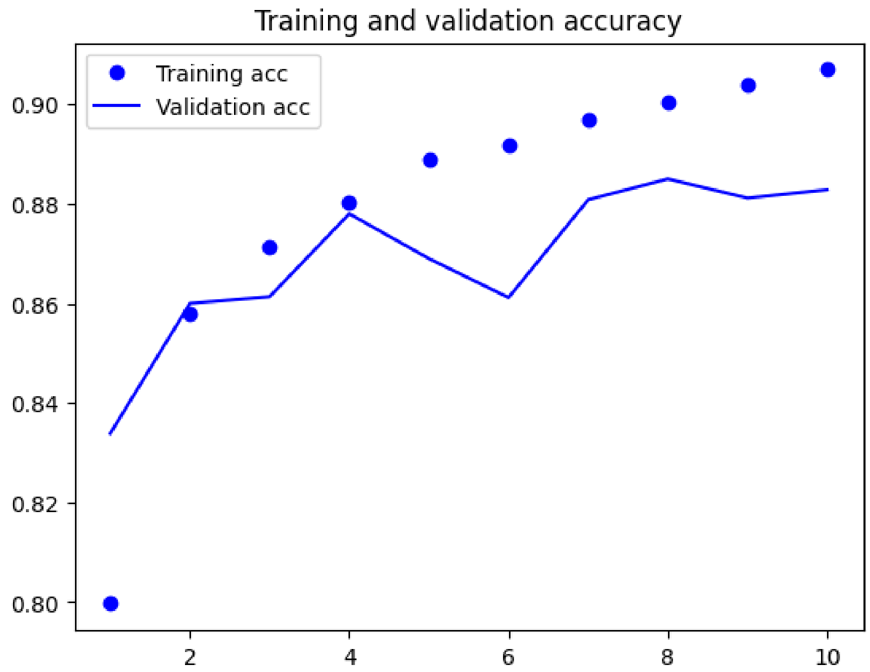
```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

history = model.fit(x_train, y_train, batch_size = 64, epochs = 10, validation_data = (x_valid,y_valid))

score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

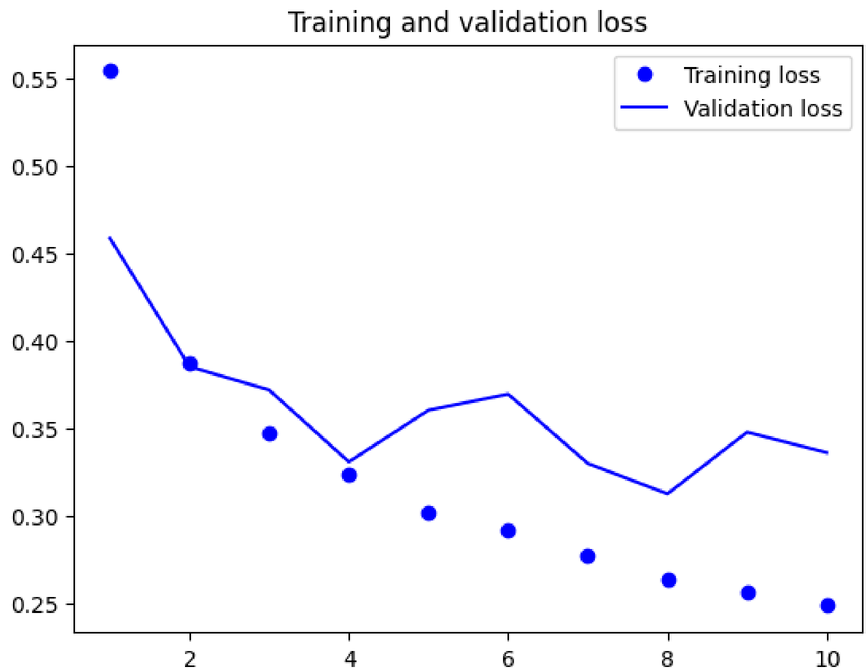
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_8 (Flatten)         (None, 784)               0

 dense_18 (Dense)            (None, 128)               100480

 dense_19 (Dense)            (None, 128)               16512

 dense_20 (Dense)            (None, 64)                8256

 dense_21 (Dense)            (None, 64)                4160

 dense_22 (Dense)            (None, 32)                2080

 dense_23 (Dense)            (None, 10)                330

=================================================================
Total params: 131818 (514.91 KB)
Trainable params: 131818 (514.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
844/844 [==============================] - 5s 4ms/step - loss: 0.5538 - accuracy: 0.7996 - val_loss: 0.4585 - val_accuracy: 0.8338
Epoch 2/10
844/844 [==============================] - 4s 5ms/step - loss: 0.3875 - accuracy: 0.8581 - val_loss: 0.3851 - val_accuracy: 0.8600
Epoch 3/10
844/844 [==============================] - 3s 4ms/step - loss: 0.3475 - accuracy: 0.8713 - val_loss: 0.3718 - val_accuracy: 0.8613
Epoch 4/10
```

```
844/844 [==============================] - 3s 4ms/step - loss: 0.3237 - accuracy: 0.8804 - val_loss: 0.3308 - val_accuracy: 0.8780
Epoch 5/10
844/844 [==============================] - 4s 5ms/step - loss: 0.3020 - accuracy: 0.8889 - val_loss: 0.3603 - val_accuracy: 0.8690
Epoch 6/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2914 - accuracy: 0.8919 - val_loss: 0.3694 - val_accuracy: 0.8612
Epoch 7/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2771 - accuracy: 0.8968 - val_loss: 0.3299 - val_accuracy: 0.8808
Epoch 8/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2637 - accuracy: 0.9004 - val_loss: 0.3125 - val_accuracy: 0.8850
Epoch 9/10
844/844 [==============================] - 4s 5ms/step - loss: 0.2561 - accuracy: 0.9039 - val_loss: 0.3477 - val_accuracy: 0.8812
Epoch 10/10
844/844 [==============================] - 3s 4ms/step - loss: 0.2487 - accuracy: 0.9070 - val_loss: 0.3360 - val_accuracy: 0.8828
Precyzja:  0.8799999952316284
```

