Sztuczna Inteligencja

Sieci neuronowe

Rafał Nowak

Instytut Informatyki UWr

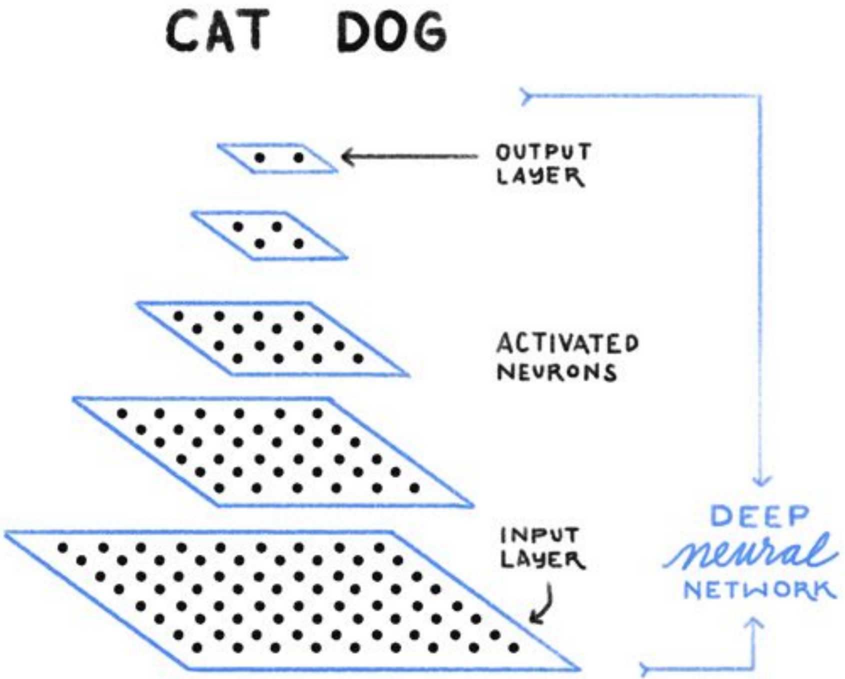
25.04.2018

Sieci neuronowe

Krótki przegląd

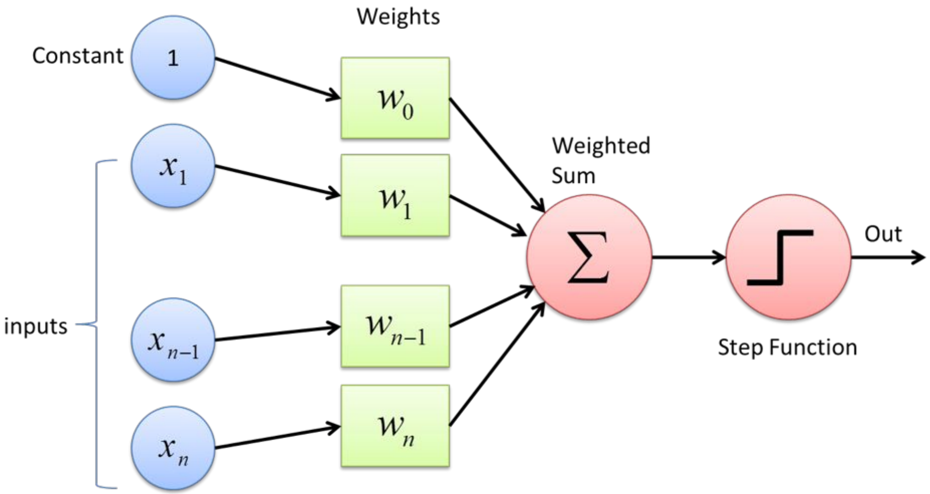
- klasyfikacja
- perceptron
- perceptron wielowarstwowy
- funkcje aktywacji
- backpropagation
- klasyfikacja

Klasyfikacja



Perceptron

- wejście - np. cechy jakiegoś obiektu
- wyjście - liczba rzeczywista
- zastosowanie
  - klasyfikacja
  - uczenie z nadzorem
- demo z użyciem biblioteki sklearn



Model

```
[ ] from sklearn.linear_model import Perceptron
    model = Perceptron(max_iter=1000, verbose=False)
```

Training

```
[ ] model.fit(X_train, y_train)
```

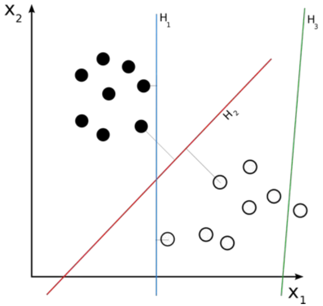
```
Perceptron(alpha=0.0001, class_weight=None, eta0=1.0, fit_intercept=True,
      max_iter=1000, n_iter=None, n_jobs=1, penalty=None, random_state=0,
      shuffle=True, tol=None, verbose=False, warm_start=False)
```

```
[ ] model.score(X_train, y_train)
```

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$\boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}$$

$$\boldsymbol{x} \in \mathbb{R}^n$$

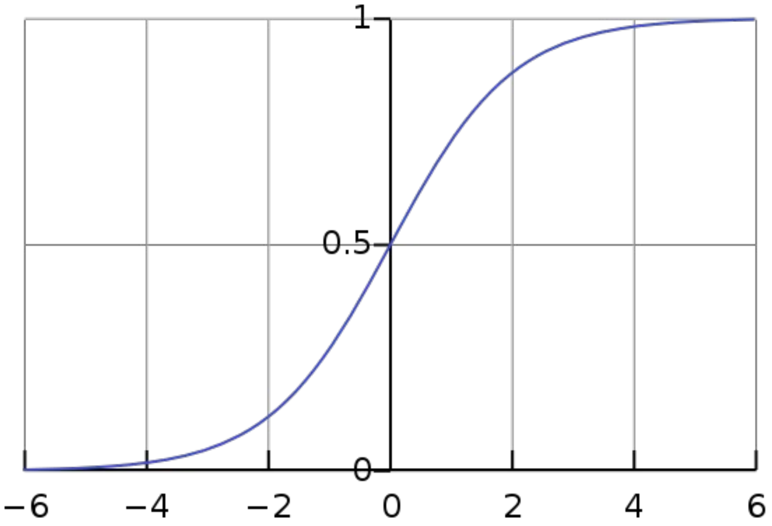


Funkcja sigmoid

- $\sigma(x) = \dfrac{1}{1+e^{-x}}$

- interpretacja probabilistyczna
- łatwo obliczyć pochodną

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$



Regresja logistyczna

$$f(\boldsymbol{x};\ \boldsymbol{w}, b) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$$

$$p(y = 1 \mid \boldsymbol{x}, \boldsymbol{w}, b) = f(\boldsymbol{x};\ \boldsymbol{w}, b)$$

