Głębokie Sieci Neuronowe

```
from keras.api.layers import Dense, Input, BatchNormalization
from keras.api.layers import Dropout, GaussianNoise
from keras.api.layers import LayerNormalization
from keras.api.models import Sequential
from keras.api.optimizers import Adam, SGD
from keras.api.regularizers import l2, l1
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
def plot_model_metrics(model, epochs=1000):
    historia = model.history.history
    floss_train = historia['loss']
    floss_test = historia['val_loss']
    acc_train = historia['accuracy']
    acc_test = historia['val_accuracy']
    fig,ax = plt.subplots(1,2, figsize=(20,10))
    epo = np.arange(0, epochs)
    ax[0].plot(epo, floss_train, label = 'floss_train')
    ax[0].plot(epo, floss_test, label = 'floss_test')
    ax[0].set_title('Funkcje strat')
    ax[0].legend()
    ax[1].set_title('Dokladnosci')
    ax[1].plot(epo, acc_train, label = 'acc_train')
    ax[1].plot(epo, acc_test, label = 'acc_test')
    ax[1].legend()
```

```
def plot_model_metrics(model, epochs=1000):
    historia = model.history.history
    floss_train = historia['loss']
    floss_test = historia['val_loss']
    acc_train = historia['accuracy']
    acc_test = historia['val_accuracy']
    fig,ax = plt.subplots(1,2, figsize=(20,10))
    epo = np.arange(0, epochs)
    ax[0].plot(epo, floss_train, label = 'floss_train')
    ax[0].plot(epo, floss_test, label = 'floss_test')
    ax[0].set_title('Funkcje strat')
    ax[0].legend()
    ax[1].set_title('Dokladnosci')
    ax[1].plot(epo, acc_train, label = 'acc_train')
    ax[1].plot(epo, acc_test, label = 'acc_test')
    ax[1].legend()
```

```
model = Sequential()
model.add( Dense(128, activation='relu',
            input_shape = (10,),
            kernel_regularizer = l2(0.01)) )
```

```
neuron_num  = 64
do_rate = 0.5
noise = 0.1
learning_rate = 0.001
```

```
block = [   Dense,
            LayerNormalization,
            BatchNormalization,
            Dropout,
            GaussianNoise ]

args = [ (neuron_num,'selu'),(),(),(do_rate,),(noise,)]
model = Sequential()
model.add( Dense(neuron_num, activation='relu', input_shape = (10,)) )
repeat_num = 2

for i in range(repeat_num):
    for layer,arg in zip(block, args):
        model.add(layer(*arg))

model.add( Dense(1, activation='sigmoid') )
model.compile( optimizer= Adam(learning_rate), loss='binary_crossentropy',
               metrics=('accuracy', 'Recall', 'Precision') )
```

Rozpoznawanie Obrazów, Sieci Konwolucyjne

```
from keras.api.layers import Conv2D, Flatten, Dense, AveragePooling2D,
     MaxPooling2D
from keras.api.models import Sequential
from keras.api.optimizers import Adam
from keras.api.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

def plot_model_metrics(model, epochs=1000):
      historia = model.history.history
      floss_train = historia['loss']
      floss_test = historia['val_loss']
      acc_train = historia['accuracy']
      acc_test = historia['val_accuracy']

      fig,ax = plt.subplots(1,2, figsize=(20,10))
      epo = np.arange(0, epochs)
      ax[0].plot(epo, floss_train, label = 'floss_train')
      ax[0].plot(epo, floss_test, label = 'floss_test')
      ax[0].set_title('Funkcje strat')
      ax[0].legend()
      ax[1].set_title('Dokladnosci')
      ax[1].plot(epo, acc_train, label = 'acc_train')
      ax[1].plot(epo, acc_test, label = 'acc_test')
      ax[1].legend()
      plt.show()

def cm_for_nn(model, X_test, y_test):
    # y_pred jest 10 wymiarowym wektorem, będącym rozkładem
    # prawdopodobieństwa (softmax w ostatniej warstwie)
    y_pred = model.predict(X_test)
