

# Gender\_prediction

April 3, 2024

## GENDER PREDICTION APPLICATION

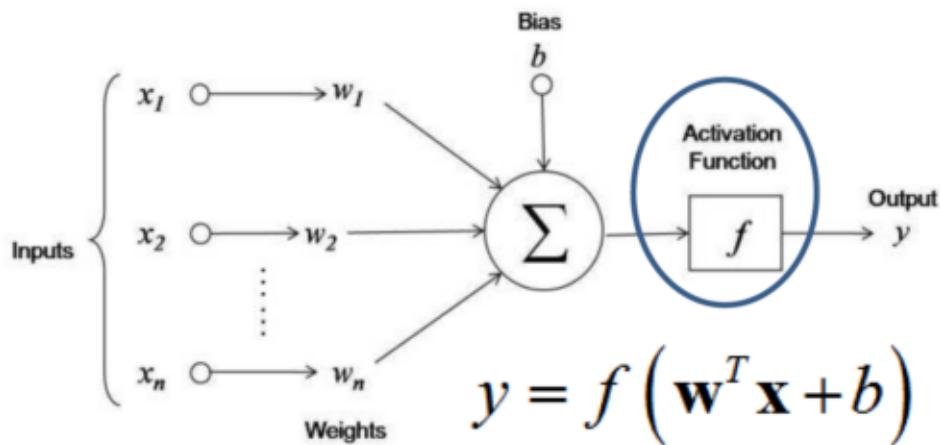
### OGÓLNY ZARYS:

- Omów schemat neuronu: Neuron w sztucznych sieciach neuronowych składa się z kilku elementów: sygnałów wejściowych, wag, sumatora, funkcji aktywacji oraz sygnału wyjściowego. Sygnały wejściowe są przemnażane przez odpowiadające im wagi, a następnie sumowane w sumatorze. Wynik tej sumy jest przekazywany do funkcji aktywacji, która decyduje, czy neuron zostanie aktywowany czy nie. Neurony są połączone w warstwy, tworząc w ten sposób sieć neuronową.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fundefined.photos%2Fphoto-gallery%2Fimitation>

```
[127]: import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/artificial_neuron.
˓→png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
```



- Przedstaw algorytm uczenia neuronu:

Inicjalizacja wag: Wagi dla każdego wejścia neuronu są inicjalizowane losowo lub ustalone na początku na małe wartości bliskie zeru.

Przekazanie wejścia: Dla każdego przykładu treningowego, wartości wejściowe są podawane na perceptron.

Obliczenie sumy ważonej: Obliczana jest suma ważona wejść przez odpowiadające wagi.

Przejście przez funkcję aktywacji: Wynik sumy ważonej jest poddawany funkcji aktywacji. W przypadku perceptronu jest to zazwyczaj funkcja skoku Heaviside'a, która zwraca 1, jeśli wynik jest większy lub równy zeru, a w przeciwnym razie zwraca 0.

Porównanie wyniku z oczekiwany wynikiem: Wynik przewidywany przez perceptron jest porównywany z rzeczywistym oczekiwany wynikiem. Jeśli wynik jest zgodny z oczekiwany, perceptron nie modyfikuje swoich wag. W przeciwnym razie, wagi są aktualizowane w celu zmniejszenia różnicy między przewidywanym a rzeczywistym wynikiem.

Aktualizacja wag: Wagi są aktualizowane zgodnie z regułą uczenia, która zmniejsza błąd. Jedną z najczęściej stosowanych reguł jest reguła uczenia perceptronu (perceptron learning rule), która modyfikuje każdą wagę według wzoru:  $w_{\{i\}} = w_{\{i\}} + \alpha(d - y) \times x_{\{i\}}$

Powtarzanie procesu: Proces ten jest powtarzany dla każdego przykładu treningowego przez określoną liczbę epok lub do momentu, gdy błąd jest wystarczająco mały.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Ftowardsdatascience.com%2Fperceptron-learning>

```
[128]: img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/algorithm.png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
```

---

**Algorithm** Perceptron learning algorithm

---

**Input:**

A set of training examples  $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Learning rate  $0 < \alpha < 1$

Number of epochs  $epochs$

```
1: Initialize the weight vector  $\mathbf{w}$  with random values
2: Initialize the bias  $b \leftarrow 0$ 
3: for  $i \leftarrow 1$  to  $epochs$  do
4:    $err \leftarrow 0$                                  $\triangleright$  The number of misclassifications
5:   for each training example  $(\mathbf{x}_i, y_i) \in D$  do
6:      $z_i \leftarrow \mathbf{w}^T \mathbf{x}_i + b$ 
7:      $o_i \leftarrow \begin{cases} 1 & z_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$ 
8:     if  $y_i \neq o_i$  then
9:        $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - o_i)\mathbf{x}_i$ 
10:       $b \leftarrow b + \alpha(y_i - o_i)$ 
11:       $err \leftarrow err + 1$ 
12:    if  $err = 0$  then break
```

---

- Wyjaśnienie działania sieci neuronowej:

Sieć neuronowa składa się z warstw neuronów, które przetwarzają sygnały wejściowe, przekazując je przez serię operacji matematycznych i funkcji aktywacji. Te operacje są wykonywane w sposób sekwencyjny od warstwy wejściowej do warstwy wyjściowej. Podczas treningu sieć dostosowuje swoje wagi (algorytm wstecznej propagacji) na podstawie dostarczonych danych treningowych i oczekiwanych wyników, aby minimalizować błąd predykcji.

Na początku, dane wejściowe są przekazywane do pierwszej warstwy neuronów, nazywanej warstwą wejściową. Każdy neuron w warstwie wejściowej przyjmuje jedną wartość z wektora wejściowego. Następnie te wartości są przekazywane do kolejnych warstw neuronów poprzez wagi, które określają, jak bardzo każda wartość wejściowa ma wpływ na aktywację neuronów w kolejnej warstwie.

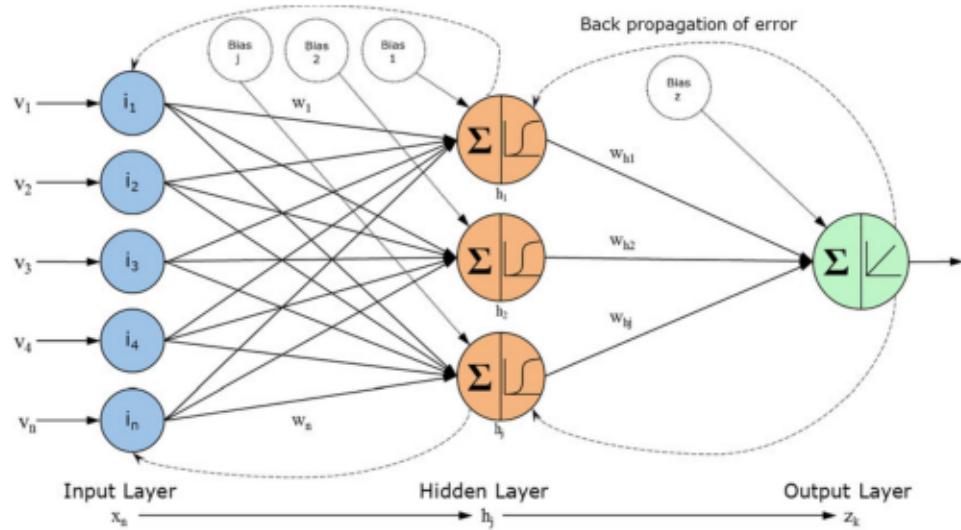
W każdej warstwie neuronów, każdy neuron oblicza ważoną sumę swoich wejść, a następnie stosuje funkcję aktywacji do tej sumy, aby wygenerować wynik. Funkcja aktywacji decyduje, czy neuron powinien zostać aktywowany (wygenerować wartość niezerową) lub zdezaktywowany (wygenerować wartość bliską zeru).

Podczas treningu, sieć neuronowa dostosowuje swoje wagi, aby minimalizować różnicę między przewidywanymi a rzeczywistymi wynikami. Jest to osiągane poprzez algorytm wstecznej propagacji błędu. Algorytm ten oblicza gradient funkcji straty (różnice między przewidywanymi a rzeczywistymi wynikami) i propaguje ten gradient wstecz przez sieć, aby obliczyć, jak każda waga w sieci powinna być zmieniona, aby zmniejszyć ten błąd. Wagi są następnie aktualizowane zgodnie z tymi gradientami, prowadząc do poprawy zdolności sieci do dokładnego przewidywania.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FWorkflow-diag>

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fprogrammerhumor.io%2Fmemes%2Fneural-network%2F>

```
[129]: img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/  
    ↪neural_network_algorithm.png"))  
plt.imshow(img)  
plt.axis('off')  
plt.grid(False)  
plt.show()  
  
img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/soul.png"))  
plt.imshow(img)  
plt.axis('off')  
plt.grid(False)  
plt.show()
```



## How Neural Networks work?

### Neurons:

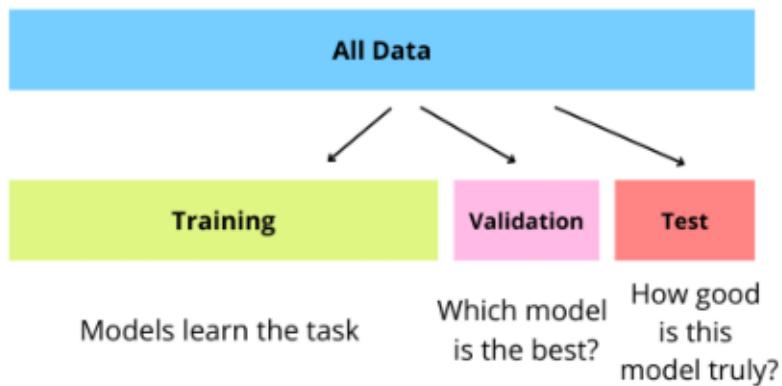


ProgrammerHumor.io

- Na jakie podzbiory i dlaczego stosuje się podział danych: Danych używanych do treningu sieci neuronowej często dzieli się na trzy podzbiory: treningowy, walidacyjny i testowy [U nas są tylko dwa, gdzie zbiór testowy jest nazwany walidacyjnym - no cóż tak ktoś nazwał na kaggle...] Zbiór treningowy służy do dostosowania wag sieci, zbiór walidacyjny do oceny jej wydajności podczas treningu (np. do dostosowywania parametrów uczenia) i zbiór testowy do ostatecznej oceny skuteczności sieci na danych, których nie widziała wcześniej. Podział ten zapewnia niezależne zestawy danych do treningu, oceny i testowania, co pomaga uniknąć przeuczenia (overfitting) modelu.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fmedium.com%2F40rahulchavan4894%2Funderstand>