Wykresy precyzji i błędu

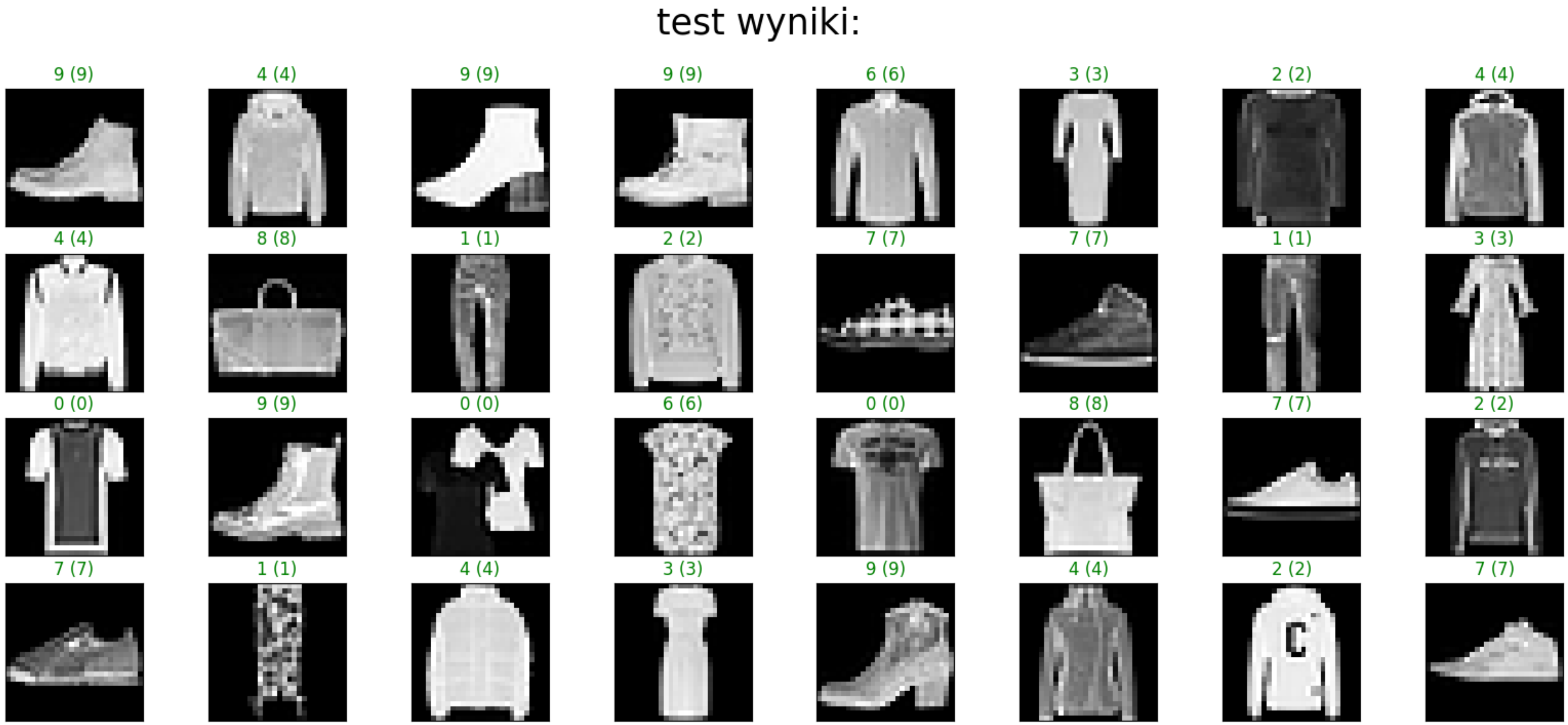
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.88
0.86
0.84
0.82
0.80
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.55
0.50
0.45
0.40
0.35
0.30
0.25
2
4
6
8
10

```
visualize_model_predictions(model, x_test, y_test, "test" )
```

```
313/313 [==============================] - 1s 2ms/step
```



2.3 Konwolucja

W dotychczasowych przykładach przed warstwą gęstą dodawaliśmy warstę płaską. Tutaj będzie podobnie, ale przed warstwą płaską dodamy warstwy konwolucyjne i maxpool.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=4, kernel_size=2,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Conv2D(filters=2, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

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

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 4)         20        
                                                                 
 max_pooling2d (MaxPooling2  (None, 14, 14, 4)         0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 14, 14, 2)         34        
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 2)           0         
 g2D)                                                            
                                                                 
 flatten_9 (Flatten)         (None, 98)                0         
                                                                 
 dense_24 (Dense)            (None, 128)               12672     
                                                                 
 dense_25 (Dense)            (None, 64)                8256      
                                                                 
 dense_26 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 21632 (84.50 KB)
Trainable params: 21632 (84.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(x_train, y_train, batch_size=128, epochs=25, validation_data=(x_valid, y_valid))
```

```
Epoch 1/25
422/422 [==============================] - 6s 5ms/step - loss: 0.8460 - accuracy: 0.6909 - val_loss: 0.5659 - val_accuracy: 0.7833
Epoch 2/25
```

```
422/422 [==============================] - 2s 4ms/step - loss: 0.5310 - accuracy: 0.8006 - val_loss: 0.4907 - val_accuracy: 0.8155
Epoch 3/25
422/422 [==============================] - 2s 4ms/step - loss: 0.4643 - accuracy: 0.8292 - val_loss: 0.4464 - val_accuracy: 0.8360
Epoch 4/25
422/422 [==============================] - 2s 4ms/step - loss: 0.4304 - accuracy: 0.8422 - val_loss: 0.4248 - val_accuracy: 0.8415
Epoch 5/25
422/422 [==============================] - 2s 4ms/step - loss: 0.4082 - accuracy: 0.8496 - val_loss: 0.4117 - val_accuracy: 0.8502
Epoch 6/25
422/422 [==============================] - 2s 5ms/step - loss: 0.3921 - accuracy: 0.8572 - val_loss: 0.3825 - val_accuracy: 0.8593
Epoch 7/25
422/422 [==============================] - 2s 6ms/step - loss: 0.3769 - accuracy: 0.8616 - val_loss: 0.3723 - val_accuracy: 0.8607
Epoch 8/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3629 - accuracy: 0.8662 - val_loss: 0.3712 - val_accuracy: 0.8628
Epoch 9/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3506 - accuracy: 0.8703 - val_loss: 0.3608 - val_accuracy: 0.8635
Epoch 10/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3402 - accuracy: 0.8727 - val_loss: 0.3554 - val_accuracy: 0.8702
Epoch 11/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3330 - accuracy: 0.8768 - val_loss: 0.3616 - val_accuracy: 0.8672
Epoch 12/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3254 - accuracy: 0.8794 - val_loss: 0.3630 - val_accuracy: 0.8658
Epoch 13/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3177 - accuracy: 0.8810 - val_loss: 0.3492 - val_accuracy: 0.8723
Epoch 14/25
422/422 [==============================] - 3s 6ms/step - loss: 0.3125 - accuracy: 0.8829 - val_loss: 0.3430 - val_accuracy: 0.8733
Epoch 15/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3064 - accuracy: 0.8849 - val_loss: 0.3284 - val_accuracy: 0.8758
Epoch 16/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2992 - accuracy: 0.8893 - val_loss: 0.3327 - val_accuracy: 0.8757
Epoch 17/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2923 - accuracy: 0.8909 - val_loss: 0.3256 - val_accuracy: 0.8778
Epoch 18/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2900 - accuracy: 0.8908 - val_loss: 0.3242 - val_accuracy: 0.8797
Epoch 19/25
422/422 [==============================] - 2s 5ms/step - loss: 0.2844 - accuracy: 0.8936 - val_loss: 0.3248 - val_accuracy: 0.8775
Epoch 20/25
422/422 [==============================] - 2s 5ms/step - loss: 0.2782 - accuracy: 0.8963 - val_loss: 0.3130 - val_accuracy: 0.8817
Epoch 21/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2759 - accuracy: 0.8971 - val_loss: 0.3142 - val_accuracy: 0.8853
Epoch 22/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2712 - accuracy: 0.8978 - val_loss: 0.3178 - val_accuracy: 0.8813
Epoch 23/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2672 - accuracy: 0.8993 - val_loss: 0.3177 - val_accuracy: 0.8837
Epoch 24/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2667 - accuracy: 0.8993 - val_loss: 0.3158 - val_accuracy: 0.8863
Epoch 25/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2621 - accuracy: 0.9018 - val_loss: 0.3237 - val_accuracy: 0.8765
```

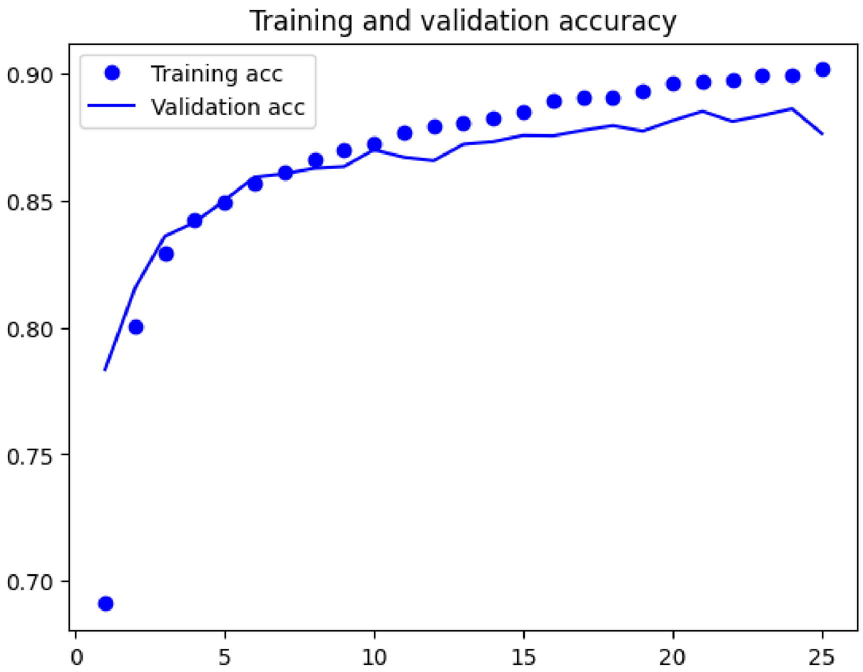
Ewaluacja modelu

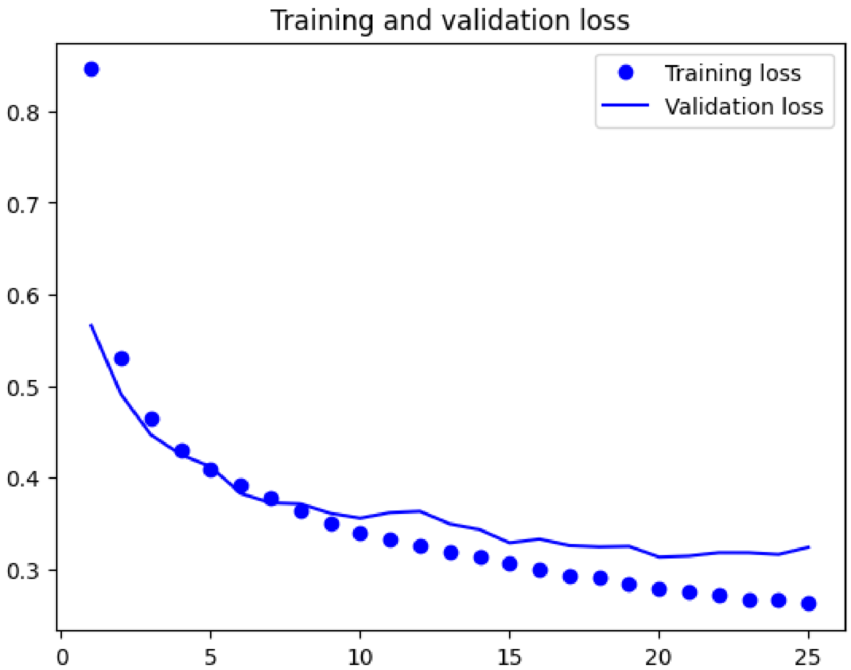
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.8812000155448914
```

