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.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)
```

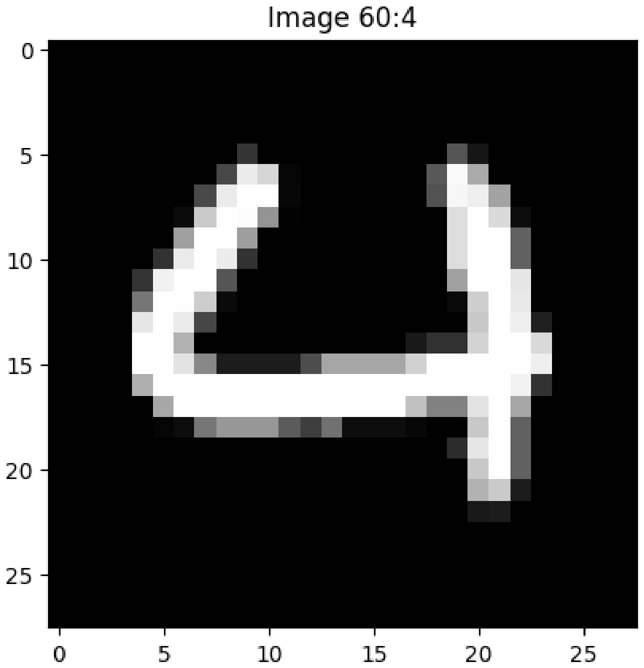
```
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
    11490434/11490434 [==============================] - 0s 0us/step
```

```
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 wykorzystywali 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 wykorzystywali 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 przekształci 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

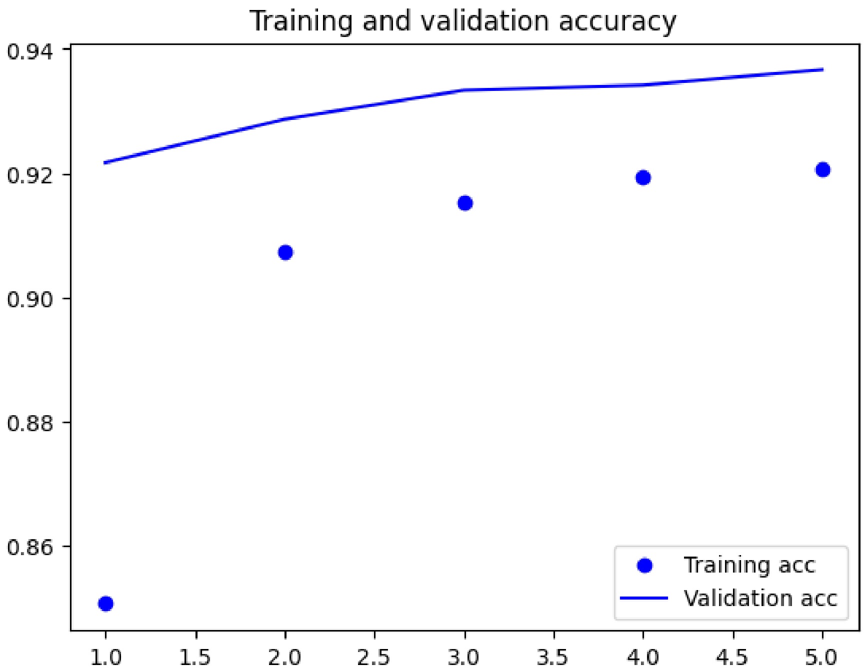
Model uczymy wykorzystując fit().

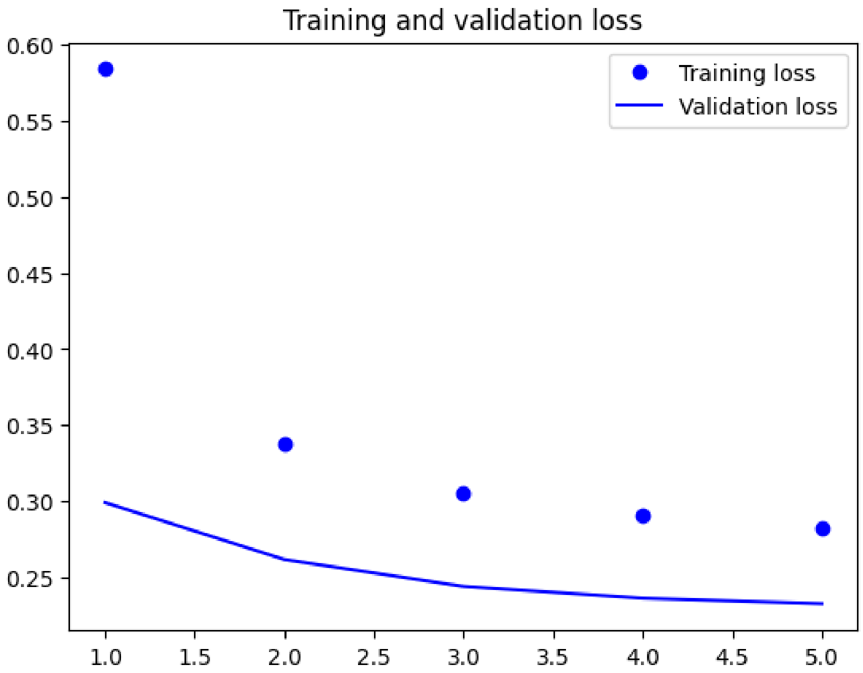
```
history = model.fit(x_train, y_train, batch_size = 64, epochs = 5, validation_data = (x_valid,y_valid))
```

```
Epoch 1/5
844/844 [==============================] - 9s 8ms/step - loss: 0.5838 - accuracy: 0.8506 - val_loss: 0.2990 - val_accuracy: 0.9217
Epoch 2/5
844/844 [==============================] - 5s 6ms/step - loss: 0.3377 - accuracy: 0.9074 - val_loss: 0.2616 - val_accuracy: 0.9287
Epoch 3/5
844/844 [==============================] - 6s 7ms/step - loss: 0.3056 - accuracy: 0.9153 - val_loss: 0.2440 - val_accuracy: 0.9333
Epoch 4/5
844/844 [==============================] - 3s 3ms/step - loss: 0.2911 - accuracy: 0.9193 - val_loss: 0.2362 - val_accuracy: 0.9342
Epoch 5/5
844/844 [==============================] - 2s 3ms/step - loss: 0.2823 - accuracy: 0.9207 - val_loss: 0.2326 - val_accuracy: 0.9367
```

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
0.94
0.92
0.90
0.88
0.86
Training acc
Validation acc
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Training and validation loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
Training loss
Validation loss
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])
```

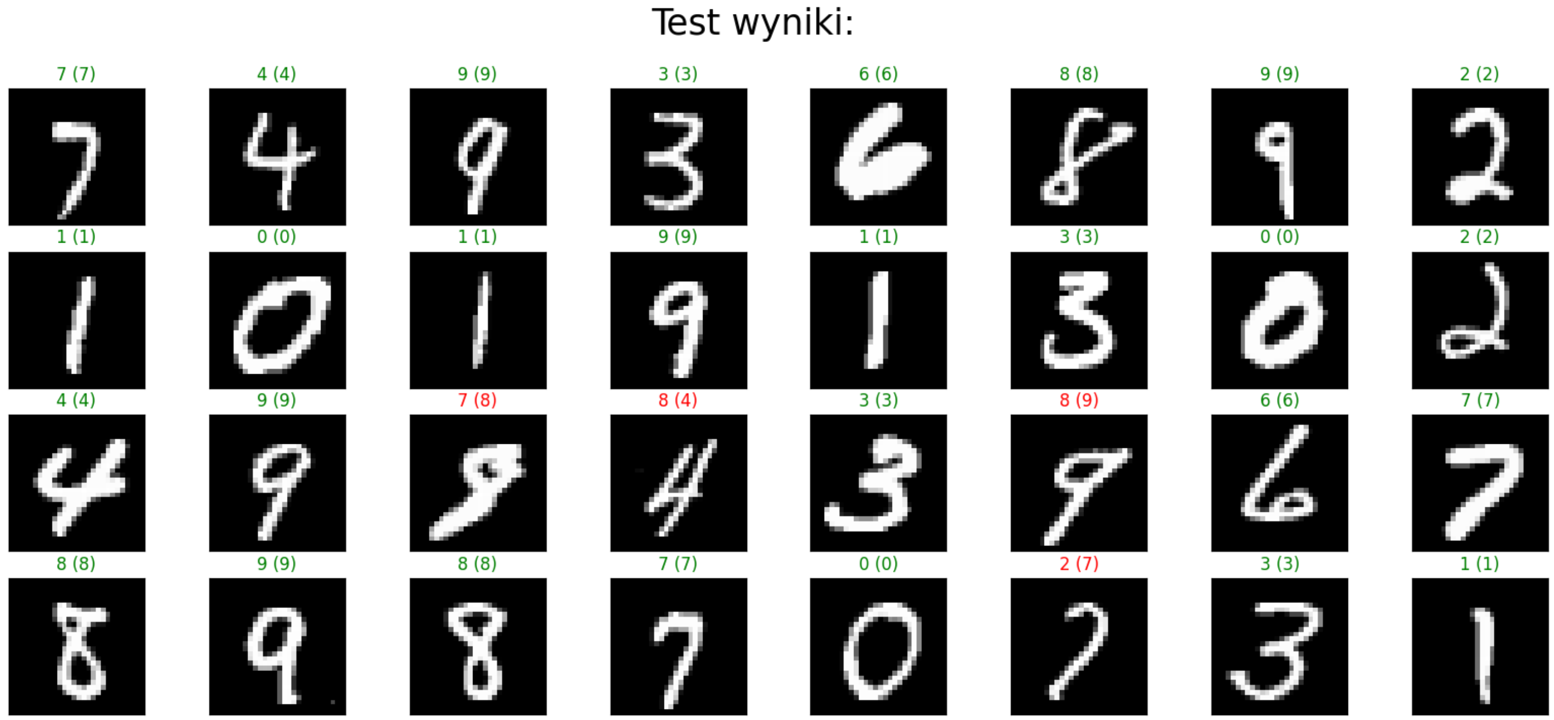
```
Test accuracy: 0.9244999885559082
```

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')
```

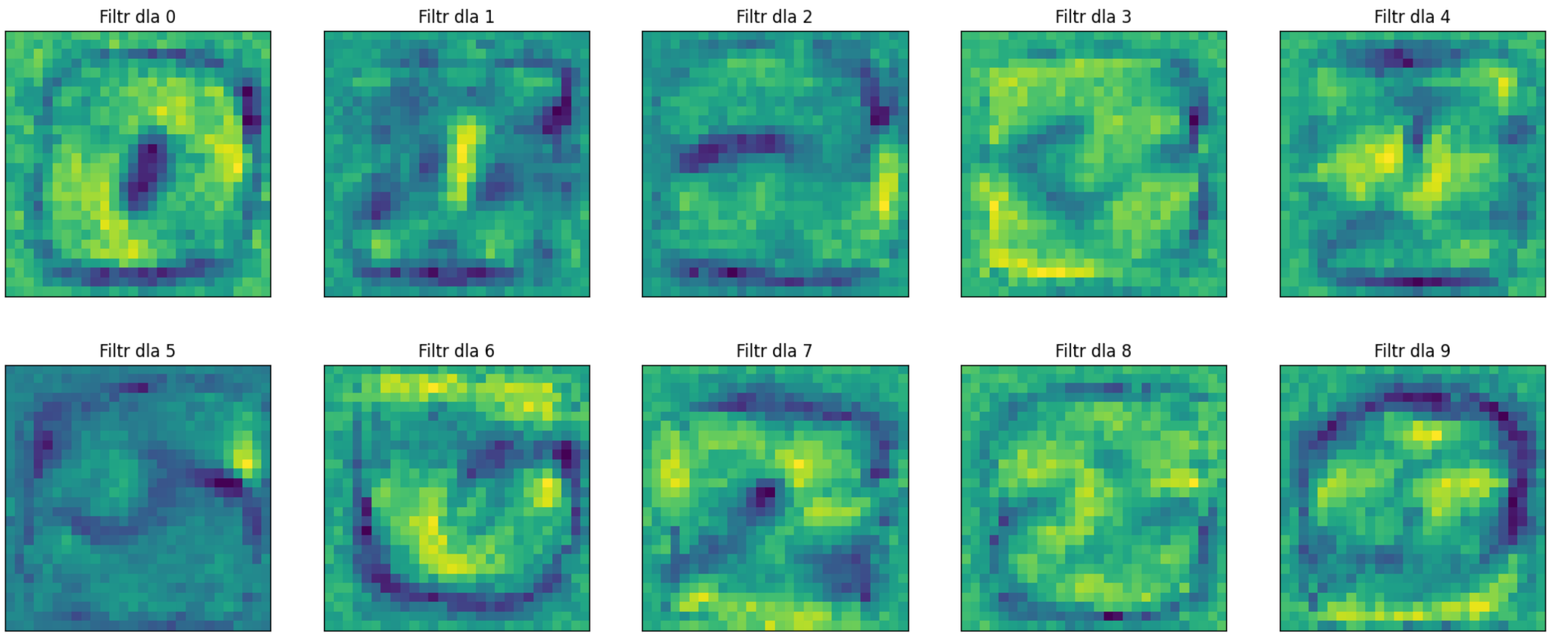
```
313/313 [==============================] - 1s 1ms/step
```



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().

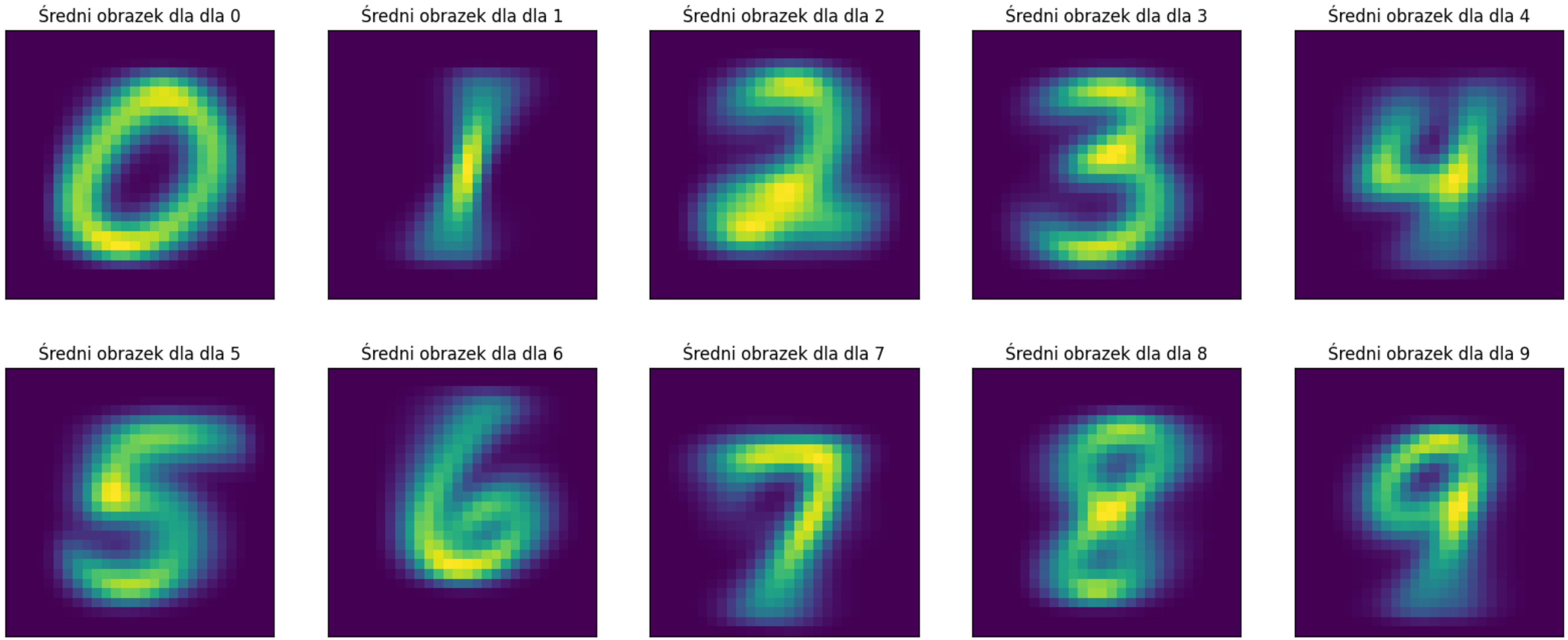
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

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_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_6 (Flatten)         (None, 784)               0

 dense_15 (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().

