Sieci neuronowe i sztuczna inteligencja – laboratorium 9

Monika Błyszcz, 236623

Kod:

```python
# import os utilities
import os
import requests, zipfile, io

# import numpy
import numpy as np

# images
import skimage
from skimage import transform
from skimage.color import rgb2gray

# Import the `pyplot` module
import matplotlib.pyplot as plt

# import random
import random

# import tf
import tensorflow as tf
import tf_slim as slim
import keras
tf.compat.v1.disable_eager_execution()



# function to load data
def load_data(data_directory):
    """Loads sign images data from their folder.

    Returns:
        images: list of images, i.e., signs
        labels: list of labels, i.e., signs IDs
    """
    # We need back labels and the row images
    images = []
    labels = []

    # We have one folder per sign type
    directories = []
    for d in os.listdir(data_directory):
        if os.path.isdir(os.path.join(data_directory, d)):
            directories.append(d)

    # In each foder there are not only images but also csv description
```

```
    # files
    for d in directories:
        label_directory = os.path.join(data_directory, d)
        file_names = [
            os.path.join(label_directory, f)
            for f in os.listdir(label_directory)
            if f.endswith(".ppm")
        ]

        for f in file_names:
            images.append(skimage.io.imread(f))
            labels.append(int(d))

    return images, labels


ROOT_PATH = os.getcwd()

# Download training data

train_data_directory = os.path.join(ROOT_PATH, "Training")
test_data_directory = os.path.join(ROOT_PATH, "Testing")

images, labels = load_data(train_data_directory)
test_images, test_labels = load_data(test_data_directory)

print(labels)

## The following commented lines were reported in the DataCamp materials
## but they does not work here
# print(images.ndim)
# print(images.size)
images[0]
print(len(images))
print(len(labels))

# this should be a bar plot but an histogram with the same number of
# bins that that unique levels of the labels list should be fine :-)
unique_labels = set(labels)
n_labels = max(unique_labels) + 1

# Make a histogram with 62 bins of the `labels` data
plt.hist(labels, n_labels)
```

```python

# Show the plot
plt.show()

# Determine the (random) indexes of the images that you want to see
traffic_signs = [300, 2250, 3650, 4000]

# Fill out the subplots with the random images that you defined
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i + 1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print(
        "shape: {0}, min value: {1}, max value: {2}".format(
            images[traffic_signs[i]].shape,
            images[traffic_signs[i]].min(),
            images[traffic_signs[i]].max()
        )
    )


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

# Plot a grid with a sample of all the signs
plt.figure(figsize=(15, 15))

i = 1

for label in unique_labels:
    # pick the first image for the label.
    #
    # The index() method searches an element in the list and returns its
    # index. In simple terms, index() method finds the given element in
    # a list and returns its position. However, if the same element is
    # present more than once, index() method returns its smallest/first
    # position.
    image = images[labels.index(label)]
```

```
    # We have 62 images. Hence, define a 64 grid sub-plots
    plt.subplot(8, 8, i)

    # Don't include axes
    plt.axis('off')

    # Add a title to each subplot
    #
    # The count() method returns the number of elements with the
    # specified value.
    plt.title("Label {0} ({1})".format(label, labels.count(label)))

    # Add 1 to the counter
    i += 1

    # Plot this first image
    plt.imshow(image)

plt.show()

# To tackle the differing image sizes, you’re going to rescale the images
images_28 = [
    transform.resize(image, (28, 28))
    for image in images
]

# Convert `images28` to an array
images_28 = np.array(images_28)

# Convert `images28` to grayscale
images_28 = rgb2gray(images_28)

for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i + 1)
    plt.axis('off')
    plt.imshow(images_28[traffic_signs[i]], cmap="gray")
    plt.subplots_adjust(wspace=0.5)

plt.show()

# Test set
```

```python
# Transform the images to 28 by 28 pixels
test_images_28 = [
    transform.resize(image, (28, 28))
    for image in test_images
]
# Convert to grayscale
test_images_28 = rgb2gray(np.array(test_images_28))


# Lets start tensorflow!!

# Define placeholders for the inputs and labels
x = tf.compat.v1.placeholder(dtype=tf.float32, shape=[None, 28, 28])
y = tf.compat.v1.placeholder(dtype=tf.int32, shape=[None])

# Flatten the images for the imputs of ANN
images_flat = tf.keras.layers.Flatten()(x)

# Fully connected layer output is 62 as the different signs
# this will be the network!!
logits = slim.layers.fully_connected(images_flat, 62, tf.nn.relu)

# Define a loss function
loss = tf.reduce_mean(
    tf.nn.sparse_softmax_cross_entropy_with_logits(
        labels=y,
        logits=logits
    )
)

# Neural Network
#
# Define an optimizer
train_op = tf.compat.v1.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

# Convert logits to label indexes.
# NOTE: this will be the final classifier which output will be the
#       predicted labels!!
correct_pred = tf.argmax(logits, 1)