    # Znajdź w każdym wierszu macierzy y_pred indeks elementu, który
    # zawiera największą wartość i zwróć numer indeksu.
    y_pred_classes = np.argmax(y_pred, axis=1)
```

```
    cm = confusion_matrix(y_test, y_pred_classes)
    sns.heatmap(cm, annot=True, fmt='d', cmap='viridis')
    plt.xlabel('Wartosci przewidziane')
    plt.ylabel('Wartości rzeczywiste')
    plt.title('Confusion Matrix')
    plt.show()

train, test = mnist.load_data()
X_train, y_train = train[0], train[1]
X_test, y_test = test[0], test[1]
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)
class_cnt = np.unique(y_train).shape[0]
filter_cnt = 32
neuron_cnt = 32
learning_rate = 0.0001
act_func = 'relu'
kernel_size = (3,3)
pooling_size = (2,2)
model = Sequential()
conv_rule = 'same'

model.add( Conv2D(input_shape = X_train.shape[1:],
            filters=filter_cnt,
            kernel_size = kernel_size,
            padding = conv_rule,
            activation = act_func) )

model.add(MaxPooling2D(pooling_size))
model.add(Flatten())
model.add(Dense(class_cnt, activation='softmax'))

model.compile( optimizer=Adam(learning_rate),
               loss='SparseCategoricalCrossentropy',
               metrics=['accuracy'] )

model.fit( x = X_train, y = y_train, epochs = class_cnt ,
            validation_data=(X_test, y_test) )

plot_model_metrics(model, 10)
cm_for_nn(model, X_test, y_test)
```

Rozszerzenie Zbioru Uczącego (Augmentacja)

```
from keras.api.datasets import fashion_mnist
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train.shape
y_train.shape

plt.imshow(x_train[0], cmap='gray')
plt.imshow(x_train[1], cmap='gray')

#--------------------
```

```
from keras.api.models import Model
from keras.api.layers import Input, Dense, Dropout, Reshape,
    BatchNormalization, Lambda
from keras.api.optimizers import Adam
from keras.api.datasets import fashion_mnist
import pandas as pd
import numpy as np

data = fashion_mnist.load_data()
X_train, y_train = data[0][0], data[0][1]
X_test, y_test = data[1][0], data[1][1]
X_train = np.expand_dims(X_train, axis = -1)
X_test = np.expand_dims(X_test, axis = -1)
y_train = pd.get_dummies(pd.Categorical(y_train)).values
y_test = pd.get_dummies(pd.Categorical(y_test)).values

act_func = 'selu'
hidden_dims = 64
encoder_layers = [ Reshape((28*28,)),
                   BatchNormalization(),
                   Dense(512,activation=act_func),
                   Dense(128,activation=act_func),
                   Dense(64, activation=act_func),
                   Dense(hidden_dims, activation=act_func) ]

tensor = encoder_input = Input(shape = (28,28))

for layer in encoder_layers:
    tensor = layer(tensor)

encoder_output = tensor
encoder = Model(inputs=encoder_input, outputs=encoder_output)

decoder_layers = [ Dense(128,activation=act_func),
                   Dense(512,activation=act_func),
                   Dense(784,activation='sigmoid'),
                   Reshape((28,28)),
                   Lambda(lambda x: x*255) ]

decoder_input = Input(shape=(hidden_dims,))
tensor = decoder_input

for layer in decoder_layers:
    tensor = layer(tensor)

decoder_output = tensor
decoder = Model(inputs = decoder_input, outputs = decoder_output)

aec_output = decoder(encoder(encoder_input))
gen_autoencoder = Model(inputs = encoder_input, outputs = aec_output)
gen_autoencoder.compile(optimizer =  Adam(0.01), loss = 'MeanSquaredError')
#Adam(x), x - learning rate

gen_autoencoder.fit(x=X_train,y=X_train, validation_data=(X_test, X_test),
                    batch_size=256, epochs=10)

#--------------------

from keras.api.datasets import fashion_mnist
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train.shape
y_train.shape

plt.imshow(x_train[0], cmap='gray')
plt.imshow(x_train[1], cmap='gray')
plt.imshow(x_train[9], cmap='gray')