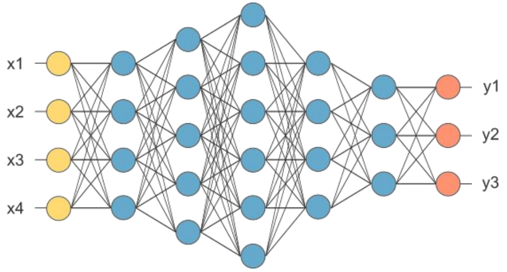
- klasyfikator binarny
- perceptron + sigmoid
- uczenie = estymacja maksymalnej wiarygodności
- funkcja straty - entropia krzyżowa (lepsza od błędu średniokwadratowego)

$$\mathcal{L}(\hat{y}, y) = -\Big[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\Big]$$

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{w}, b), \quad y \in \{0, 1\}$$

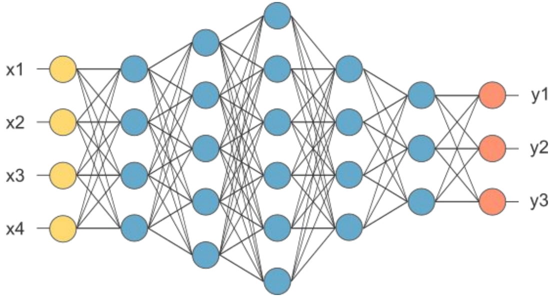
- demo

Wielowarstwowy perceptron (MLP)



- *m* - liczba danych wejściowych (*n*-wymiarowych) $X \in \mathbb{R}^{m \times n}$
- L - liczba warstw ukrytych
- $h_1, h_2, \ldots, h_L$ - liczba neuronów w kolejnych warstwach
- wagi warstw ukrytych $W_i \in \mathbb{R}^{h_{i-1} \times h_i} \quad (i = 1, 2, \ldots, L;\ h_0 = n)$
- wagi warstwy wyjściowej $W_{L+1} \in \mathbb{R}^{h_L \times K}$
- obciążenia $\boldsymbol{b}_i$

Wielowarstwowy perceptron (MLP)



- propagacja w przód $\quad (i = 1, 2, \ldots, L+1)$

$$Z_i := A_{i-1} W_i + \boldsymbol{b}_i$$
$$A_i := g_i(Z_i) \qquad\qquad A_0 := X \in \mathbb{R}^{m \times n}$$

- $g_1, g_2, \ldots, g_{L+1}$ - funkcje aktywacji
- *m* - liczba danych wejściowych (*n*-wymiarowych)
- wartości jednostek (po aktywacji) $\quad A_i \in \mathbb{R}^{m \times h_i}, \quad i = 1, 2, \ldots, L+1$
- na wyjściu otrzymujemy *K* wartości dla każdej obserwacji

$$\hat{Y} := A_{L+1} \in \mathbb{R}^{m \times K}$$

Funkcje aktywacji

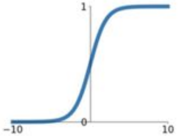
$$[\text{softmax}(\boldsymbol{z})]_j = \frac{\exp(z_j)}{\sum_i \exp(z_i)}$$

- w warstwie wyjściowej najczęściej stosuje się funkcję aktywacji *sigmoid* (K<2) albo [softmax]() (K≥2)
- w warstwach ukrytych można stosować następujące funkcje aktywacji

$$\begin{bmatrix}1.2\\0.9\\0.4\end{bmatrix} \xrightarrow{} \boxed{\text{Softmax}} \xrightarrow{} \begin{bmatrix}0.46\\0.34\\0.20\end{bmatrix}$$

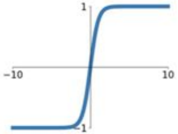
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



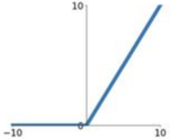
tanh

$\tanh(x)$



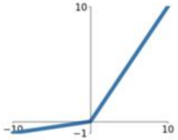
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

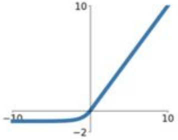


Maxout

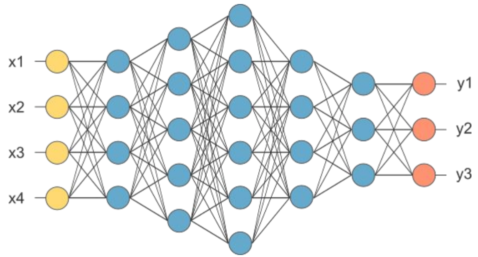
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Funkcja kosztu



- $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ - wyjście sieci neuronowej, $\boldsymbol{y} \in \{0,1\}^K$ - binarnie zakodowana przynależność do klasy
- entropia krzyżowa

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k = -\log(\hat{y}_r) \qquad ([\boldsymbol{y}]_r = 1)$$

- funkcja straty

$$J(\boldsymbol{\theta}) := \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$$

Trenowanie

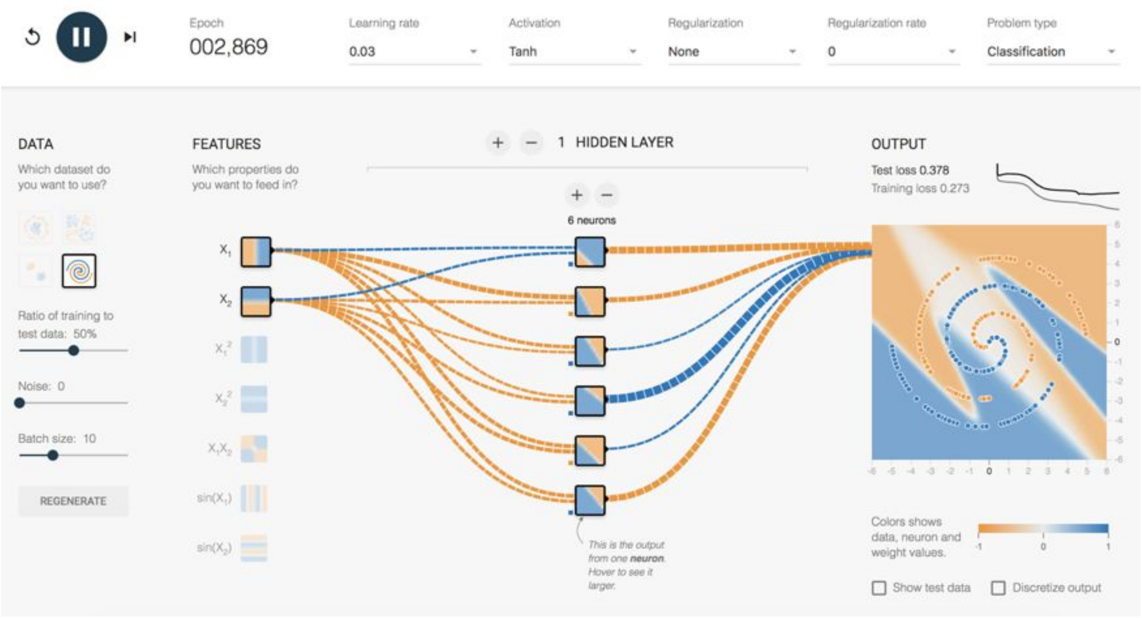
$$J(\boldsymbol{\theta}) := \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$$