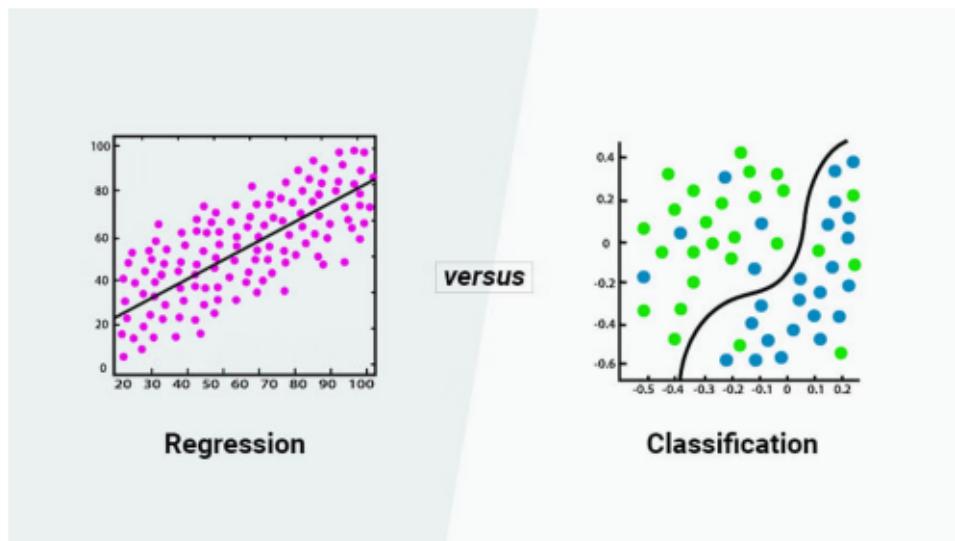
```
[130]: img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/train_test.png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
```



- Czym się różni problem regresji od problemu klasyfikacji: Problem regresji polega na przewidywaniu wartości ciągły, np. cen nieruchomości, na podstawie danych wejściowych. Natomiast problem klasyfikacji polega na przypisywaniu danych wejściowych do określonych klas lub kategorii, np. rozpoznawanie gatunku roślin na podstawie cech morfologicznych. Różnica polega więc na rodzaju danych wyjściowych, ciągłych w przypadku regresji i dyskretnych (klas lub kategorii) w przypadku klasyfikacji.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.simplilearn.com%2Fregression-vs-classification>

```
[131]: img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/regression_classification.png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
```



Przedstawiony przez nas problem będzie obejmował klasyfikację płci na podstawie zdjecia twarzy. W tym celu na wstępnie przetworzonych danych wyszkolimy model uczenia maszynowego przy wykorzystaniu biblioteki PyTorch. Następnie wykorzystamy go w praktycznym celu do napisania aplikacji mobilnej w języku Swift (iOS) rozpoznajacej płeć badanej osoby na podstawie inputu z tylniej kamery urządzenia.

Pobranie przy pomocy klucza API z Kaggle zbioru danych zawierającego zdjecia twarzy mężczyzn i kobiet.

```
[185]: import os
import kaggle # Upewnij się, że plik kaggle.json znajduje się w katalogu .
           ↪kaggle w Twoim katalogu domowym

dataset_path = 'Datasets/gender-classification-dataset'
if not os.path.exists(dataset_path):
    kaggle.api.dataset_download_files('cashutosh/
           ↪gender-classification-dataset', path=dataset_path, unzip=True)
else:
    print("Zbior danych został już pobrany")
```

Zbior danych został już pobrany

Ścieżka do katalogu z pobranym zbiorem danych:

```
[186]: #Dataset path:
PATH = 'Datasets/gender-classification-dataset'
```

Usunięcie podwójnego rozszerzenia “.jpg.jpg” z plików:

```
[187]: def remove_double_jpg(directory):
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg.jpg"):
                old_path = os.path.join(root, file)
                new_path = old_path.rsplit('.', 1)[0]
                os.rename(old_path, new_path)
remove_double_jpg(PATH)
```

Zmiana nazwy plików na liczby od 1 do n:

```
[188]: def rename_images(directory):
    for root, dirs, files in os.walk(directory):
        counter = 1
        for file in sorted(files):
            if file.endswith(".jpg"):
                old_path = os.path.join(root, file)
                new_path = os.path.join(root, f"{counter}.jpg")
                os.rename(old_path, new_path)
                counter += 1
rename_images(PATH)
```

Określenie minimalnych i maksymalnych wymiarów zdjęć w zbiorze danych:

```
[189]: from PIL import Image
import os

def check_shape(directory):
    min_width = float('inf')
    min_height = float('inf')
    max_width = float('-inf')
    max_height = float('-inf')
    min_width_image = None
    min_height_image = None
    max_width_image = None
    max_height_image = None

    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg"):
                path = os.path.join(root, file)
                image = Image.open(path)
                width, height = image.size

                if width < min_width:
                    min_width = width
                    min_width_image = path
```

```

        if height < min_height:
            min_height = height
            min_height_image = path

        if width > max_width:
            max_width = width
            max_width_image = path

        if height > max_height:
            max_height = height
            max_height_image = path

image_1 = Image.open(min_width_image)
image_2 = Image.open(min_height_image)
image_11 = Image.open(max_width_image)
image_22 = Image.open(max_height_image)

print("Image with the smallest width:", min_width_image.split('/')[-1], image_1.size)
print("Image with the smallest height:", min_height_image.split('/')[-1], image_2.size)
print("Image with the biggest width:", max_width_image.split('/')[-1], image_11.size)
print("Image with the biggest height:", max_height_image.split('/')[-1], image_22.size)

check_shape(PATH)

```

Image with the smallest width: 20049.jpg (52, 62)  
 Image with the smallest height: 22773.jpg (52, 58)  
 Image with the biggest width: 20779.jpg (111, 131)  
 Image with the biggest height: 3137.jpg (105, 151)

Zmiana rozmiaru zdjęć na 100x100 w celu zachowania spójności wymiarów:

```
[190]: def resize_image(directory, x=100, y=100):
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg"):
                path = os.path.join(root, file)
                image = Image.open(path)
                resized_image = image.resize((x, y))
                resized_image = resized_image.convert("RGB")
                resized_image.save(path)

resize_image(PATH)
```

Sprawdzenie czy wszystkie zdjęcia są w formacie RGB w celu zachowania jednakowej ilości cech wejściowych do sieci:

```
[191]: def is_rgb_image(directory):
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg"):
                path = os.path.join(root, file)
                image = Image.open(path)
                if image.mode != 'RGB':
                    print(f'Image {path} is not in RGB format')
    print('All images are in RGB format')

is_rgb_image(PATH)
```

All images are in RGB format

Podział zbioru danych na zbiór treningowy i walidacyjny (ratio: ~0.2 Validation // ~0.8 Training) na podstawie struktury katalogów w pobranym zbiorze danych:

```
[192]: categories = ['Training', 'Validation']
genders = ['male', 'female']

images = []
labels = []
purposes = []

image_files = {
    category: {
        gender: [
            os.path.join(PATH, category, gender, file)
            for file in os.listdir(os.path.join(PATH, category, gender))
            if file.endswith(".jpg")
        ]
        for gender in genders
    }
    for category in categories
}
```

Funkcja do załadowywania zdjęć z rozszerzeniami:

```
[193]: import cv2

def load_image_with_extension(path):
    image = cv2.imread(path)
    if image is not None:
        return image
    return None
```

Rozdzielenie cech na macierz pixeli, etykiety płci oraz cel: zbiór treningowy/walidacyjny:

```
[194]: def select_faces(image_files):
    images = []
```

```

labels = []
purposes = []
for purpose, gender_dict in image_files.items():
    for gender, files in gender_dict.items():
        for file in files:
            image = load_image_with_extension(file)
            if image is not None:
                images.append(image)
                labels.append(gender)
                purposes.append(purpose)
return images, labels, purposes

images, labels, purposes = select_faces(image_files)

```

Utworzenie ramki danych z pozyskanych cech z kolumnami: [Image,Gender,Purpose]:

```
[195]: import pandas as pd

df = pd.DataFrame({"Image":images, "Gender":labels, "Purpose":purposes})
print(df)
```

	Image	Gender	Purpose
0	[[[18, 20, 28], [21, 23, 31], [21, 25, 30], [2...	male	Training
1	[[[25, 31, 30], [23, 31, 30], [24, 33, 36], [2...	male	Training
2	[[[0, 3, 21], [0, 9, 29], [24, 37, 59], [62, 7...	male	Training
3	[[[172, 166, 185], [173, 167, 186], [174, 170,...	male	Training
4	[[[78, 109, 178], [82, 113, 182], [86, 118, 19...	male	Training
...	...	...	...
58653	[[[44, 50, 63], [43, 49, 62], [41, 45, 56], [3...	female	Validation
58654	[[[57, 68, 95], [49, 60, 87], [41, 53, 77], [3...	female	Validation
58655	[[[29, 43, 65], [26, 40, 62], [25, 41, 64], [2...	female	Validation
58656	[[[40, 52, 56], [38, 50, 54], [35, 47, 51], [3...	female	Validation
58657	[[[38, 38, 52], [29, 29, 43], [22, 22, 36], [2...	female	Validation
58658 rows x 3 columns]			

Przykładowa próbka:

```
[196]: import matplotlib.pyplot as plt

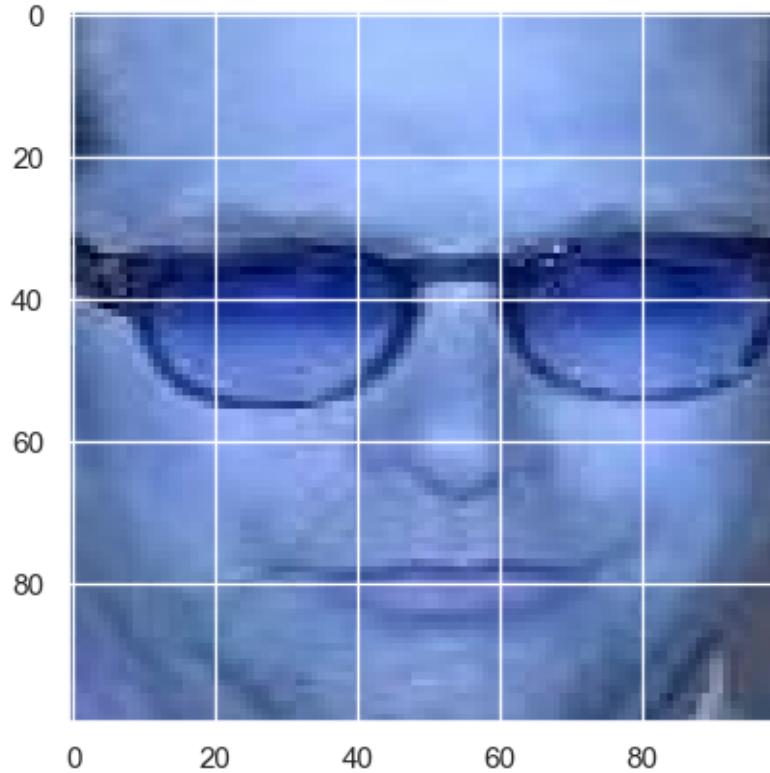
def check_loading_correctness(df):
    rand = df.sample(1)

    print(f"Example image // sex: {rand['Gender'].values[0]} //"
    ↪{rand['Purpose'].values[0]})

    image = rand['Image'].values[0]
    plt.imshow(image)
    plt.show()
```

```
print("Shape: ", image.shape)
check_loading_correctness(df)
```