Wykresy precyzji i błędu

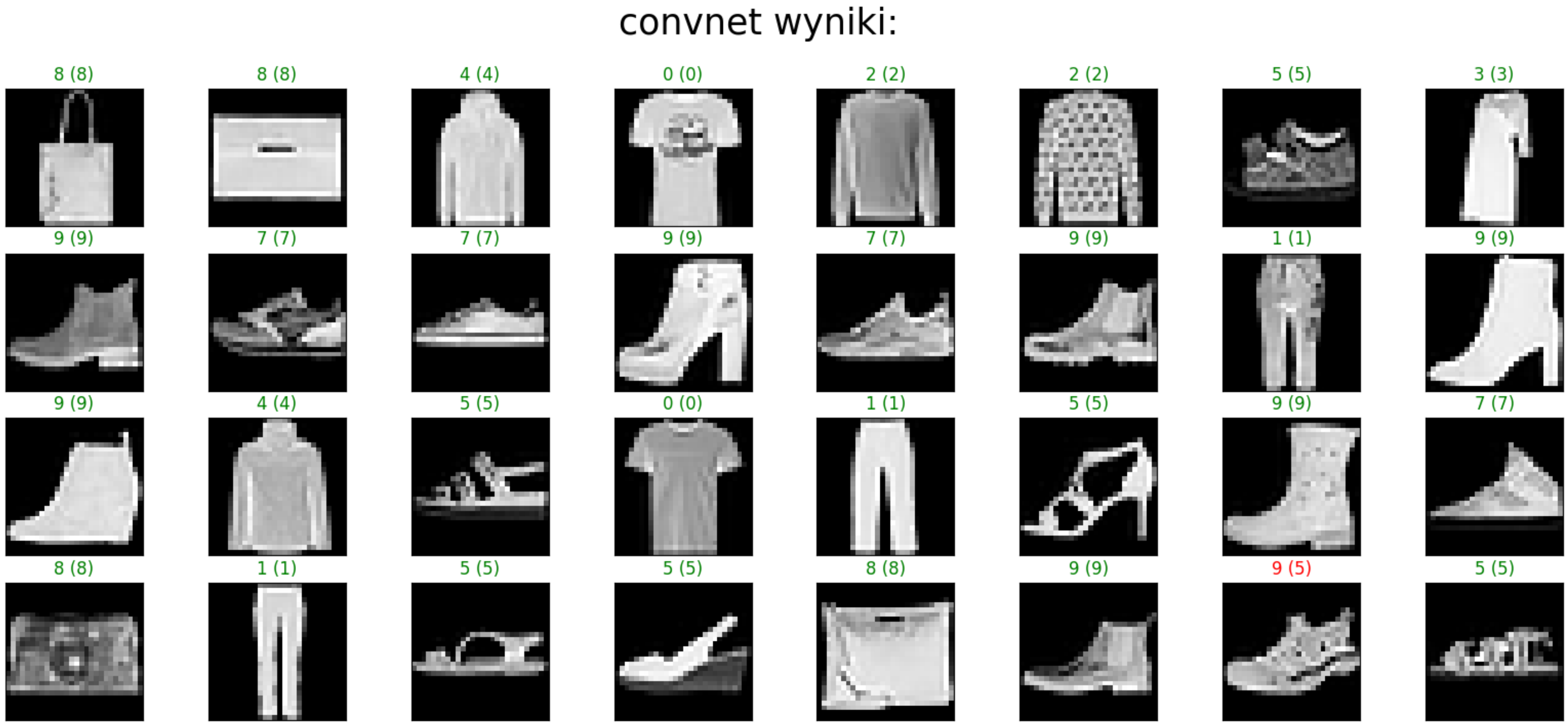
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.85
0.80
0.75
0.70
0
5
10
15
20
25


Training and validation loss
Training loss
Validation loss
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25

```
visualize_model_predictions(model, x_test, y_test,"convnet")
```

```
313/313 [==============================] - 1s 3ms/step
```



2.3 Konwolucja model v2

W dotychczasowych przykładach przed warstwą gęstą dodawaliśmy warstę płaską. Tutaj będzie podobnie, ale przed warstwą płaską dodamy warstwy konwolucyjne i maxpool.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=4, kernel_size=2,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Conv2D(filters=2, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Conv2D(filters=2, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

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

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 4)         20

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 4)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 14, 14, 2)         34