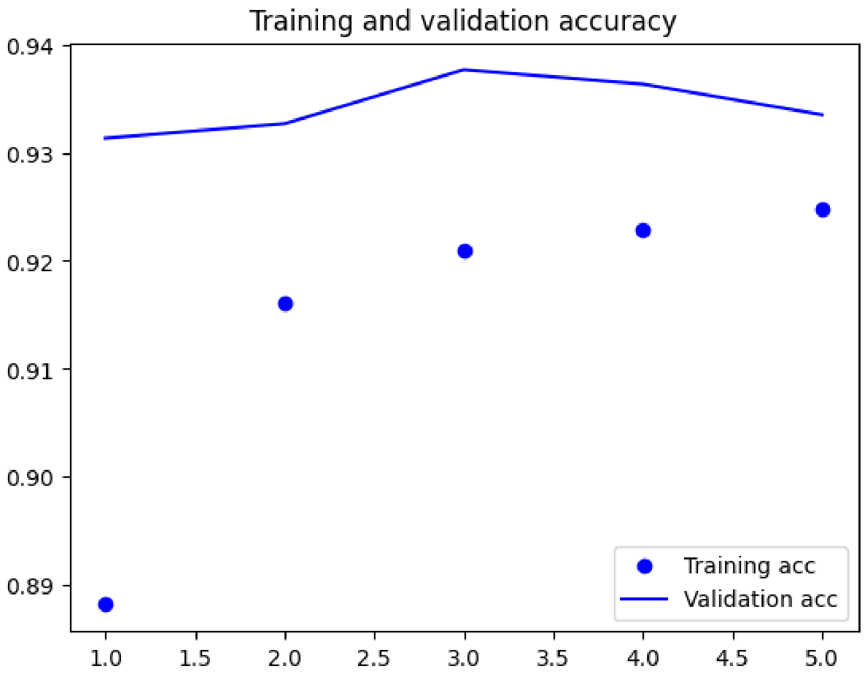
```
history = model.fit(x_train, y_train, batch_size = 32, epochs = 5, validation_data = (x_valid,y_valid))
```

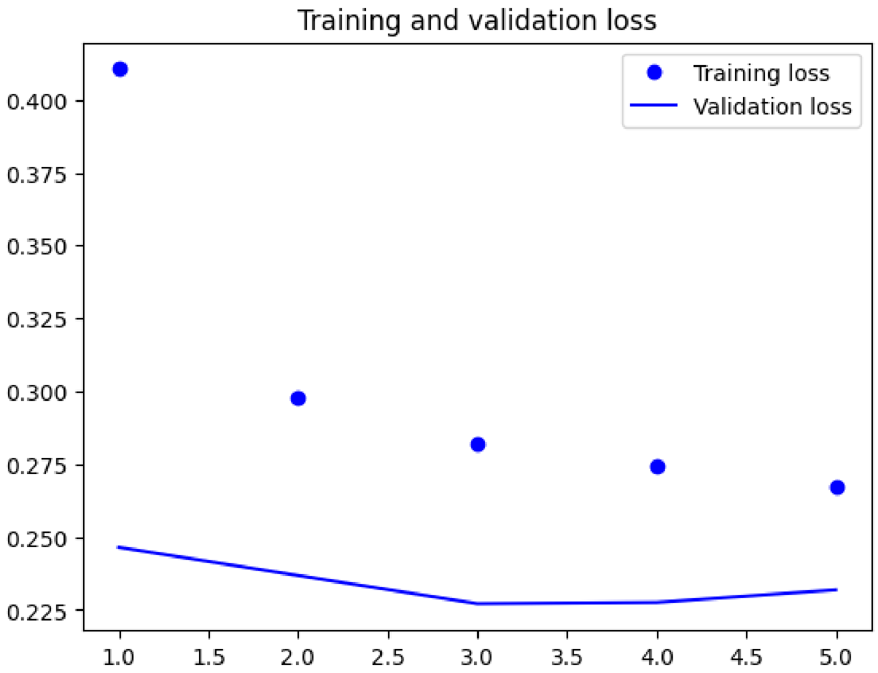
```
Epoch 1/5
1688/1688 [==============================] - 6s 3ms/step - loss: 0.4105 - accuracy: 0.8882 - val_loss: 0.2464 - val_accuracy: 0.9313
Epoch 2/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.2977 - accuracy: 0.9161 - val_loss: 0.2368 - val_accuracy: 0.9327
```

```
Epoch 3/5
1688/1688 [==============================] - 5s 3ms/step - loss: 0.2817 - accuracy: 0.9210 - val_loss: 0.2271 - val_accuracy: 0.9377
Epoch 4/5
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2742 - accuracy: 0.9228 - val_loss: 0.2275 - val_accuracy: 0.9363
Epoch 5/5
1688/1688 [==============================] - 4s 3ms/step - loss: 0.2673 - accuracy: 0.9247 - val_loss: 0.2318 - val_accuracy: 0.9335
```

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
0.94
0.93
0.92
0.91
0.90
0.89
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Training acc
Validation acc


Training and validation loss
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Training loss
Validation loss

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])
```

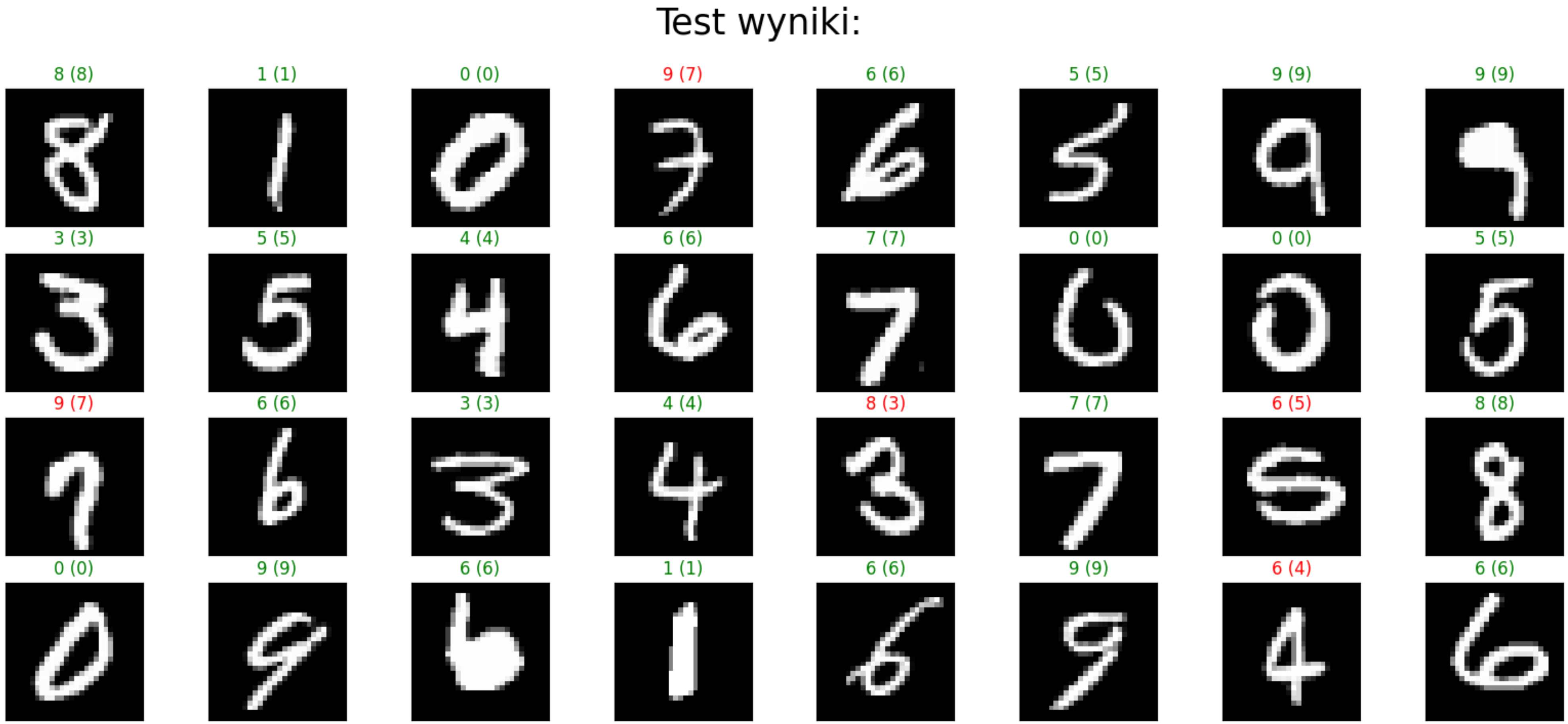
```
Test accuracy: 0.9204000234603882
```

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')
```

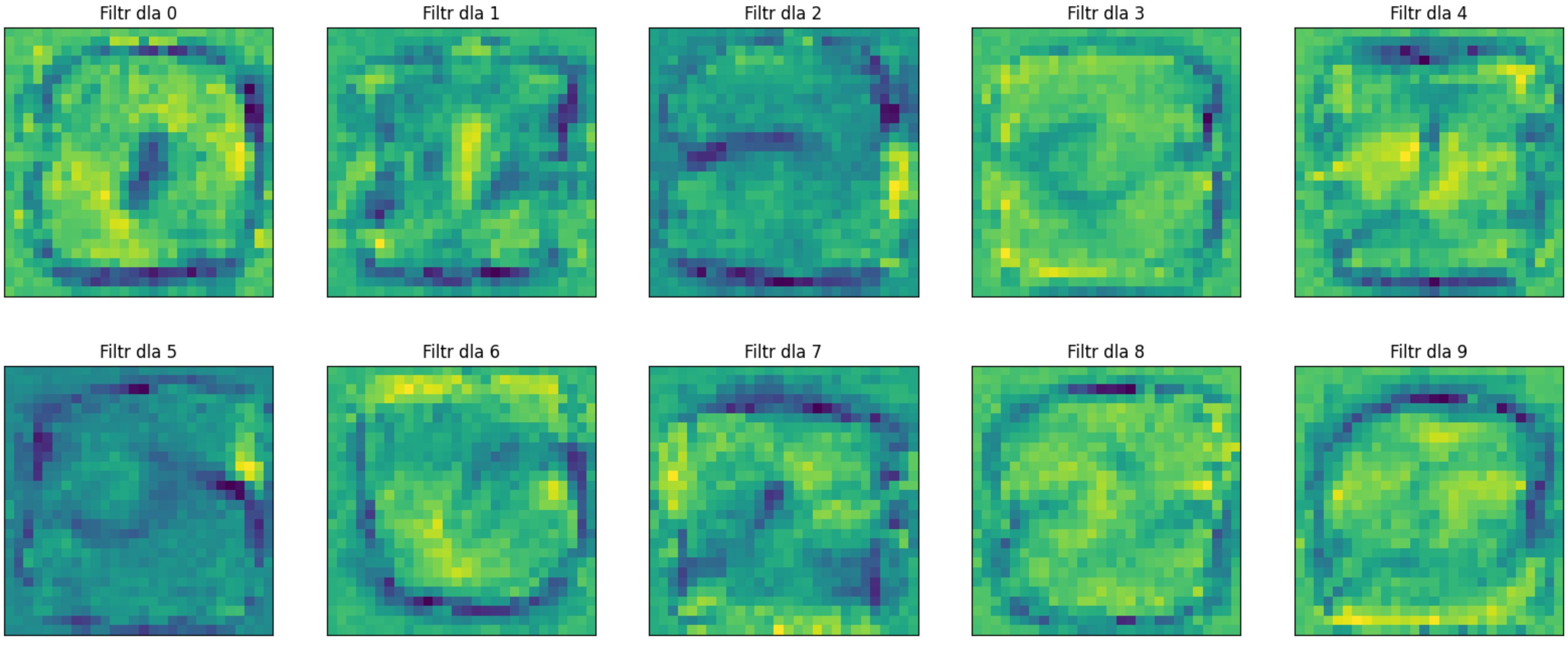
```
313/313 [==============================] - 0s 1ms/step
```



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().

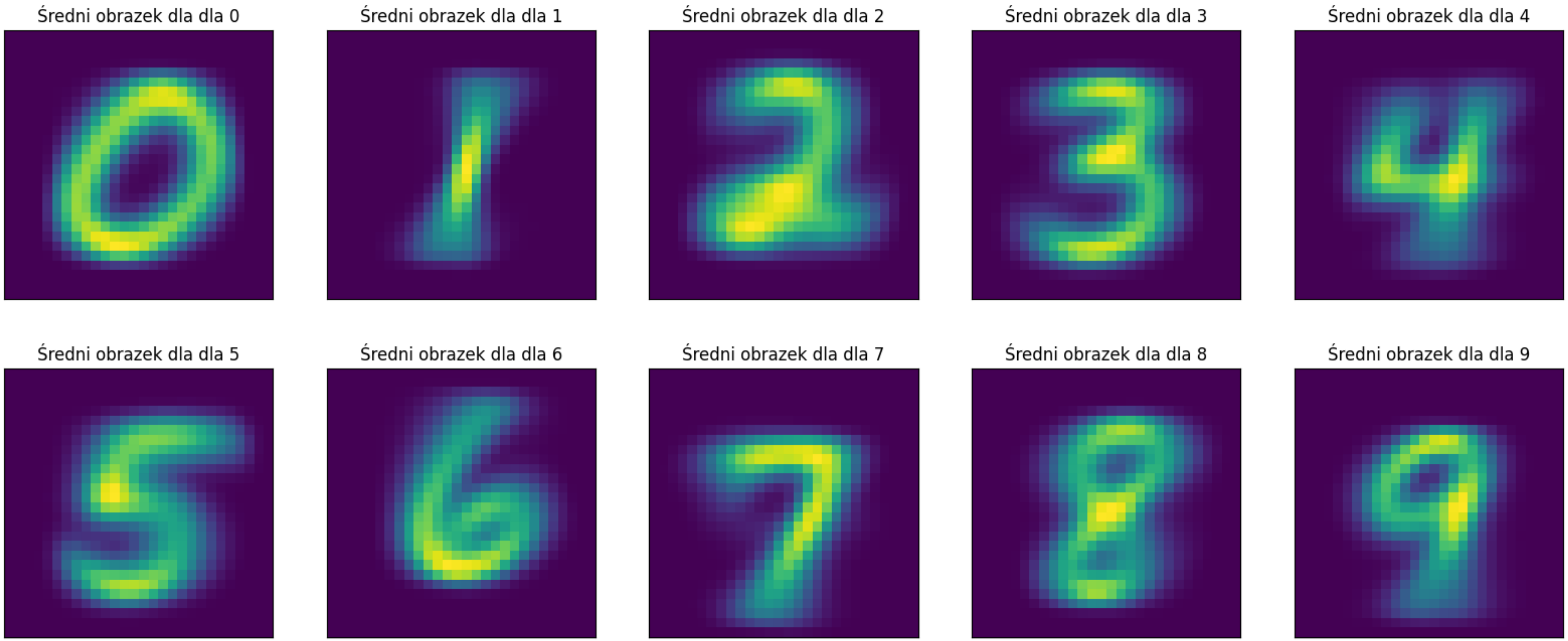
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

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_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_17 (Flatten)        (None, 784)               0

 dense_26 (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

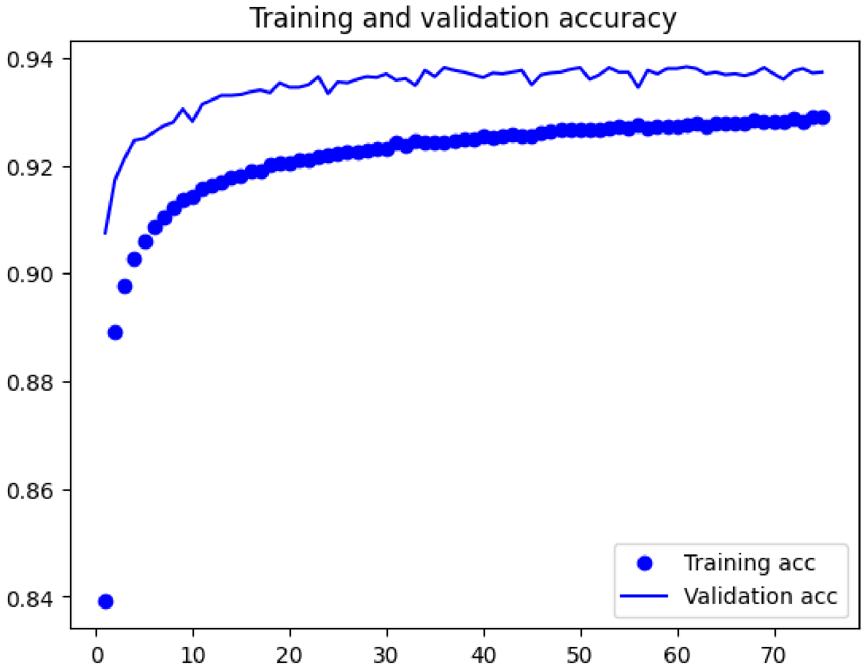
Model uczymy wykorzystując fit().

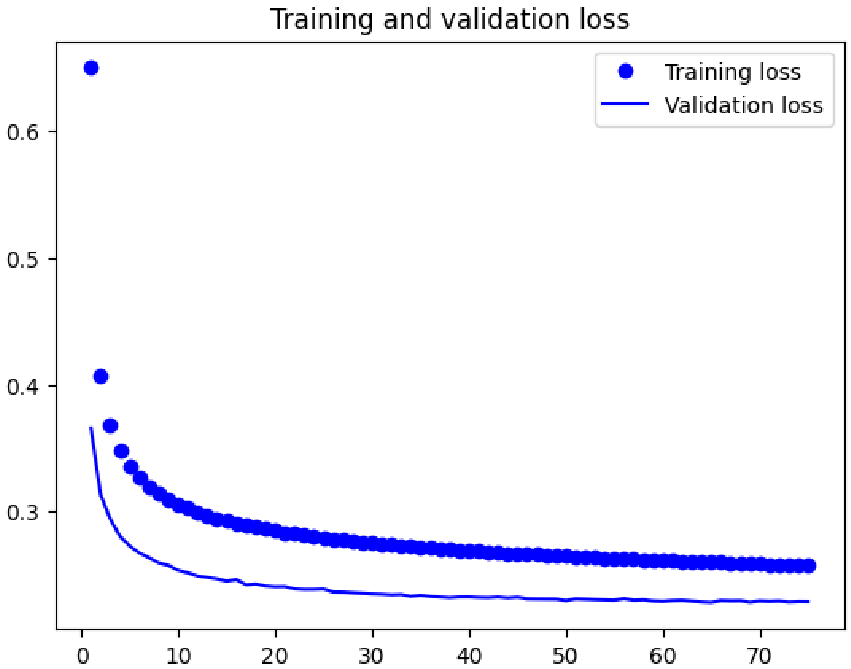
```
history = model.fit(x_train, y_train, batch_size = 64, epochs = 75, validation_data = (x_valid,y_valid))
```

```
844/844 [==============================] - 2s 3ms/step - loss: 0.2573 - accuracy: 0.9290 - val_loss: 0.2287 - val_accuracy: 0.9373
```