Example image // sex: male // Training



Shape: (100, 100, 3)

Określenie ilości próbek należących do poszczególnych klas:

```
[197]: from tabulate import tabulate

size = len(df)
print("Total samples:", size)

men = df[df["Gender"] == "male"]
women = df[df["Gender"] == "female"]

data = [['Male', len(men)], ['Female', len(women)]]
table = tabulate(data, headers=['Gender', 'Count'], tablefmt='pretty')

print(table)
```

Total samples: 58658

Gender	Count
Male	29574
Female	29084

Wyrównanie ilości próbek w klasie mniejszościowej (kobiety) za pomocą oversamplingu:

```
[198]: # Oversampling
from sklearn.utils import resample

df_male = df[df['Gender'] == 'male']
df_female = df[df['Gender'] == 'female']

df_female_oversampled = resample(df_female,
                                 replace=True,
                                 n_samples=len(df_male),
                                 random_state=123)

df_oversampled = pd.concat([df_male, df_female_oversampled])
```

Preferowana paleta kolorów:

```
[199]: import matplotlib.pyplot as plt
import seaborn as sns
sns.set_palette('cubebehelix')
sns.color_palette("cubebehelix")
```

```
[199]: [(0.10231025194333628, 0.13952898866828906, 0.2560120319409181),
(0.10594361078604106, 0.3809739011595331, 0.27015111282899046),
(0.4106130272672762, 0.48044780541672255, 0.1891154277778484),
(0.7829183382530567, 0.48158303462490826, 0.48672451968362596),
(0.8046168329276406, 0.6365733569301846, 0.8796578402926125),
(0.7775608374378459, 0.8840392521212448, 0.9452007992345052)]
```

Histogram przedstawiający rozkład płci w populacji po dokonanej korekcji zbioru danych:

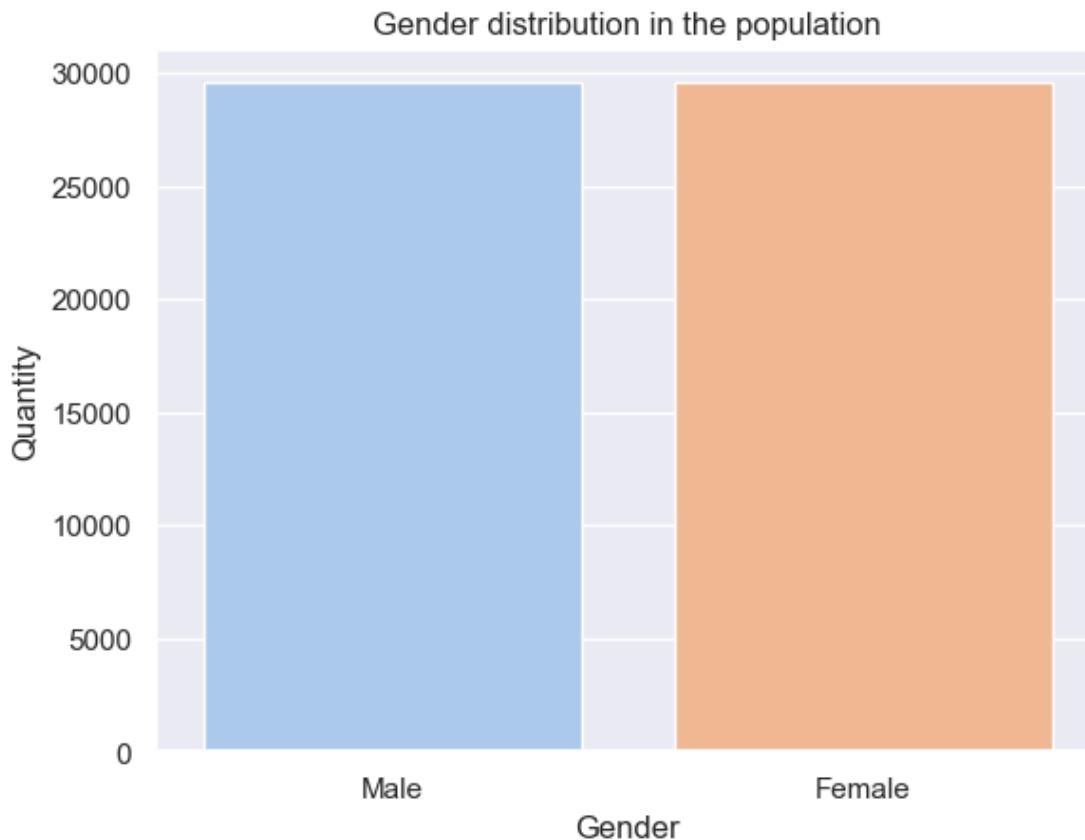
```
[200]: import seaborn as sns

sns.set_theme()
palette = sns.color_palette("pastel", n_colors=len(df_oversampled['Gender']).
                             unique())
sns.countplot(data=df_oversampled, x='Gender', hue='Gender', palette=palette, u
               ↪legend=False)
plt.xlabel('Gender')
plt.ylabel('Quantity')
plt.title("Gender distribution in the population")
plt.gca().set_xticks([0, 1])
```

```

plt.gca().set_xticklabels(['Male', 'Female'])
plt.savefig("../Data/gender_distribution.png")
plt.show()

```



```

[201]: gender_counts = df_oversampled['Gender'].value_counts().reset_index()
gender_counts.columns = ['Gender', 'Count']
table = tabulate(gender_counts, headers='keys', tablefmt='pretty')
print(table)

```

```

+---+-----+---+
|   | Gender | Count |
+---+-----+---+
| 0 | male   | 29574 |
| 1 | female | 29574 |
+---+-----+---+

```

Sprawdzenie czy Oversampling nie zaburzył (znacząco podziału na zbiór treningowy i testowy - proporcja odbiega znacząco od 0.2):

```
[202]: training_samples = len(df[df['Purpose'] == 'Training'])
validation_samples = len(df[df['Purpose'] == 'Validation'])

validation_ratio = validation_samples / (training_samples + validation_samples)
print(validation_ratio)
```

0.19859183743052952

Kodowanie zmiennej jakościowej dla celów szkoleniowych (model potrzebuje danych numerycznych):

```
[81]: df['Gender'] = df['Gender'].replace({'female': 1, 'male': 0})
```

Model sieci #1 jest zaimplementowany przy użyciu biblioteki PyTorch. Jest to konwolucyjna sieć neuronowa (CNN), która jest często używana do przetwarzania obrazów.

- Pierwszą warstwą w tej sieci jest warstwa konwolucyjna (self.conv1), która przyjmuje na wejściu obrazy o trzech kanałach i aplikuje zestaw filtrów, aby uzyskać 16 map cech. Filtry te mają rozmiar jądra 3x3 i są używane z paddingiem równym 1, co pozwala utrzymać rozmiar obrazu na wyjściu. Następnie, po każdej warstwie konwolucyjnej, stosowany jest nieliniowy element aktywacji ReLU (self.relu), który wprowadza nieliniowość do modelu poprzez zastosowanie funkcji max(0, x) do każdej wartości piksela w mapach cech.
- Po warstwie konwolucyjnej następuje warstwa poolingowa (self.pool), która redukuje rozmiar map cech przez wybieranie maksymalnych wartości z okna o rozmiarze 2x2 i kroku 2. Pozwala to na zmniejszenie przestrzeni pikseli i osiągnięcie inwariantności na translację w modelu.
- Następnie wynik z warstwy poolingowej jest spłaszczany (x.reshape(-1, 505016)) i przekazywany do dwóch warstw w pełni połączonych (self.fc1, self.fc2). Pierwsza z nich zawiera 128 neuronów i działa jako warstwa ukryta, przetwarzając cechy wyodrębnione przez warstwy konwolucyjne. Druga warstwa zawiera tylko jeden neuron, ponieważ model ma na celu klasyfikację binarną (np. czy obraz zawiera dany obiekt czy nie).
- Po każdej z warstw w pełni połączonych stosowana jest funkcja aktywacji ReLU, z wyjątkiem ostatniej warstwy, gdzie używana jest funkcja sigmoidalna (self.sigmoid). Sigmoida przekształca wyniki na przedział (0,1), co jest użyteczne w przypadku problemów binarnych, gdzie model musi zwrócić prawdopodobieństwo przynależności do danej klasy.
- Dodatkowo, w trakcie treningu, przed podaniem danych do warstwy wyjściowej (self.fc2), stosowana jest warstwa dropout (self.dropout), która losowo wyłącza część neuronów z określonym prawdopodobieństwem. To pomaga w regularyzacji modelu i zapobiega przeuczeniu poprzez zmniejszenie współzależności między neuronami.
- Ostatecznie, model zwraca wynik, który jest interpretowany jako prawdopodobieństwo przynależności obrazu do danej klasy, gdzie wartość bliższa 1 oznacza większe prawdopodobieństwo, a bliższa 0 - mniejsze.

Sprawdzenie czy na urządzeniu jest dostępna karta graficzna (cuda) w celu przyspieszenia procesu uczenia:

```
[82]: import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

cpu

```
[83]: import torch.nn as nn

class Net_1(nn.Module):
    def __init__(self, dropout_prob=0.5):
        super(Net_1, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(50*50*16, 128)
        self.fc2 = nn.Linear(128, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = x.reshape(-1, 50*50*16)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.sigmoid(self.fc2(x))
        return x
```

Model sieci #2 to modyfikacja oryginalnego modelu:

- Warstwa konwolucyjna: Zwiększo liczbę cech wyjściowych z 16 do 32 (out\_channels=32), co oznacza, że w wyniku działania tej warstwy uzyskujemy więcej map cech.
- Warstwa w pełni połączona (fc1): Zmieniono liczbę neuronów w tej warstwie na 256 (nn.Linear(50\*50\*32, 256)), co zwiększa złożoność modelu poprzez dodanie większej liczby parametrów do uczenia.

