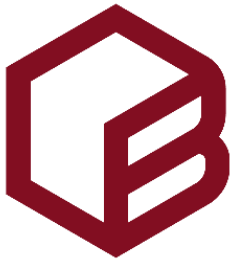
Sztuczna Inteligencja

Projekt

2023



POLITECHNIKA
BYDGOSKA
im. Jana i Jędrzeja Śniadeckich

Łukasz Rydz 118849

Michał Pawlak 118839

Informatyka Stosowana, grupa 3, semestr 4, rok 2023

1.Dane projektu

a. **Temat: System detekcji płci.**

b. **Język programowania: Python**

c. **Cel: Głównym celem projektu jest stworzenie modelu uczenia maszynowego, który będzie w stanie rozpoznawać płeć na podstawie przetwarzanych obrazów. Do tego celu wykorzystane zostaną biblioteki TensorFlow i Keras.**

d. **Model uczenia maszynowego: W projekcie używany jest model sekwencyjny, który będzie zawierał warstwy konwolucyjne, warstwy poolingowe oraz warstwy gęste. Dodatkowo, wykorzystamy techniki augmentacji danych, aby zwiększyć różnorodność zbioru uczącego.**

e. **Proces uczenia: Proces uczenia modelu będzie składał się z określonej liczby epok, podczas których będziemy przetwarzać paczki obrazów. W trakcie treningu modelu będziemy także monitorować metryki, takie jak dokładność (accuracy) modelu na zbiorze treningowym i walidacyjnym.**

f. **Zastosowanie w praktyce: Po zakończeniu procesu uczenia, zapiszemy model na dysku w celu dalszego wykorzystania. Następnie aplikacja będzie pobierać klatki z kamery, przetwarzać je w celu wykrycia twarzy, a następnie na podstawie zastosowanego modelu określać płeć.**

2.Lista zależności

- TensorFlow:
  Zastosowanie: TensorFlow jest wykorzystywany do tworzenia i trenowania modeli uczenia maszynowego, w tym sieci neuronowych. Oferuje narzędzia do definiowania struktury modelu, obliczeń na tensorach, automatycznego różniczkowania i optymalizacji modelu.

  Instalacja: pip install tensorflow

- Kerass:
  Zastosowanie: Keras jest wysokopoziomowym interfejsem programistycznym (API) dla budowania sieci neuronowych. W tym projekcie Keras jest używany do definiowania architektury modelu, dodawania warstw, kompilacji modelu i trenowania.

  Instalacja: pip install keras

- cikit-learn:
  Zastosowanie: scikit-learn jest biblioteką do uczenia maszynowego w języku Python. W tym projekcie scikit-learn jest wykorzystywany do podziału danych na zbiór treningowy i testowy za pomocą funkcji train_test_split.

  Instalacja: pip install scikit-learn

- matplotlib:
  Zastosowanie: matplotlib jest biblioteką do wizualizacji danych w języku Python. W projekcie używany jest do generowania wykresu historii procesu trenowania modelu, przedstawiającego zmiany w funkcji straty (loss) i dokładności (accuracy) w kolejnych epokach.