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 2)           0
 g2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 2)           18

 max_pooling2d_4 (MaxPoolin  (None, 3, 3, 2)           0
 g2D)

 flatten_10 (Flatten)        (None, 18)                0

 dense_27 (Dense)            (None, 128)               2432

 dense_28 (Dense)            (None, 64)                8256

 dense_29 (Dense)            (None, 10)                650
```

```
=================================================================
Total params: 11410 (44.57 KB)
Trainable params: 11410 (44.57 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(x_train, y_train, batch_size=128, epochs=25, validation_data=(x_valid, y_valid))
```

```
Epoch 1/25
422/422 [==============================] - 4s 5ms/step - loss: 1.1637 - accuracy: 0.5790 - val_loss: 0.7959 - val_accuracy: 0.6807
Epoch 2/25
422/422 [==============================] - 2s 5ms/step - loss: 0.7650 - accuracy: 0.7042 - val_loss: 0.6997 - val_accuracy: 0.7293
Epoch 3/25
422/422 [==============================] - 2s 5ms/step - loss: 0.6898 - accuracy: 0.7345 - val_loss: 0.6478 - val_accuracy: 0.7452
Epoch 4/25
422/422 [==============================] - 3s 6ms/step - loss: 0.6401 - accuracy: 0.7561 - val_loss: 0.5995 - val_accuracy: 0.7725
Epoch 5/25
422/422 [==============================] - 2s 5ms/step - loss: 0.6076 - accuracy: 0.7702 - val_loss: 0.5994 - val_accuracy: 0.7697
Epoch 6/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5841 - accuracy: 0.7803 - val_loss: 0.5755 - val_accuracy: 0.7847
Epoch 7/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5668 - accuracy: 0.7889 - val_loss: 0.5513 - val_accuracy: 0.7888
Epoch 8/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5510 - accuracy: 0.7934 - val_loss: 0.5310 - val_accuracy: 0.8002
Epoch 9/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5404 - accuracy: 0.7985 - val_loss: 0.5261 - val_accuracy: 0.7992
Epoch 10/25
422/422 [==============================] - 2s 6ms/step - loss: 0.5260 - accuracy: 0.8053 - val_loss: 0.5091 - val_accuracy: 0.8112
Epoch 11/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5172 - accuracy: 0.8079 - val_loss: 0.5029 - val_accuracy: 0.8143
Epoch 12/25
422/422 [==============================] - 2s 5ms/step - loss: 0.5076 - accuracy: 0.8123 - val_loss: 0.5075 - val_accuracy: 0.8095
Epoch 13/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4981 - accuracy: 0.8141 - val_loss: 0.5007 - val_accuracy: 0.8112
Epoch 14/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4893 - accuracy: 0.8189 - val_loss: 0.4824 - val_accuracy: 0.8185
Epoch 15/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4806 - accuracy: 0.8219 - val_loss: 0.4747 - val_accuracy: 0.8210
Epoch 16/25
422/422 [==============================] - 3s 6ms/step - loss: 0.4753 - accuracy: 0.8247 - val_loss: 0.4730 - val_accuracy: 0.8222
Epoch 17/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4711 - accuracy: 0.8226 - val_loss: 0.4662 - val_accuracy: 0.8267
Epoch 18/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4608 - accuracy: 0.8279 - val_loss: 0.4752 - val_accuracy: 0.8212
Epoch 19/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4599 - accuracy: 0.8277 - val_loss: 0.4673 - val_accuracy: 0.8235
Epoch 20/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4538 - accuracy: 0.8313 - val_loss: 0.4557 - val_accuracy: 0.8322
Epoch 21/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4494 - accuracy: 0.8326 - val_loss: 0.4492 - val_accuracy: 0.8372
```

```
Epoch 22/25
422/422 [==============================] - 3s 6ms/step - loss: 0.4456 - accuracy: 0.8330 - val_loss: 0.4507 - val_accuracy: 0.8318
Epoch 23/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4417 - accuracy: 0.8341 - val_loss: 0.4515 - val_accuracy: 0.8302
Epoch 24/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4376 - accuracy: 0.8367 - val_loss: 0.4476 - val_accuracy: 0.8372
Epoch 25/25
422/422 [==============================] - 2s 5ms/step - loss: 0.4344 - accuracy: 0.8360 - val_loss: 0.4490 - val_accuracy: 0.8320
```

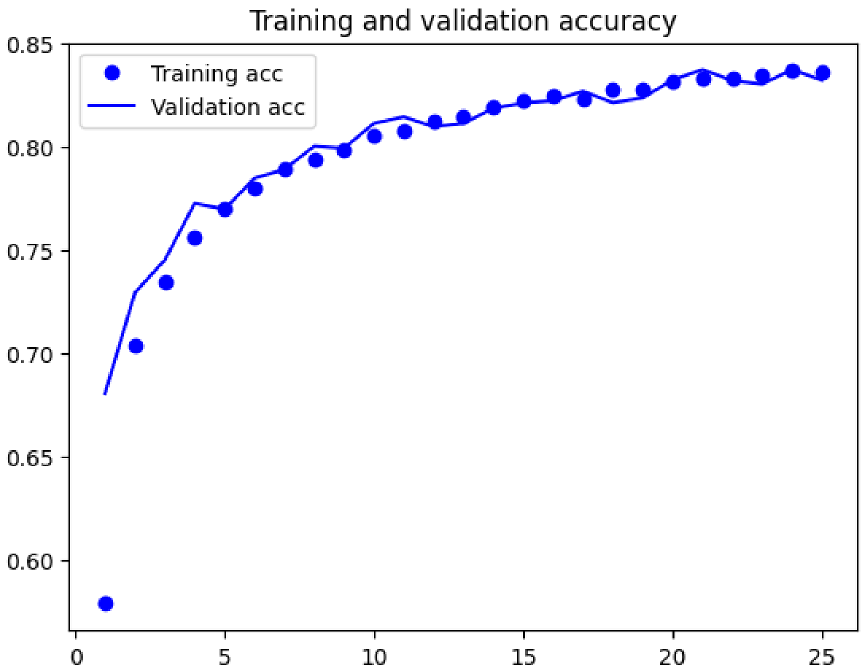
Ewaluacja modelu

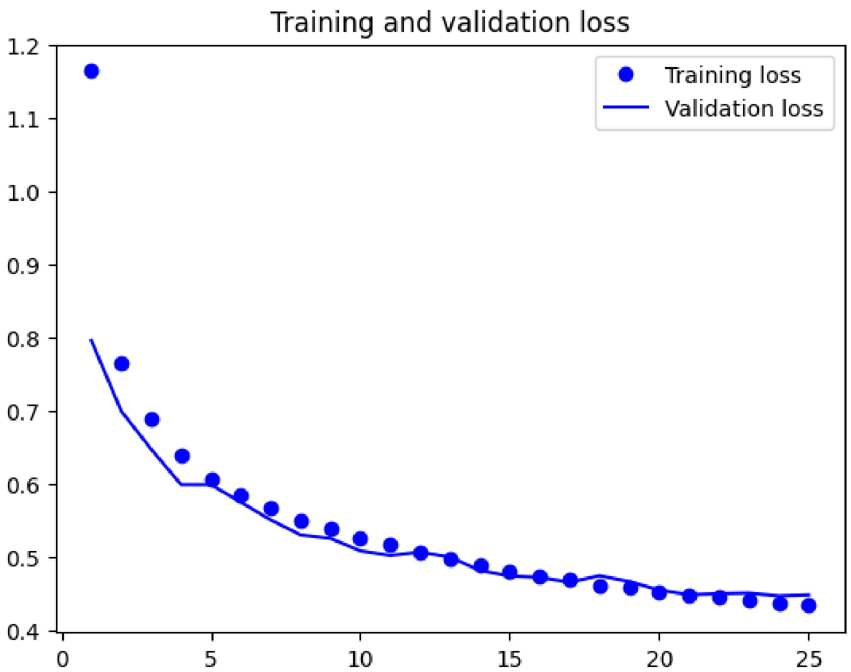
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.8307999968528748
```