```
[84]: import torch.nn as nn

class Net_2(nn.Module):
    def __init__(self, dropout_prob=0.5):
        super(Net_2, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(50*50*32, 256)
        self.fc2 = nn.Linear(256, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, x):
```

```

x = self.relu(self.conv1(x))
x = self.pool(x)
x = x.reshape(-1, 50*50*32)
x = self.relu(self.fc1(x))
x = self.dropout(x)
x = self.sigmoid(self.fc2(x))
return x

```

Model sieci #3 wprowadza kolejną modyfikację w porównaniu do poprzednich dwóch podejść:

- Pierwsza warstwa konwolucyjna: Zmieniono liczbę cech wyjściowych generowanych przez tą wartswę na 16.
- Druga warstwa konwolucyjna: Dodano kolejną warstwę konwolucyjną (self.conv2), która ma 32 cechy wyjściowe.

```
[85]: import torch.nn as nn

class Net_3(nn.Module):
    def __init__(self, dropout_prob=0.5):
        super(Net_3, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(25*25*32, 128)
        self.fc2 = nn.Linear(128, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.max_pool2d(x)
        x = self.relu(self.conv2(x))
        x = self.max_pool2d(x)
        x = x.view(-1, 25*25*32)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.sigmoid(self.fc2(x))
        return x
```

Model sieci #4 wprowadza modyfikacje względem swojego poprzednika:

- Pierwsza warstwa konwolucyjna: Zmieniono liczbę cech wyjściowych generowanych przez tą wartswę na 32.
- Druga warstwa konwolucyjna: Ma ona teraz 64 cechy wyjściowe.

```
[86]: import torch.nn as nn
```

```

class Net_4(nn.Module):
    def __init__(self, dropout_prob=0.5):
        super(Net_4, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(25*25*64, 256)
        self.fc2 = nn.Linear(256, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.max_pool2d(x)
        x = self.relu(self.conv2(x))
        x = self.max_pool2d(x)
        x = x.view(-1, 25*25*64)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.sigmoid(self.fc2(x))
        return x

```

Model sieci #5 wprowadza kolejne zmiany w strukturze w porównaniu do poprzednich modeli. Oto szczegółowo:

Trzecia warstwa konwolucyjna: Model posiada teraz trzecią warstwę konwolucyjną (self.conv3), która ma 64 cechy wyjściowe. Dodanie tej warstwy pozwala na dalsze zwiększenie złożoności modelu poprzez ekstrakcję bardziej skomplikowanych cech.

```
[87]: import torch.nn as nn

class Net_5(nn.Module):
    def __init__(self, dropout_prob=0.5):
        super(Net_5, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(12*12*64, 128)
        self.fc2 = nn.Linear(128, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
```

```

def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.max_pool2d(x)
    x = self.relu(self.conv2(x))
    x = self.max_pool2d(x)
    x = self.relu(self.conv3(x))
    x = self.max_pool2d(x)
    x = x.view(-1, 12*12*64)
    x = self.relu(self.fc1(x))
    x = self.dropout(x)
    x = torch.sigmoid(self.fc2(x))
    return x

```

Stworzenie instancji modeli:

```
[88]: Gender_model_1 = Net_1().to(device)
Gender_model_2 = Net_2().to(device)
Gender_model_3 = Net_3().to(device)
Gender_model_4 = Net_4().to(device)
Gender_model_5 = Net_5().to(device)
```

```
[89]: Gender_model_1
```

```
[89]: Net_1(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=40000, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)
```

```
[90]: Gender_model_2
```

```
[90]: Net_2(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=80000, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)
```

```
[91]: Gender_model_3
```

```
[91]: Net_3(  
    (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (fc1): Linear(in_features=20000, out_features=128, bias=True)  
    (fc2): Linear(in_features=128, out_features=1, bias=True)  
    (relu): ReLU()  
    (dropout): Dropout(p=0.5, inplace=False)  
    (max_pool2d): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
        ceil_mode=False)  
)
```

```
[92]: Gender_model_4
```

```
[92]: Net_4(  
    (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (fc1): Linear(in_features=40000, out_features=256, bias=True)  
    (fc2): Linear(in_features=256, out_features=1, bias=True)  
    (relu): ReLU()  
    (dropout): Dropout(p=0.5, inplace=False)  
    (max_pool2d): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
        ceil_mode=False)  
)
```

```
[93]: Gender_model_5
```

```
[93]: Net_5(  
    (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (fc1): Linear(in_features=9216, out_features=128, bias=True)  
    (fc2): Linear(in_features=128, out_features=1, bias=True)  
    (relu): ReLU()  
    (dropout): Dropout(p=0.5, inplace=False)  
    (max_pool2d): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
        ceil_mode=False)  
)
```

Definicja funkcji straty - Binary Cross Entropy Loss wydaje się być najlepsze do problemu klasyfikacji binarnej (karze model za bycie pewnym i mylnym. Jeśli model przewiduje bardzo wysokie prawdopodobieństwo dla niewłaściwej klasy, funkcja straty będzie bardzo wysoka):

```
[94]: criterion = nn.BCELoss()  
print(criterion)
```

```
BCELoss()
```

Definicja optymalizatora - aktualizującego wagę modelu w procesie uczenia (ulepszony Adam -> AdamW). W klasycznym Adamie, regularyzacja wag jest włączana bezpośrednio do aktualizacji

momentów, co może prowadzić do niewłaściwego skalowania gradientów. Z drugiej strony, AdamW stosuje regularyzację wag bezpośrednio do parametrów modelu, co jest bardziej zgodne z intuicją stojącą za regularyzacją wag.

Dzięki temu AdamW często osiąga lepsze wyniki w porównaniu do klasycznego Adama, szczególnie w przypadku głębokich sieci neuronowych.

```
[95]: import torch.optim as optim
optimizer_1 = optim.AdamW(Gender_model_1.parameters(), lr=0.001) # lr=learning rate
optimizer_2 = optim.AdamW(Gender_model_2.parameters(), lr=0.001)
optimizer_3 = optim.AdamW(Gender_model_3.parameters(), lr=0.001)
optimizer_4 = optim.AdamW(Gender_model_4.parameters(), lr=0.001)
optimizer_5 = optim.AdamW(Gender_model_5.parameters(), lr=0.001)
```

Utworzenie tensorów z obrazami oraz etykietami płci zarówno dla zbioru treningowego jak i walidacyjnego oraz przeprowadzenie ich normalizacji (wartość pixeli 0-1):

```
[96]: import torch

train_df = df[df['Purpose'] == 'Training']
test_df = df[df['Purpose'] == 'Validation']

x_train = torch.tensor(train_df['Image'].values.tolist(), dtype=torch.float32).
    reshape(-1,3,100,100) / 255.0
y_train = torch.tensor(train_df['Gender'].values.tolist(), dtype=torch.float32).
    reshape(-1, 1)
x_test = torch.tensor(test_df['Image'].values.tolist(), dtype=torch.float32).
    reshape(-1,3,100,100) / 255.0
y_test = torch.tensor(test_df['Gender'].values.tolist(), dtype=torch.float32).
    reshape(-1, 1)
```

Utworzenie Loaderów danych treningowych i walidacyjnych zadaną wielkością paczki (batchsize):

```
[97]: from torch.utils.data import TensorDataset, DataLoader

train_dataset = TensorDataset(x_train, y_train)
test_dataset = TensorDataset(x_test, y_test)

batch_size = 64
trainloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
testloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
```

Definicja funkcji trenującej (model monitoruje wartość accuracy i jeśli ten parametr nie ulega poprawie przez zadaną liczbę epok [epochs=15/patience=3], proces uczenia zostaje przerwany i zapisany zostaje model z największą dokładnością (największą ilością trafnych określeń) na zbiorze testowym). Mierzona jest też wartość straty na zbiorze testowym, co spowodowane jest koniecznością oceny ogólnej wydajności modelu podczas procesu uczenia. Zmniejszenie straty na zbiorze testowym jest jednym z głównych celów uczenia maszynowego, ponieważ wskazuje ona na zdolność

modelu do generalizacji wzorców na danych, których nie widział podczas treningu.

```
[35]: import torch
from sklearn.metrics import accuracy_score

def train(Gender_model,optimizer,num_epochs=15,patience=3):

    losses = []
    accuracies = []

    best_model_weights = None
    no_improvement_count = 0
    best_accuracy = 0.0
    best_accuracy_loss = 0.0

    for epoch in range(num_epochs):
        for inputs, labels in trainloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = Gender_model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            with torch.no_grad():
                Gender_model.eval()
                gender_predictions = []
                true_labels = []
                test_losses = []

                for inputs, labels in testloader:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    outputs = Gender_model(inputs)
                    test_loss = criterion(outputs, labels)
                    test_losses.append(test_loss.item())
                    gender_predictions.extend((outputs > 0.5).int().numpy())
                    true_labels.extend(labels.numpy())

                accuracy = accuracy_score(true_labels, gender_predictions)
                avg_test_loss = sum(test_losses) / len(test_losses)
                losses.append(avg_test_loss)
                accuracies.append(accuracy)
```

```

        print(f"Epoch {epoch + 1}: Loss {round(avg_test_loss,3)}, Accuracy={round(accuracy,3)}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_accuracy_loss = avg_test_loss
        best_model_weights = Gender_model.state_dict()
        no_improvement_count = 0
    else:
        no_improvement_count += 1

    if no_improvement_count >= patience:
        print(f"Stop learning, no improvement for {patience} epochs.")
        break

    if best_model_weights is not None:
        Gender_model.load_state_dict(best_model_weights)

return best_accuracy, best_accuracy_loss, losses, accuracies

```

Miary modelu po procesie uczenia wraz z historią zmian straty i dokładności na zbiorze testowym dla każdego z modeli:

[36]: best\_accuracy1, best\_accuracy\_loss1, losses1, accuracies1 =  
 ↪train(Gender\_model\_1,optimizer\_1)

```

Epoch 1: Loss 0.294, Accuracy 0.873
Epoch 2: Loss 0.206, Accuracy 0.915
Epoch 3: Loss 0.195, Accuracy 0.923
Epoch 4: Loss 0.186, Accuracy 0.922
Epoch 5: Loss 0.197, Accuracy 0.921
Epoch 6: Loss 0.199, Accuracy 0.917
Stop learning, no improvement for 3 epochs.

```

[37]: best\_accuracy2, best\_accuracy\_loss2, losses2, accuracies2 =  
 ↪train(Gender\_model\_2,optimizer\_2)