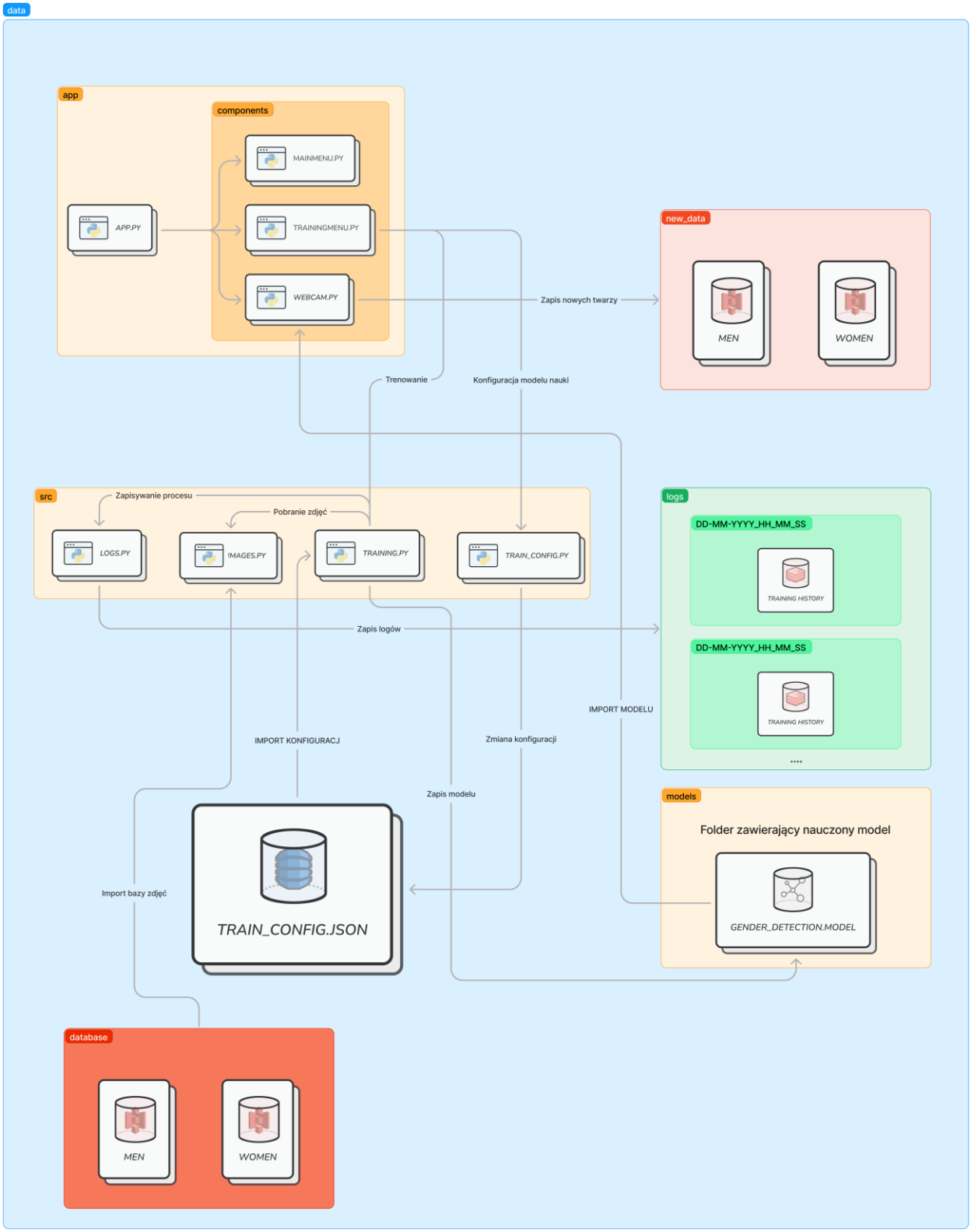
  Instalacja: pip install matplotlib

- numpy:
  Zastosowanie: numpy jest biblioteką do obliczeń naukowych w języku Python. W projekcie jest używany do przetwarzania obrazów poprzez konwersję ich na tablicy numpy, normalizację wartości pikseli oraz manipulację danymi numerycznymi.

  Instalacja: pip install numpy
- OpenCV:
  Zastosowanie: OpenCV to biblioteka do przetwarzania obrazów i analizy wizualnej. W tym projekcie OpenCV jest wykorzystywany do wczytania obrazów, zmiany ich rozmiaru oraz konwersji na tablicy numpy.

  Instalacja: pip install opencv-python

3.Struktura Projektu



4.Opis programów

A. Klasa Training

a. Zmienne klasy:

```
images = Images()                                   # Obiekt klasy Images  (obrazy)
logs = Logs()                                       # Obiekt klasy Logs    (logi)
trainParameters = None                              # Obiekt klasy TrainParameters (parametry treningu)
CLASSIFICATION_MODEL = None                         # Model klasyfikacji
```

1. Program pobiera obiekt klasy Images który zawiera metody pobierania i obrabiania danych treningowych.
2. Następnie pobiera obiekt klasy Logs który zawiera metody tworzące oraz edytujące logi procesu nauki.
3. Następnie inicjalizuje zmienne potrzebne w późniejszym czasie.

b. Inicjalizacja klasy:

```
def __init__(self, conf):
    TOTAL_TRAINING_DURATION = time.time()

    # Pobranie parametrów treningu
    self.trainParameters = conf

    # Zapisanie parametrów treningu do pliku
    self.logs.add_value("Status", "Unfinished")
    self.logs.add_value("Training parameters: ", self.trainParameters.get_parameters())


    self.__prepareImages__()                                                    # Przygotowanie obrazów
    self.__buildModel__()                                                       # Zbudowanie modelu

    TIMER = time.time()
    self.__trainModel__()                                                       # Trening modelu

    self.logs.add_value("Training duration: ", str(time.time() - TIMER) + "s")  # Zapisanie czasu treningu
    self.logs.add_value("Total duration: ", str(time.time() - TOTAL_TRAINING_DURATION) + "s")   # Zapisanie całkowitego czasu treningu

    print(colored("Training finished", 'green'))
    input(colored("Press Enter to continue...", 'magenta'))

    self.logs.add_value("Status", "Finished")
    K.clear_session()                                                           # Wyczyszczenie sesji
```