y_train[0:9]

#x_train = np.expand_dims(x_train, axis=-1) poszerzenie zbioru treningowego
o dodatkowy wymiar

#x_test = np.expand_dims(x_test, axis=-1)

x_train.shape

k = pd.Categorical(y_train)

p = pd.get_dummies(k).values

y_train = p.astype(int)
y_train.shape

#plt.imshow(x_train[0,:,:], cmap='gray')
#plt.imshow(x_train[0,:,5:15], cmap='gray')
#plt.imshow(x_train[0,1:10,:], cmap='gray')
#plt.imshow(x_train[0,2:20:2,:], cmap='gray')
#plt.imshow(x_train[0,::-1,:], cmap='gray')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

obr = x_train[0,:,:]
obr = np.expand_dims(obr, axis=-1)
#obr.shape
data_gen = ImageDataGenerator(
    rotation_range=30,        # Obrót do 30 stopni
    width_shift_range=0.2,    # Przesunięcie w poziomie
    height_shift_range=0.2,   # Przesunięcie w pionie
    shear_range=0.2,          # Ścinanie
    zoom_range=0.2,           # Zoom
    horizontal_flip=True,     # Odbicie poziome
    fill_mode='nearest'
)

img_gen = data_gen.flow(
    np.expand_dims(obr, axis = 0),
    batch_size = 1
)

obrazki = np.zeros((10, 28, 28)) #tablica 10 X 28 X 28

for i in range(10):
    img = next(img_gen)[0]
    obrazki[i] = img[:,:,0]

plt.imshow(obrazki[0], cmap='gray')
```

```
#AUGEMNTACJA

from keras.api.models import Model
from keras.api.layers import Input, Dense, Dropout, Reshape,
    BatchNormalization, Lambda
from keras.api.optimizers import Adam
from keras.api.datasets import fashion_mnist
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras import backend as K

def adding_noise(tensor):
    noise = tf.random.normal(shape=(tf.shape(tensor)), mean=0, stddev=1.5)
    return tensor + noise

def filter_data(data, iteration_num, autoencoder):
    augmented_data = data.copy()
    for i in range(iteration_num):
        augmented_data = autoencoder.predict(augmented_data)
    return augmented_data

data = fashion_mnist.load_data()
X_train, y_train = data[0][0], data[0][1]
X_test, y_test = data[1][0], data[1][1]
X_train = np.expand_dims(X_train, axis = -1)
X_test = np.expand_dims(X_test, axis = -1)
y_train = pd.get_dummies(pd.Categorical(y_train)).values
y_test = pd.get_dummies(pd.Categorical(y_test)).values

act_func = 'selu'
hidden_dims = 64
encoder_layers = [ Reshape((28*28,)),
                   BatchNormalization(),
                   Dense(512,activation=act_func),
                   Dense(128,activation=act_func),
                   Dense(64, activation=act_func),
                   Dense(hidden_dims, activation=act_func) ]

tensor = encoder_input = Input(shape = (28,28))

for layer in encoder_layers:
    tensor = layer(tensor)

encoder_output = tensor
encoder = Model(inputs=encoder_input, outputs=encoder_output)

decoder_layers = [ Dense(128,activation=act_func),
                   Dense(512,activation=act_func),
                   Dense(784,activation='sigmoid'),
                   Reshape((28,28)),
                   Lambda(lambda x: x*255) ]

decoder_input = Input(shape=(hidden_dims,))
tensor = decoder_input

for layer in decoder_layers:
    tensor = layer(tensor)
```

```
decoder_output = tensor
decoder = Model(inputs = decoder_input, outputs = decoder_output)

aec_output = decoder(encoder(encoder_input))
gen_autoencoder = Model(inputs = encoder_input, outputs = aec_output)
gen_autoencoder .compile(optimizer = Adam(0.01), loss = 'MeanSquaredError')
#Adam(x), x - learning rate

gen_autoencoder .fit(x=X_train,y=X_train, validation_data=(X_test, X_test),
    batch_size=256, epochs=1)

noised_encoder_output = Lambda(adding_noise,
    output_shape=(hidden_dims,))(encoder_output)

augmenter_output = decoder(noised_encoder_output)
augmenter = Model(inputs = encoder_input, outputs = augmenter_output)

start = 50
end = start + 10

for i in range(10):
    test_for_augm = X_train[i*10:i*10+10,...]