```

Epoch 1: Loss 0.249, Accuracy 0.905
Epoch 2: Loss 0.195, Accuracy 0.924
Epoch 3: Loss 0.224, Accuracy 0.908
Epoch 4: Loss 0.209, Accuracy 0.919
Epoch 5: Loss 0.179, Accuracy 0.929
Epoch 6: Loss 0.214, Accuracy 0.916
Epoch 7: Loss 0.192, Accuracy 0.933
Epoch 8: Loss 0.273, Accuracy 0.909
Epoch 9: Loss 0.213, Accuracy 0.93
Epoch 10: Loss 0.214, Accuracy 0.934
Epoch 11: Loss 0.254, Accuracy 0.93
Epoch 12: Loss 0.248, Accuracy 0.931

```

```
Epoch 13: Loss 0.276, Accuracy 0.919  
Stop learning, no improvement for 3 epochs.
```

```
[75]: best_accuracy3, best_accuracy_loss3, losses3, accuracies3 =  
      ↪train(Gender_model_3,optimizer_3)
```

```
Epoch 1: Loss 0.267, Accuracy 0.894  
Epoch 2: Loss 0.252, Accuracy 0.893  
Epoch 3: Loss 0.218, Accuracy 0.915  
Epoch 4: Loss 0.21, Accuracy 0.916  
Epoch 5: Loss 0.212, Accuracy 0.915  
Epoch 6: Loss 0.196, Accuracy 0.922  
Epoch 7: Loss 0.182, Accuracy 0.927  
Epoch 8: Loss 0.283, Accuracy 0.89  
Epoch 9: Loss 0.181, Accuracy 0.929  
Epoch 10: Loss 0.19, Accuracy 0.927  
Epoch 11: Loss 0.205, Accuracy 0.931  
Epoch 12: Loss 0.198, Accuracy 0.928  
Epoch 13: Loss 0.231, Accuracy 0.924  
Epoch 14: Loss 0.243, Accuracy 0.925  
Stop learning, no improvement for 3 epochs.
```

```
[83]: best_accuracy4, best_accuracy_loss4, losses4, accuracies4 =  
      ↪train(Gender_model_4,optimizer_4)
```

```
Epoch 1: Loss 0.221, Accuracy 0.909  
Epoch 2: Loss 0.193, Accuracy 0.923  
Epoch 3: Loss 0.194, Accuracy 0.921  
Epoch 4: Loss 0.186, Accuracy 0.927  
Epoch 5: Loss 0.184, Accuracy 0.93  
Epoch 6: Loss 0.178, Accuracy 0.931  
Epoch 7: Loss 0.205, Accuracy 0.926  
Epoch 8: Loss 0.202, Accuracy 0.932  
Epoch 9: Loss 0.238, Accuracy 0.92  
Epoch 10: Loss 0.236, Accuracy 0.929  
Epoch 11: Loss 0.292, Accuracy 0.929  
Stop learning, no improvement for 3 epochs.
```

```
[84]: best_accuracy5, best_accuracy_loss5, losses5, accuracies5 =  
      ↪train(Gender_model_5,optimizer_5)
```

```
Epoch 1: Loss 0.284, Accuracy 0.88  
Epoch 2: Loss 0.24, Accuracy 0.904  
Epoch 3: Loss 0.199, Accuracy 0.922  
Epoch 4: Loss 0.196, Accuracy 0.922  
Epoch 5: Loss 0.215, Accuracy 0.915  
Epoch 6: Loss 0.18, Accuracy 0.93  
Epoch 7: Loss 0.19, Accuracy 0.927  
Epoch 8: Loss 0.19, Accuracy 0.927
```

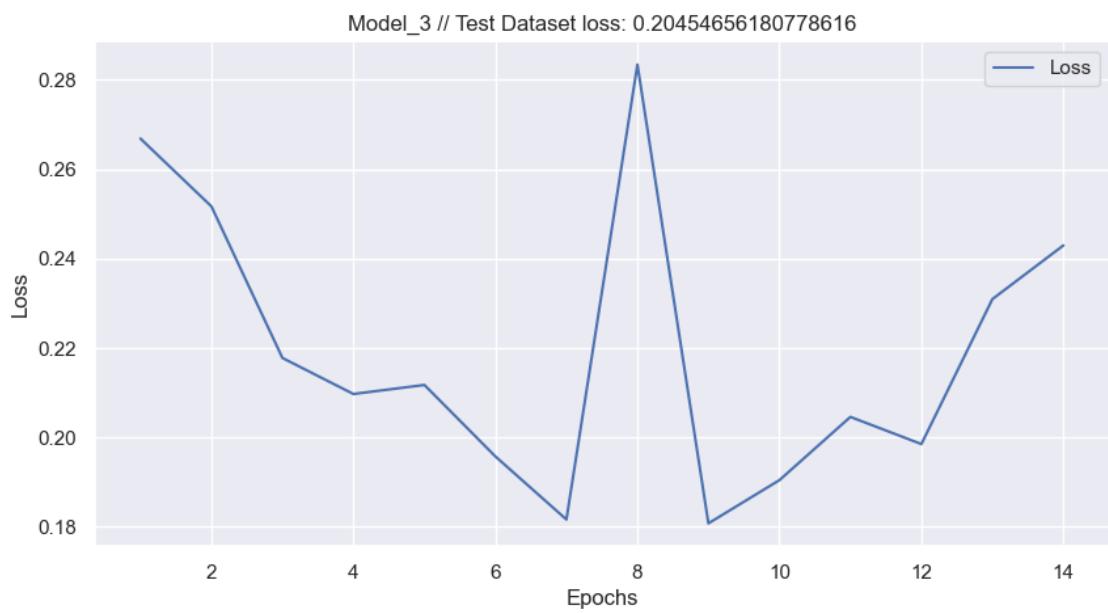
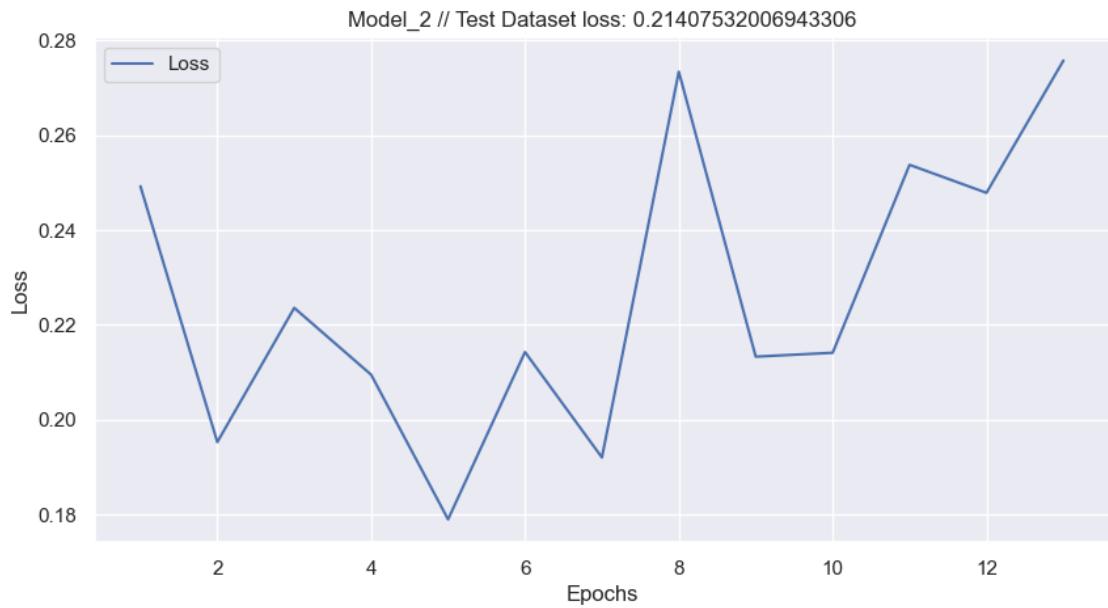
```
Epoch 9: Loss 0.18, Accuracy 0.932
Epoch 10: Loss 0.19, Accuracy 0.927
Epoch 11: Loss 0.214, Accuracy 0.922
Epoch 12: Loss 0.21, Accuracy 0.933
Epoch 13: Loss 0.203, Accuracy 0.928
Epoch 14: Loss 0.222, Accuracy 0.928
Epoch 15: Loss 0.243, Accuracy 0.919
Stop learning, no improvement for 3 epochs.
```

Wykresy zmiany straty i dokładności na zbiorze testowym dla każdego z modeli:

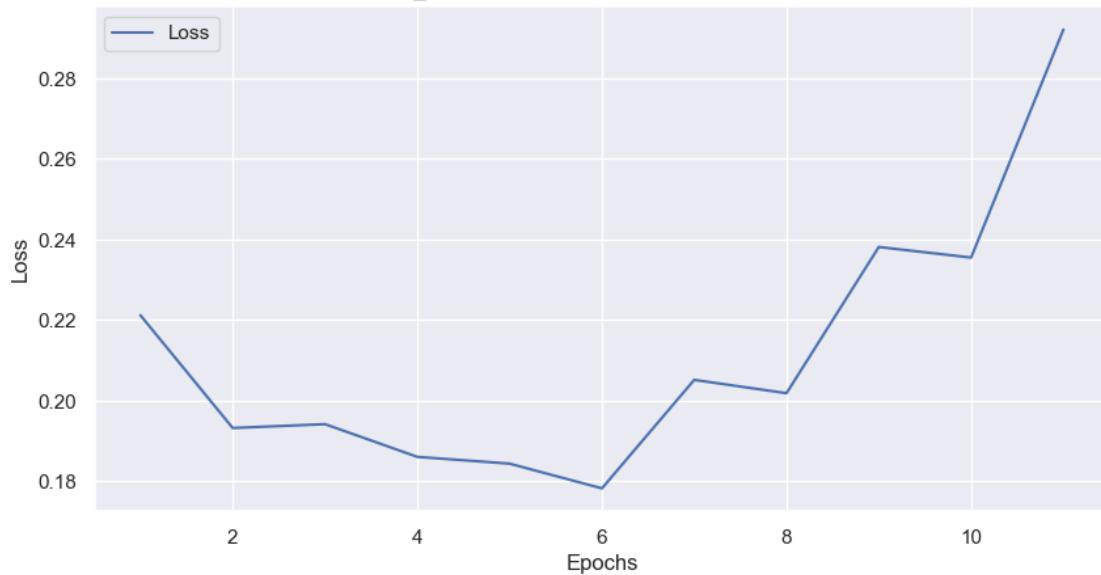
```
[87]: def loss_chart(losses,best_accuracy_loss,model_num):
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, len(losses) + 1), losses, label='Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title(f"Model_{model_num} // Test Dataset loss: {best_accuracy_loss}")
    plt.savefig(f"../Data/Loss_{model_num}.png")
    plt.show()