Wykresy precyzji i błędu

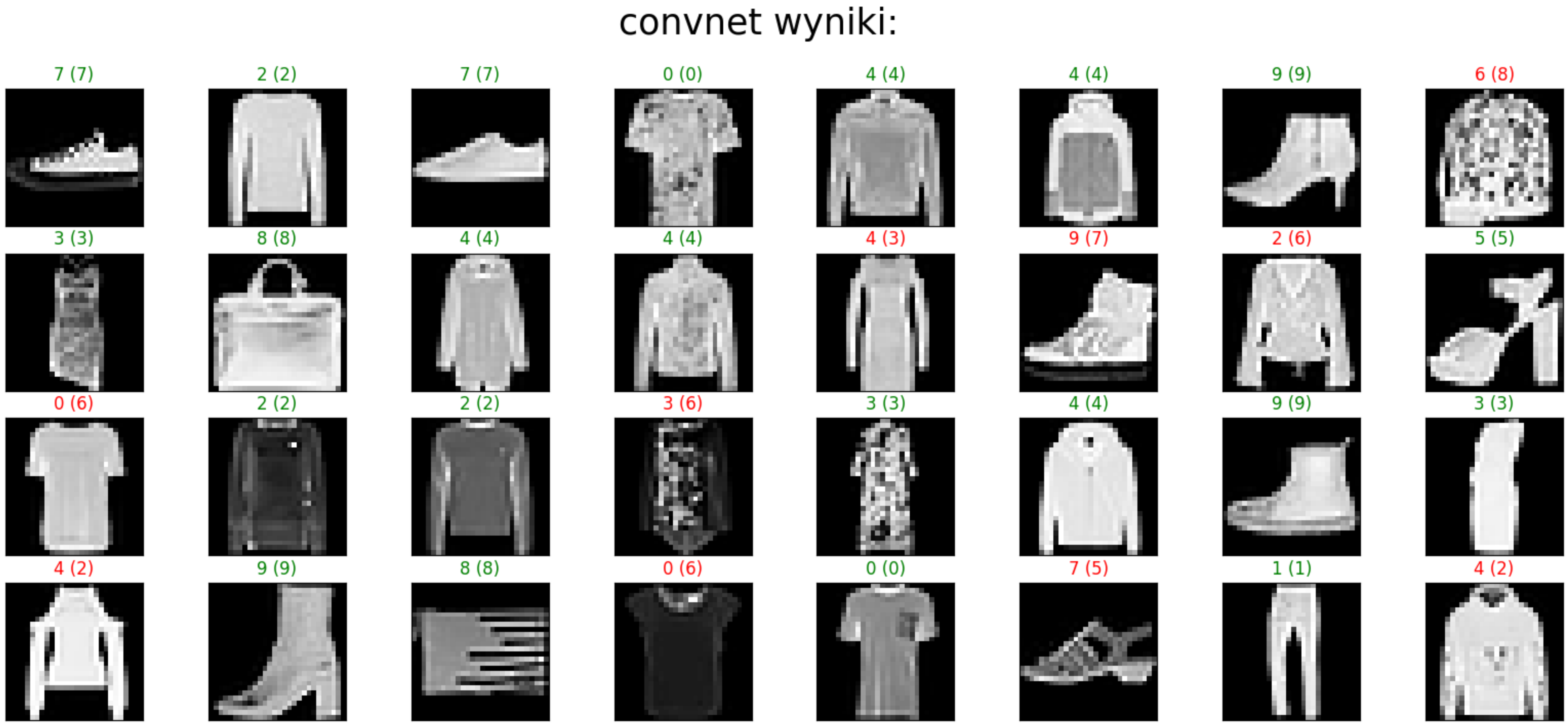
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25


Training and validation loss
Training loss
Validation loss
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25

```
visualize_model_predictions(model, x_test, y_test,"convnet")
```

```
313/313 [==============================] - 1s 2ms/step
```



2.3 Konwolucja model v3

W dotychczasowych przykładach przed warstwą gęstą dodawaliśmy warstę płaską. Tutaj będzie podobnie, ale przed warstwą płaską dodamy warstwy konwolucyjne i maxpool.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))


model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

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

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 64)        320       
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 14, 14, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_6 (Conv2D)           (None, 14, 14, 32)        8224      
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 7, 7, 32)          0         
 g2D)                                                            
                                                                 
 conv2d_7 (Conv2D)           (None, 7, 7, 16)          2064      
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 3, 3, 16)          0         
 g2D)                                                            
                                                                 
 flatten_11 (Flatten)        (None, 144)               0         
                                                                 
 dense_30 (Dense)            (None, 128)               18560     
                                                                 
 dense_31 (Dense)            (None, 64)                8256      
                                                                 
 dense_32 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 38074 (148.73 KB)
Trainable params: 38074 (148.73 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(x_train, y_train, batch_size=128, epochs=25, validation_data=(x_valid, y_valid))
```

```
Epoch 1/25
422/422 [==============================] - 5s 7ms/step - loss: 0.7195 - accuracy: 0.7354 - val_loss: 0.4585 - val_accuracy: 0.8332
Epoch 2/25
422/422 [==============================] - 3s 7ms/step - loss: 0.4166 - accuracy: 0.8487 - val_loss: 0.4034 - val_accuracy: 0.8548
Epoch 3/25
422/422 [==============================] - 3s 6ms/step - loss: 0.3568 - accuracy: 0.8701 - val_loss: 0.3831 - val_accuracy: 0.8572
Epoch 4/25
422/422 [==============================] - 2s 6ms/step - loss: 0.3261 - accuracy: 0.8821 - val_loss: 0.3415 - val_accuracy: 0.8787
Epoch 5/25
422/422 [==============================] - 2s 6ms/step - loss: 0.3085 - accuracy: 0.8870 - val_loss: 0.3166 - val_accuracy: 0.8847
Epoch 6/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2871 - accuracy: 0.8949 - val_loss: 0.2970 - val_accuracy: 0.8910
Epoch 7/25
422/422 [==============================] - 3s 7ms/step - loss: 0.2742 - accuracy: 0.8987 - val_loss: 0.2798 - val_accuracy: 0.9002
Epoch 8/25
422/422 [==============================] - 3s 6ms/step - loss: 0.2630 - accuracy: 0.9030 - val_loss: 0.2810 - val_accuracy: 0.8962
Epoch 9/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2528 - accuracy: 0.9063 - val_loss: 0.2711 - val_accuracy: 0.8985
Epoch 10/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2439 - accuracy: 0.9103 - val_loss: 0.2829 - val_accuracy: 0.8985
Epoch 11/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2354 - accuracy: 0.9130 - val_loss: 0.2660 - val_accuracy: 0.9013
Epoch 12/25
422/422 [==============================] - 3s 7ms/step - loss: 0.2300 - accuracy: 0.9159 - val_loss: 0.2536 - val_accuracy: 0.9092
Epoch 13/25
422/422 [==============================] - 3s 6ms/step - loss: 0.2216 - accuracy: 0.9183 - val_loss: 0.2626 - val_accuracy: 0.9035
Epoch 14/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2163 - accuracy: 0.9197 - val_loss: 0.2623 - val_accuracy: 0.9083
Epoch 15/25
422/422 [==============================] - 2s 6ms/step - loss: 0.2102 - accuracy: 0.9226 - val_loss: 0.2535 - val_accuracy: 0.9065
Epoch 16/25
422/422 [==============================] - 3s 6ms/step - loss: 0.2032 - accuracy: 0.9249 - val_loss: 0.2731 - val_accuracy: 0.9025
Epoch 17/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1989 - accuracy: 0.9259 - val_loss: 0.2437 - val_accuracy: 0.9140
Epoch 18/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1932 - accuracy: 0.9280 - val_loss: 0.2496 - val_accuracy: 0.9115
Epoch 19/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1908 - accuracy: 0.9287 - val_loss: 0.2471 - val_accuracy: 0.9080
Epoch 20/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1855 - accuracy: 0.9312 - val_loss: 0.2432 - val_accuracy: 0.9100
Epoch 21/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1796 - accuracy: 0.9328 - val_loss: 0.2621 - val_accuracy: 0.9055
Epoch 22/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1772 - accuracy: 0.9352 - val_loss: 0.2577 - val_accuracy: 0.9092
Epoch 23/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1702 - accuracy: 0.9368 - val_loss: 0.2515 - val_accuracy: 0.9123
```