1. Zainicjalizowanie pomiaru czasu treningu.
2. Zapisanie pobranej konfiguracji do zmiennej w klasie.
3. Rozpoczęcie zapisywania logów treningu.
4. Program rozpoczyna przygotowywanie danych (wczytanie zdjęć, obróbka, podział na moduły)
5. Rozpoczęcie treningu
6. Zapisanie logów treningu

c. Metoda __buildModel__(self):

```python
def __buildModel__(self):
        # Model sekwencyjny
        model = Sequential([
            # Warstwa konwolucyjna 2D z 32 filtrami o rozmiarze 3x3, z zachowaniem wymiarów, wejściowy kształt obrazu
            Conv2D(32, (3, 3), padding="same", input_shape=self.images.imageProperties),
            Activation("relu"),                                                    # Funkcja aktywacji ReLU
            BatchNormalization(axis=self.images.chanDim),                          # Normalizacja wsadowa na kanale
            MaxPooling2D(pool_size=(3, 3)),                                        # Warstwa poolingowa typu maksymalnego o rozmiarze 3x3
            Dropout(0.25),                                                         # Warstwa dropoutu o współczynniku 0.25

            Conv2D(64, (3, 3), padding="same"),
            Activation("relu"),
            BatchNormalization(axis=self.images.chanDim),                          # Normalizacja wsadowa na kanale

            Conv2D(64, (3, 3), padding="same"),
            Activation("relu"),
            BatchNormalization(axis=self.images.chanDim),
            MaxPooling2D(pool_size=(2, 2)),                                        # Warstwa poolingowa typu maksymalnego o rozmiarze 2x2
            Dropout(0.25),                                                         # Warstwa dropoutu o współczynniku 0.25

            Conv2D(128, (3, 3), padding="same"),
            Activation("relu"),
            BatchNormalization(axis=self.images.chanDim),

            Conv2D(128, (3, 3), padding="same"),
            Activation("relu"),
            BatchNormalization(axis=self.images.chanDim),
            MaxPooling2D(pool_size=(2, 2)),
            Dropout(0.25),

            Flatten(),                                                             # Warstwa spłaszczająca
            Dense(1024),                                                           # W pełni połączona warstwa z 1024 neuronami
            Activation("relu"),
            BatchNormalization(),
            Dropout(0.5),

            Dense(2),                                                              # W pełni połączona warstwa z 2 neuronami (klasyfikacja binarna (mężczyzna/kobieta))
            Activation("sigmoid")                                                  # Funkcja aktywacji sigmoidalna
        ])

        self.CLASSIFICATION_MODEL = model
```

1. Metoda ta buduje model sekwencyjny który zostanie użyty w procesie nauki.
2. Można go zmieniać dowolnie i testować wyniki.

d. Metoda __trainModel__(self):

```python
def __trainModel__(self):
        print(colored("\n\n\nRozpoczęcie treningu...", 'yellow'))

        # Inicjalizacja optymalizatora
        opt = tf.keras.optimizers.legacy.Adam(learning_rate=self.trainParameters.learning_rate,
                                              decay=self.trainParameters.learning_rate/self.trainParameters.epochs)

        # Kompilacja modelu
        self.CLASSIFICATION_MODEL.compile(loss="binary_crossentropy", optimizer=opt, metrics=["acc"])

        # Inicjalizacja generatora danych treningowych
        train_data_generator = self.images.IMAGE_AUGMENTER.flow(self.images.train_images, self.images.train_labels,
                                                                batch_size=self.trainParameters.batch_size)

        # Inicjalizacja generatora zapisującego wyniki
        csv_logger = CSVLogger(os.path.join(self.logs.get_folder_path(), "epochs_history.csv"))

        # Train the model
        TRAINING_HISTORY = self.CLASSIFICATION_MODEL.fit(
            train_data_generator,                                                  # Generator danych treningowych
            validation_data=(self.images.test_images, self.images.test_labels),   # Dane do walidacji modelu
            steps_per_epoch=len(self.images.train_images) // self.trainParameters.batch_size,  # Liczba kroków (mini-batches) na epokę
            epochs=self.trainParameters.epochs,                                    # Liczba epok
            verbose=1,                                                             # Poziom wypisywanych informacji
            callbacks=[csv_logger]                                                 # Lista callbacków, w tym przypadku CSVLogger
        )

        self.logs.savePlot(self.TRAINING_HISTORY, self.trainParameters)

        # Zapisanie modelu
        model_path = './model/gender_detection.model'
        self.CLASSIFICATION_MODEL.save(model_path)
```