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


Training and validation loss
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])
```

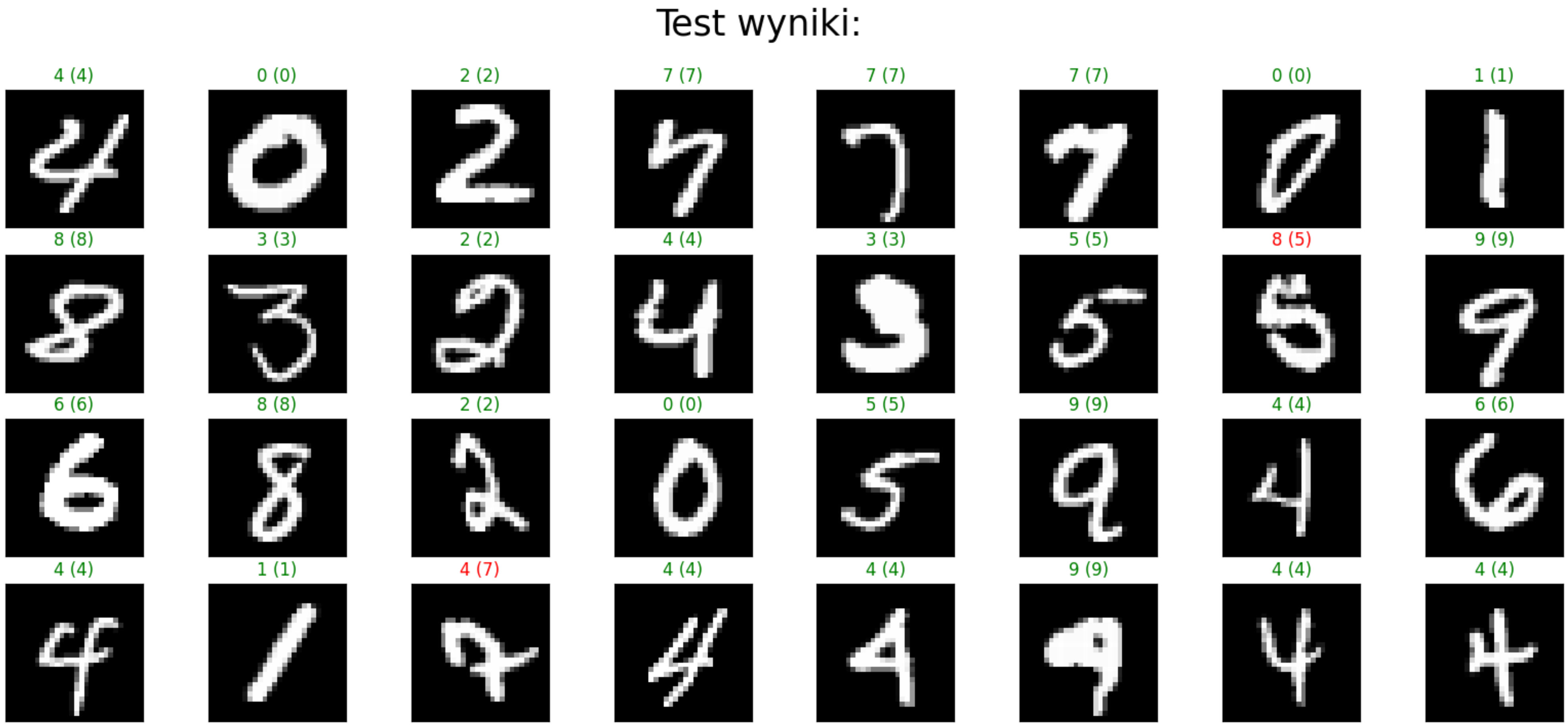
```
    Test accuracy: 0.9243000149726868
```

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')
```

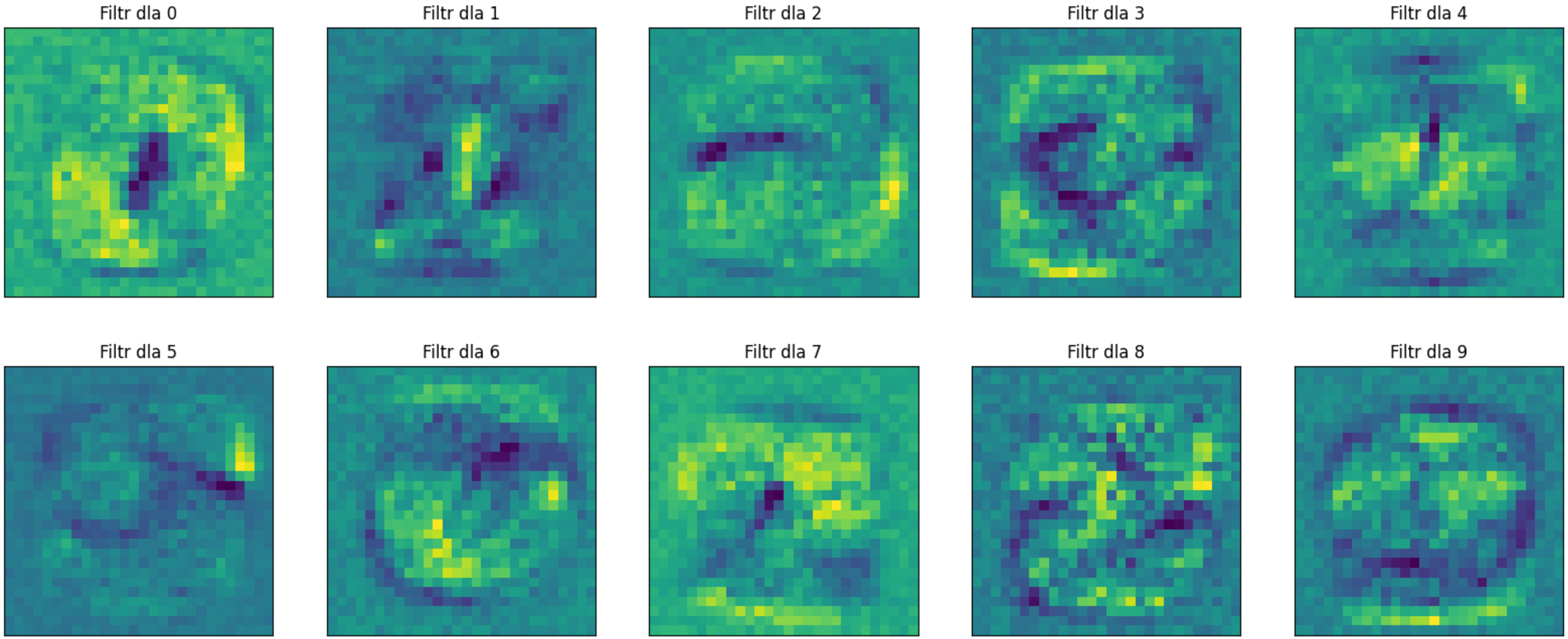
```
313/313 [==============================] - 0s 1ms/step
```



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().

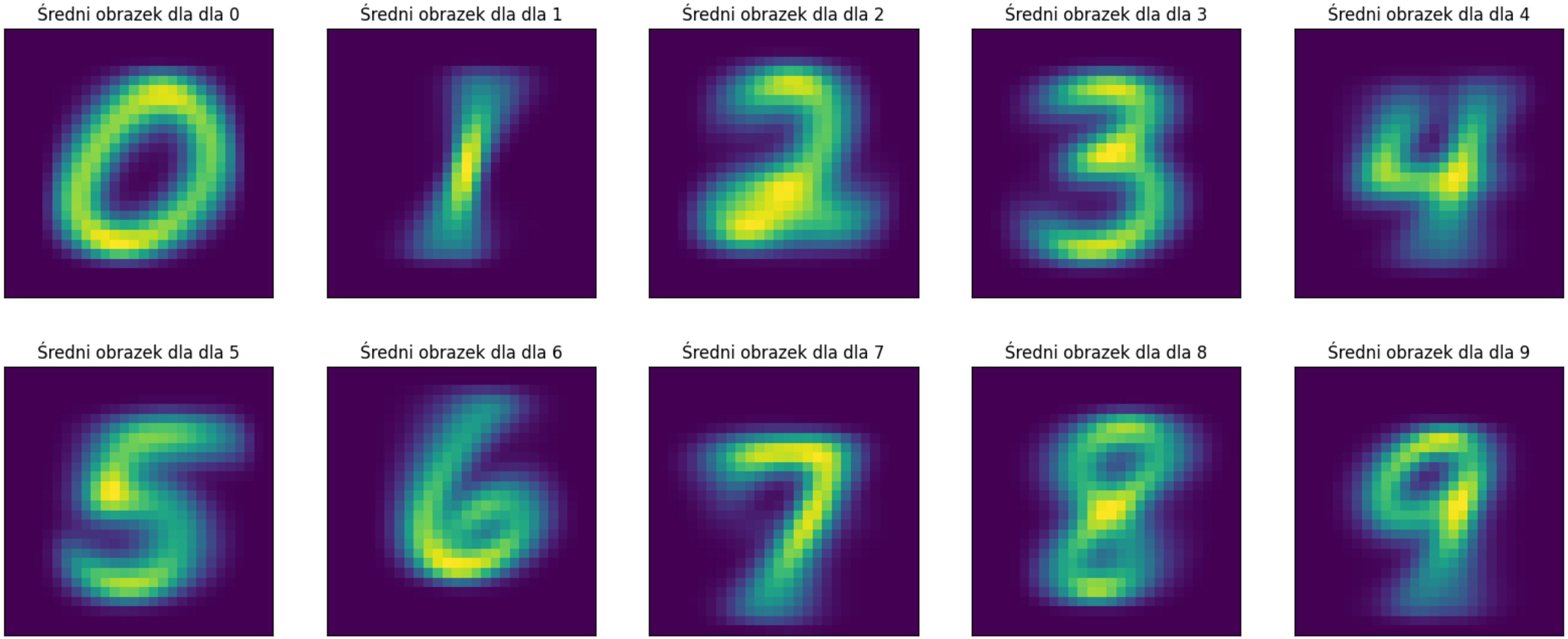
```
for layer in model.layers:
    weights = layer.get_weights()
    if len(weights) > 0:
        w,b = weights
        filters = np.reshape(w, (28,28,10))

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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_1 (Flatten)         (None, 784)               0         
                                                                 
 dense_1 (Dense)             (None, 60)                47100     
                                                                 
 dense_2 (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: 0.3693 - accuracy: 0.8986 - val_loss: 0.1734 - val_accuracy: 0.9537
Epoch 2/10
844/844 [==============================] - 3s 4ms/step - loss: 0.1755 - accuracy: 0.9497 - val_loss: 0.1264 - val_accuracy: 0.9653
Epoch 3/10
844/844 [==============================] - 3s 3ms/step - loss: 0.1296 - accuracy: 0.9629 - val_loss: 0.1046 - val_accuracy: 0.9707
Epoch 4/10
844/844 [==============================] - 3s 3ms/step - loss: 0.1047 - accuracy: 0.9691 - val_loss: 0.0971 - val_accuracy: 0.9723
Epoch 5/10
844/844 [==============================] - 3s 3ms/step - loss: 0.0858 - accuracy: 0.9745 - val_loss: 0.0955 - val_accuracy: 0.9720
Epoch 6/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0739 - accuracy: 0.9783 - val_loss: 0.0853 - val_accuracy: 0.9763
Epoch 7/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0627 - accuracy: 0.9816 - val_loss: 0.0924 - val_accuracy: 0.9735
Epoch 8/10
844/844 [==============================] - 3s 3ms/step - loss: 0.0551 - accuracy: 0.9836 - val_loss: 0.0856 - val_accuracy: 0.9763
Epoch 9/10
844/844 [==============================] - 3s 3ms/step - loss: 0.0483 - accuracy: 0.9854 - val_loss: 0.0906 - val_accuracy: 0.9750
Epoch 10/10
844/844 [==============================] - 3s 3ms/step - loss: 0.0418 - accuracy: 0.9874 - val_loss: 0.0940 - val_accuracy: 0.9747
```

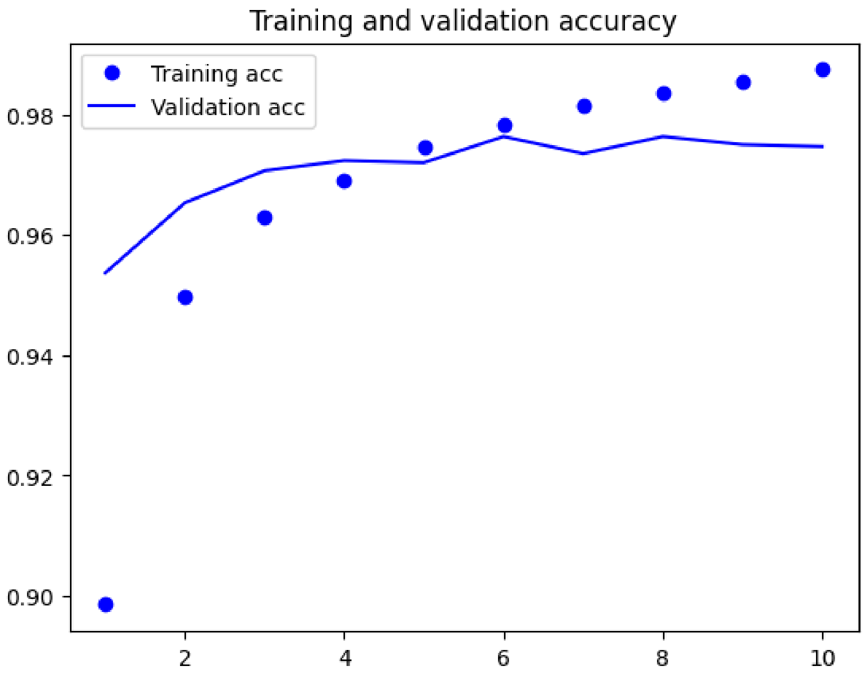
Precyzja

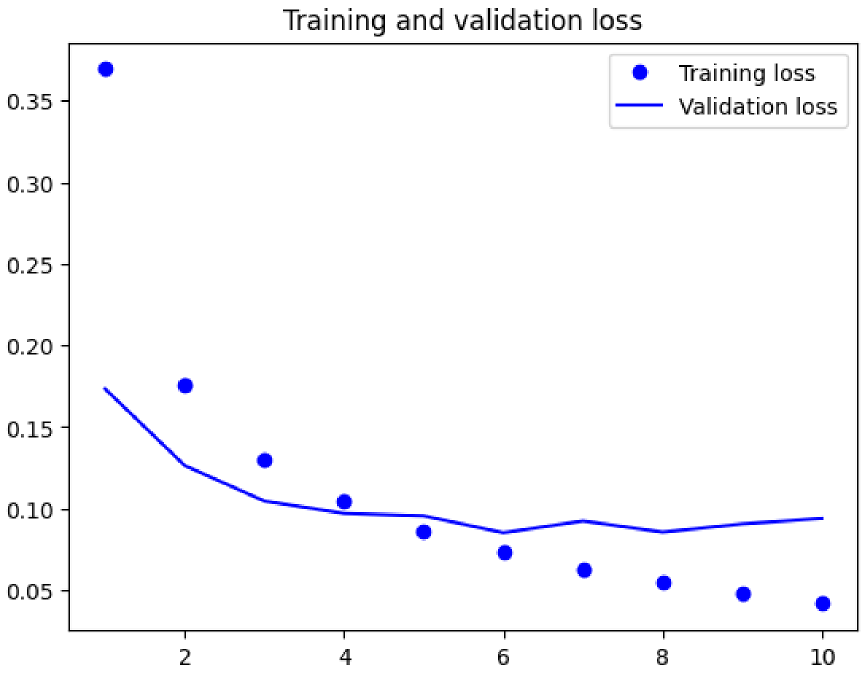
```
score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.9746000170707703
```

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
0.98
0.96
0.94
0.92
0.90
2
4
6
8
10

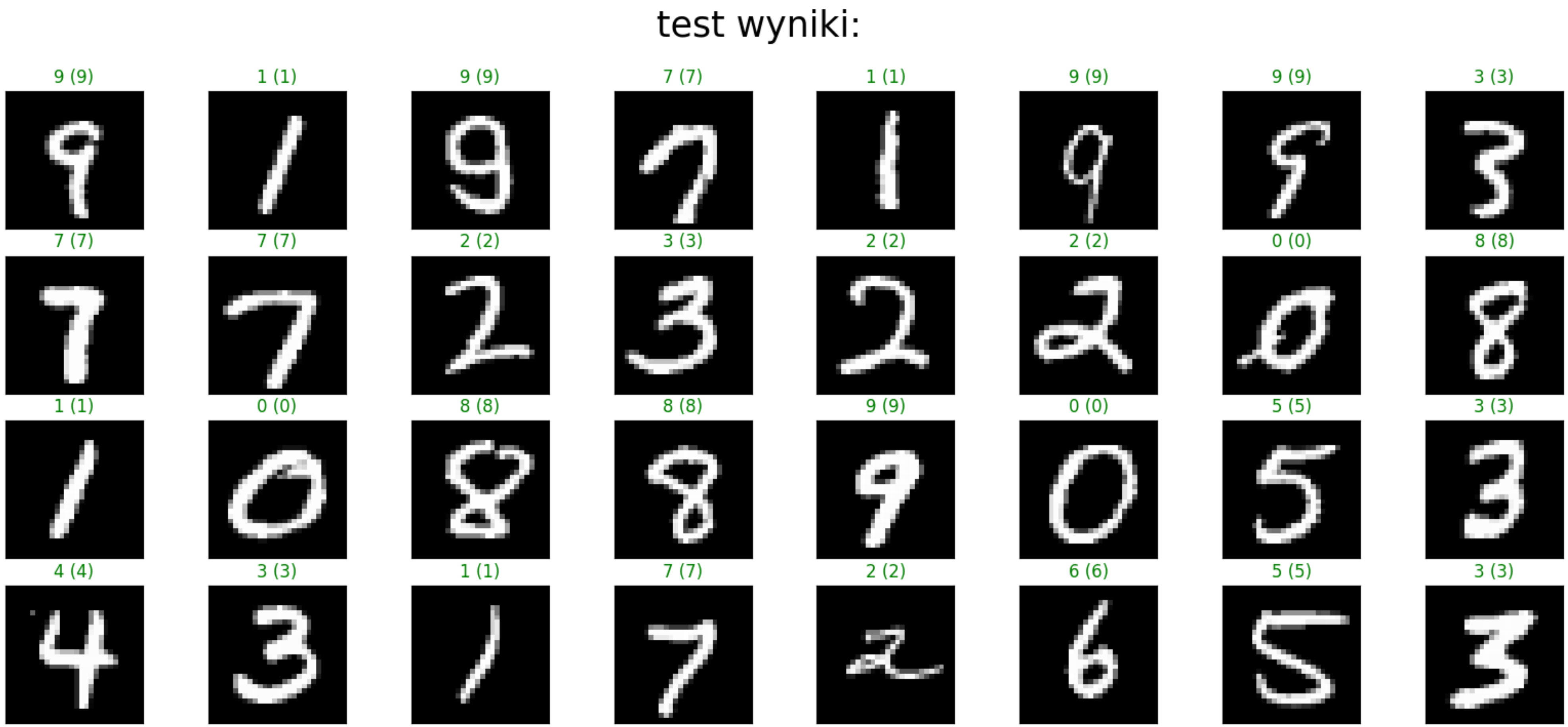

Training and validation loss
Training loss
Validation loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2
4
6
8
10

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_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_23 (Flatten)        (None, 784)               0

 dense_37 (Dense)            (None, 64)                50240

 dense_38 (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 [==============================] - 3s 3ms/step - loss: 0.2624 - accuracy: 0.9215 - val_loss: 0.1202 - val_accuracy: 0.9658
Epoch 2/13
844/844 [==============================] - 3s 3ms/step - loss: 0.1204 - accuracy: 0.9627 - val_loss: 0.1179 - val_accuracy: 0.9638
Epoch 3/13
844/844 [==============================] - 4s 4ms/step - loss: 0.0912 - accuracy: 0.9722 - val_loss: 0.0917 - val_accuracy: 0.9738
Epoch 4/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0745 - accuracy: 0.9761 - val_loss: 0.1010 - val_accuracy: 0.9730
Epoch 5/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0638 - accuracy: 0.9795 - val_loss: 0.1001 - val_accuracy: 0.9728
Epoch 6/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0556 - accuracy: 0.9818 - val_loss: 0.0992 - val_accuracy: 0.9732
Epoch 7/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0452 - accuracy: 0.9851 - val_loss: 0.1033 - val_accuracy: 0.9720
Epoch 8/13
844/844 [==============================] - 3s 4ms/step - loss: 0.0461 - accuracy: 0.9844 - val_loss: 0.1229 - val_accuracy: 0.9683
Epoch 9/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0395 - accuracy: 0.9866 - val_loss: 0.0974 - val_accuracy: 0.9768
```