- optymalizacja funkcji $\arg\min_{\theta} J(\theta)$
- wiele metod numerycznych
  - metoda spadku gradientu (ang. *gradient descent*)

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k)$$

  - metoda gradientu stochastycznego (SGD) - losowy podział danych treningowych na mini-paczki
  - istnieje wiele lepszych (szybciej zbieżnych) algorytmów:
    - SGD+momentum, Adagrad, AdaDelta, Adam

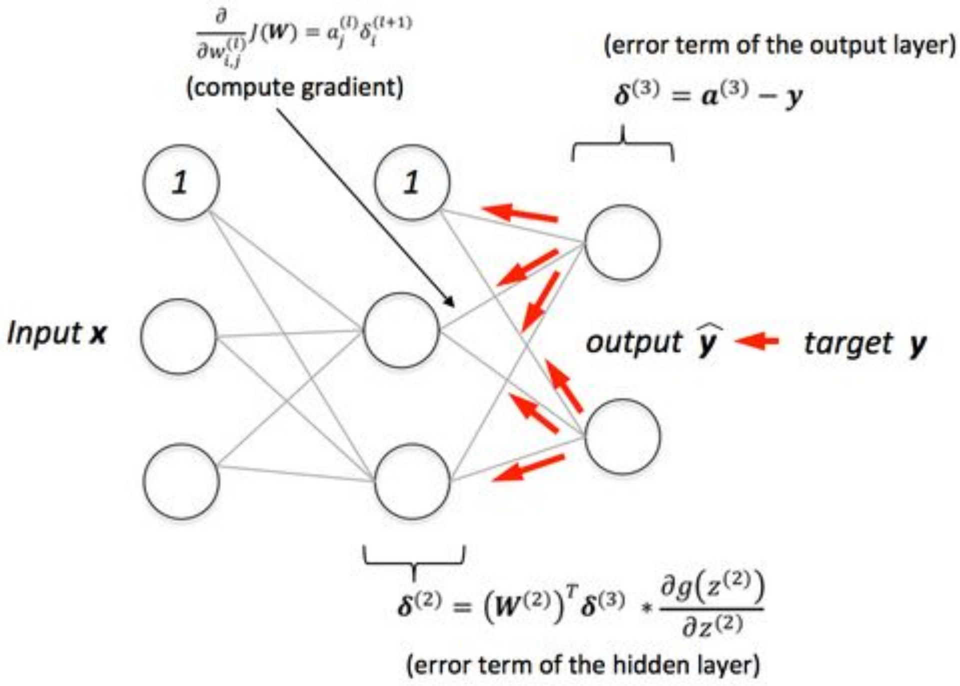
Demo - playground.tensorflow.org



Propagacja wsteczna

- efektywne obliczanie gradientu funkcji kosztu
- wykorzystanie reguły łańcuchowej

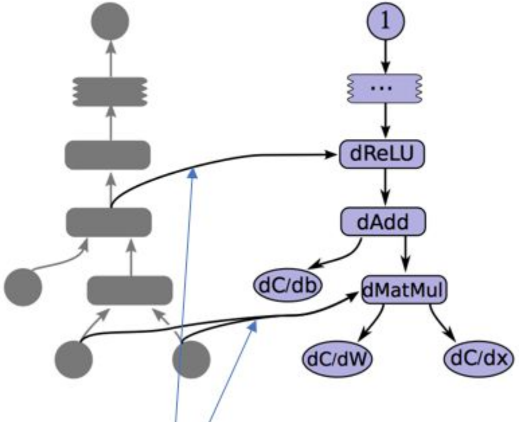
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



https://sebastianraschka.com/faq/docs/visual-backpropagation.html

Implementacja MLP z użyciem TensorFlow™

Krok 1.
Graf obliczeń




```python
tf_X = tf.placeholder('float32', shape=(None,n_features))
tf_Y = tf.placeholder('float32', shape=(None,n_classes))


tf_W1 = tf.Variable( W1.astype('float32') )
tf_b1 = tf.Variable( np.float32( b1 ) )

tf_W2 = tf.Variable( W2.astype('float32') )
tf_b2 = tf.Variable( np.float32( b2 ) )

tf_W3 = tf.Variable( W3.astype('float32') )
tf_b3 = tf.Variable( np.float32( b3 ) )

tf_Z1 = tf.matmul( tf_X, tf_W1 ) + tf_b1
tf_A1 = tf.tanh( tf_Z1 )
tf_Z2 = tf.matmul( tf_A1, tf_W2 ) + tf_b2
tf_A2 = tf.tanh( tf_Z2 )
tf_Z3 = tf.matmul( tf_A2, tf_W3 ) + tf_b3      # logits
tf_A3 = tf.nn.softmax( tf_Z3 )

tf_loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits_v2( \
                                           labels=tf_Y, logits=tf_Z3 ) )

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    print(sess.run(tf_loss, feed_dict={tf_X: X, tf_Y: y_one_hot}))
```