1. Metoda ta odpowiedzialna jest za przeprowadzenie treningu modelu.

2. Najpierw zostaje utworzony obiekt optymalizatora Adam.
3. Później wywołana zostaje funkcja compile() która kompiluje model klasyfikacji.
4. Następnie tworzy generator danych treningowych.
5. Program rozpoczyna trening modelu funkcją self.CLASSIFICATION_MODEL.fit()
6. Ostatecznie zapisuje przebieg nauki
    a. Plik .csv – surowe dane
    b. Plik .png – wykres przebiegu nauki

B. Klasa train_config.py

a. Inicjalizacja

```python
def __init__(self, config_name):
    self.config_name = './' + config_name
    self.data = self.load_config()

    self.train_parameters = self.data['trainParameters']

    self.epochs = self.train_parameters['epochs']
    self.learning_rate = self.train_parameters['learningRate']
    self.batch_size = self.train_parameters['batchSize']
    self.width = self.train_parameters['width']
    self.height = self.train_parameters['height']
    self.channels = self.train_parameters['channels']
```

1. Klasa wczytuje plik konfiguracyjny.
2. Pobiera z niego parametry procesu nauki.

b. Metoda load_config oraz save_config

```python
def load_config(self):
    with open(self.config_name, 'r') as file:
        return json.load(file)

def save_config(self):
    with open(self.config_name, 'w') as file:
        json.dump(self.data, file, indent=4)
```

1. Metoda load_config ładuje plik csv
2. Metoda save_config zapisuje plik csv

c. SETTERY I GETTERY

```python
def set_epochs(self, epochs):
        self.train_parameters['epochs'] = epochs
        self.epochs = epochs

    def set_learning_rate(self, learning_rate):
        self.train_parameters['learningRate'] = learning_rate
        self.learning_rate = learning_rate

    def set_batch_size(self, batch_size):
        self.train_parameters['batchSize'] = batch_size
        self.batch_size = batch_size

    def set_width(self, width):
        self.train_parameters['width'] = width
        self.width = width

    def set_height(self, height):
        self.train_parameters['height'] = height
        self.height = height

    def set_channels(self, channels):
        self.train_parameters['channels'] = channels
        self.channels = channels
```

```python
def get_epochs(self):
    return self.epochs

def get_learning_rate(self):
    return self.learning_rate

def get_batch_size(self):
    return self.batch_size

def get_width(self):
    return self.width

def get_height(self):
    return self.height

def get_channels(self):
    return self.channels
```

d. Metoda isInRange()

```python
def isInRange(self, value, min, max):
    if value >= min and value <= max:
        return True
    else:
        print(colored("Value must be between " + str(min) + " and " + str(max), 'red'))
        return False
```

1. Sprawdza podczas próby zmiany parametru czy podana wartości mieści się w założonych granicach normy.

e. Metoda get_parameters()

```python
def get_parameters(self):
        return self.train_parameters
```

1. Zwraca obiekt zawierający wszystkie parametry

C. Klasa Logs:

Inicjalizacja

```python
def __init__(self):
    # Create folder with actual data and time
    self.folder_name = datetime.now().strftime("%d-%m-%Y_%H-%M-%S")
    self.folder_path = os.path.join(os.getcwd(), "logs", self.folder_name)

    # Create folder
    os.mkdir(self.folder_path)

    # Create log file path
    self.json_file_path = os.path.join(self.folder_path, "log.json")

    self.jsonLog = {}
```

1. Najpierw zostaje utworzony folder który będzie przechowywał wszystkie nowe logi.

Inne metody:

```python
def get_folder_path(self):
    return self.folder_path

def get_json_file_path(self):
    return self.json_file_path

def add_value(self, key, value):
    if os.path.isfile(self.json_file_path):
        with open(self.json_file_path, 'r') as file:
            self.jsonLog = json.load(file)

    self.jsonLog[key] = value

    # Zapisywanie słownika do pliku .json
    with open(self.json_file_path, 'w') as file:
        json.dump(self.jsonLog, file)
```

1. Get_folder_path() – zwraca ścieżkę logów (folderu)
2. get_json_file_path() – zwraca ścieżkę logów (.json)
3. add_value() - Dodaje klucz: wartość