```

```python
# Define an accuracy metric
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

print("images_flat: ", images_flat)
print("logits: ", logits)
print("loss: ", loss)
print("predicted_labels: ", correct_pred)

# run the Graph
tf.random.set_seed(1234)

with tf.compat.v1.Session() as sess:
    # initialize all the variables
    sess.run(tf.compat.v1.global_variables_initializer())
    losses = []
    error_train = []
    error_test = []

    # epoch
    for i in range(201):

        # run the optimizer, accordingly to the loss defined, feeding
        # the actual graph with the input we want. In this case all the
        # samples every time.

        # NOTE: this update the weights every time, i.e. the logits,
        #       i.e. the correct_pred!!!
        _, loss_value = sess.run(
            [train_op, loss],
            feed_dict={x: images_28, y: labels}
        )

        # Just print the loss every 10 epoch
        losses.append(loss_value)
        if i % 10 == 0:
            print("Loss: ", loss_value)

        # Run predictions against the full train set.
        predicted_train = sess.run(
            [correct_pred],
            feed_dict={x: images_28}
        )[0]
```

```
        # Calculate mean test error
        train_error = 1 - np.mean([
            int(y == y_)
            for y, y_ in zip(labels, predicted_train)
        ])
        error_train.append(train_error)


        # Run predictions against the full test set.
        predicted_test = sess.run(
            [correct_pred],
            feed_dict={x: test_images_28}
        )[0]
        # Calculate mean test error
        test_error = 1 - np.mean([
            int(y == y_)
            for y, y_ in zip(test_labels, predicted_test)
        ])
        error_test.append(test_error)

    # NOTE: if de-indented the session will be closed and so you cannot
    #       run the sess.run() call

    # Pick 10 random images
    sample_indexes = random.sample(range(len(images_28)), 10)
    sample_images = [images_28[i] for i in sample_indexes]
    sample_labels = [labels[i] for i in sample_indexes]

    # To have predictions we have to run the "correct_pred" operation
    # inside the session, feeding the sample we would like to predict
    predicted = sess.run([correct_pred], feed_dict={x: sample_images})[0]

    # Print the real and predicted labels
    print(sample_labels)
    print(predicted)

    # Display the predictions and the ground truth visually.
    fig = plt.figure(figsize=(10, 10))
    for i in range(len(sample_images)):
```

```
        # i starts from 0!!
        truth = sample_labels[i]
        prediction = predicted[i]
        color = 'green' if truth == prediction else 'red'

        plt.subplot(5, 2, 1 + i)
        plt.axis('off')

        plt.text(
            x=40, y=10,
            s="Truth:        {0}\nPrediction: {1}".format(
                truth, prediction
            ),
            fontsize=12,
            color=color
        )
        plt.imshow(sample_images[i], cmap="gray")

    plt.show()


    # Print the accuracy
    print("Final test error: {:.3f}".format(test_error))

    plt.plot(error_train, "b", error_test, "r--")
    plt.axvline(
        x=error_test.index(min(error_test)),
        color="g", linestyle='--'
    )
    plt.ylabel('Overall classification error')
    plt.xlabel("Epochs")
    plt.title("Training (blue) and test (red) errors by epoch")
    plt.show()


    #Predicted
    predicted1 = sess.run([correct_pred], feed_dict={x: test_images_28})[0]

    #Calculate correct matches
    match_count = sum([int (y == y_) for y, y_ in zip(test_labels,predicted1)])

```

```python
#Calculate the accuracy
accuracy_1 = match_count / len(test_labels)

#Print
print("Accuracy:  {:.3f}".format(accuracy_1))

```

Zad 1

Wynik działania programu

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe C:\Users\Mo\Documents\Studia\SN_lab\SN\PWR_2023_SN\lab8\lab9.py
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
4575
4575
shape: (62, 61, 3), min value: 3, max value: 160
shape: (110, 96, 3), min value: 3, max value: 255
shape: (379, 153, 3), min value: 0, max value: 255
shape: (100, 68, 3), min value: 17, max value: 255
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\tensorflow\python\keras\engine\base_layer_v1.py:1694: UserW
  warnings.warn('`layer.apply` is deprecated and '
images_flat:  Tensor("flatten/Reshape:0", shape=(None, 784), dtype=float32)
logits:  Tensor("fully_connected/Relu:0", shape=(None, 62), dtype=float32)
loss:  Tensor("Mean:0", shape=(), dtype=float32)
predicted_labels:  Tensor("ArgMax:0", shape=(None,), dtype=int64)
Loss:  4.256946
Loss:  3.673935
Loss:  3.4502177
Loss:  3.251147
Loss:  3.0821376
Loss:  2.945996
Loss:  2.8358376
Loss:  2.7467303
Loss:  2.6739264
Loss:  2.6133065
Loss:  2.5620732
Loss:  2.5180707
Loss:  2.479711
Loss:  2.4457924
Loss:  2.4154627
Loss:  2.388023
Loss:  2.3630183
Loss:  2.3400445
Loss:  2.3188314
Loss:  2.2991614
Loss:  2.2808235
[19, 53, 38, 45, 46, 56, 32, 19, 27, 19]
[19 53 22 45 47 56 32 19 28 19]
Final test error: 0.458
Accuracy:  0.542