Implementacja MLP z użyciem TensorFlow™

gradient oblicza się automatycznie

Krok 2. Trenowanie

```
# construct an optimizer
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(tf_loss)
# input parameter is the learning rate

logits = tf_Z3
predict_op = tf.argmax(logits, axis=1)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for i in range(1000):
    _, cost, pred = sess.run([train_op, tf_loss, predict_op], \
                             feed_dict={tf_X: X, tf_Y: y_one_hot})

    if i%100==0:
      print("Loss after iteration %i: %7.5f (score=%.2f%%)" \
          % (i, cost, np.mean( pred==y )*100.0) )

sess.close()
```

```
Loss after iteration 0: 2.98696 (score=10.13%)
Loss after iteration 100: 2.26833 (score=14.30%)
Loss after iteration 200: 2.22988 (score=17.86%)
Loss after iteration 300: 2.17295 (score=20.20%)
Loss after iteration 400: 2.09463 (score=22.76%)
Loss after iteration 500: 2.00825 (score=28.16%)
Loss after iteration 600: 1.92877 (score=28.94%)
Loss after iteration 700: 1.86114 (score=30.50%)
Loss after iteration 800: 1.80412 (score=32.89%)
Loss after iteration 900: 1.75529 (score=35.11%)
```

Tensorflow - ciekawe materiały

- [cs224n-2017-tensorflow.pdf]
- [https://www.tensorflow.org/tutorials/]
- [https://becominghuman.ai/an-introduction-to-tensorflow-f4f31e3ea1c0]
- [tf.train]

Classes

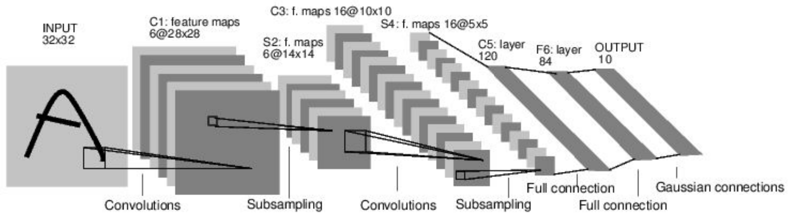
`class AdadeltaOptimizer` : Optimizer that implements the Adadelta algorithm.

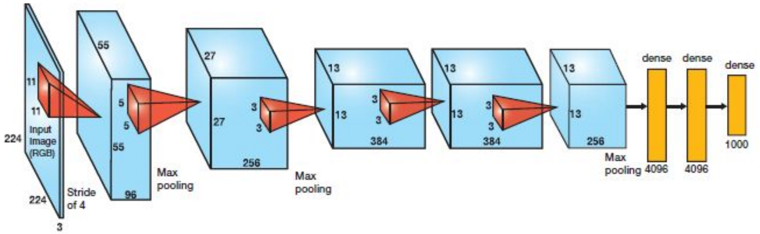
`class AdagradDAOptimizer` : Adagrad Dual Averaging algorithm for sparse linear models.

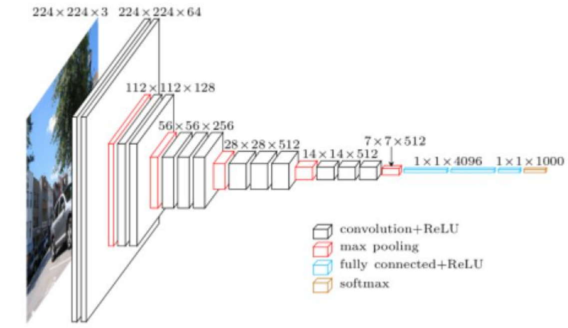
`class AdagradOptimizer` : Optimizer that implements the Adagrad algorithm.

`class AdamOptimizer` : Optimizer that implements the Adam algorithm.

Splotowe sieci neuronowe (CNN)