```
Epoch 24/25
422/422 [==============================] - 3s 6ms/step - loss: 0.1665 - accuracy: 0.9373 - val_loss: 0.2510 - val_accuracy: 0.9117
Epoch 25/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1659 - accuracy: 0.9387 - val_loss: 0.2455 - val_accuracy: 0.9143
```

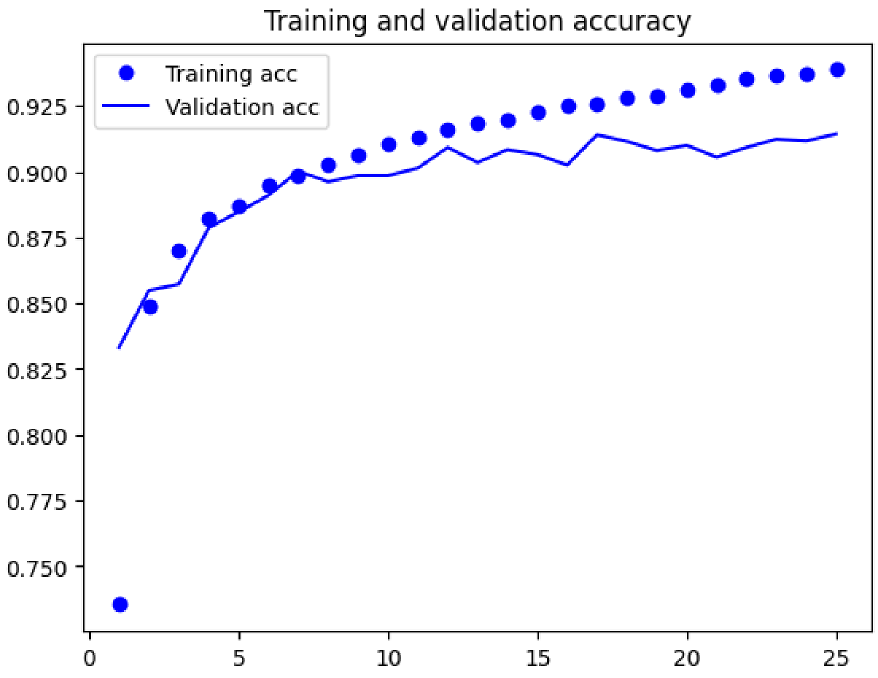
Ewaluacja modelu

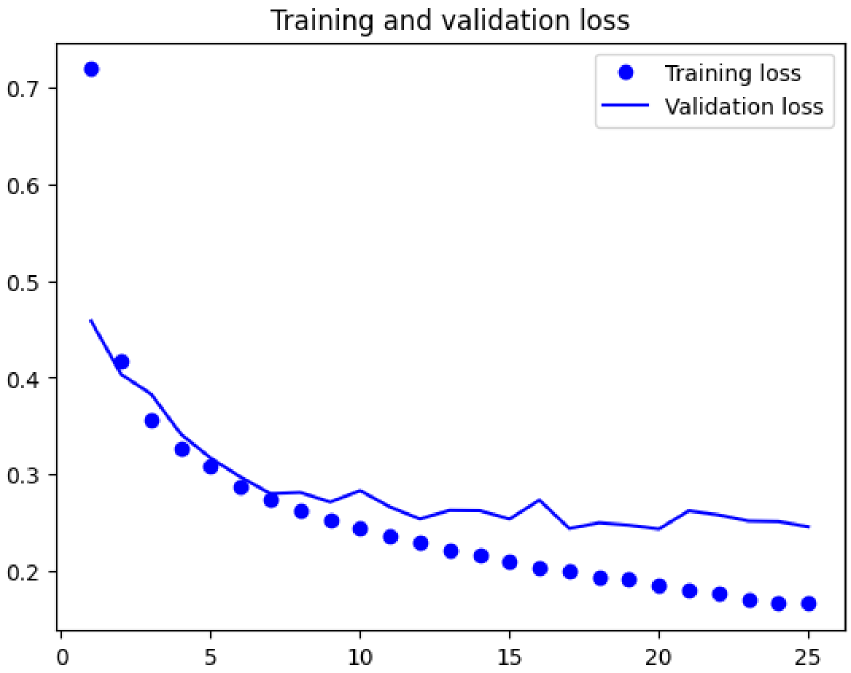
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.9099000096321106
```

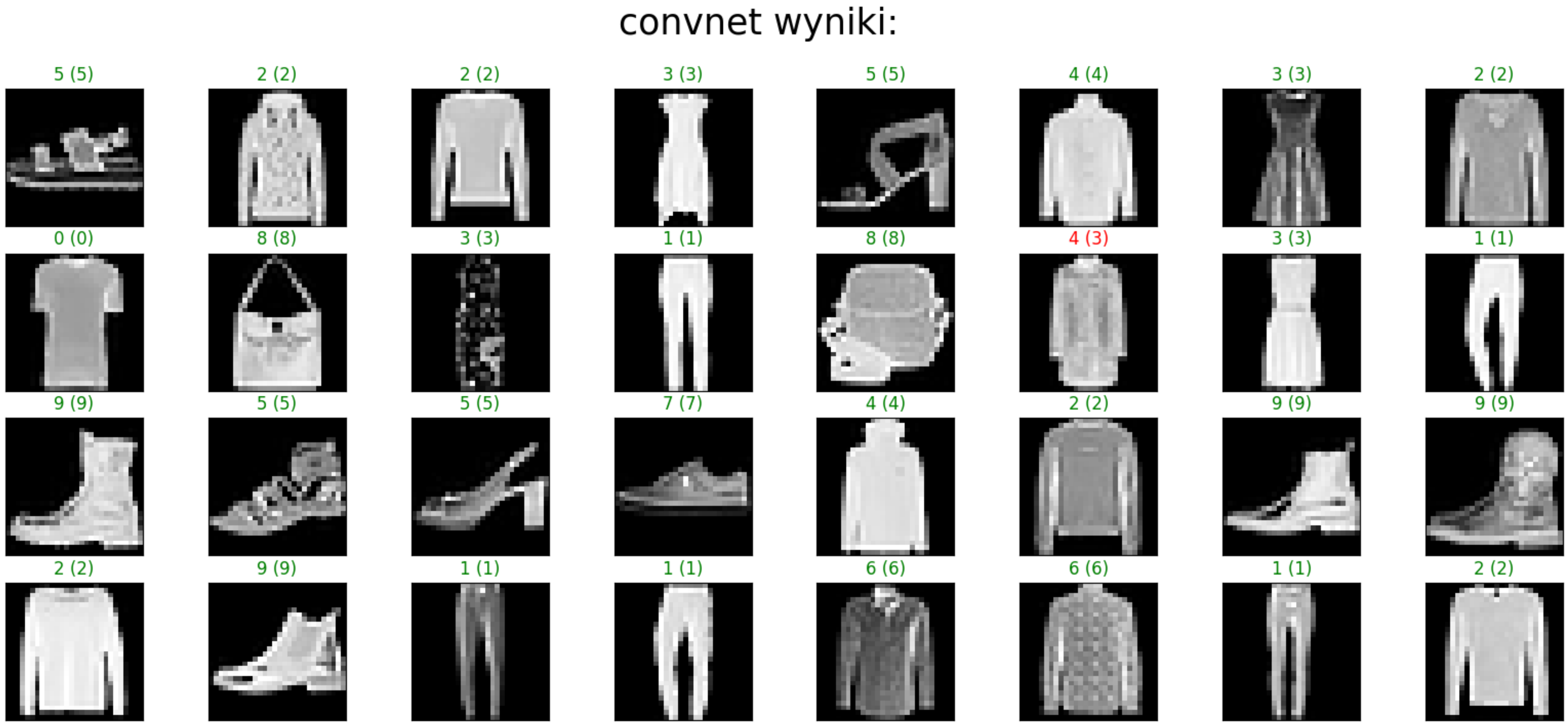
Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
5
10
15
20
25