    augmented_data = test_for_augm.copy()
    # Iterate through each image in the batch and display it individually
    for j in range(test_for_augm.shape[0]):
        plt.imshow(test_for_augm[j, :, :, 0], cmap='gray')
        # Select image j and channel 0
        plt.show()  # Display the current image
    augmented_data = augmenter.predict(augmented_data)
    # Iterate through each augmented image in the batch and display it
    # individually
    for j in range(augmented_data.shape[0]):
        # The augmented data is now likely 3-dimensional
        plt.imshow(augmented_data[j, :, :], cmap='gray')
        # Remove the extra channel index
        plt.show()  # Display the current image
    augmented_data = filter_data(augmented_data, 5, gen_autoencoder)
```

INTERFEJS FUNKCYJNY BIBLIOTEKI KERAS: ARCHITEKTURY ROZGAŁĘZIONE, CYKLICZNE, Z POŁĄCZENIAMI SKRÓTOWYMI

```
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.api.datasets import mnist, cifar10
from keras.api.models import Model
from keras.api.layers import Input, Dense, Conv2D, MaxPooling2D,\
    AveragePooling2D, GlobalAveragePooling2D, concatenate, Lambda
import math
from tensorflow.keras.utils import plot_model
# Import plot_model from tensorflow.keras.utils

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train.shape
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train.shape
```

```
x_train = np.expand_dims(x_train, axis=-1)
# axis=-1 - automatyczne dodanie osi
x_test = np.expand_dims(x_test, axis=-1)
y_train = pd.get_dummies(pd.Categorical(y_train)).values
y_test = pd.get_dummies(pd.Categorical(y_test)).values

#input_tensor = Input(shape=(28,28,1))
input_tensor = Input(shape = x_train.shape[1:])
output_tensor = input_tensor
output_tensor.shape

act_func = "selu"
#32 - filter count
#(3,3) - kernel size
#10 - classifications count

output_tensor=Conv2D(32, (3,3), padding='same',
activation = act_func)(output_tensor)

output_tensor=MaxPooling2D(pool_size=(2,2))(output_tensor)

def my_act_func(tensor):
    return tf.math.tanh(tensor) * 0.5

output_tensor = Conv2D(32, (3,3), padding='valid',
activation=my_act_func)(output_tensor)

# You need to call the Conv2D layer with the input tensor

global_average_pooling_layer = GlobalAveragePooling2D()
# Assign the layer to a variable

output_tensor = global_average_pooling_layer(output_tensor)
# Apply the layer to the tensor

# Now you can access the output shape
output_tensor.shape

output_tensor = Dense(10, activation='relu')(output_tensor)
output_tensor = Dense(10, activation='softmax')(output_tensor)

model = Model(inputs=input_tensor, outputs=output_tensor)
model.compile( loss = 'categorical_crossentropy',
               metrics = ['accuracy'], optimizer = 'adam' )

layers = [Conv2D(32, (3,3), activation = act_func),
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(10, (3,3), activation = act_func),
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(10, (3,3), activation = act_func),
          GlobalAveragePooling2D(),
          Dense(10, activation = 'softmax')]

output_tensor = input_tensor = Input(x_train.shape[1:])

for layer in layers:
    output_tensor = layer(output_tensor)
```

```
def add_inseption_module(input_tensor):
    act_func = 'relu'
    paths = [
              [ Conv2D(filters = 64, kernel_size=(1,1),
               padding='same', activation=act_func)
              ],
              [ Conv2D(filters = 96, kernel_size=(1,1),
               padding='same', activation=act_func),
               Conv2D(filters = 128, kernel_size=(3,3),
               padding='same', activation=act_func)
              ],
              [ Conv2D(filters = 16, kernel_size=(1,1),
               padding='same', activation=act_func),
               Conv2D(filters = 32, kernel_size=(5,5),
               padding='same', activation=act_func)
              ],
              [ MaxPooling2D(pool_size=(3,3),
               strides = 1, padding='same'),
               Conv2D(filters = 32, kernel_size=(1,1),
               padding='same', activation=act_func)
              ]
    ]

    for_concat = []
    for path in paths:
        output_tensor = input_tensor
        for layer in path:
            output_tensor = layer(output_tensor)
        for_concat.append(output_tensor)

    return concatenate(for_concat)

output_tensor = input_tensor = Input(x_train.shape[1:])
insept_module_cnt  = 2
for i in range(insept_module_cnt):
    output_tensor = add_inseption_module(output_tensor)

output_tensor = GlobalAveragePooling2D()(output_tensor)
output_tensor = Dense(10, activation='softmax')(output_tensor)
ANN = Model(inputs = input_tensor, outputs = output_tensor)
ANN.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'],
    optimizer = 'adam')

plot_model(ANN, show_shapes=True)

def ReLOGU(tensor):
    mask = tensor >= 1
    tensor = tf.where(mask, tensor, 1)
    tensor = tf.math.log(tensor)
    return tensor

output_tensor = input_tensor = Input(x_train.shape[1:])
insept_module_cnt  = 2
for i in range(insept_module_cnt):
    output_tensor = add_inseption_module(output_tensor)

output_tensor = Conv2D(32, (3,3))(output_tensor)
output_tensor = Lambda(ReLOGU)(output_tensor)
output_tensor = GlobalAveragePooling2D()(output_tensor)
output_tensor = Dense(10, activation='softmax')(output_tensor)
```

```
ANN = Model(inputs = input_tensor, outputs = output_tensor)
ANN.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'],
    optimizer = 'adam')

plot_model(ANN, show_shapes=True)
```