Metoda savePlot()

```python
def savePlot(self, TRAINING_HISTORY, trainParameters):
    plt.style.use("seaborn")
    plt.figure(figsize=(10, 6))
    epochs = range(1, trainParameters.epochs + 1)

    plt.plot(epochs, TRAINING_HISTORY.history["loss"], label="Training Loss", color="blue", linestyle="-")
    plt.plot(epochs, TRAINING_HISTORY.history["val_loss"], label="Validation Loss", color="red", linestyle="--")
    plt.plot(epochs, TRAINING_HISTORY.history["acc"], label="Training Accuracy", color="green", linestyle="-")
    plt.plot(epochs, TRAINING_HISTORY.history["val_acc"], label="Validation Accuracy", color="orange", linestyle="--")

    # Zaznacz punkty szczytowe dla val_acc
    val_acc = TRAINING_HISTORY.history["val_acc"]
    max_val_acc = max(val_acc)
    max_val_acc_index = val_acc.index(max_val_acc)
    plt.scatter(max_val_acc_index + 1, max_val_acc, color='red', label=f"Peak: {max_val_acc:.4f}")
    plt.text(max_val_acc_index + 1, max_val_acc, f"({max_val_acc_index + 1}, {max_val_acc:.4f})", ha='right', va='bottom')

    plt.title("Model Training History")
    plt.xlabel("Epoch")
    plt.ylabel("Loss/Accuracy")
    plt.legend(loc="lower right")

    # Zapisz wykres do folderu
    plot_path = os.path.join(self.folder_path, "plot.png")
    plt.savefig(plot_path)
    plt.close()
```

1. Odpowiedzialna za sporządzenie wykresu na podstawie danych z procesu nauki.

D. Klasa Images:

Zmienne:

```
logs = None                                 # Obiekt klasy Logs

IMAGE_PATHS = []                            # Tablica z ścieżkami do obrazów
IMAGE_DATA = []                             # Lista z obrazami które zostaną przetworzone
IMAGE_LABELS = []                           # Lista z etykietami dla przetworzonych obrazów
AUGUMENTED_IMAGE_DATA = []                  # Lista z przetworzonymi obrazami

train_images = []                           # Obrazy do uczenia
train_labels = []                           # Etykiety obrazów do uczenia
test_images = []                            # Obrazy do testowania
test_labels = []                            # Etykiety obrazów do testowania

chanDims = 0                                # Oś kanałów
imageProperties = ()                        # Wymiary obrazu

trainParameters = None                      # Parametry uczenia
```

Inicjalizacja:

```
def start(self, trainParameters, logs):
    self.logs = logs
    self.trainParameters = trainParameters
    TOTAL_IMG_PREPARE_DURATION = time.time()

    TIMER = time.time()
    self.__loadImages__()
    self.logs.add_value("Load images duration", str(time.time() - TIMER) + "s")

    TIMER = time.time()
    self.__convertImages__()
    self.logs.add_value("Convert images duration", str(time.time() - TIMER) + "s")

    self.__imageAugmentation__()

    self.logs.add_value("Number of images: ", len(self.IMAGE_PATHS))
    self.logs.add_value("Total image prepare duration", str(time.time() - TOTAL_IMG_PREPARE_DURATION) + "s")
```

1. Inicjalizacja logów
2. Inicjalizacja parametrów treningowych
3. Inicjalizacja pomiaru czasu przygotowania danych
4. Rozpoczęcie ładowania obrazów
5. Rozpoczęcie konwertowania obrazów
6. Zapisanie danych do logów

Metoda ładowania danych __loadImages__():

```
def __loadImages__(self):
    print(colored("Rozpoczęcie ładowania zdjęć...", "yellow"))
    for root, dirs, files in os.walk('./database'):
        for file in files:
            file_path = os.path.join(root, file)
            if not os.path.isdir(file_path):
                self.IMAGE_PATHS.append(file_path)
```

Metoda konwertowania danych __convertImages__()

```python
def __convertImages__(self):

        for img in tqdm(self.IMAGE_PATHS, desc=colored("Konwertowanie zdjęć", "magenta")):
            # Wczytanie obrazu
            image = cv2.imread(img)
            # Zmiana rozmiaru obrazu na wymiary zdefiniowane
            image = cv2.resize(image, (self.trainParameters.width, self.trainParameters.height))
            # Konwertuje obraz na tablicę i dodaje go do listy obrazów
            image = img_to_array(image)
            self.IMAGE_DATA.append(image)

            # Etykietowanie obrazów 1 == kobieta, 0 == mężczyzna
            label = img.split(os.path.sep)[-2]
            if label == "women":
                label = 1
            else:
                label = 0
            self.IMAGE_LABELS.append([label])

        # Normalizacja która ma na celu skalowanie wartości pikseli do zakresu od 0 do 1, co ułatwia uczenie modelu.
        self.IMAGE_DATA = np.array(self.IMAGE_DATA, dtype="float") / 255.0
        # Konwersja etykiet do tablicy numpy
        self.IMAGE_LABELS = np.array(self.IMAGE_LABELS)

        # Podział danych na zbiór treningowy i testowy 20% dla testów, 80% dla treningu
        (self.train_images, self.test_images, self.train_labels, self.test_labels) = train_test_split(self.IMAGE_DATA, self.IMAGE_LABELS, test_size=0.2, random_state=42)

        # Konwersja etykiet do postaci kategorycznej
        self.train_labels = to_categorical(self.train_labels, num_classes=2)
        self.test_labels = to_categorical(self.test_labels, num_classes=2)

        # Ustawienie wymiarów wejściowych oraz osi kanałów
        if backend.image_data_format() == "channels_first":
            self.imageProperties = (self.trainParameters.channels,
                                    self.trainParameters.height,
                                    self.trainParameters.width)
            self.chanDim = 1
        else:
            self.imageProperties = (self.trainParameters.height,
                                    self.trainParameters.width,
                                    self.trainParameters.channels)
            self.chanDim = -1
```