[88]: loss_chart(losses1,best_accuracy_loss1,1)
loss_chart(losses2,best_accuracy_loss2,2)
loss_chart(losses3,best_accuracy_loss3,3)
loss_chart(losses4,best_accuracy_loss4,4)
loss_chart(losses5,best_accuracy_loss5,5)
```

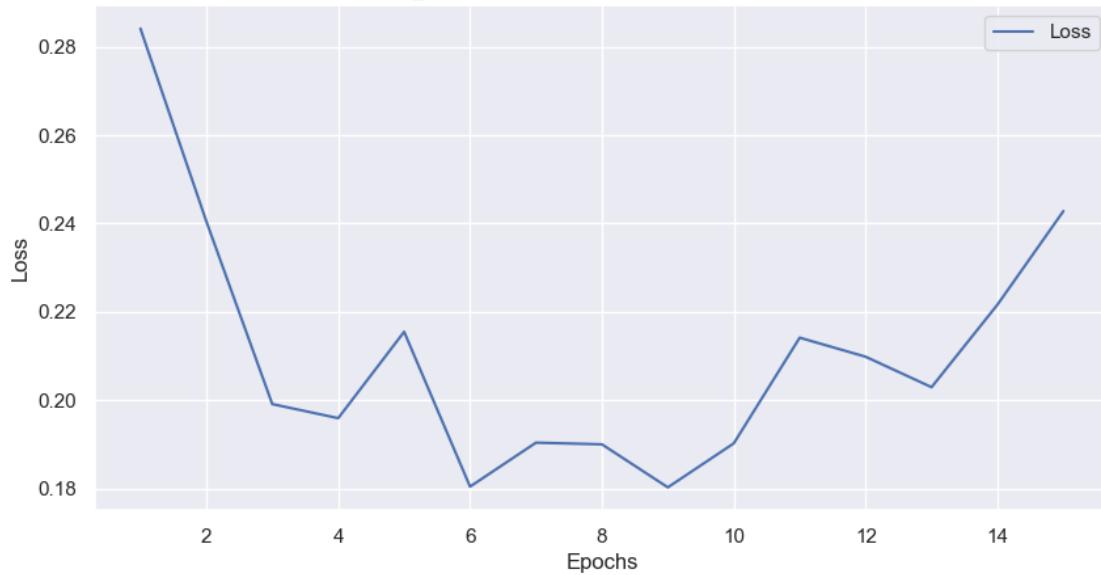




Model\_4 // Test Dataset loss: 0.2018577304031504



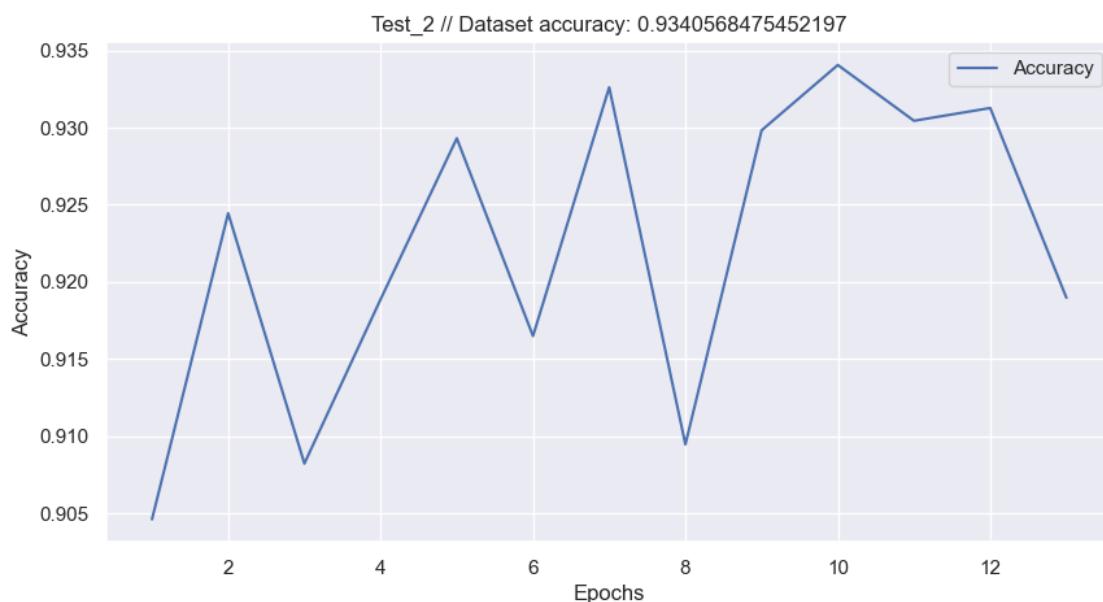
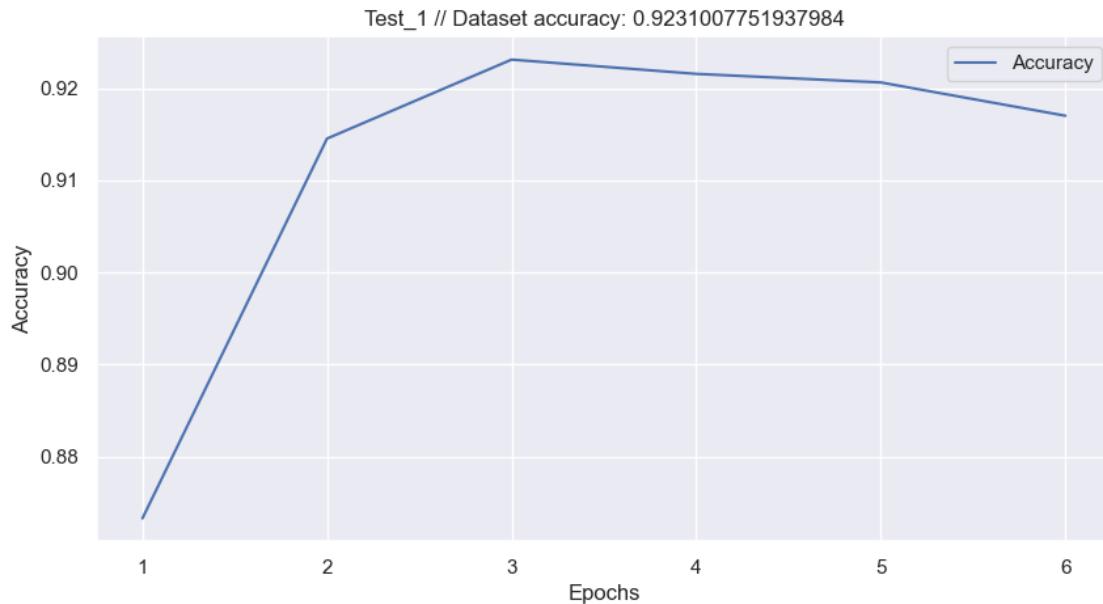
Model\_5 // Test Dataset loss: 0.2098041240855022



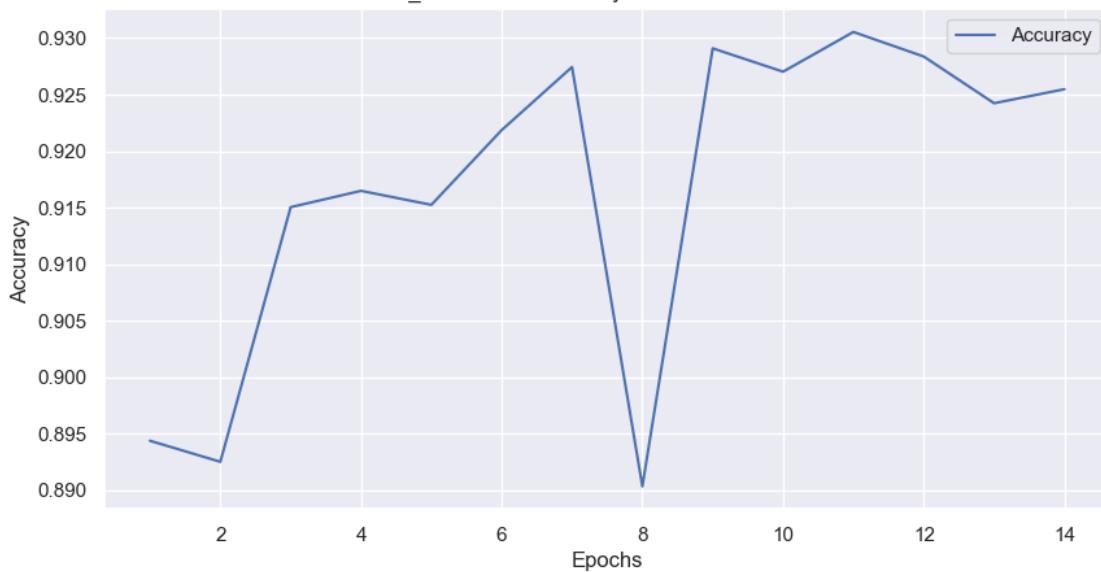
```
[89]: def accuracy_chart(accuracies,best_accuracy,model_num):
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, len(accuracies) + 1), accuracies, label='Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
plt.title(f"Test_{model_num} // Dataset accuracy: {best_accuracy}")
plt.savefig(f'../Data/Accuracy_{model_num}.png')
plt.show()
```

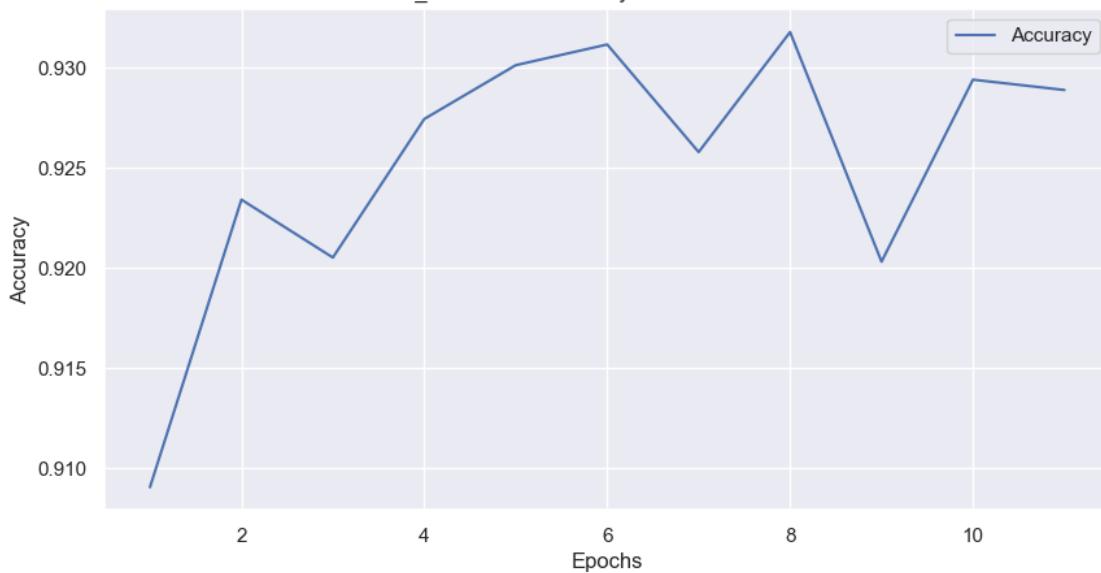
```
[90]: accuracy_chart(accuracies1,best_accuracy1,1)
accuracy_chart(accuracies2,best_accuracy2,2)
accuracy_chart(accuracies3,best_accuracy3,3)
accuracy_chart(accuracies4,best_accuracy4,4)
accuracy_chart(accuracies5,best_accuracy5,5)
```

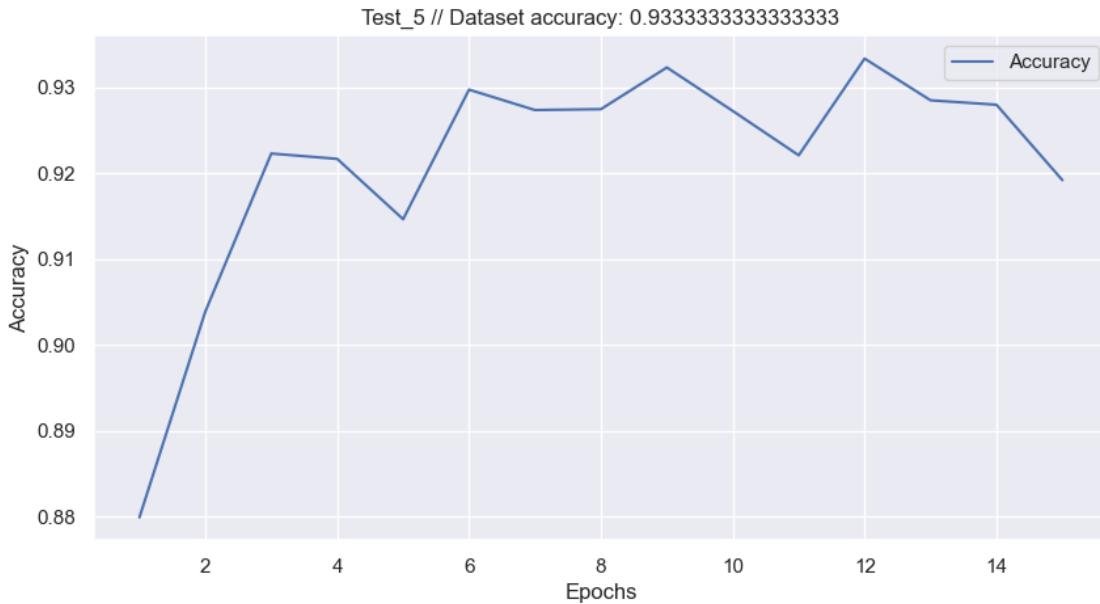


Test\_3 // Dataset accuracy: 0.9305426356589147



Test\_4 // Dataset accuracy: 0.931782945736434





Powyższe wykresy ukazują, że mimo zmiany architektur modelu wyniki dokładności na zbiorze testowym w wszystkich przypadkach są zbliżone. Oznacza to, że na dostarczonych danych model prawdopodobnie osiągnął swoją granicę wydajności (~93%) co dodatkowo potwierdza rosnącą wartość straty pod sam koniec uczenia (model zaczyna się przeuczać).

Zapisanie wag poszczególnych modeli w celu późniejszego wykorzystania:

