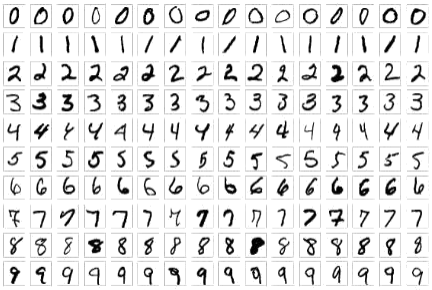


# Rozpoznawanie pisma ręcznego

Sem. I/2019 – Mateusz Broja

**Cel projektu:** Rozpoznanie pisma ręcznego z obrazków

**Dane:** pozyskane jako zbiór wcześniej przygotowanego zestawu 70 tysięcy znaków pisanych ręcznie (60 tys. zbiór treningowy oraz 10 tys. zbiór testowy). Znaki zostały ujednolicone do stałego rozmiaru obrazka oraz scentrowane.

**Model:** Do osiągniecia celu zostanie użyta sieć neuronowa, która badając 60 tysięcy znaków będzie miała za zadania odpowiednie rozpoznanie następnych, przedstawionych w zestawie testowym. Jako narzędzie zostanie użyty open-source'owy program Python.

**Miary ewaluacji:** Do ewaluacji zostanie użyte 10 tys. obrazków z grupy testowej, dodatkowo dane te pochodzą z grupy 250 osób, niebędących w grupie treningowej. W praktyce zostanie zastosowana funkcja *model.evalute* z jednego z pakietów programu Python. Wynik będzie procentową zgodnością ewaluacji modelu, tj. ile % model zgadł odpowiednich znaków. Przyjęty próg zadowalający to 95%.

**Cel:** Rozpoznanie pisma ręcznego

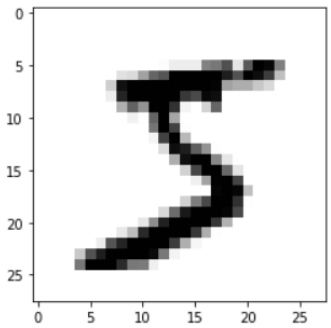
**Dane**: źródłem danych jest wbudowany w bibliotekę Keras zestaw danych MNIST, wywołany przy użyciu kodu:

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

**Wstępna ocena danych:** Wizualizacja danych, dzięki pakietowi matplotlib, pokazuje jak dane się prezentują:

```
first_number = train_images[0]
plt.imshow(first_number, cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x1e361b7bfc8>
```




Dane zostały już wcześniej odpowiednie scentrowane oraz podzielone na testowe oraz treningowe. Posiadają one przypisane etykiety z wartościami. Posiadają one kształt:

```
(60000, 28, 28)
(10000, 28, 28)
```

Dane są od lat obiektem zainteresowania analityków, tym samym nie wymagają większej obróbki same w sobie. Na potrzeby jednak przepuszczenia danych przez sieć neuronową zostaną lekko zmodyfikowane.

**Przygotowanie danych:** Ażeby mogły zostać użyte do trenowania sieci neuronowej, zostaną one transformowane do postaci dwuwymiarowej oraz typu float32, poprzez komendy `reshape(60000, 28 * 28)` dla treningowych oraz `astype`.

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Dodatkowo zmienne train_labels oraz test_labels muszą zostać ukazane w formie wektora wypełnionego samymi zerami z liczbą 1 umieszczoną w miejscu indeksu etykiety. Jest to tzw. one hot-encoding. Tam, gdzie znajduje się jedynka, taka klasa zostanie nadana. W naszym przypadku wektor będzie miał 10 wymiarów, tj. kategorii (od cyfr 0-9).

**Modelowanie:** Do rozpoznania pisma zostanie użyte głęboką sieć neuronowa z dwoma warstwami właściwymi oraz jedną czynnością **dropout**.

```
dpt_model = models.Sequential()
network_1.add(layers.Dense(512,activation='relu', input_shape=(28*28,)))
dpt_model.add(layers.Dropout(0.5))
network_1.add(layers.Dense(10,activation='softmax'))
```

Metoda aktywacji **softmax** pozwala na normalizację, dzięki czemu sprawdza się dobrze przy dylematach klasyfikacji, gdzie wartości końcowe mogą być interpretowane jako prawdopodobieństwa.

```
network_1.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

network_1.fit(train_images, train_labels, epochs=4, batch_size=128)
```

Ustalone zostały 4 epoki, ze względu na osiągnięcie zbyt dużego zjawiska przetrenowania przy większej ilości warst.

**Ewaluacja:** sprowadza się do obliczenia wartości straty. Całość dostosowania się sieci do jej optymalnej formy prezentuje się w 5 punktach:
1. Generowanie wsadu z próbek treningowych wraz z podpisami
2. Uruchomienie sieci na ww. danych
3. Obliczenie wartości straty i pomiar różnicy między predykcją, a właściwymi wartościami
4. Obliczyć gradient straty
5. Przesunięcie wartości parametrów, tj. wagi w kierunku przeciwnym do gradientu, co powinno zredukować stratę.

Dodatkowo istotne jest, ażeby znaleźć globalne ekstremum, a nie lokalne. Można w tym celu skorzystać z gradientu wraz z tzw. pędem.
W naszym modelu użyty zostanie optymalizator **RMSprop**, który dostosuje to w jakim stopniu wagi będą zmienione, dzięki bardziej dynamicznemu parametrowi **learning_rate**.

```
test_loss, test_acc = network_1.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
test_acc: 0.9787
```

Wartość "**test_acc**" dla testowego jest niższa niż treningowego, gdyż wystąpił nieznaczny problem przetrenowania, który jednak poprzez zmniejszenie ilości warstw jest na optymalnym poziomie.

**Wdrożenie (wnioski):** Wynik uzyskany przy klasyfikacji wydaje się być satysfakcjonujący, jednakże sama problematyka, jak na dzisiejsze czasy, niewymagająca. Początkowe założenia zostały spełnione (>95% skuteczności), jednakże wynik można by poprawić przy użyciu konwolucyjnej sieci neuronowej, lub poprzez zwiększenie liczby warstw.