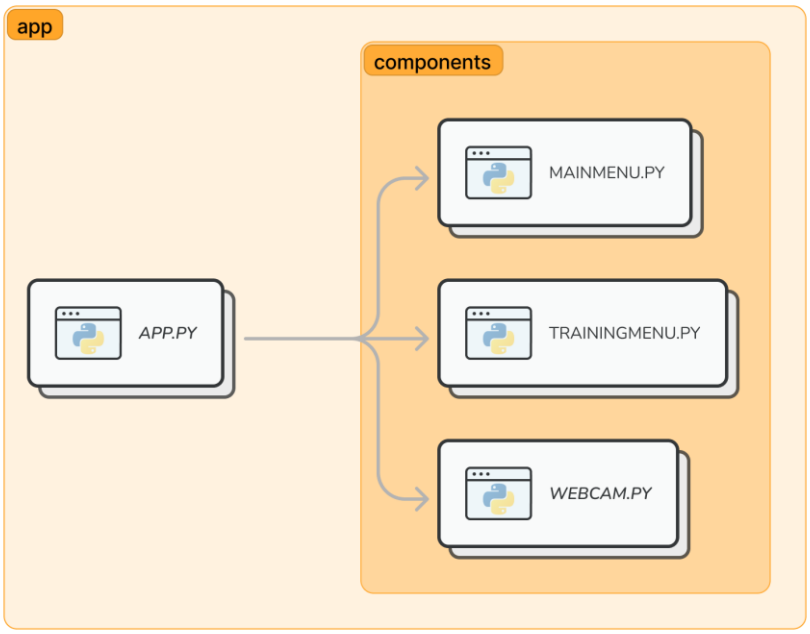
1. Najpierw wczytuje I przetwarza obrazy z ścieżek:
    a. Zmienia ich wielkość na podstawie parametrów treningu
    b. Konwertuje obrazy na tablicę
    c. Przypisuje etykiety do danych w tablicy [‘women’, ‘men’]
2. Następnie normalizuje zero jedynkowo piksele.
3. Dzieli zbiór danych na testowy i treningowy 2:8
4. Ustawia wymiary wejściowe

Metoda __imageAugmentation__():

```python
def __imageAugmentation__(self):
    print(colored("Rozpoczęcie augmentacji zdjęć...", "yellow"))
    self.IMAGE_AUGMENTER = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.08,
        height_shift_range=0.08,
        shear_range=0.15,
        zoom_range=0.15,
        horizontal_flip=True,
        fill_mode="nearest")
    print(colored("Zakończono augmentację zdjęć.", "green"))
```

1. Przerabia zdjęcia aby usprawnić proces treningu.

5.Główna aplikacja



Inicjalizacja:

```
class App:
    def __init__(self):
        self.mainMenu = MainMenu().run()

App()
```