Training and validation loss
Training loss
Validation loss
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25

```
visualize_model_predictions(model, x_test, y_test,"convnet")

    313/313 [==============================] - 1s 2ms/step
```



3 Regularyzacja

Nasz model ma obecnie dużo stopni swobody (ma DUŻO parametrów i dlatego może dopasować się do niemal każdej funkcji, jeśli tylko będziemy trenować wystarczająco długo). Oznacza to, że nasza sieć jest również podatna na przeuczenie.

W tej sekcji dodajmy warstwy dropout pomiędzy głównymi warstwami naszej sieci, aby uniknąć przeuczenia.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=2,padding='same',activation='relu'))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

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

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 64)        320

 max_pooling2d_8 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_9 (Conv2D)           (None, 14, 14, 32)        8224

 max_pooling2d_9 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 dropout_1 (Dropout)         (None, 7, 7, 32)          0

 conv2d_10 (Conv2D)          (None, 7, 7, 16)          2064

 flatten_12 (Flatten)        (None, 784)               0

 dropout_2 (Dropout)         (None, 784)               0
```

```
 dense_33 (Dense)            (None, 128)               100480

 dense_34 (Dense)            (None, 64)                8256

 dense_35 (Dense)            (None, 10)                650

=================================================================
Total params: 119994 (468.73 KB)
Trainable params: 119994 (468.73 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(x_train,
          y_train,
          batch_size=128,
          epochs=25,
          validation_data=(x_valid, y_valid))
```

```
Epoch 1/25
422/422 [==============================] - 7s 9ms/step - loss: 0.8686 - accuracy: 0.6681 - val_loss: 0.5237 - val_accuracy: 0.8052
Epoch 2/25
422/422 [==============================] - 3s 8ms/step - loss: 0.5599 - accuracy: 0.7899 - val_loss: 0.4507 - val_accuracy: 0.8330
Epoch 3/25
422/422 [==============================] - 3s 8ms/step - loss: 0.5045 - accuracy: 0.8107 - val_loss: 0.3926 - val_accuracy: 0.8550
Epoch 4/25
422/422 [==============================] - 3s 7ms/step - loss: 0.4702 - accuracy: 0.8233 - val_loss: 0.3652 - val_accuracy: 0.8642
Epoch 5/25
422/422 [==============================] - 3s 7ms/step - loss: 0.4425 - accuracy: 0.8353 - val_loss: 0.3641 - val_accuracy: 0.8647
Epoch 6/25
422/422 [==============================] - 3s 8ms/step - loss: 0.4225 - accuracy: 0.8417 - val_loss: 0.3312 - val_accuracy: 0.8783
Epoch 7/25
422/422 [==============================] - 3s 8ms/step - loss: 0.4103 - accuracy: 0.8459 - val_loss: 0.3324 - val_accuracy: 0.8770
Epoch 8/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3966 - accuracy: 0.8532 - val_loss: 0.3089 - val_accuracy: 0.8878
Epoch 9/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3856 - accuracy: 0.8555 - val_loss: 0.3170 - val_accuracy: 0.8802
Epoch 10/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3761 - accuracy: 0.8600 - val_loss: 0.2988 - val_accuracy: 0.8873
Epoch 11/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3657 - accuracy: 0.8637 - val_loss: 0.2918 - val_accuracy: 0.8910
Epoch 12/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3620 - accuracy: 0.8656 - val_loss: 0.2881 - val_accuracy: 0.8932
Epoch 13/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3504 - accuracy: 0.8682 - val_loss: 0.2857 - val_accuracy: 0.8942
Epoch 14/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3457 - accuracy: 0.8706 - val_loss: 0.2854 - val_accuracy: 0.8912
Epoch 15/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3398 - accuracy: 0.8734 - val_loss: 0.2701 - val_accuracy: 0.8997
Epoch 16/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3339 - accuracy: 0.8742 - val_loss: 0.2892 - val_accuracy: 0.8913
```