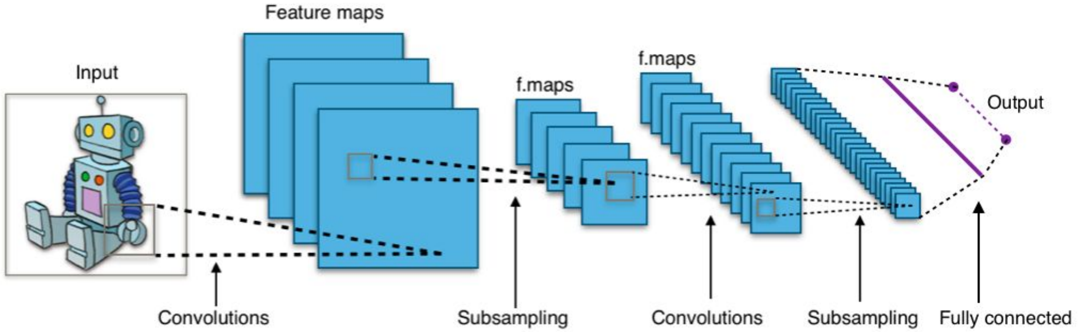






Warstwa splotowa

1. Etap splotu (ang. convolution)
2. Etap wykrywania (ang. activation)
3. Etap redukcji (ang. pooling / downsampling)



Splot

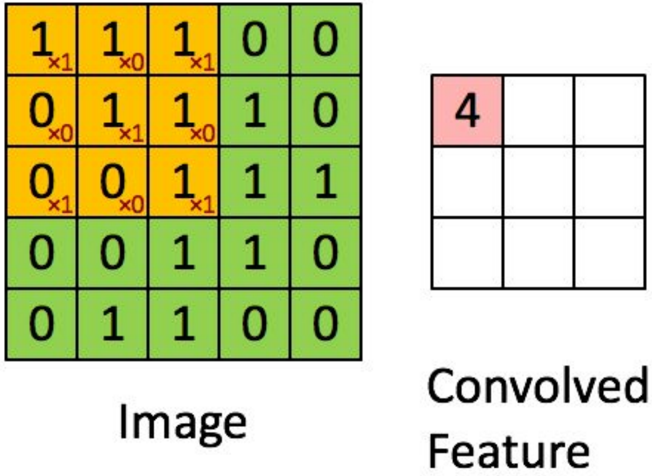
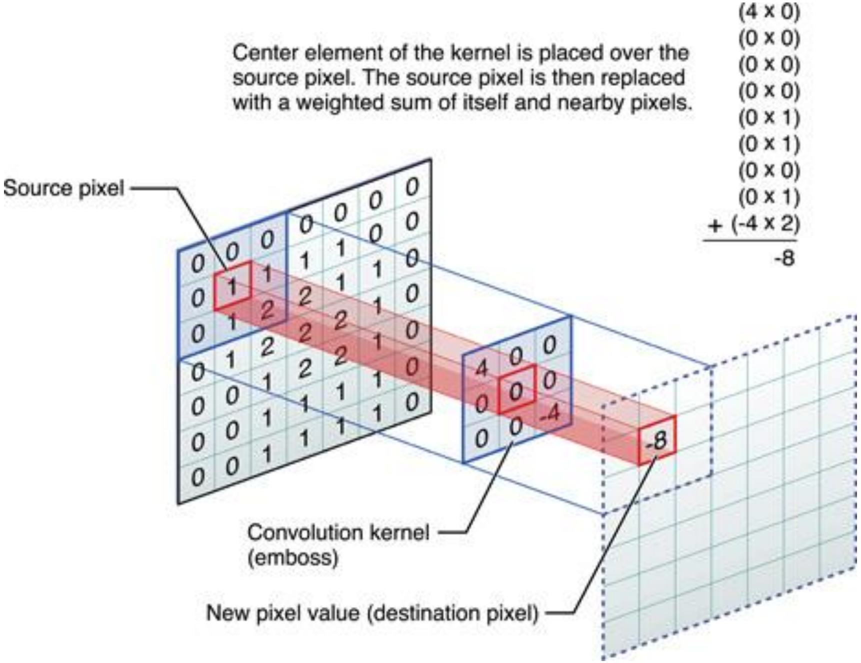
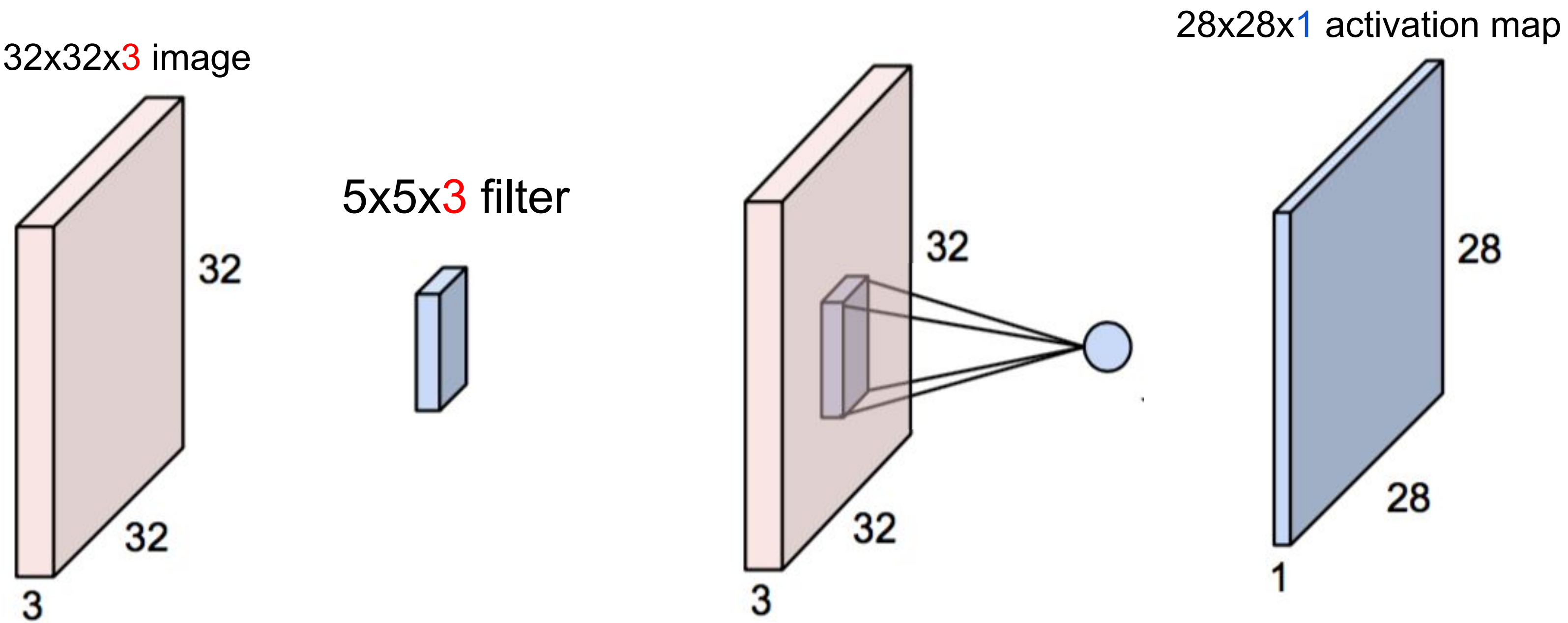


Image
Convolved Feature



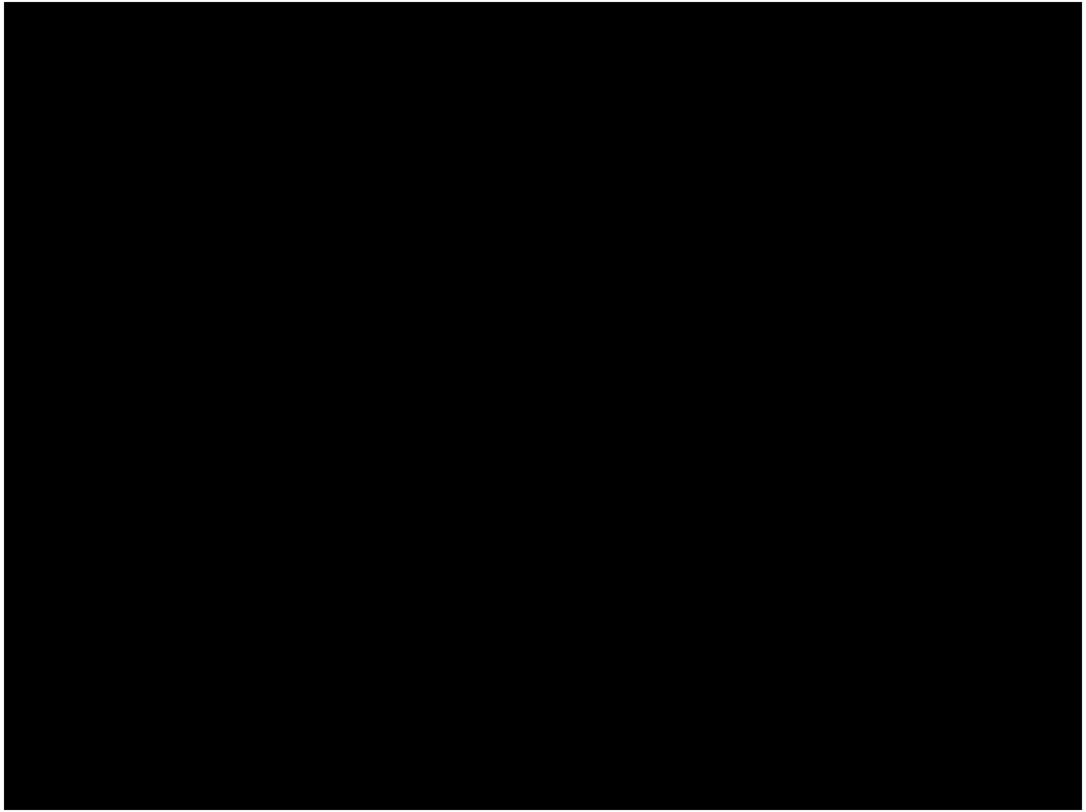
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Etap splotu