```
Epoch 10/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0330 - accuracy: 0.9885 - val_loss: 0.1032 - val_accuracy: 0.9768
Epoch 11/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0317 - accuracy: 0.9893 - val_loss: 0.1274 - val_accuracy: 0.9725
Epoch 12/13
844/844 [==============================] - 3s 4ms/step - loss: 0.0275 - accuracy: 0.9904 - val_loss: 0.1137 - val_accuracy: 0.9743
Epoch 13/13
844/844 [==============================] - 3s 3ms/step - loss: 0.0234 - accuracy: 0.9920 - val_loss: 0.1196 - val_accuracy: 0.9765
```

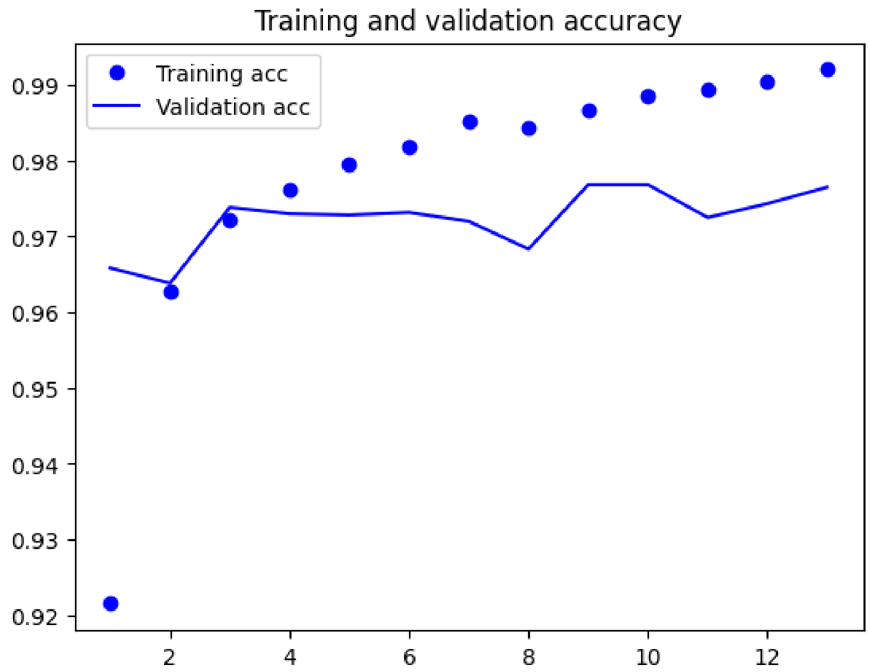
Precyzja

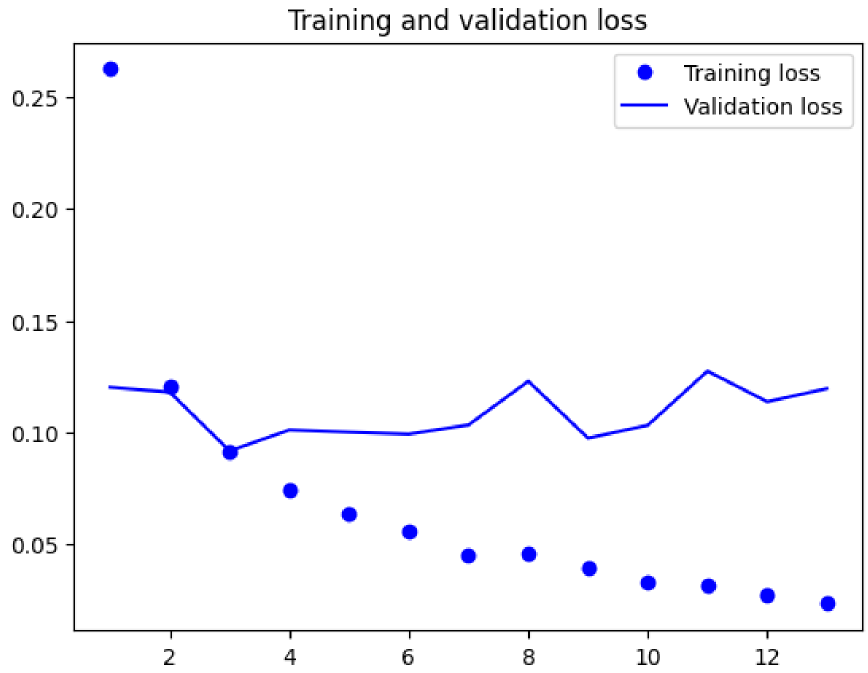
```
score = model.evaluate(x_test,y_test,verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.972100019454956
```

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
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
2
4
6
8
10
12


Training and validation loss
Training loss
Validation loss
0.25
0.20
0.15
0.10
0.05
2
4
6
8
10
12

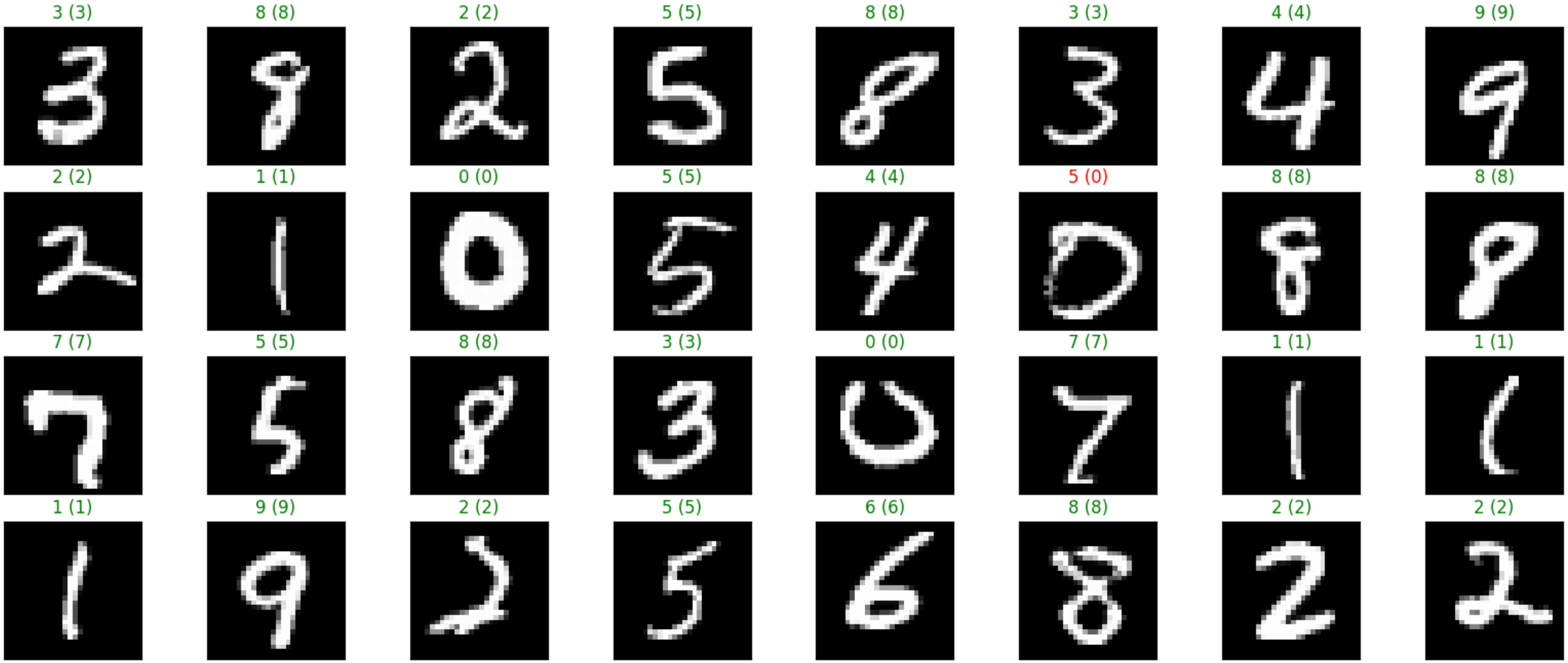
Przewidywania modelu

Sprawdźmy jakie są przewidywania naszego modelu

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

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

test wyniki:



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 3ms/step - loss: 0.0447 - accuracy: 0.9882 - val_loss: 0.0841 - val_accuracy: 0.9760
Epoch 39/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0432 - accuracy: 0.9884 - val_loss: 0.0833 - val_accuracy: 0.9767
Epoch 40/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0420 - accuracy: 0.9893 - val_loss: 0.0826 - val_accuracy: 0.9770
Epoch 41/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0411 - accuracy: 0.9899 - val_loss: 0.0818 - val_accuracy: 0.9770
Epoch 42/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0399 - accuracy: 0.9898 - val_loss: 0.0821 - val_accuracy: 0.9767
Epoch 43/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0387 - accuracy: 0.9904 - val_loss: 0.0819 - val_accuracy: 0.9768
Epoch 44/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0376 - accuracy: 0.9905 - val_loss: 0.0819 - val_accuracy: 0.9777
Epoch 45/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0369 - accuracy: 0.9906 - val_loss: 0.0828 - val_accuracy: 0.9775
Epoch 46/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0359 - accuracy: 0.9909 - val_loss: 0.0825 - val_accuracy: 0.9782
Epoch 47/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0350 - accuracy: 0.9916 - val_loss: 0.0827 - val_accuracy: 0.9775
Epoch 48/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0337 - accuracy: 0.9917 - val_loss: 0.0836 - val_accuracy: 0.9773
Epoch 49/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0331 - accuracy: 0.9920 - val_loss: 0.0811 - val_accuracy: 0.9787
Epoch 50/50
422/422 [==============================] - 1s 3ms/step - loss: 0.0322 - accuracy: 0.9924 - val_loss: 0.0821 - val_accuracy: 0.9773
```

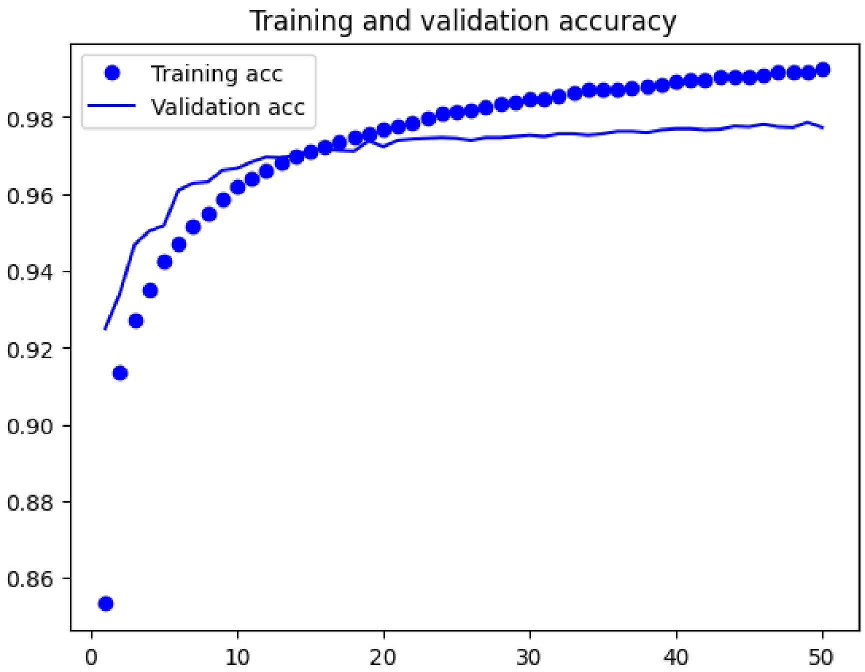
Precyzja

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

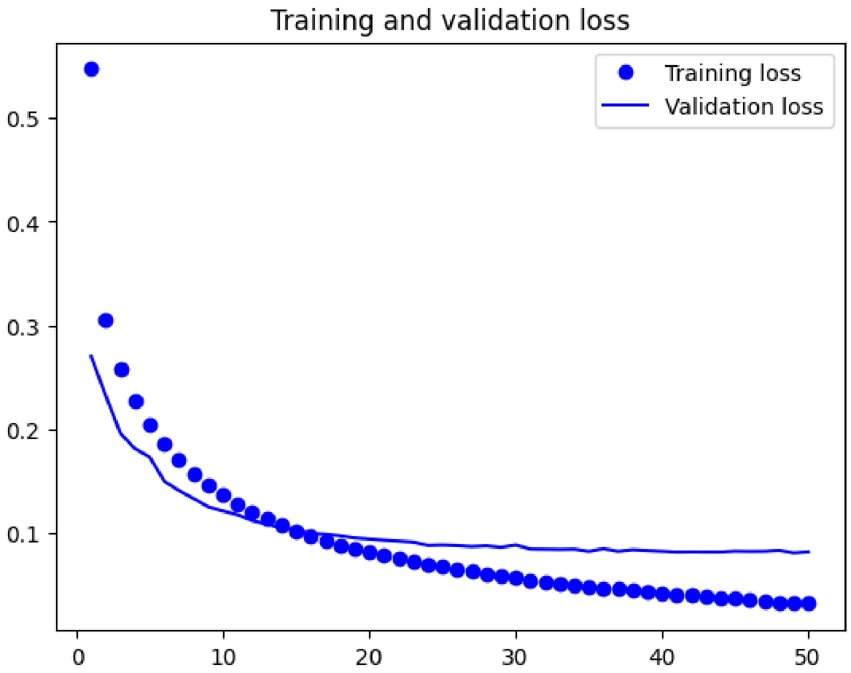
```
Precyzja:  0.9751999974250793
```

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
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0
10
20
30
40
50



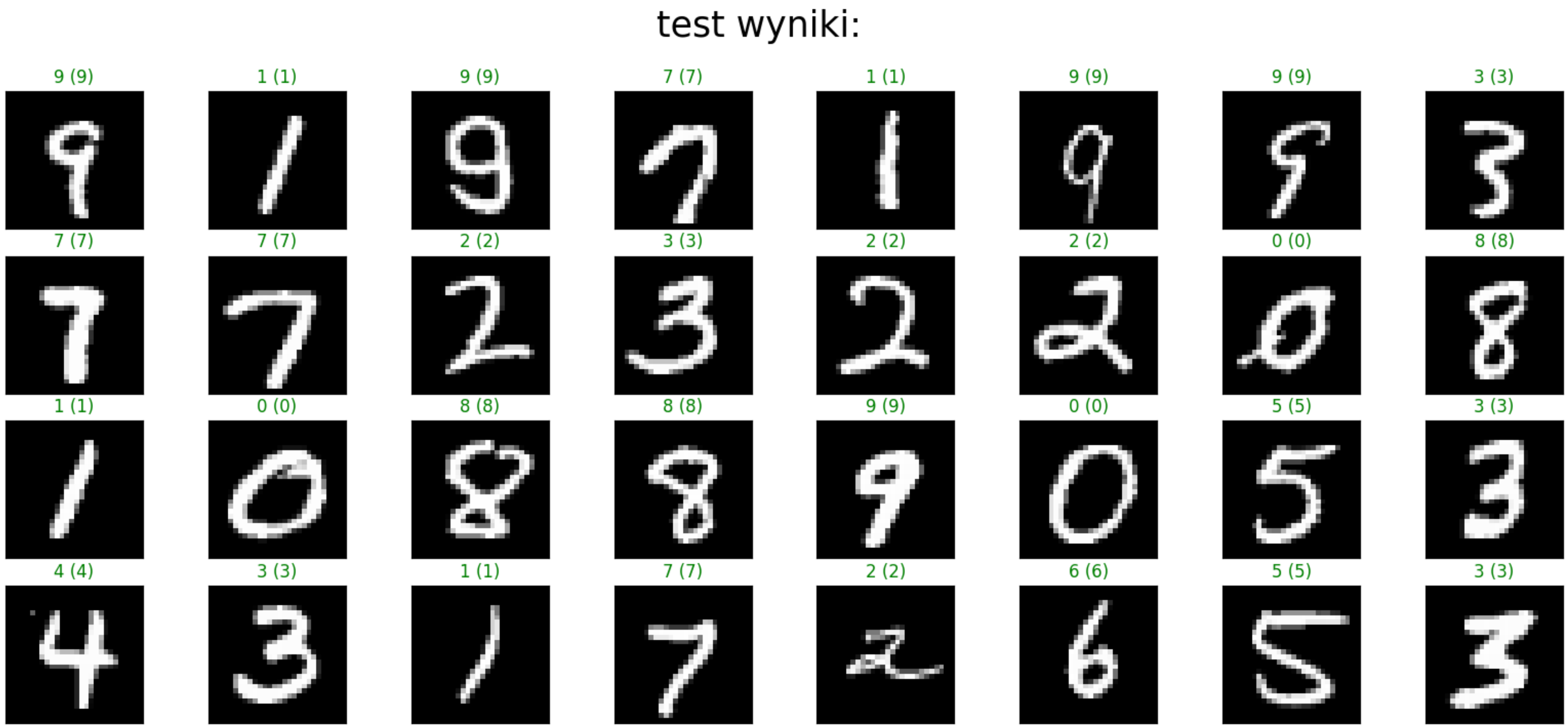
Training and validation loss
Training loss
Validation loss
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50

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.

