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)