32x32x3 image
32
32
3
5x5x3 filter
32
32
3
28x28x1 activation map
28
28
1

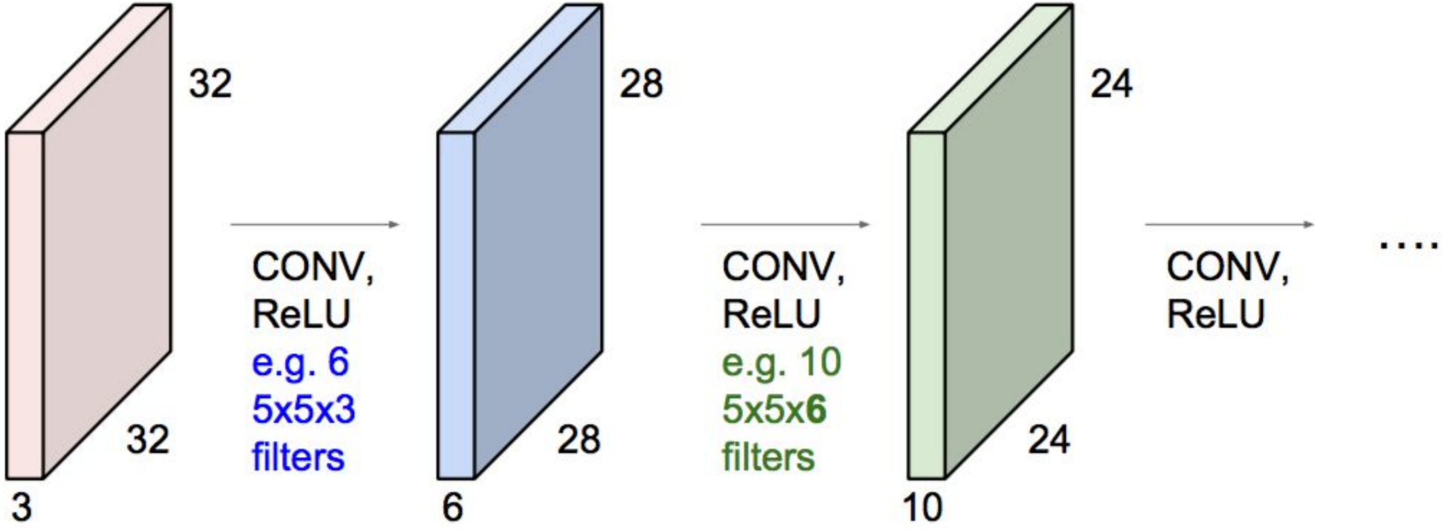
Etap splotu



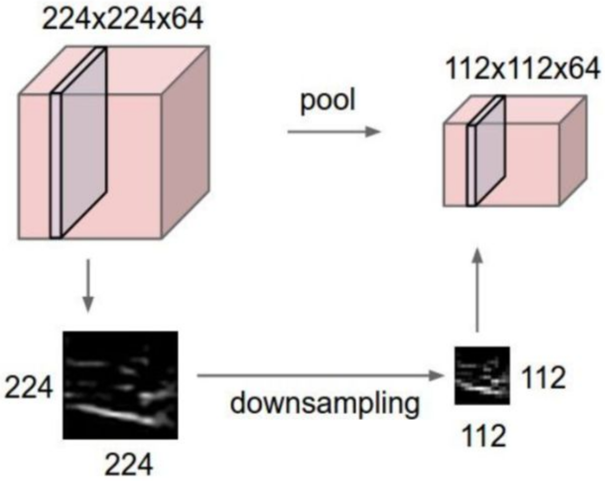
źródło: [Deep Learning, CNN, Coursera Course]

Etap splotu (więcej filtrów)