```
Epoch 17/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3320 - accuracy: 0.8759 - val_loss: 0.2703 - val_accuracy: 0.8988
Epoch 18/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3276 - accuracy: 0.8782 - val_loss: 0.2721 - val_accuracy: 0.9003
Epoch 19/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3185 - accuracy: 0.8808 - val_loss: 0.2638 - val_accuracy: 0.9025
Epoch 20/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3200 - accuracy: 0.8796 - val_loss: 0.2626 - val_accuracy: 0.9017
Epoch 21/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3187 - accuracy: 0.8794 - val_loss: 0.2589 - val_accuracy: 0.9028
Epoch 22/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3165 - accuracy: 0.8826 - val_loss: 0.2549 - val_accuracy: 0.9025
Epoch 23/25
422/422 [==============================] - 3s 8ms/step - loss: 0.3134 - accuracy: 0.8820 - val_loss: 0.2575 - val_accuracy: 0.9025
Epoch 24/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3082 - accuracy: 0.8850 - val_loss: 0.2508 - val_accuracy: 0.9058
Epoch 25/25
422/422 [==============================] - 3s 7ms/step - loss: 0.3082 - accuracy: 0.8843 - val_loss: 0.2527 - val_accuracy: 0.9033
<keras.src.callbacks.History at 0x7f529797fd90>
```

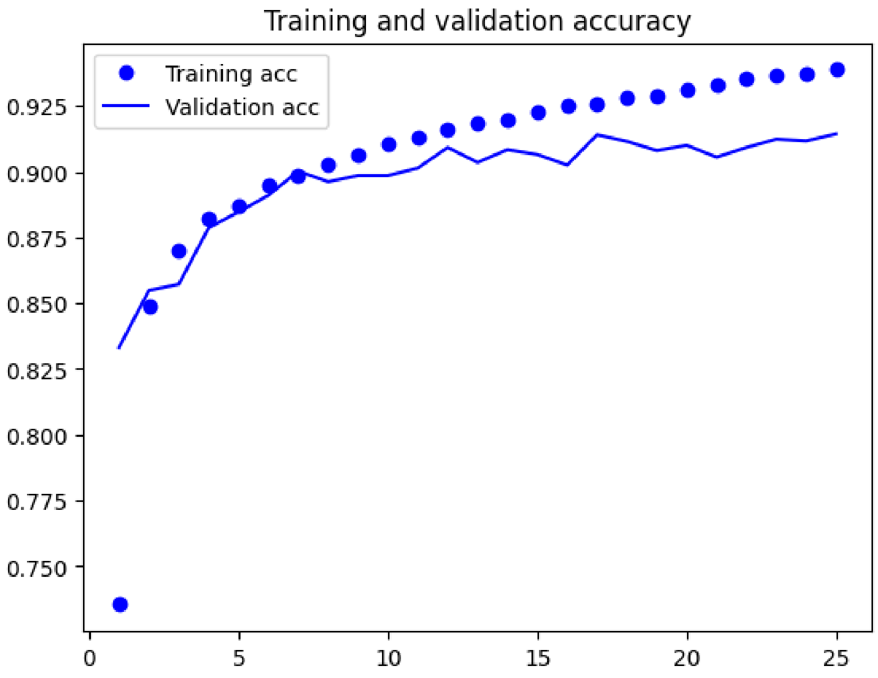
Evaluate model:

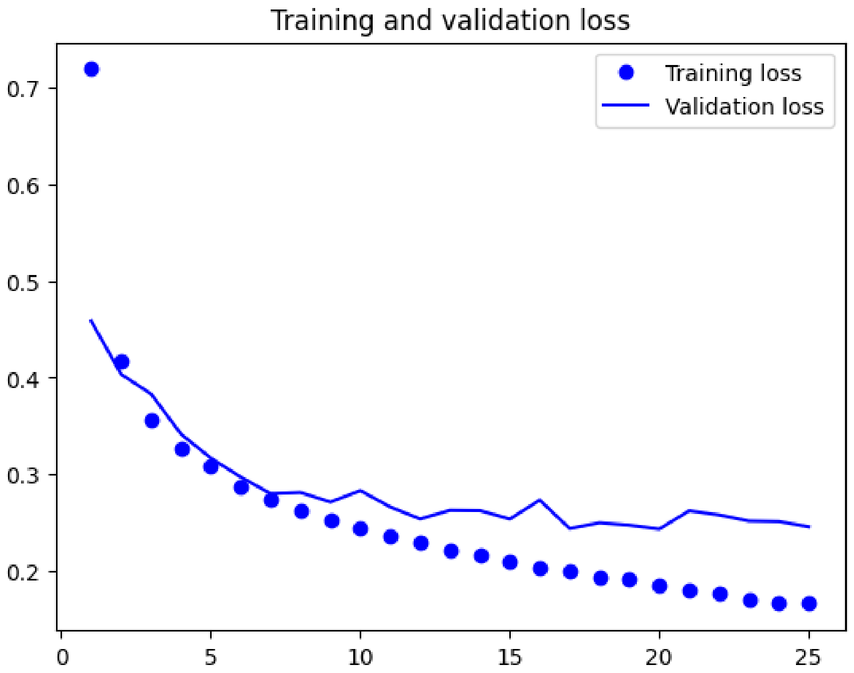
```
test_score = model.evaluate(x_test, y_test, verbose=0)
train_score = model.evaluate(x_train, y_train, verbose=0)

print('Train accuracy: ',train_score[1],' Test accuracy: ',test_score[1])
```

```
Train accuracy:  0.9166481494903564  Test accuracy:  0.9003000259399414
```

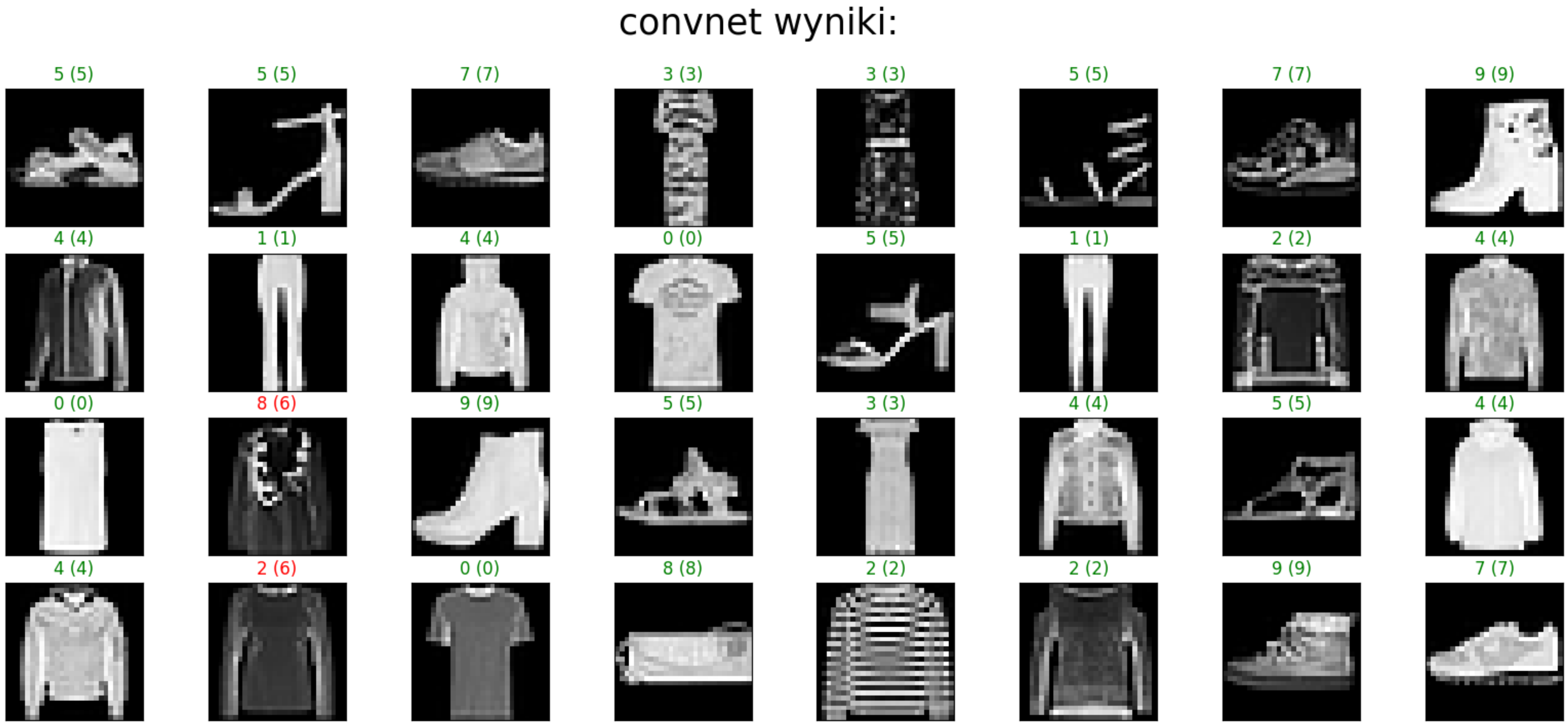
Wykresy precyzji i błędu

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
5
10
15
20
25


Training and validation loss
Training loss
Validation loss
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25

```
visualize_model_predictions(model, x_test, y_test,"convnet")

    313/313 [==============================] - 1s 2ms/step
```



3 Regularyzacja v2

Nasz model ma obecnie dużo stopni swobody (ma DUŻO parametrów i dlatego może dopasować się do niemal każdej funkcji, jeśli tylko będziemy trenować wystarczająco długo). Oznacza to, że nasza sieć jest również podatna na przeuczenie.

W tej sekcji dodajmy warstwy dropout pomiędzy warstwami naszej sieci, aby uniknąć przeuczenia.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2,padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=2,padding='same',activation='relu'))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation="softmax"))

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

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 28, 28, 64)        320

 max_pooling2d_10 (MaxPooli  (None, 14, 14, 64)        0
 ng2D)

 dropout_3 (Dropout)         (None, 14, 14, 64)        0

 conv2d_12 (Conv2D)          (None, 14, 14, 32)        8224
```