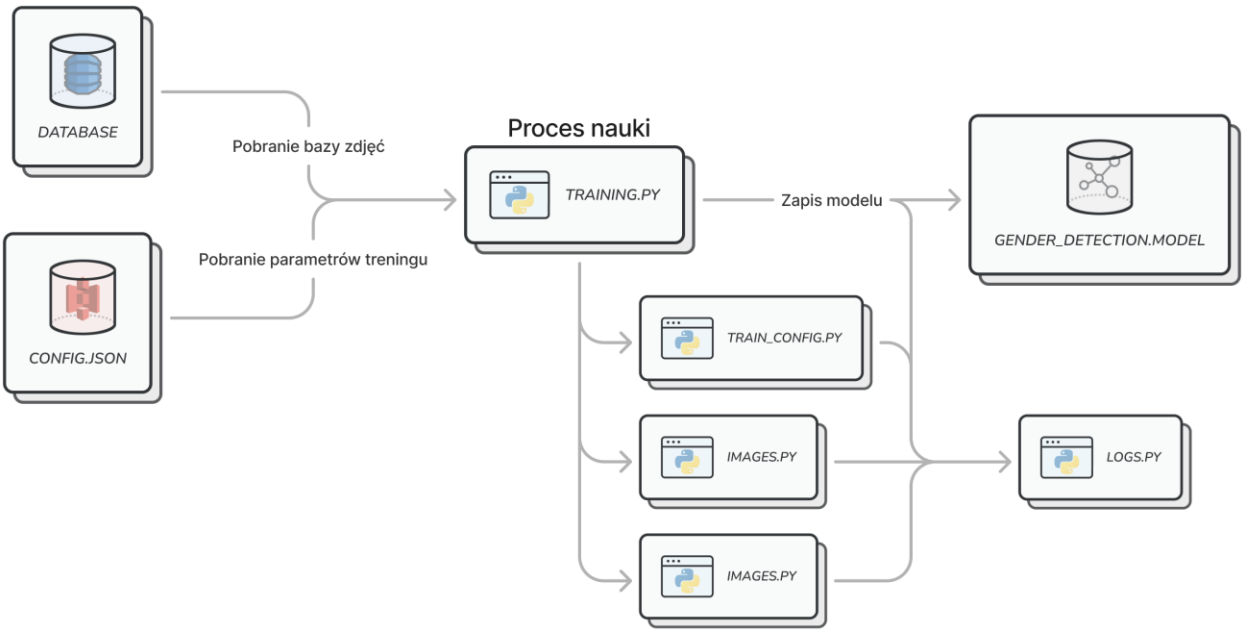
Menu główne:

```python
class MainMenu:
    options = ["1. Training", "2. Camera App", "3. Exit"]
    choice = None

    def printMenu(self):
        os.system('cls')
        for option in self.options:
            print(colored(option, 'magenta'))

    def getOption(self):
        self.choice = input(colored("\n»Enter option: ", 'light_cyan'))

    def checkOption(self):
        if self.choice == "1":
            TrainingMenu().run()
            return True
        elif self.choice == "2":
            cameraApp()
            return True
        elif self.choice == "3":
            print(colored("""
==============================
              Exit
            Goodbye!
==============================""", 'light_yellow'))
            return False
        else:
            os.system('cls')
            print(colored("Invalid option", 'red'))
            input("Press any key to continue...")
            return True

    def run(self):
        while True:
            self.printMenu()
            self.getOption()
            if not self.checkOption():
                break


```

Menu treningu:

```python
class TrainingMenu:

    trainingConfig = TrainConfig('train_config.json')

    options = ["1. Start training", "2. Configuration", "3. Back"]
    choice = None

    def printMenu(self):
        os.system('cls')
        for option in self.options:
            print(colored(option, 'magenta'))

    def getOption(self):
        self.choice = input(colored("\n»Enter option: ", 'light_cyan'))

    def checkOption(self):
        if self.choice == "1":
            Training(self.trainingConfig)
            return True

        elif self.choice == "2":
            self.config()
            return True
        elif self.choice == "3":
            return False
        else:
            os.system('cls')
            print(colored("Invalid option", 'red'))
            input("Press any key to continue...")
            return True

    def config(self):

        configOptions = ["1. Epochs: ", "2. Learning Rate: ", "3. Batch Size: ", "4. Width x Height: ", "5. Channels: ", "6. Back"]

        while True:
            self.choice = None
            configParameters = [
                self.trainingConfig.get_epochs(),
                self.trainingConfig.get_learning_rate(),
                self.trainingConfig.get_batch_size(),
                self.trainingConfig.get_height(),
                self.trainingConfig.get_channels(),
                ""
            ]

            value = None
            configParameters = configParameters

            os.system('cls')
            print("Current configuration:")

            temp = 0
            for option in configOptions:
                print(colored(option, 'magenta'), colored(configParameters[temp], 'light_blue'))
                temp = temp + 1

            self.choice = None
            self.getOption()

            if self.choice == "1":
                value = int(input(colored("Enter epochs [1, 100]: ", 'light_cyan')))
                if not self.trainingConfig.isInRange(value, 1, 100):
                    input("Press any key to continue...")
                else:
                    self.trainingConfig.set_epochs(value)

            elif self.choice == "2":
                value = float(input(colored("Enter learning rate [0.000001 - 0.01]: ",'light_cyan')))
                if not self.trainingConfig.isInRange(value, 0.000001, 0.01):
                    input("Press any key to continue...")
                else:
                    self.trainingConfig.set_learning_rate(value)

            elif self.choice == "3":
                value = int(input(colored("Enter batch size [8 - 128]: ",'light_cyan')))
                if not self.trainingConfig.isInRange(value, 8, 128):
                    input("Press any key to continue...")
                else:
                    self.trainingConfig.set_batch_size(value)


            elif self.choice == "4":
                value = int(input(colored("Enter width x height [64 - 144]: ",'light_cyan')))
                if not self.trainingConfig.isInRange(value, 64, 144):
                    input("Press any key to continue...")
                else:
                    self.trainingConfig.set_height(value)
                    self.trainingConfig.set_width(value)

            elif self.choice == "5":
                value = int(input(colored("Enter channels [1 - 3]: ",'light_cyan')))
                if not self.trainingConfig.isInRange(value, 1, 3):
                    input("Press any key to continue...")
                else:
                    self.trainingConfig.set_channels(value)

            elif self.choice == "6":
                break

            else:
                os.system('cls')
                print(colored("Invalid option", 'red'))
                input("Press any key to continue...")
                self.config()

            self.trainingConfig.save_config()

    def run(self):
        while True:
            self.printMenu()
            self.getOption()
            if not self.checkOption():
                break
```