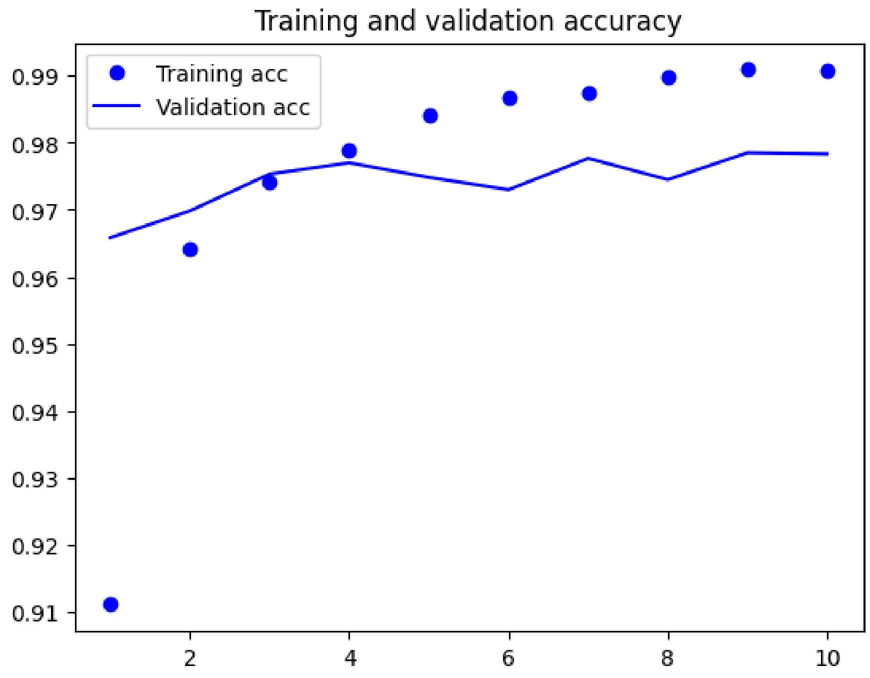
```
model = tf.keras.Sequential()

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])
```

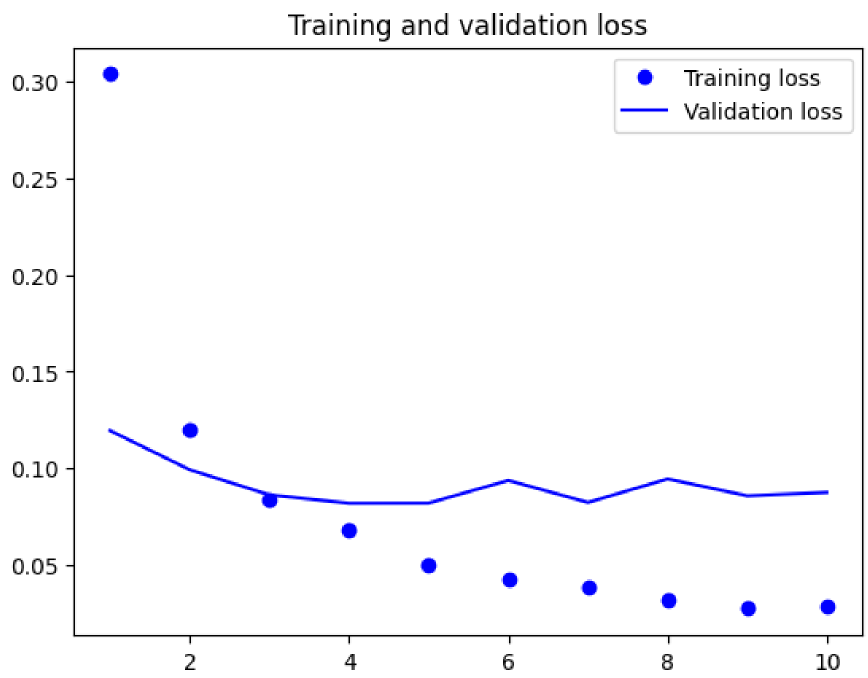
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_3 (Dense)             (None, 128)               100480

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 64)                4160

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (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 [==============================] - 5s 4ms/step - loss: 0.3040 - accuracy: 0.9111 - val_loss: 0.1194 - val_accuracy: 0.9658
Epoch 2/10
844/844 [==============================] - 4s 4ms/step - loss: 0.1197 - accuracy: 0.9641 - val_loss: 0.0991 - val_accuracy: 0.9698
Epoch 3/10
844/844 [==============================] - 4s 4ms/step - loss: 0.0836 - accuracy: 0.9741 - val_loss: 0.0861 - val_accuracy: 0.9753
Epoch 4/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0677 - accuracy: 0.9789 - val_loss: 0.0818 - val_accuracy: 0.9770
```

```
Epoch 5/10
844/844 [==============================] - 4s 5ms/step - loss: 0.0501 - accuracy: 0.9842 - val_loss: 0.0818 - val_accuracy: 0.9748
Epoch 6/10
844/844 [==============================] - 4s 5ms/step - loss: 0.0426 - accuracy: 0.9868 - val_loss: 0.0935 - val_accuracy: 0.9730
Epoch 7/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0381 - accuracy: 0.9874 - val_loss: 0.0822 - val_accuracy: 0.9777
Epoch 8/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0318 - accuracy: 0.9898 - val_loss: 0.0943 - val_accuracy: 0.9745
Epoch 9/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0272 - accuracy: 0.9909 - val_loss: 0.0856 - val_accuracy: 0.9785
Epoch 10/10
844/844 [==============================] - 4s 5ms/step - loss: 0.0283 - accuracy: 0.9908 - val_loss: 0.0873 - val_accuracy: 0.9783
Precyzja:  0.9760000109672546
```

Wykresy precyzji i błędu

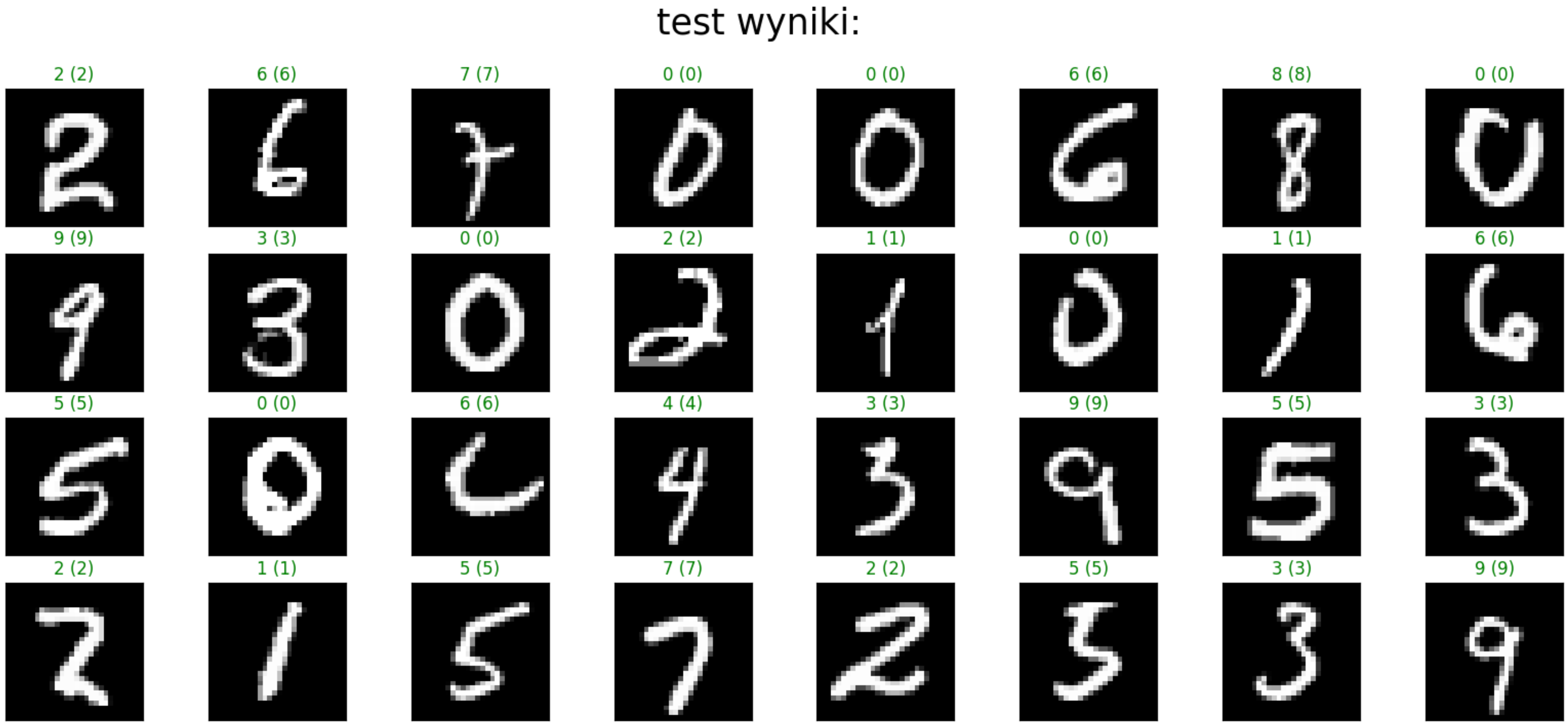
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.30
0.25
0.20
0.15
0.10
0.05
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()
```

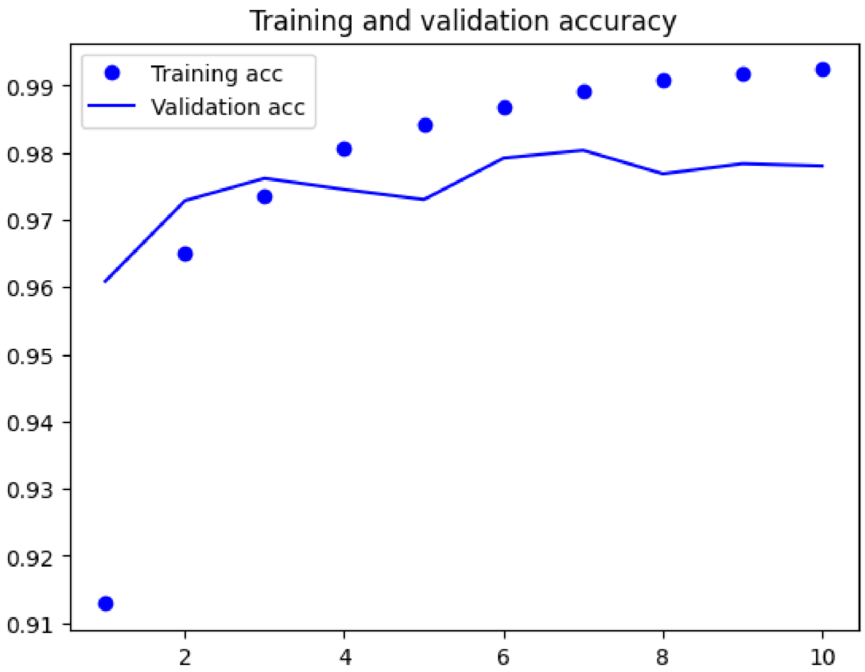
```
model.add(tf.keras.layers.Flatten(input_shape=(28,28,1)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

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])
```

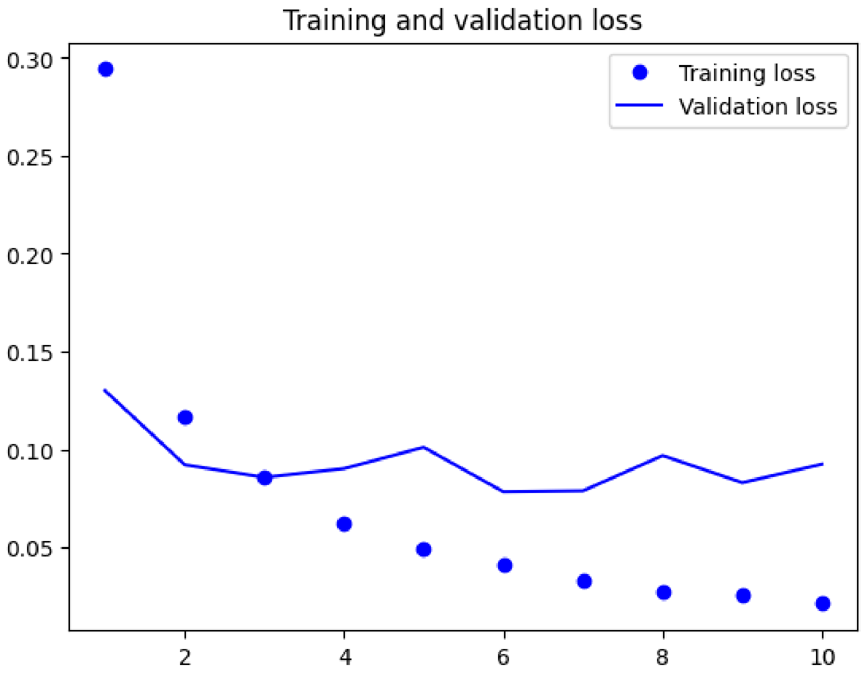
```
Model: "sequential_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_28 (Flatten)        (None, 784)               0

 dense_47 (Dense)            (None, 128)               100480

 dense_48 (Dense)            (None, 64)                8256

 dense_49 (Dense)            (None, 32)                2080

 dense_50 (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 [==============================] - 4s 4ms/step - loss: 0.2942 - accuracy: 0.9129 - val_loss: 0.1300 - val_accuracy: 0.9608
Epoch 2/10
844/844 [==============================] - 4s 5ms/step - loss: 0.1168 - accuracy: 0.9649 - val_loss: 0.0922 - val_accuracy: 0.9728
Epoch 3/10
844/844 [==============================] - 4s 4ms/step - loss: 0.0854 - accuracy: 0.9734 - val_loss: 0.0857 - val_accuracy: 0.9762
Epoch 4/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0623 - accuracy: 0.9807 - val_loss: 0.0902 - val_accuracy: 0.9745
Epoch 5/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0492 - accuracy: 0.9841 - val_loss: 0.1011 - val_accuracy: 0.9730
Epoch 6/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0413 - accuracy: 0.9867 - val_loss: 0.0784 - val_accuracy: 0.9792
Epoch 7/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0334 - accuracy: 0.9891 - val_loss: 0.0789 - val_accuracy: 0.9803
Epoch 8/10
```

```
844/844 [==============================] - 3s 4ms/step - loss: 0.0277 - accuracy: 0.9909 - val_loss: 0.0969 - val_accuracy: 0.9768
Epoch 9/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0257 - accuracy: 0.9916 - val_loss: 0.0830 - val_accuracy: 0.9783
Epoch 10/10
844/844 [==============================] - 4s 4ms/step - loss: 0.0213 - accuracy: 0.9923 - val_loss: 0.0925 - val_accuracy: 0.9780
Precyzja:  0.9775000214576721
```

Wykresy precyzji i błędu

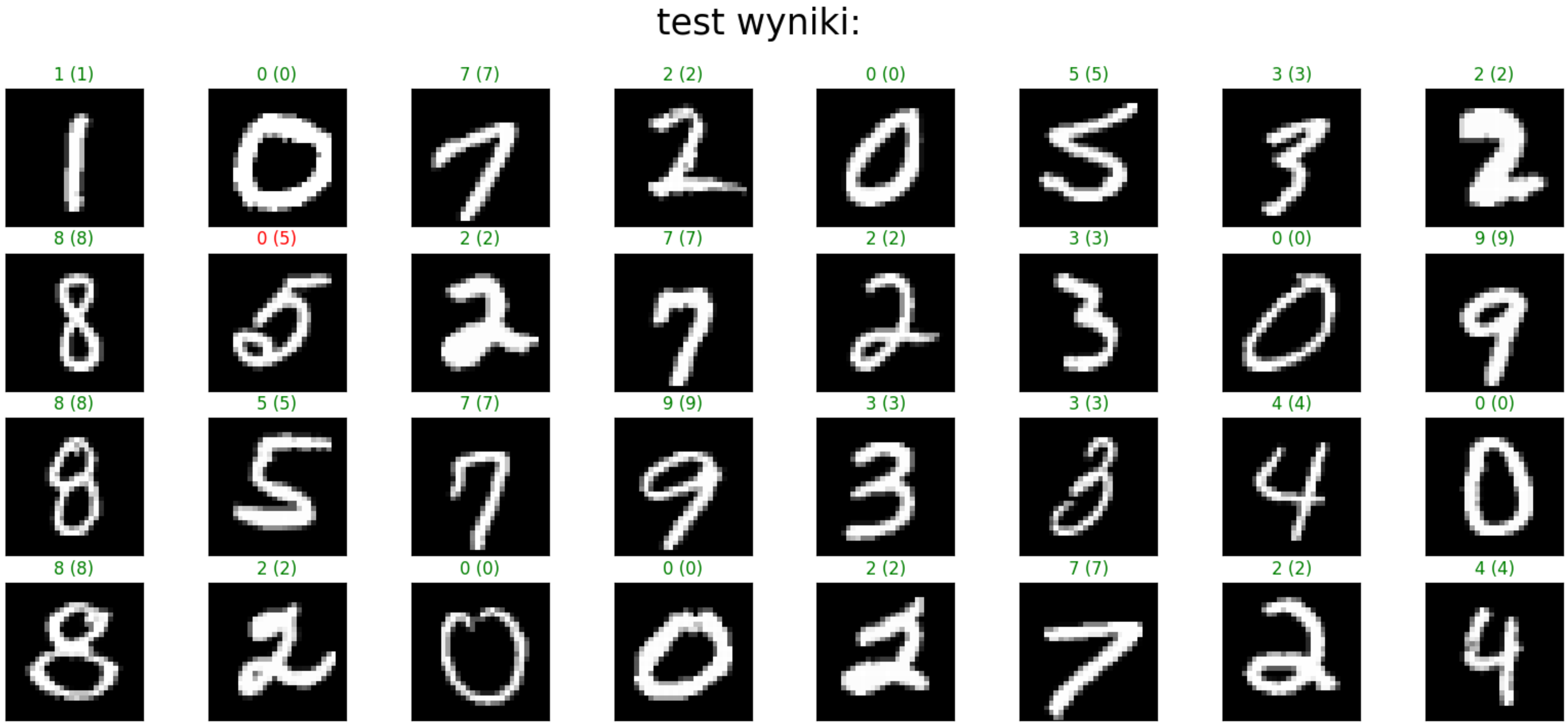
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.30
0.25
0.20
0.15
0.10
0.05
2
4
6
8
10

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

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



2.2 Pogłębiamy model v3

Dodajmy 4 warstwy gęste.