```
[98]: torch.save({'weights': Gender_model_1.state_dict()}, '../Models/gender_model_1.  
        pth')  
torch.save({'weights': Gender_model_2.state_dict()}, '../Models/gender_model_2.  
        pth')  
torch.save({'weights': Gender_model_3.state_dict()}, '../Models/gender_model_3.  
        pth')  
torch.save({'weights': Gender_model_4.state_dict()}, '../Models/gender_model_4.  
        pth')  
torch.save({'weights': Gender_model_5.state_dict()}, '../Models/gender_model_5.  
        pth')
```

Wczytanie wag modeli:

```
[107]: models = [Gender_model_1, Gender_model_2, Gender_model_3, Gender_model_4,  
             ↪Gender_model_5]  
  
for i, model in enumerate(models, start=1):  
    model.load_state_dict(torch.load(f"../Models/gender_model_{i}.  
        pth")['weights'])  
    model.eval()
```

Macierze konfuzji dla poszczególnych modeli:

```
[118]: from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion_matrix(model, testloader, ax, title):
    model.eval()
    true_labels = []
    pred_labels = []

    for images, labels in testloader:
        output = model(images)
        predicted = (output > 0.5).float()
        true_labels.extend(labels.tolist())
        pred_labels.extend(predicted.tolist())

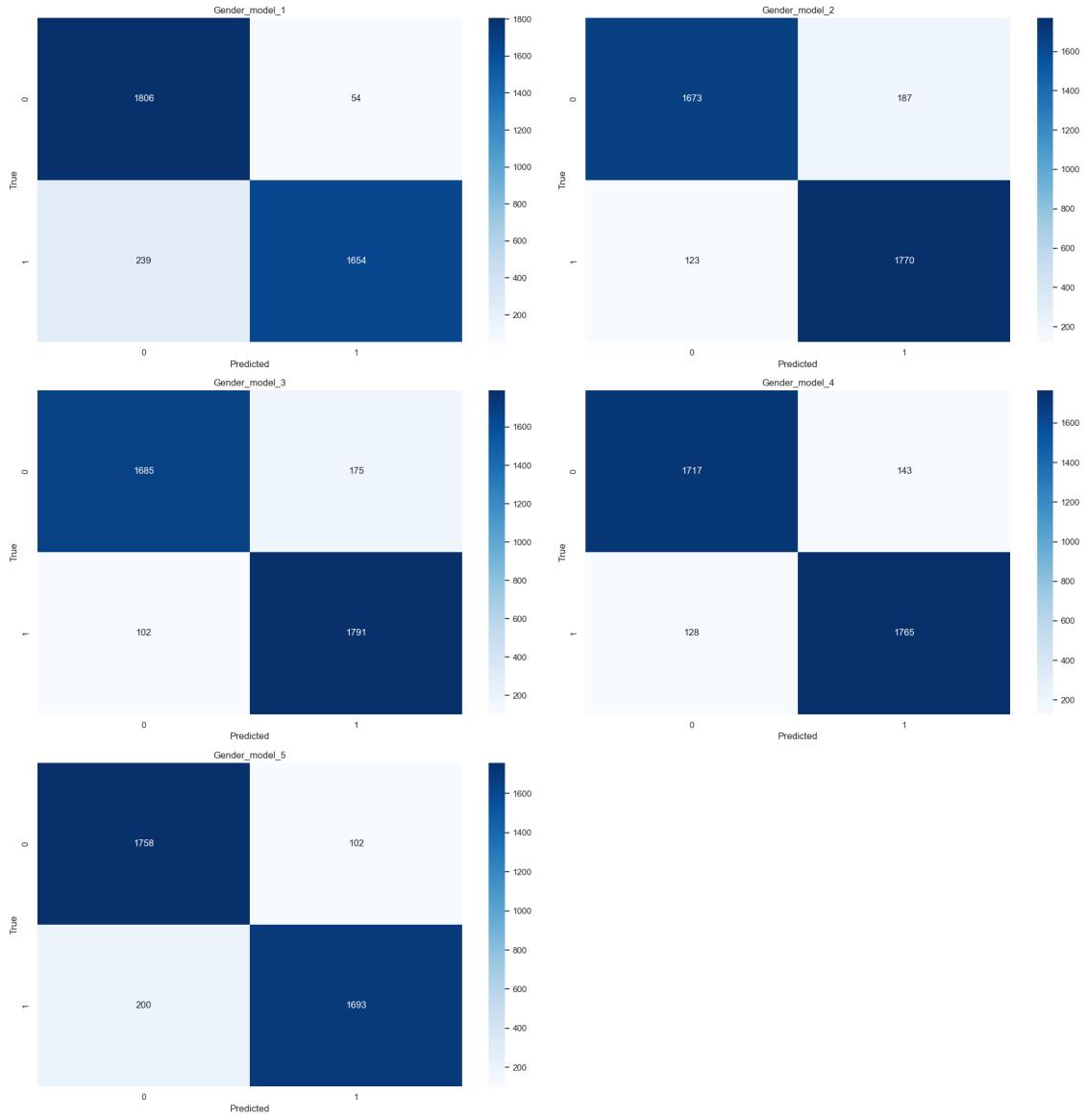
    cm = confusion_matrix(true_labels, pred_labels)
    sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
    ax.set_title(title)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')

fig, axs = plt.subplots(3, 2, figsize=(20, 20))

plot_confusion_matrix(Gender_model_1, testloader, axs[0,0], 'Gender_model_1')
plot_confusion_matrix(Gender_model_2, testloader, axs[0,1], 'Gender_model_2')
plot_confusion_matrix(Gender_model_3, testloader, axs[1,0], 'Gender_model_3')
plot_confusion_matrix(Gender_model_4, testloader, axs[1,1], 'Gender_model_4')
plot_confusion_matrix(Gender_model_5, testloader, axs[2,0], 'Gender_model_5')

fig.delaxes(axs[2,1])

plt.tight_layout()
plt.show()
```



Sprawdzenie predykcji modelów na 10 losowych próbkach:

```
[121]: import random
import matplotlib.pyplot as plt
import torch

def check_model(Gender_model):
    model = Gender_model
    model.eval()

    random_samples = random.sample(list(df["Image"]), 10)
```

```

fig, axs = plt.subplots(2, 5, figsize=(15, 6))

for i, image in enumerate(random_samples):
    image_tensor = torch.from_numpy(image)
    image_tensor = image_tensor.view(3, 100, 100) / 255.0