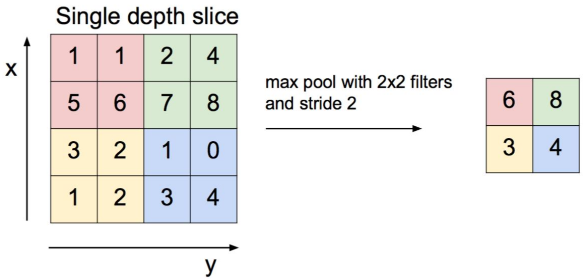
Etap wykrywania



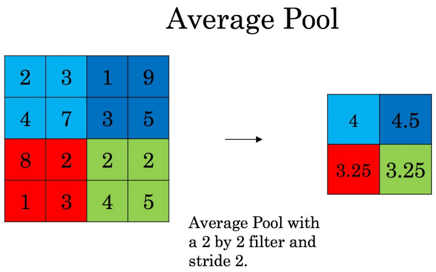
Etap redukcji



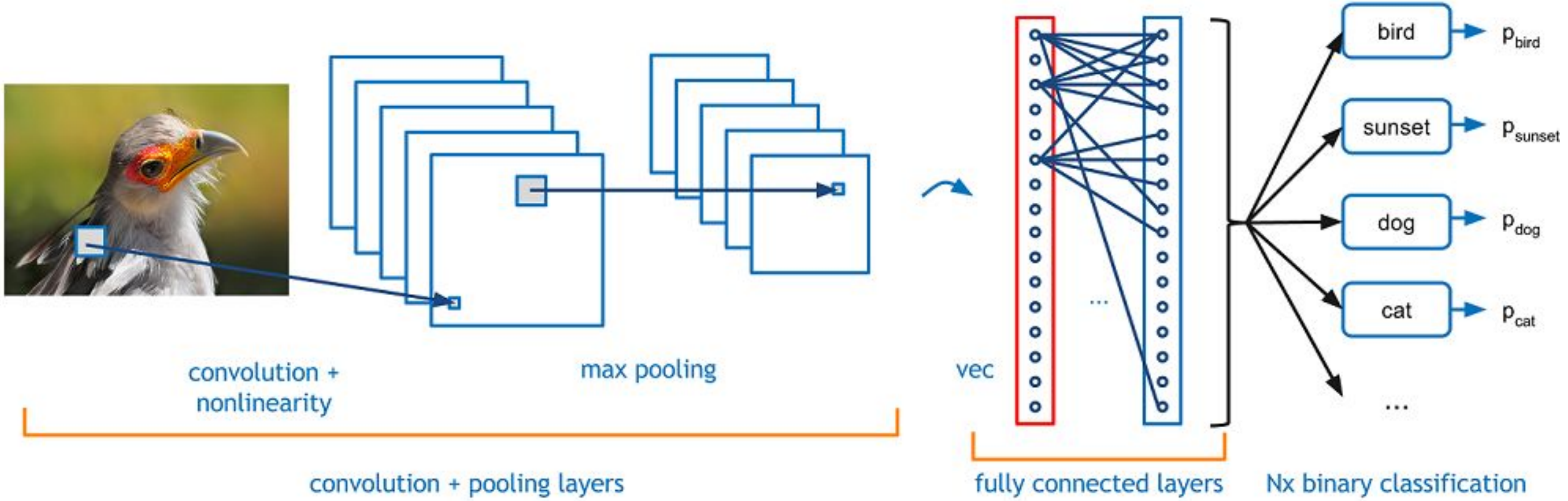
- MAX pooling



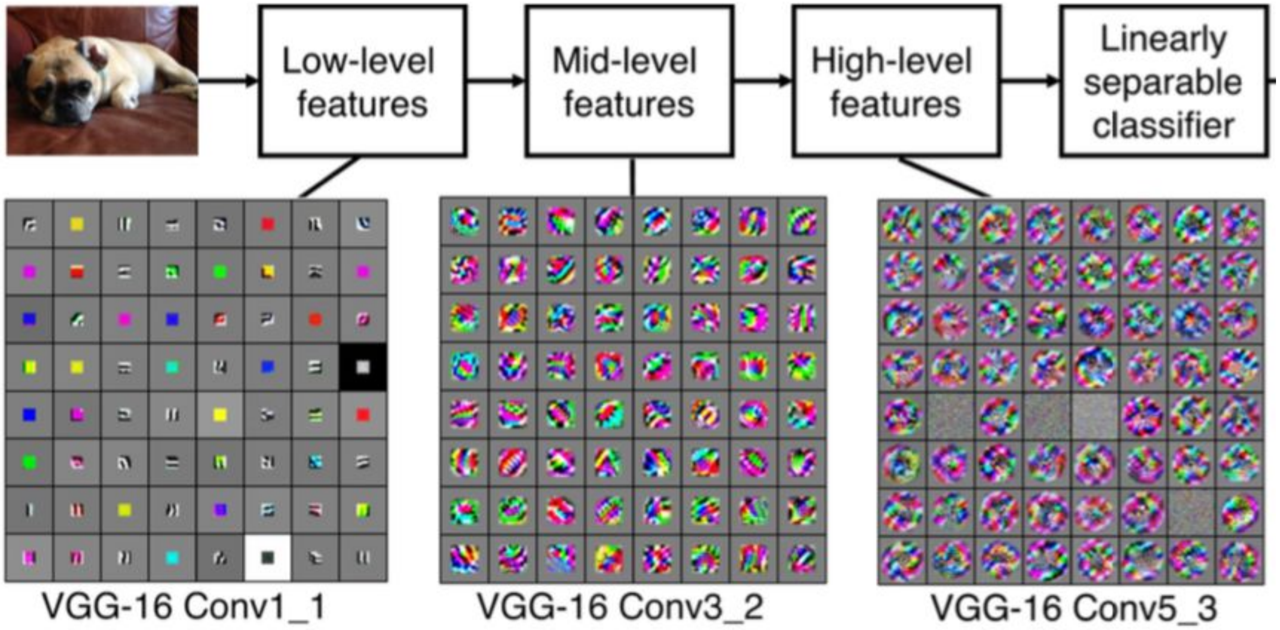
- AVG pooling



Splotowa sieć neuronowa



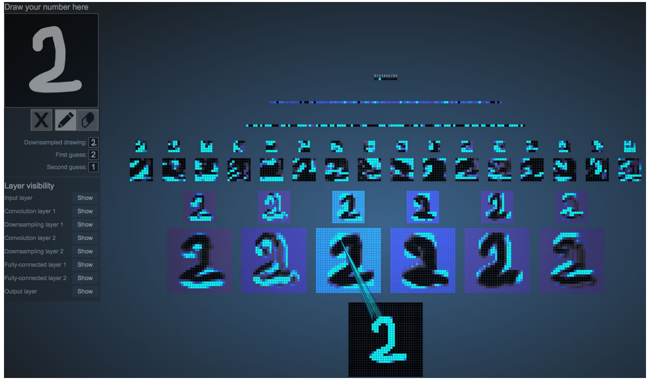
Wizualizacja warstw splotowych



Demo

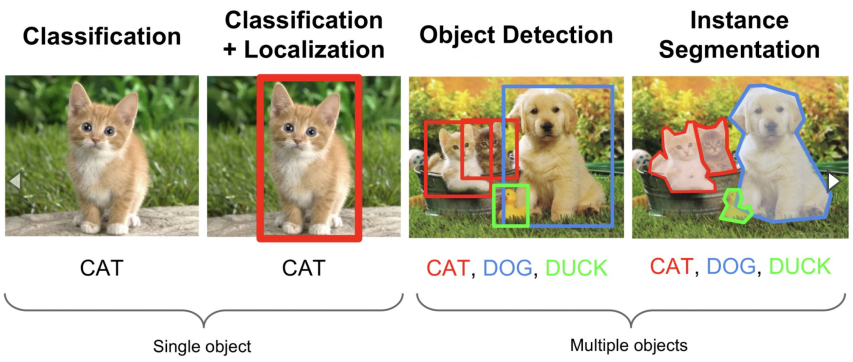
- https://transcranial.github.io/keras-js/#/mnist-cnn