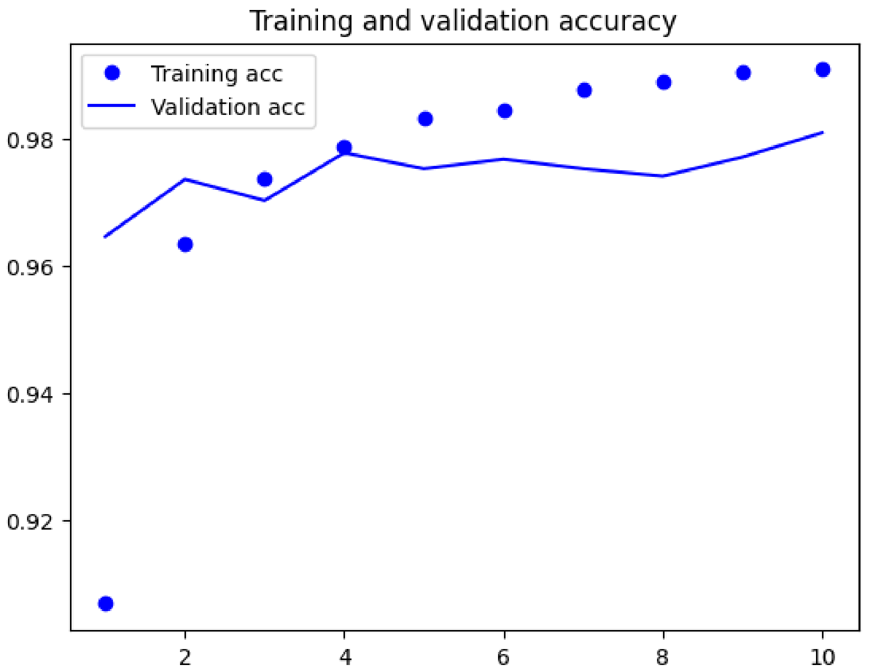
```
model = tf.keras.Sequential()

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])
```

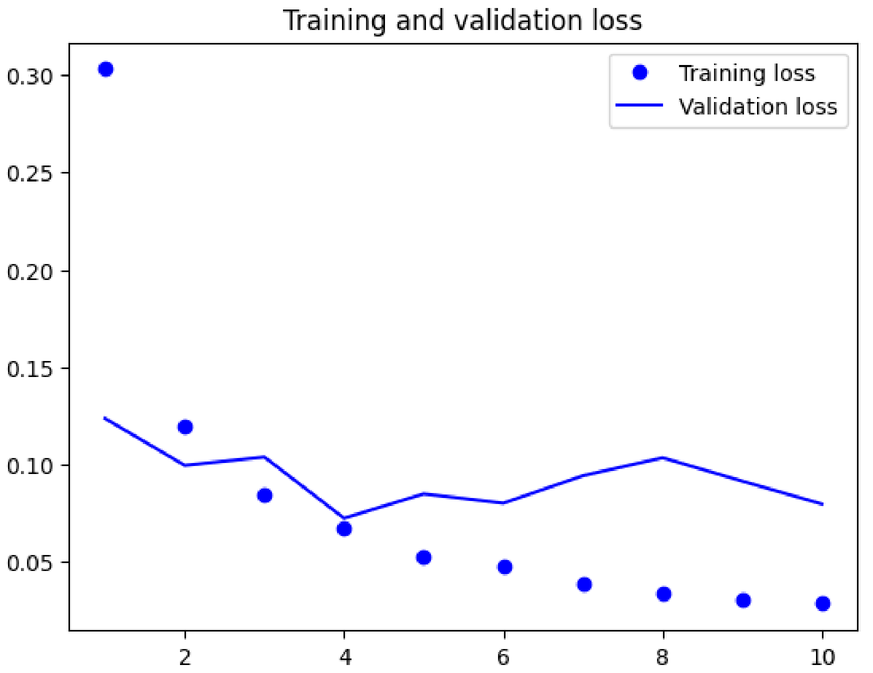
```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_29 (Flatten)        (None, 784)               0

 dense_51 (Dense)            (None, 128)               100480

 dense_52 (Dense)            (None, 128)               16512

 dense_53 (Dense)            (None, 64)                8256

 dense_54 (Dense)            (None, 64)                4160

 dense_55 (Dense)            (None, 32)                2080

 dense_56 (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 [==============================] - 6s 5ms/step - loss: 0.3030 - accuracy: 0.9069 - val_loss: 0.1236 - val_accuracy: 0.9647
Epoch 2/10
844/844 [==============================] - 3s 4ms/step - loss: 0.1200 - accuracy: 0.9635 - val_loss: 0.0996 - val_accuracy: 0.9737
Epoch 3/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0849 - accuracy: 0.9738 - val_loss: 0.1039 - val_accuracy: 0.9703
Epoch 4/10
```

```
844/844 [==============================] - 3s 4ms/step - loss: 0.0672 - accuracy: 0.9789 - val_loss: 0.0725 - val_accuracy: 0.9778
Epoch 5/10
844/844 [==============================] - 4s 5ms/step - loss: 0.0528 - accuracy: 0.9832 - val_loss: 0.0849 - val_accuracy: 0.9753
Epoch 6/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0474 - accuracy: 0.9846 - val_loss: 0.0802 - val_accuracy: 0.9768
Epoch 7/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0387 - accuracy: 0.9878 - val_loss: 0.0943 - val_accuracy: 0.9753
Epoch 8/10
844/844 [==============================] - 4s 5ms/step - loss: 0.0339 - accuracy: 0.9890 - val_loss: 0.1035 - val_accuracy: 0.9742
Epoch 9/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0303 - accuracy: 0.9905 - val_loss: 0.0915 - val_accuracy: 0.9772
Epoch 10/10
844/844 [==============================] - 3s 4ms/step - loss: 0.0286 - accuracy: 0.9909 - val_loss: 0.0797 - val_accuracy: 0.9810
Precyzja:  0.978600025177002
```

Wykresy precyzji i błędu

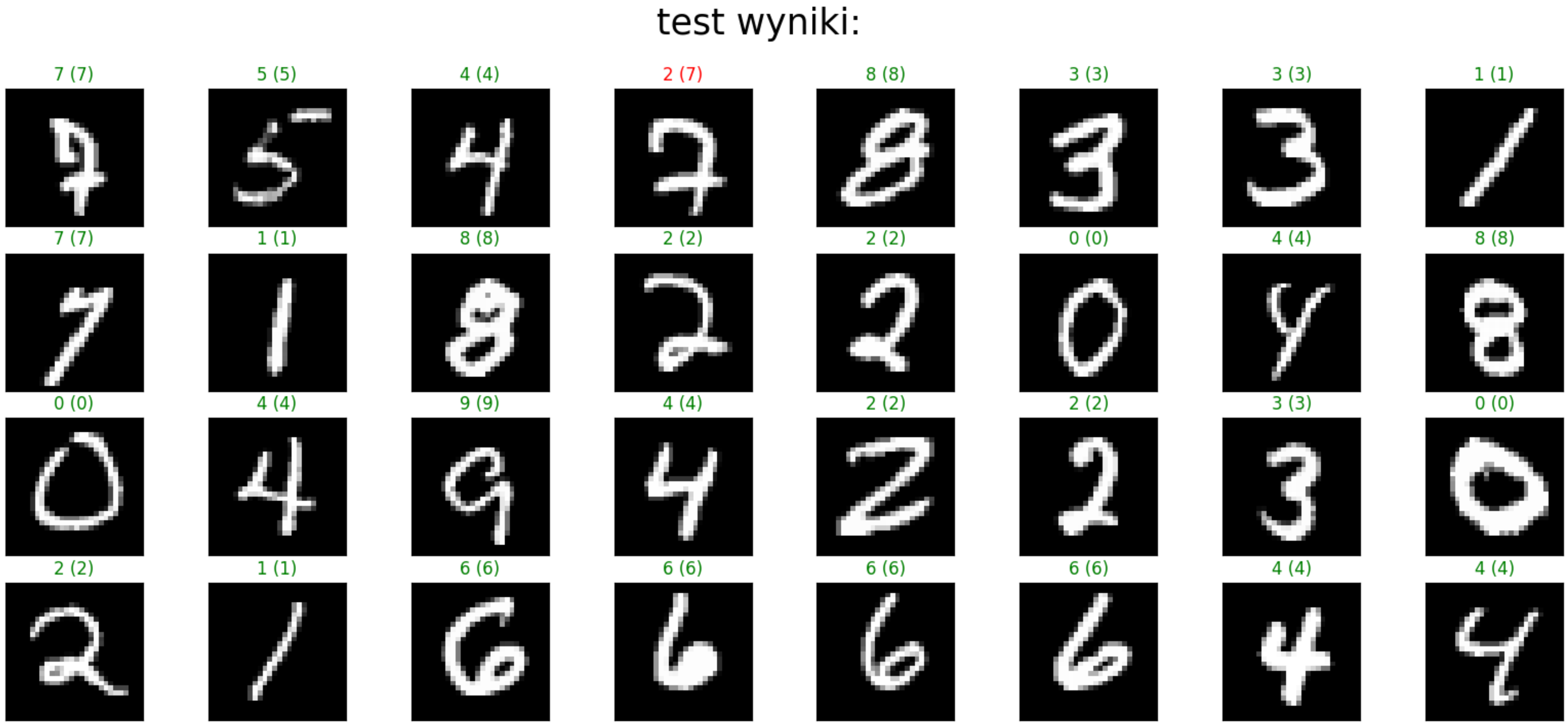
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.98
0.96
0.94
0.92
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.30
0.25
0.20
0.15
0.10
0.05
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_3"
_________________________________________________________________
 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_3 (Flatten)         (None, 98)                0

 dense_8 (Dense)             (None, 128)               12672

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (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: 0.6456 - accuracy: 0.8120 - val_loss: 0.2564 - val_accuracy: 0.9202
Epoch 2/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2554 - accuracy: 0.9202 - val_loss: 0.1794 - val_accuracy: 0.9455
Epoch 3/25
422/422 [==============================] - 2s 4ms/step - loss: 0.1903 - accuracy: 0.9398 - val_loss: 0.1564 - val_accuracy: 0.9518
Epoch 4/25
```

```
422/422 [==============================] - 2s 4ms/step - loss: 0.1560 - accuracy: 0.9503 - val_loss: 0.1370 - val_accuracy: 0.9587
Epoch 5/25
422/422 [==============================] - 2s 4ms/step - loss: 0.1347 - accuracy: 0.9570 - val_loss: 0.1276 - val_accuracy: 0.9618
Epoch 6/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1176 - accuracy: 0.9624 - val_loss: 0.1130 - val_accuracy: 0.9675
Epoch 7/25
422/422 [==============================] - 2s 4ms/step - loss: 0.1051 - accuracy: 0.9662 - val_loss: 0.1122 - val_accuracy: 0.9672
Epoch 8/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0954 - accuracy: 0.9688 - val_loss: 0.1011 - val_accuracy: 0.9723
Epoch 9/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0891 - accuracy: 0.9710 - val_loss: 0.1059 - val_accuracy: 0.9703
Epoch 10/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0811 - accuracy: 0.9736 - val_loss: 0.0932 - val_accuracy: 0.9740
Epoch 11/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0757 - accuracy: 0.9755 - val_loss: 0.1021 - val_accuracy: 0.9713
Epoch 12/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0702 - accuracy: 0.9768 - val_loss: 0.0919 - val_accuracy: 0.9745
Epoch 13/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0659 - accuracy: 0.9791 - val_loss: 0.1043 - val_accuracy: 0.9717
Epoch 14/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0615 - accuracy: 0.9796 - val_loss: 0.0942 - val_accuracy: 0.9735
Epoch 15/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0579 - accuracy: 0.9803 - val_loss: 0.0941 - val_accuracy: 0.9752
Epoch 16/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0523 - accuracy: 0.9828 - val_loss: 0.0899 - val_accuracy: 0.9760
Epoch 17/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0494 - accuracy: 0.9837 - val_loss: 0.0909 - val_accuracy: 0.9752
Epoch 18/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0469 - accuracy: 0.9847 - val_loss: 0.0942 - val_accuracy: 0.9740
Epoch 19/25
422/422 [==============================] - 2s 5ms/step - loss: 0.0436 - accuracy: 0.9854 - val_loss: 0.0986 - val_accuracy: 0.9753
Epoch 20/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0408 - accuracy: 0.9864 - val_loss: 0.0929 - val_accuracy: 0.9763
Epoch 21/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0389 - accuracy: 0.9869 - val_loss: 0.1009 - val_accuracy: 0.9770
Epoch 22/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0362 - accuracy: 0.9876 - val_loss: 0.1025 - val_accuracy: 0.9748
Epoch 23/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0345 - accuracy: 0.9885 - val_loss: 0.1083 - val_accuracy: 0.9752
Epoch 24/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0320 - accuracy: 0.9896 - val_loss: 0.1095 - val_accuracy: 0.9757
Epoch 25/25
422/422 [==============================] - 2s 4ms/step - loss: 0.0312 - accuracy: 0.9893 - val_loss: 0.1088 - val_accuracy: 0.9732
```

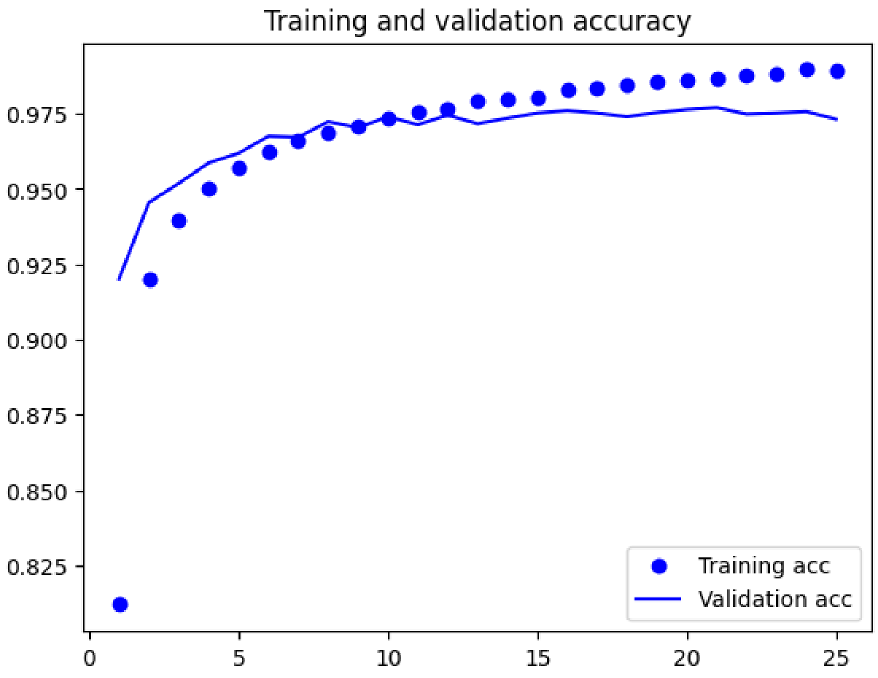
Ewaluacja modelu

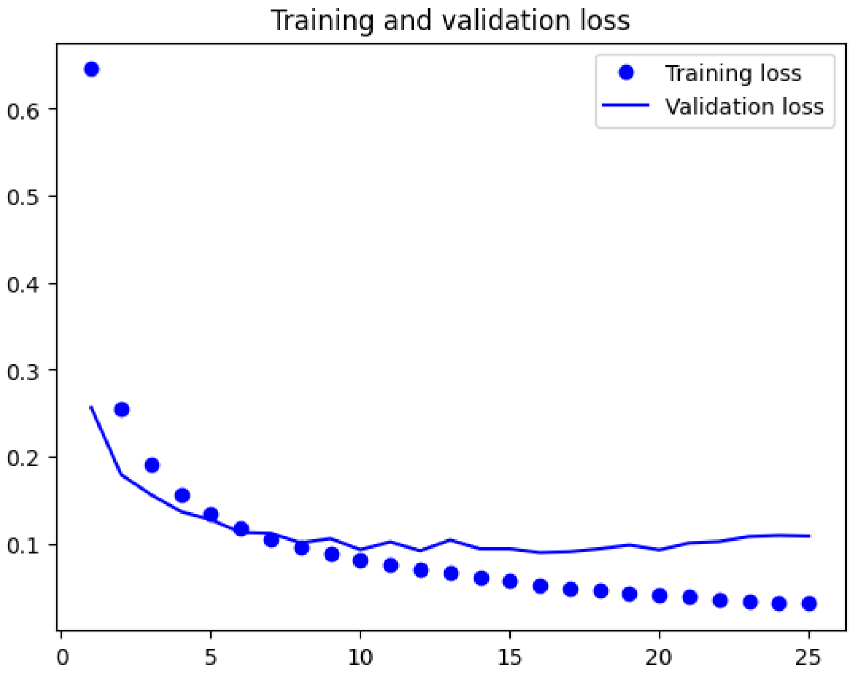
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.9750000238418579
```

Wykresy precyzji i błędu

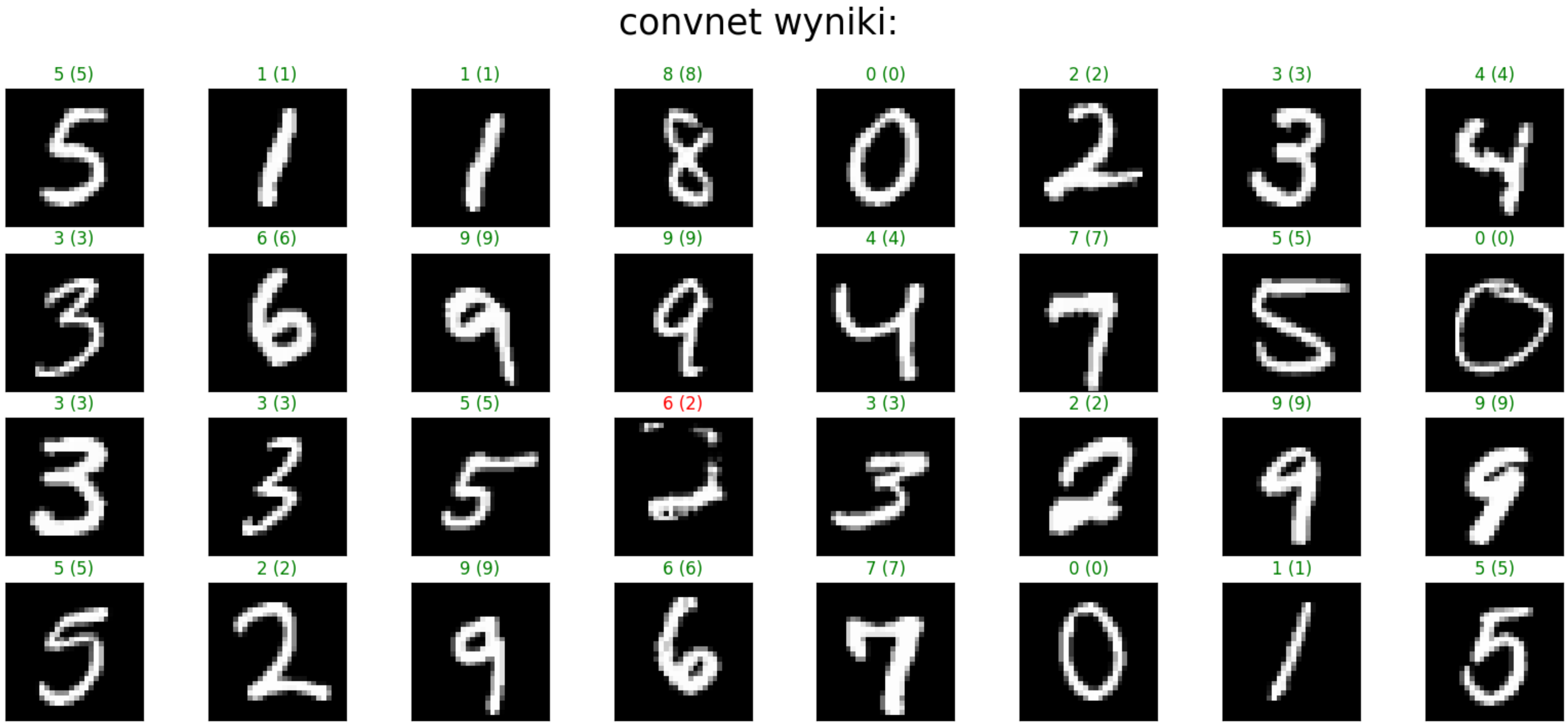
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0
5
10
15
20
25