Process finished with exit code 0
```

Uzyskano accuracy = 0,542. Oznacza to, że sieć neuronowa nie uzyskała bardzo dużego dopasowania, ale jest stosunkowe wysokie jak na pierwszą próbę. Możnaby poprawić kod by uzyskać wyższą wartość accuraccy.

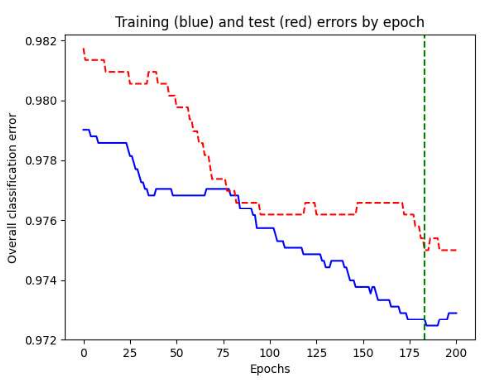
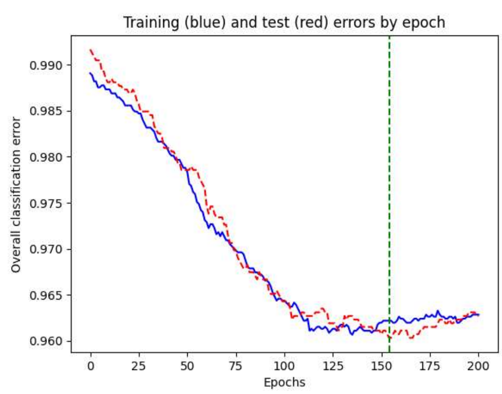
Zad 2

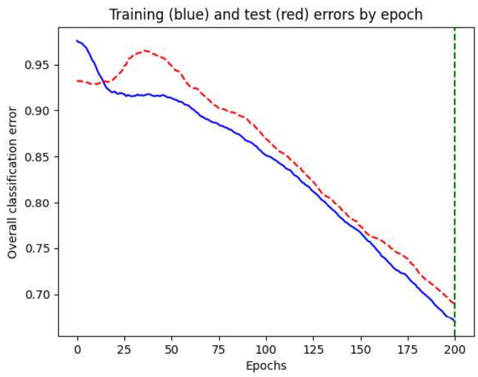
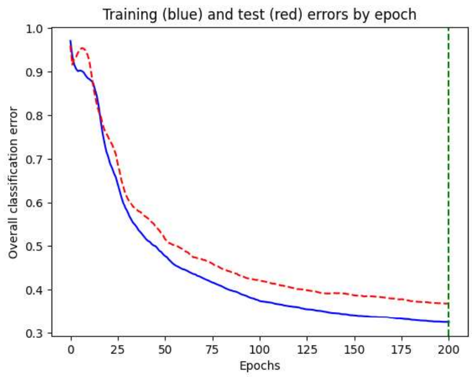
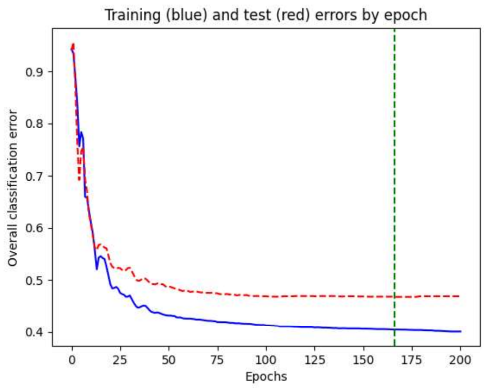
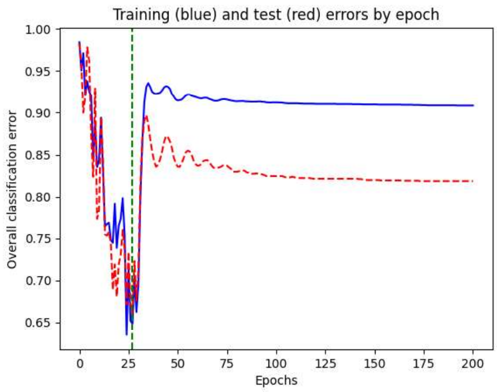
Postanowiono sprawdzić czy da się zoptymalizować wyniki. I uzyskać bardziej dopasowane wykresy epok.

```python
# Neural Network
#
# Define an optimizer
train_op = tf.compat.v1.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

```

Zmieniano wartości learning_rate na następujące: 0,000001; 0,00001; 0,0001; 0,001; 0,01; 0,1.
I porównywano wykresy między sobą.

learning_rate = 0,000001	learning_rate = 0,00001
	

learning_rate = 0,0001	**learning_rate = 0,001**
	
learning_rate = 0,01	learning_rate = 0,1
	

Najlepsze wyniki dopasowania uzyskano dla learning rate = 0,001. Wykresy treningowe (Training) oraz testowe (Test) niemal się pokrywają. Największe różnice wyszły dla learning rate = 0,1. Im wyższa wartość, tym bardziej się rozmijają wynik