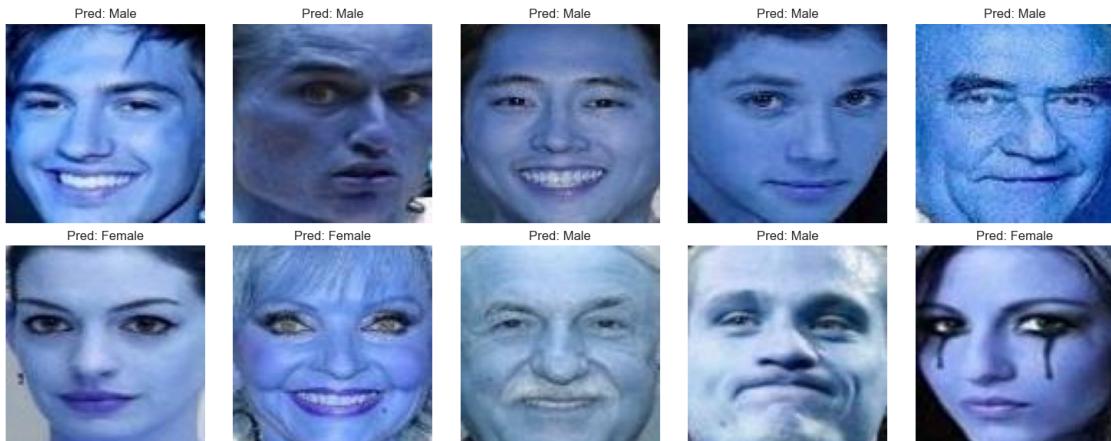
    output = model(image_tensor.unsqueeze(0))
    predicted_gender = 'Male' if output.item() < 0.5 else 'Female'

    ax = axs[i // 5, i % 5]
    ax.imshow(image)
    ax.set_title(f'Pred: {predicted_gender}')
    ax.axis('off')

plt.tight_layout()
plt.show()

check_model(Gender_model_1)
check_model(Gender_model_2)
check_model(Gender_model_3)
check_model(Gender_model_4)
check_model(Gender_model_5)

```







```
[123]: from sklearn.metrics import accuracy_score, precision_score

def calculate_metrics(model, testloader):
    model.eval()
    true_labels = []
    pred_labels = []

    for images, labels in testloader:
        output = model(images)
        predicted = (output > 0.5).float()
        true_labels.extend(labels.tolist())
        pred_labels.extend(predicted.tolist())

    accuracy = accuracy_score(true_labels, pred_labels)
    precision_male = precision_score(true_labels, pred_labels, pos_label=1) # ↪ Precision for 'Male'
    precision_female = precision_score(true_labels, pred_labels, pos_label=0) # ↪ Precision for 'Female'

    return accuracy, precision_male, precision_female

models = [Gender_model_1, Gender_model_2, Gender_model_3, Gender_model_4, ↪ Gender_model_5]
model_names = ['Gender_model_1', 'Gender_model_2', 'Gender_model_3', ↪ 'Gender_model_4', 'Gender_model_5']

for model, name in zip(models, model_names):
    accuracy, precision_male, precision_female = calculate_metrics(model, ↪ testloader)
    print(f'{name}: Accuracy = {accuracy:.2f}, Precision (Male) = {precision_male:.2f}, Precision (Female) = {precision_female:.2f}' )
```

```
Gender_model_1: Accuracy = 0.92, Precision (Male) = 0.97, Precision (Female) =
0.88
Gender_model_2: Accuracy = 0.92, Precision (Male) = 0.90, Precision (Female) =
0.93
Gender_model_3: Accuracy = 0.93, Precision (Male) = 0.91, Precision (Female) =
0.94
Gender_model_4: Accuracy = 0.93, Precision (Male) = 0.93, Precision (Female) =
0.93
Gender_model_5: Accuracy = 0.92, Precision (Male) = 0.94, Precision (Female) =
0.90
```

Gender\_model\_1: jest najbardziej precyzyjny w przewidywaniu mężczyzn, ale ma najwyższą precyzję dla kobiet. Gender\_model\_2: ma najwyższą precyzję dla mężczyzn, ale dobrą precyzję dla kobiet. Gender\_model\_3: ma najwyższą dokładność i dobrą precyzję dla obu płci. Gender\_model\_4: jest najbardziej zrównoważony, z równą precyzją dla mężczyzn i kobiet. Gender\_model\_5: ma dobrą precyzję dla mężczyzn, ale nieco niższą precyzję dla kobiet.

Zapisanie notatnik jako pliku PDF:

```
[204]: !jupyter nbconvert --to pdf --output-dir='../../PDF' Gender_prediction.ipynb
```

```
[NbConvertApp] Converting notebook Gender_prediction.ipynb to pdf
/usr/local/lib/python3.11/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (3.1.2).
Your version must be at least (1.12.1) but less than (3.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts...
    check_pandoc_version()
[NbConvertApp] Support files will be in Gender_prediction_files/
[NbConvertApp] Making directory ./Gender_prediction_files
[NbConvertApp] Writing 139541 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 4679048 bytes to ../../PDF/Gender_prediction.pdf
```

Ukończono proces treningu :)

```
[125]: import os
import platform

if platform.system() == 'Darwin':  # macOS
    os.system('say "Model training completed"')
elif platform.system() == 'Windows':  # Windows
```

```

os.system('PowerShell -Command "Add-Type -TypeDefinition \'public class [Speech] { public static void Speak(string text) { new System.Speech.Synthesis.SpeechSynthesizer().Speak(text); } }\' ; [Speech]::Speak(\'Model training completed\')")
else: # Linux/UNIX
    os.system('echo "Model training completed" | espeak')

```

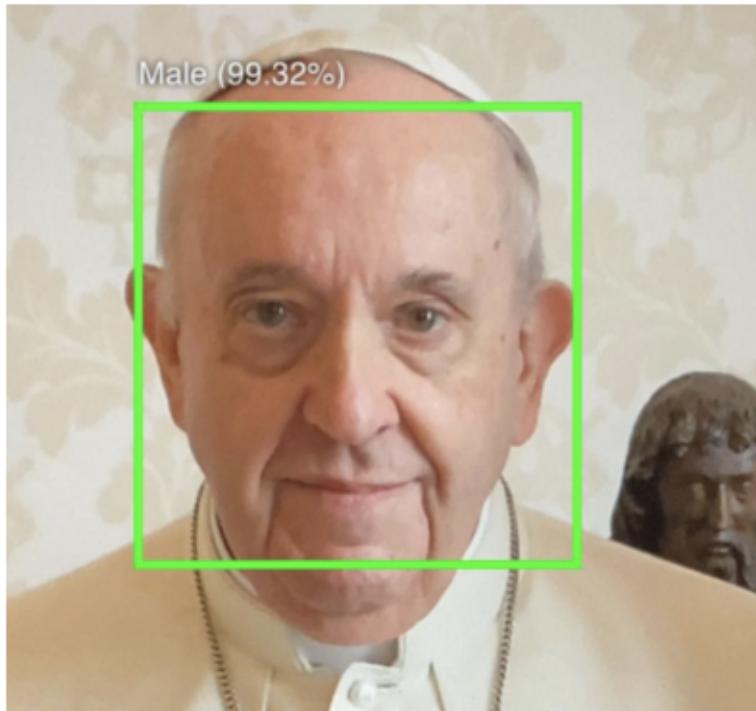
Działanie aplikacji mobilnej (iOS) - wykorzystującej wytrenowany model nr.2

link github: - [https://github.com/przemek890/Gender\\_prediction/tree/master/IOS](https://github.com/przemek890/Gender_prediction/tree/master/IOS)

```
[132]: import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/men.png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
print("https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FPope&psig=A0vVaw20iu3pYaI4nEx19UxS4U-W&ust=1712182259187000&source=images&cd=v

```

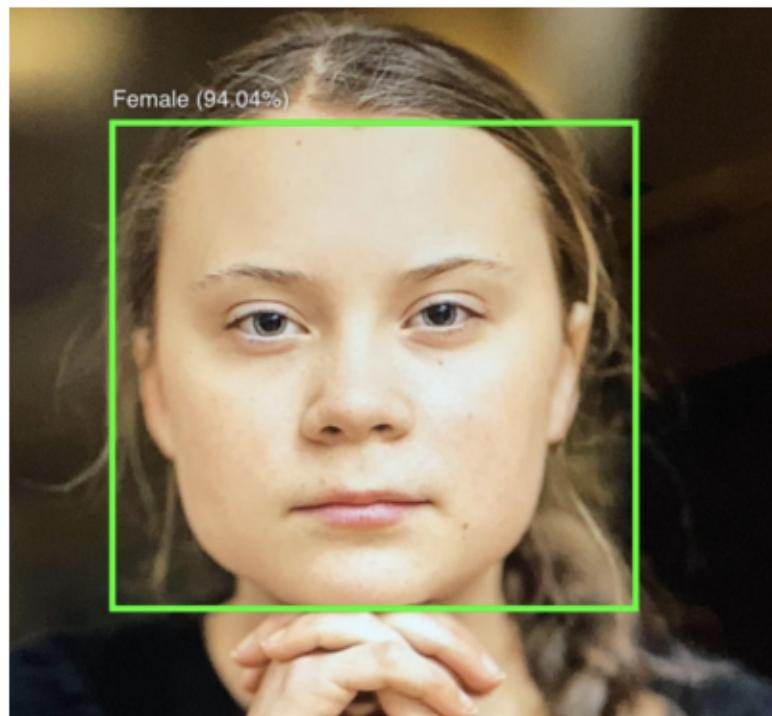


<https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FPope&psig=A0vVaw20iu3pYaI4nEx19UxS4U->

W&ust=1712182259187000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCKCGnI  
zGpIUDFQAAAAAdAAAAABAE

```
[133]: import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

img = mpimg.imread(os.path.join(os.getcwd(), "../Data/init/women.png"))
plt.imshow(img)
plt.axis('off')
plt.grid(False)
plt.show()
print("https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.wysokieobcasy.
    ↪pl%2Fakcje-specjalne%2F7%2C168569%2C24832423%2Cgreta-thunberg-chce-zebyscie-czuli-strach-kt
    ↪oryja-czuje.html&psig=A0vVaw0ruvK6bg87wnCoCCmxugAN&ust=1712181626708000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCNjKqd_DpIUDFQAAAAAdAAAAABAE")
```



https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.wysokieobcasy.pl%2Fakcje-
specjalne%2F7%2C168569%2C24832423%2Cgreta-thunberg-chce-zebyscie-czuli-strach-
ktryja-czuje.html&psig=A0vVaw0ruvK6bg87wnCoCCmxugAN&ust=1712181626708000&sourc
e=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCNjKqd\_DpIUDFQAAAAAdAAAAABAE