- http://www.scs.ryerson.ca/~aharley/vis/conv/flat.html

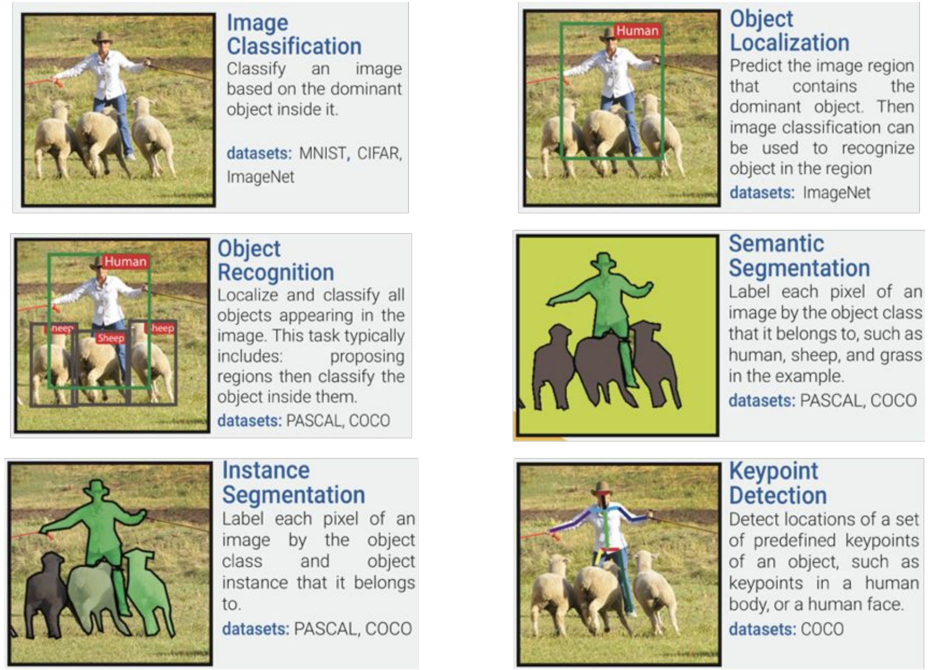


Zastosowania

Kilka zastosowań w wizji komputerowej i przetwarzaniu języka

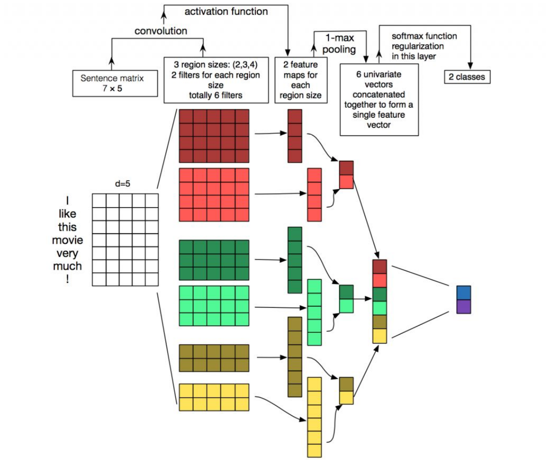


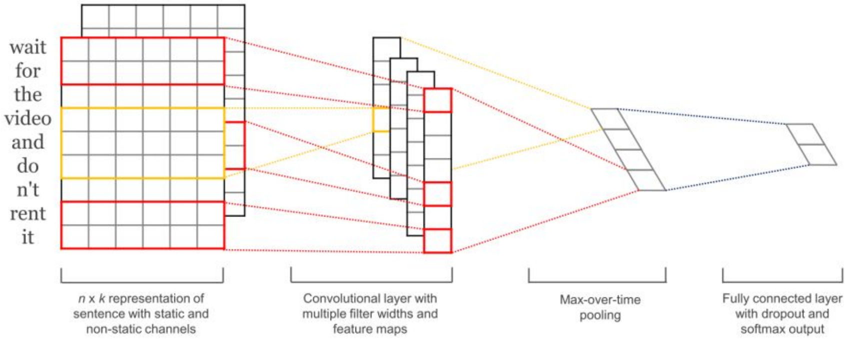
Zastosowania w wizji komputerowej



Zastosowania w przetwarzaniu tekstu

- rozpoznawanie mowy (speech recognition)
- klasyfikacja tekstu, np. analiza sentymentu, porównywanie podobieństw pomiędzy tekstami

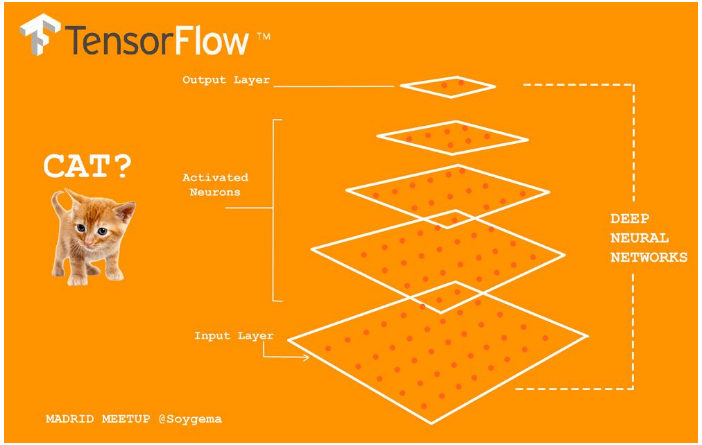




http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Implementacja z użyciem keras

> pip install keras



keras - MLP - klasyfikacja binarna

```
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

https://keras.io/getting-started/sequential-model-guide/

keras - MLP - wiele klas

```
# For a single-input model with 10 classes (categorical classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))

# Convert labels to categorical one-hot encoding
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

keras - CNN - wiele klas

```
from keras.layers import Input, Conv2D, MaxPooling2D, Activation
```

```
model = Sequential()

model.add(Conv2D(32, (5, 5), input_shape=(h,w,1),\                   # - input +
                 padding='same', strides=(1,1)))                     #   convolution
model.add(Activation("relu"))                                        # - activation
model.add(MaxPooling2D(pool_size=(2, 2)))                            # - pooling

model.add(Conv2D(16, (3, 3), padding='same', strides=(1,1)))         # - convolution
model.add(Activation("relu"))                                        # - activation
model.add(MaxPooling2D(pool_size=(2, 2)))                            # - pooling

model.add(Flatten())                                                 # flatten
model.add(Dense(128, activation='sigmoid'))                          # fully connected

model.add(Dense(n_classes, activation='softmax'))                    # output layer

print( model.summary() )

model.compile( loss='categorical_crossentropy', \
               optimizer="adam", \
               metrics=['categorical_accuracy'] )

model.fit(X_train, y_train, epochs=100, batch_size=32)
```