Training and validation loss
Training loss
Validation loss
0.6
0.5
0.4
0.3
0.2
0.1
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_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_14 (Conv2D)          (None, 28, 28, 4)         20        
                                                                 
 max_pooling2d_11 (MaxPooli  (None, 14, 14, 4)         0         
 ng2D)                                                           
                                                                 
 conv2d_15 (Conv2D)          (None, 14, 14, 2)         34        
                                                                 
 max_pooling2d_12 (MaxPooli  (None, 7, 7, 2)           0         
 ng2D)                                                           
                                                                 
 conv2d_16 (Conv2D)          (None, 7, 7, 2)           18        
                                                                 
 max_pooling2d_13 (MaxPooli  (None, 3, 3, 2)           0         
 ng2D)                                                           
                                                                 
 flatten_33 (Flatten)        (None, 18)                0         
                                                                 
 dense_66 (Dense)            (None, 128)               2432      
                                                                 
 dense_67 (Dense)            (None, 64)                8256      
                                                                 
 dense_68 (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 6ms/step - loss: 0.8775 - accuracy: 0.7366 - val_loss: 0.3990 - val_accuracy: 0.8743
Epoch 2/25
422/422 [==============================] - 3s 6ms/step - loss: 0.4237 - accuracy: 0.8690 - val_loss: 0.3221 - val_accuracy: 0.8957
Epoch 3/25
422/422 [==============================] - 2s 4ms/step - loss: 0.3533 - accuracy: 0.8911 - val_loss: 0.2699 - val_accuracy: 0.9163
Epoch 4/25
422/422 [==============================] - 2s 5ms/step - loss: 0.3125 - accuracy: 0.9053 - val_loss: 0.2398 - val_accuracy: 0.9233
Epoch 5/25
422/422 [==============================] - 2s 5ms/step - loss: 0.2787 - accuracy: 0.9141 - val_loss: 0.2199 - val_accuracy: 0.9333
Epoch 6/25
422/422 [==============================] - 2s 5ms/step - loss: 0.2535 - accuracy: 0.9216 - val_loss: 0.2120 - val_accuracy: 0.9320
Epoch 7/25
422/422 [==============================] - 2s 4ms/step - loss: 0.2363 - accuracy: 0.9268 - val_loss: 0.1925 - val_accuracy: 0.9370
Epoch 8/25
422/422 [==============================] - 3s 6ms/step - loss: 0.2213 - accuracy: 0.9307 - val_loss: 0.1857 - val_accuracy: 0.9400
Epoch 9/25
422/422 [==============================] - 2s 5ms/step - loss: 0.2109 - accuracy: 0.9335 - val_loss: 0.1840 - val_accuracy: 0.9407
Epoch 10/25
422/422 [==============================] - 2s 5ms/step - loss: 0.1997 - accuracy: 0.9374 - val_loss: 0.1729 - val_accuracy: 0.9435
Epoch 11/25
422/422 [==============================] - 2s 5ms/step - loss: 0.1901 - accuracy: 0.9395 - val_loss: 0.1750 - val_accuracy: 0.9435
Epoch 12/25
422/422 [==============================] - 2s 4ms/step - loss: 0.1848 - accuracy: 0.9411 - val_loss: 0.1595 - val_accuracy: 0.9478
Epoch 13/25
422/422 [==============================] - 2s 4ms/step - loss: 0.1783 - accuracy: 0.9439 - val_loss: 0.1611 - val_accuracy: 0.9467
Epoch 14/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1728 - accuracy: 0.9449 - val_loss: 0.1623 - val_accuracy: 0.9458
Epoch 15/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1694 - accuracy: 0.9457 - val_loss: 0.1513 - val_accuracy: 0.9522
Epoch 16/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1626 - accuracy: 0.9479 - val_loss: 0.1536 - val_accuracy: 0.9505
Epoch 17/25
422/422 [==============================] - 2s 5ms/step - loss: 0.1605 - accuracy: 0.9482 - val_loss: 0.1481 - val_accuracy: 0.9528
Epoch 18/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1578 - accuracy: 0.9501 - val_loss: 0.1495 - val_accuracy: 0.9508
Epoch 19/25
422/422 [==============================] - 4s 8ms/step - loss: 0.1535 - accuracy: 0.9511 - val_loss: 0.1503 - val_accuracy: 0.9513
Epoch 20/25
422/422 [==============================] - 2s 6ms/step - loss: 0.1507 - accuracy: 0.9519 - val_loss: 0.1469 - val_accuracy: 0.9523
Epoch 21/25
422/422 [==============================] - 2s 5ms/step - loss: 0.1488 - accuracy: 0.9530 - val_loss: 0.1425 - val_accuracy: 0.9537
```

```
Epoch 22/25
422/422 [==============================] - 2s 5ms/step - loss: 0.1447 - accuracy: 0.9536 - val_loss: 0.1412 - val_accuracy: 0.9553
Epoch 23/25
422/422 [==============================] - 3s 6ms/step - loss: 0.1428 - accuracy: 0.9538 - val_loss: 0.1436 - val_accuracy: 0.9560
Epoch 24/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1421 - accuracy: 0.9536 - val_loss: 0.1441 - val_accuracy: 0.9540
Epoch 25/25
422/422 [==============================] - 3s 6ms/step - loss: 0.1373 - accuracy: 0.9554 - val_loss: 0.1466 - val_accuracy: 0.9518
```

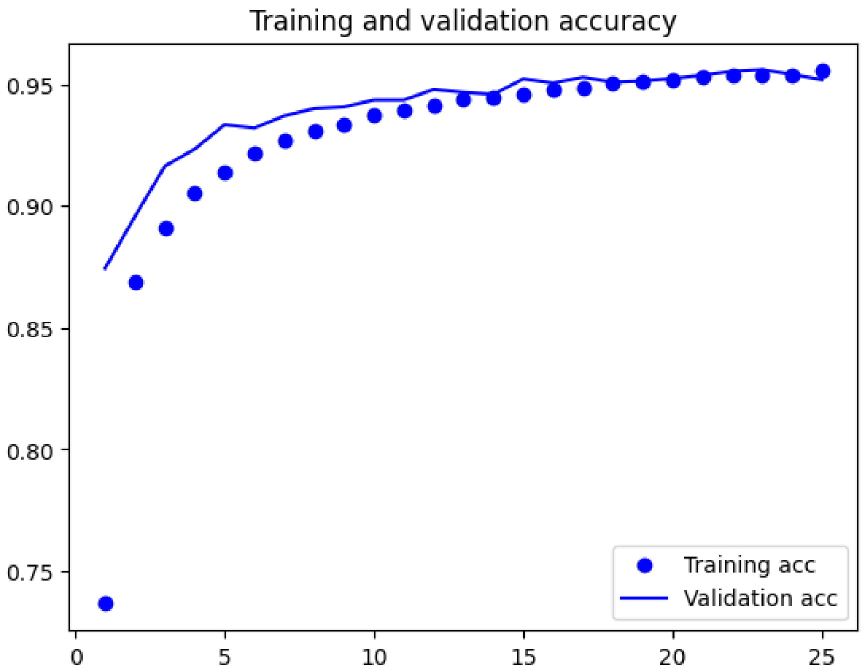
Ewaluacja modelu

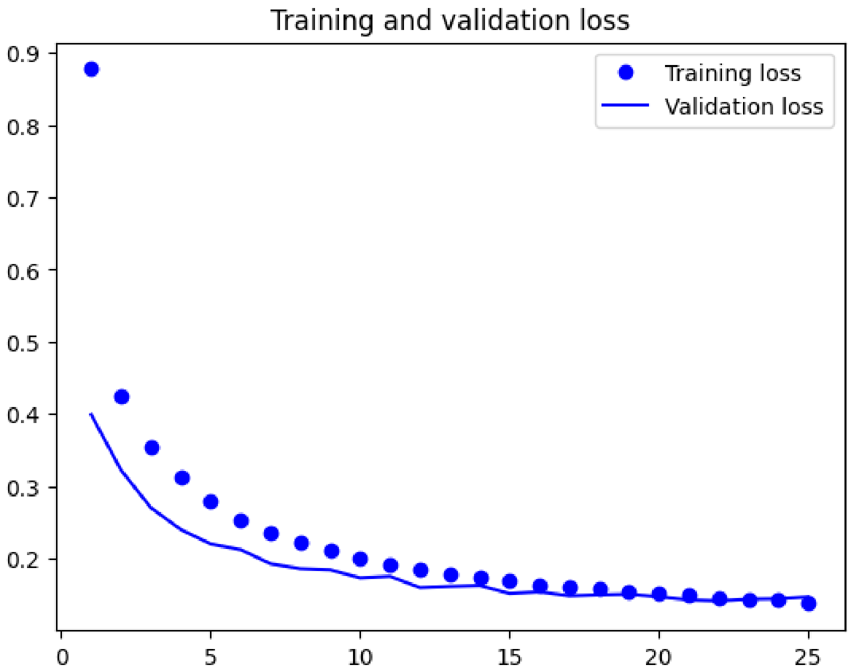
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.9480000138282776
```

Wykresy precyzji i błędu

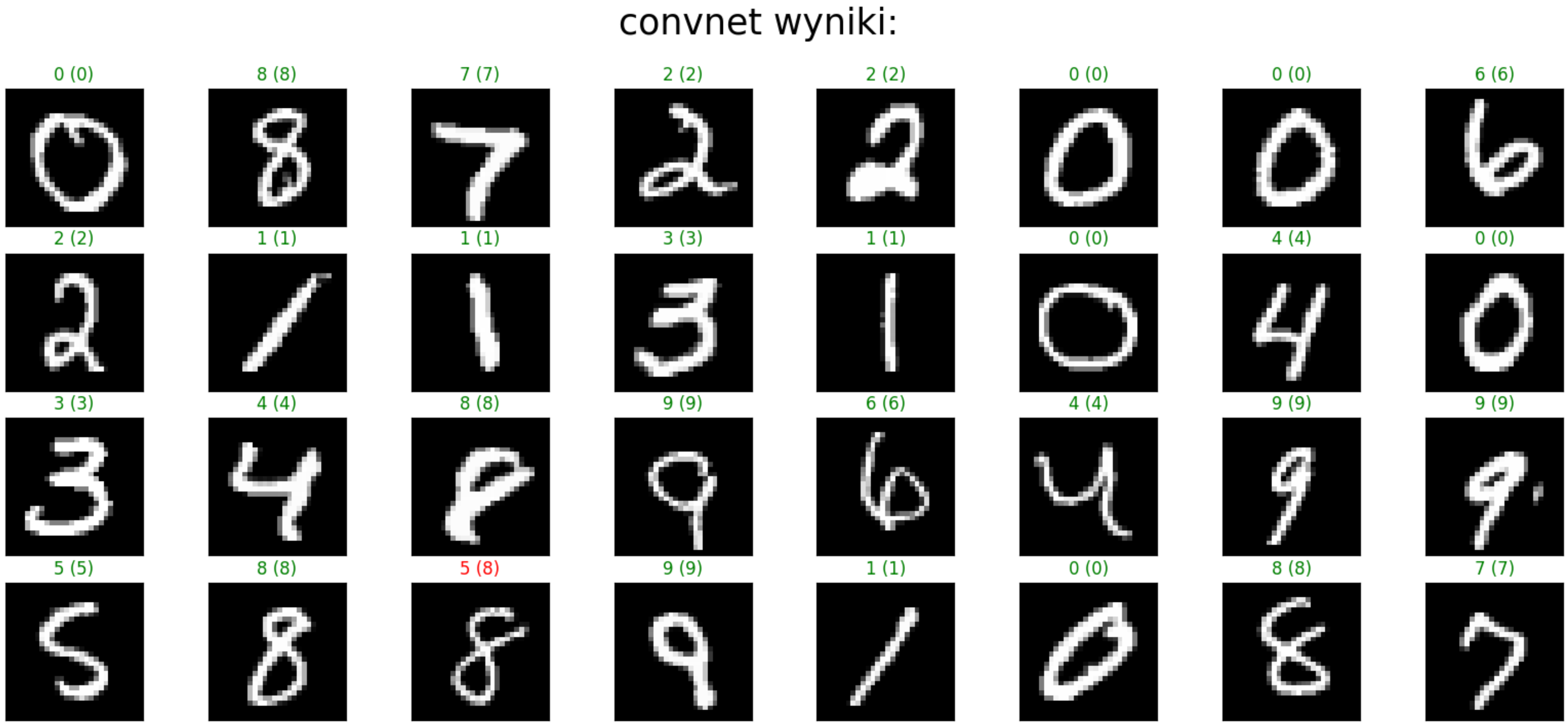
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
Training acc
Validation acc


Training and validation loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
Training loss
Validation loss

```
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_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_17 (Conv2D)          (None, 28, 28, 64)        320       
                                                                 
 max_pooling2d_14 (MaxPooli  (None, 14, 14, 64)        0         
 ng2D)                                                           
                                                                 
 conv2d_18 (Conv2D)          (None, 14, 14, 32)        8224      
                                                                 
 max_pooling2d_15 (MaxPooli  (None, 7, 7, 32)          0         
 ng2D)                                                           
                                                                 
 conv2d_19 (Conv2D)          (None, 7, 7, 16)          2064      
                                                                 
 max_pooling2d_16 (MaxPooli  (None, 3, 3, 16)          0         
 ng2D)                                                           
                                                                 
 flatten_34 (Flatten)        (None, 144)               0         
                                                                 
 dense_69 (Dense)            (None, 128)               18560     
                                                                 
 dense_70 (Dense)            (None, 64)                8256      
                                                                 
 dense_71 (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 [==============================] - 6s 8ms/step - loss: 0.3922 - accuracy: 0.8789 - val_loss: 0.1028 - val_accuracy: 0.9685
Epoch 2/25
422/422 [==============================] - 3s 6ms/step - loss: 0.1002 - accuracy: 0.9681 - val_loss: 0.0702 - val_accuracy: 0.9792
Epoch 3/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0714 - accuracy: 0.9776 - val_loss: 0.0678 - val_accuracy: 0.9785
Epoch 4/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0596 - accuracy: 0.9811 - val_loss: 0.0638 - val_accuracy: 0.9812
Epoch 5/25
422/422 [==============================] - 4s 8ms/step - loss: 0.0479 - accuracy: 0.9846 - val_loss: 0.0465 - val_accuracy: 0.9845
Epoch 6/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0433 - accuracy: 0.9863 - val_loss: 0.0469 - val_accuracy: 0.9850
Epoch 7/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0377 - accuracy: 0.9880 - val_loss: 0.0464 - val_accuracy: 0.9853
Epoch 8/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0327 - accuracy: 0.9897 - val_loss: 0.0532 - val_accuracy: 0.9843
Epoch 9/25
422/422 [==============================] - 4s 9ms/step - loss: 0.0311 - accuracy: 0.9898 - val_loss: 0.0478 - val_accuracy: 0.9865
Epoch 10/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0280 - accuracy: 0.9911 - val_loss: 0.0412 - val_accuracy: 0.9878
Epoch 11/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0262 - accuracy: 0.9917 - val_loss: 0.0480 - val_accuracy: 0.9845
Epoch 12/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0230 - accuracy: 0.9922 - val_loss: 0.0362 - val_accuracy: 0.9902
Epoch 13/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0209 - accuracy: 0.9932 - val_loss: 0.0338 - val_accuracy: 0.9893
Epoch 14/25
422/422 [==============================] - 3s 6ms/step - loss: 0.0201 - accuracy: 0.9928 - val_loss: 0.0513 - val_accuracy: 0.9872
Epoch 15/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0180 - accuracy: 0.9938 - val_loss: 0.0433 - val_accuracy: 0.9898
Epoch 16/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0158 - accuracy: 0.9947 - val_loss: 0.0382 - val_accuracy: 0.9888
Epoch 17/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0141 - accuracy: 0.9951 - val_loss: 0.0341 - val_accuracy: 0.9910
Epoch 18/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0144 - accuracy: 0.9950 - val_loss: 0.0321 - val_accuracy: 0.9915
Epoch 19/25
422/422 [==============================] - 3s 6ms/step - loss: 0.0116 - accuracy: 0.9960 - val_loss: 0.0480 - val_accuracy: 0.9872
Epoch 20/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0122 - accuracy: 0.9958 - val_loss: 0.0483 - val_accuracy: 0.9877
Epoch 21/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0130 - accuracy: 0.9955 - val_loss: 0.0352 - val_accuracy: 0.9908
Epoch 22/25
422/422 [==============================] - 3s 6ms/step - loss: 0.0113 - accuracy: 0.9960 - val_loss: 0.0423 - val_accuracy: 0.9888
Epoch 23/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0110 - accuracy: 0.9966 - val_loss: 0.0361 - val_accuracy: 0.9910
```

```
Epoch 24/25
422/422 [==============================] - 3s 6ms/step - loss: 0.0087 - accuracy: 0.9970 - val_loss: 0.0417 - val_accuracy: 0.9892
Epoch 25/25
422/422 [==============================] - 2s 6ms/step - loss: 0.0104 - accuracy: 0.9965 - val_loss: 0.0380 - val_accuracy: 0.9905
```

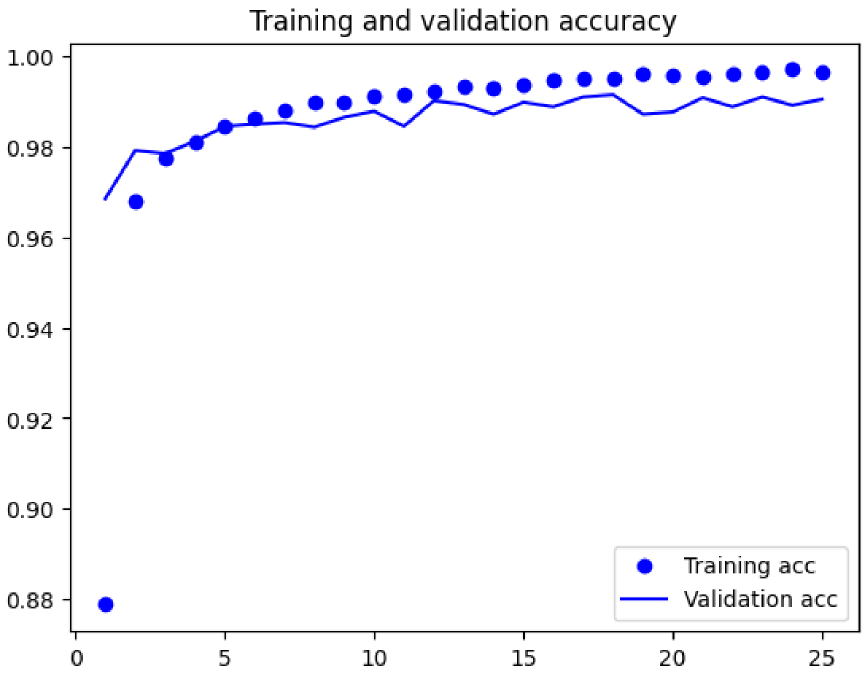
Ewaluacja modelu

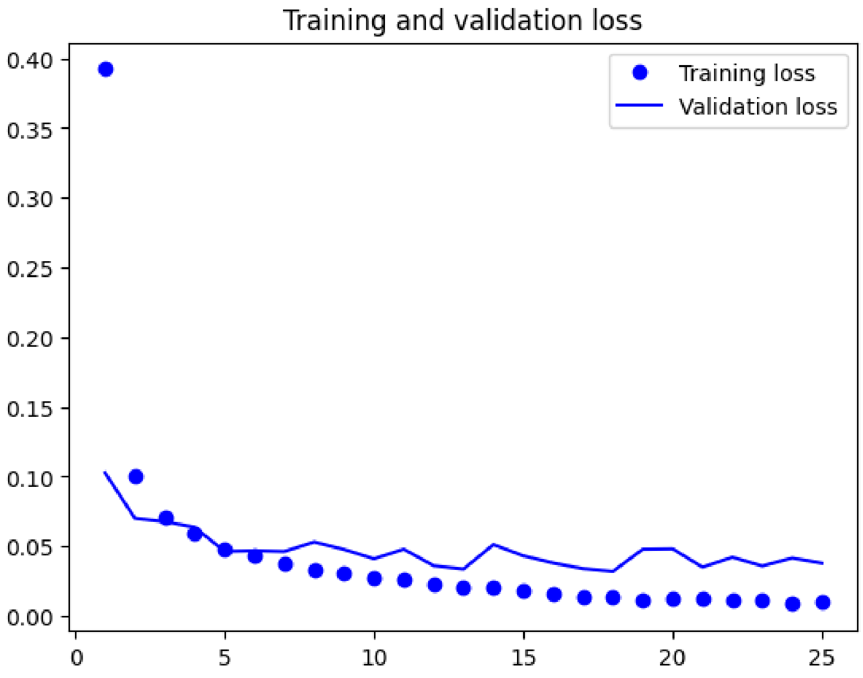
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Precyzja: ',score[1])
```

```
Precyzja:  0.9902999997138977
```

Wykresy precyzji i błędu

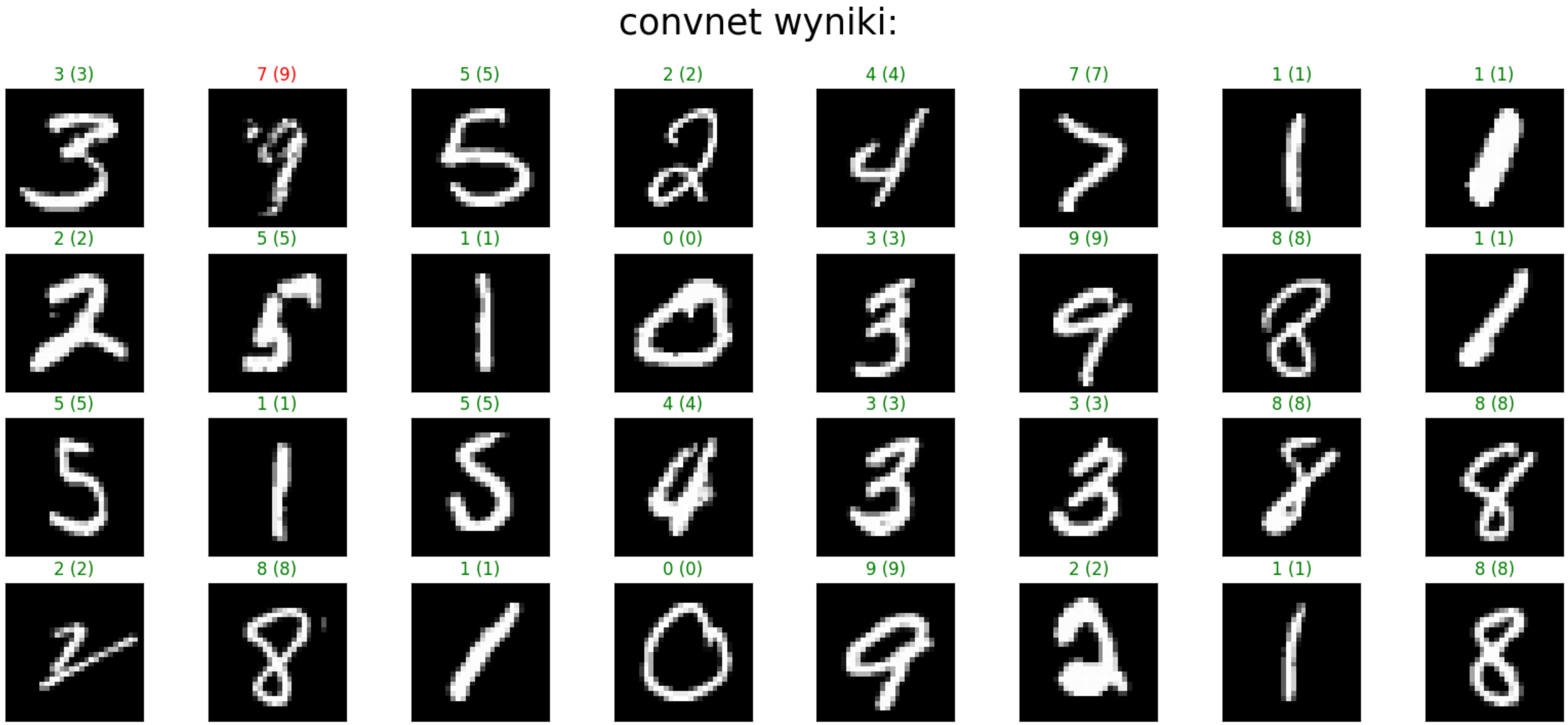
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
5
10
15
20
25
Training acc
Validation acc


Training and validation loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
Training loss
Validation loss

```
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_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 64)        320

 max_pooling2d_4 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

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

 conv2d_6 (Conv2D)           (None, 14, 14, 32)        8224

 max_pooling2d_5 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 dropout_4 (Dropout)         (None, 7, 7, 32)          0

 conv2d_7 (Conv2D)           (None, 7, 7, 16)          2064

 flatten_30 (Flatten)        (None, 784)               0

 dropout_5 (Dropout)         (None, 784)               0
```

```
 dense_57 (Dense)            (None, 128)               100480

 dense_58 (Dense)            (None, 64)                8256

 dense_59 (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 [==============================] - 6s 9ms/step - loss: 0.6871 - accuracy: 0.7697 - val_loss: 0.1493 - val_accuracy: 0.9603
Epoch 2/25
422/422 [==============================] - 4s 10ms/step - loss: 0.2656 - accuracy: 0.9146 - val_loss: 0.0969 - val_accuracy: 0.9705
Epoch 3/25
422/422 [==============================] - 3s 7ms/step - loss: 0.2066 - accuracy: 0.9339 - val_loss: 0.0831 - val_accuracy: 0.9742
Epoch 4/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1753 - accuracy: 0.9437 - val_loss: 0.0647 - val_accuracy: 0.9808
Epoch 5/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1546 - accuracy: 0.9506 - val_loss: 0.0589 - val_accuracy: 0.9847
Epoch 6/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1381 - accuracy: 0.9556 - val_loss: 0.0532 - val_accuracy: 0.9835
Epoch 7/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1296 - accuracy: 0.9588 - val_loss: 0.0505 - val_accuracy: 0.9863
Epoch 8/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1240 - accuracy: 0.9605 - val_loss: 0.0492 - val_accuracy: 0.9860
Epoch 9/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1163 - accuracy: 0.9628 - val_loss: 0.0448 - val_accuracy: 0.9877
Epoch 10/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1094 - accuracy: 0.9654 - val_loss: 0.0418 - val_accuracy: 0.9873
Epoch 11/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1035 - accuracy: 0.9666 - val_loss: 0.0421 - val_accuracy: 0.9882
Epoch 12/25
422/422 [==============================] - 3s 7ms/step - loss: 0.1044 - accuracy: 0.9664 - val_loss: 0.0404 - val_accuracy: 0.9888
Epoch 13/25
422/422 [==============================] - 3s 8ms/step - loss: 0.1016 - accuracy: 0.9683 - val_loss: 0.0393 - val_accuracy: 0.9888
Epoch 14/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0960 - accuracy: 0.9684 - val_loss: 0.0390 - val_accuracy: 0.9883
Epoch 15/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0930 - accuracy: 0.9702 - val_loss: 0.0351 - val_accuracy: 0.9905
Epoch 16/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0900 - accuracy: 0.9710 - val_loss: 0.0358 - val_accuracy: 0.9898
```

```
Epoch 17/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0872 - accuracy: 0.9719 - val_loss: 0.0354 - val_accuracy: 0.9890
Epoch 18/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0874 - accuracy: 0.9716 - val_loss: 0.0334 - val_accuracy: 0.9912
Epoch 19/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0854 - accuracy: 0.9728 - val_loss: 0.0352 - val_accuracy: 0.9895
Epoch 20/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0838 - accuracy: 0.9731 - val_loss: 0.0358 - val_accuracy: 0.9895
Epoch 21/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0793 - accuracy: 0.9743 - val_loss: 0.0313 - val_accuracy: 0.9907
Epoch 22/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0810 - accuracy: 0.9736 - val_loss: 0.0330 - val_accuracy: 0.9898
Epoch 23/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0761 - accuracy: 0.9749 - val_loss: 0.0343 - val_accuracy: 0.9903
Epoch 24/25
422/422 [==============================] - 3s 7ms/step - loss: 0.0764 - accuracy: 0.9749 - val_loss: 0.0303 - val_accuracy: 0.9905
Epoch 25/25
422/422 [==============================] - 3s 8ms/step - loss: 0.0739 - accuracy: 0.9760 - val_loss: 0.0310 - val_accuracy: 0.9915
<keras.src.callbacks.History at 0x7fdb787b2e60>
```

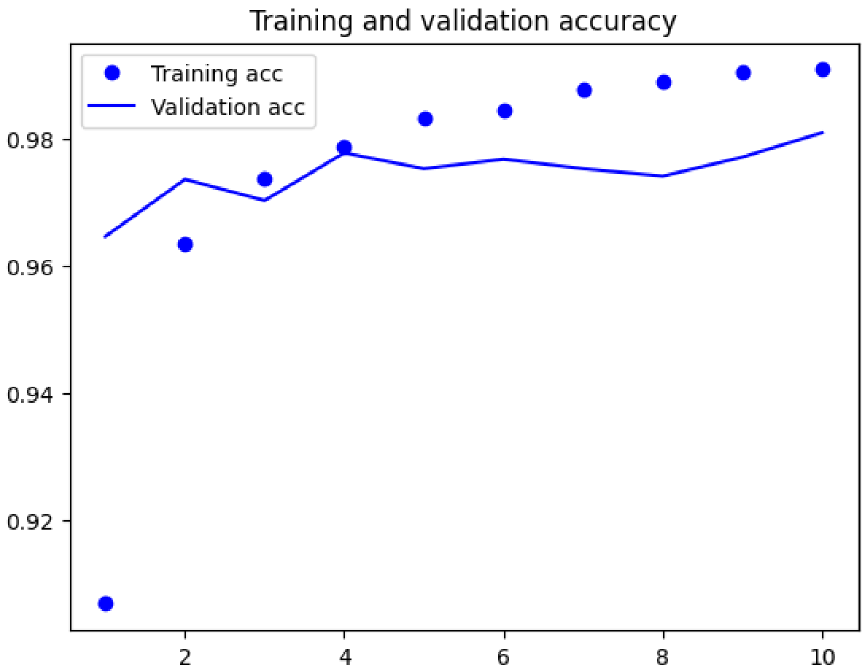
Evaluate model:

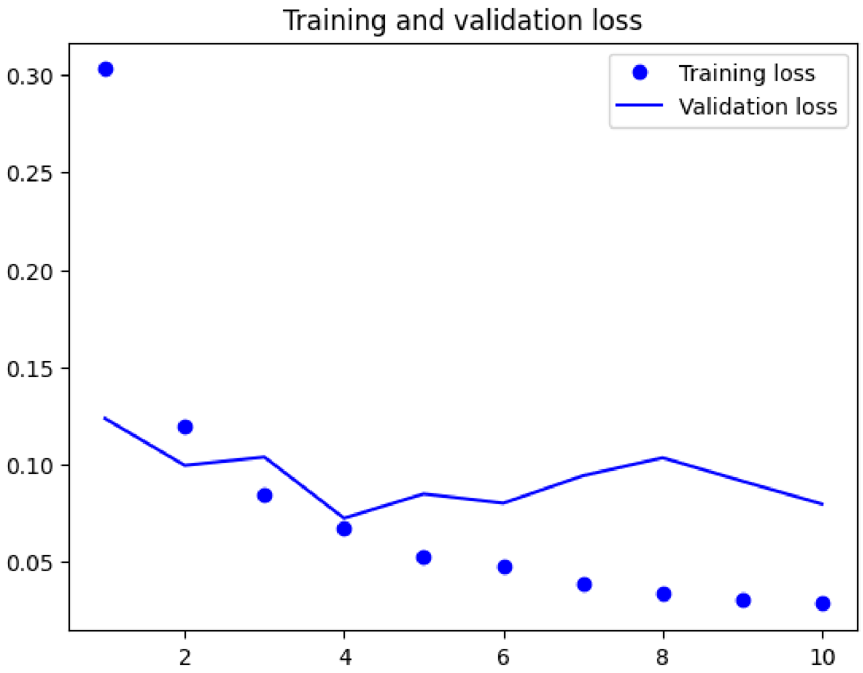
```
test_score = model.evaluate(x_test, y_test, verbose=0)
train_score = model.evaluate(x_train, y_train, verbose=0)

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

```
Train accuracy:  0.9929259419441223  Test accuracy:  0.9897000193595886
```

Wykresy precyzji i błędu

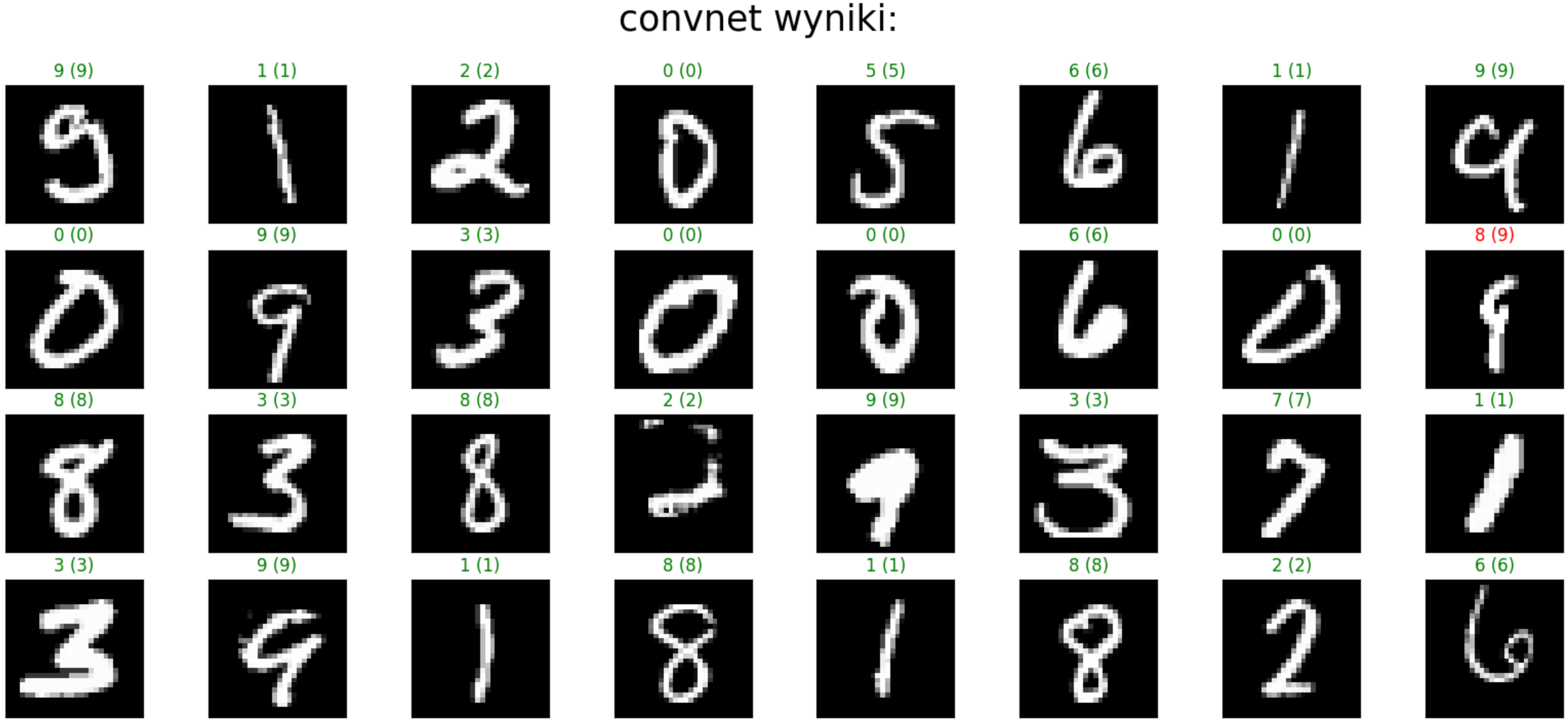
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation 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
0.98
0.96
0.94
0.92
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.30
0.25
0.20
0.15
0.10
0.05
2
4
6
8
10

```
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_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 64)        320       
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 14, 14, 64)        0         
 g2D)                                                            
                                                                 
 dropout_6 (Dropout)         (None, 14, 14, 64)        0         
                                                                 
 conv2d_9 (Conv2D)           (None, 14, 14, 32)        8224      
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 7, 7, 32)          0         
 g2D)                                                            
```