6.Proces nauki

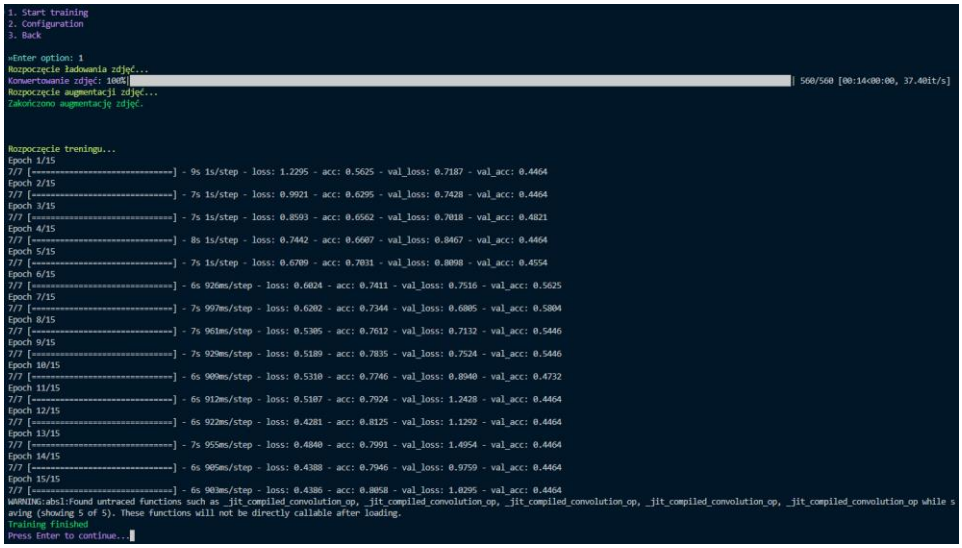


Przebieg działania programu

1. Inicjalizacja parametrów treningowych: W pliku config.py wczytywane są parametry treningowe z pliku konfiguracyjnego.
2. Ładowanie i konwersja obrazów: W pliku images.py inicjalizowane są ścieżki do obrazów, a następnie obrazy są wczytywane i konwertowane na odpowiedni format. Obrazy są również podzielone na zbiór treningowy i testowy.
3. Augmentacja danych: W pliku images.py przeprowadzana jest augmentacja danych, czyli generowanie dodatkowych wersji obrazów w celu rozszerzenia zbioru treningowego.
4. Budowa modelu: W pliku training.py definiowany jest model sieci neuronowej. Wykorzystywane są różne warstwy takie jak warstwy konwolucyjne, normalizacyjne, poolingowe, dropout itp.
5. Inicjalizacja optymalizatora i kompilacja modelu: W pliku training.py inicjalizowany jest optymalizator (np. Adam) oraz kompilowany jest model, określając funkcję straty, optymalizator i metryki do monitorowania.
6. Trening modelu: W pliku training.py model jest trenowany na danych treningowych. Wykorzystywany jest generator danych treningowych, który generuje mini-batche danych w trakcie treningu. Podczas treningu model jest walidowany na danych testowych. Trening jest przeprowadzany przez określoną liczbę epok.
7. Zapisanie modelu i wyników: Po zakończeniu treningu model jest zapisywany do pliku. Ponadto, zapisywane są także statystyki treningu, takie jak historia dokładności i funkcji straty w trakcie kolejnych epok.
8. Czyszczenie sesji: Po zakończeniu treningu sesja jest czyszczona, aby zwolnić zasoby pamięci.

Przykładowe zapisane dane po treningu

konsola



Wykres

Plik .json

```json
{
    "Status": "Unfinished",
    "Training parameters: ": {
        "epochs": 15,
        "learningRate": 0.001,
        "batchSize": 64,
        "width": 96,
        "height": 96,
        "channels": 3
    },
    "Load images duration": "0.039537429809570315",
    "Convert images duration": "15.098411798477173s",
    "Number of images: ": 560,
    "Total image prepare duration": "15.152655839920044s",
    "Training duration: ": "109.48233842849731s",
    "Total duration: ": "124.85995483398438s"
}
```

5.Test modeli:

Dane testów:

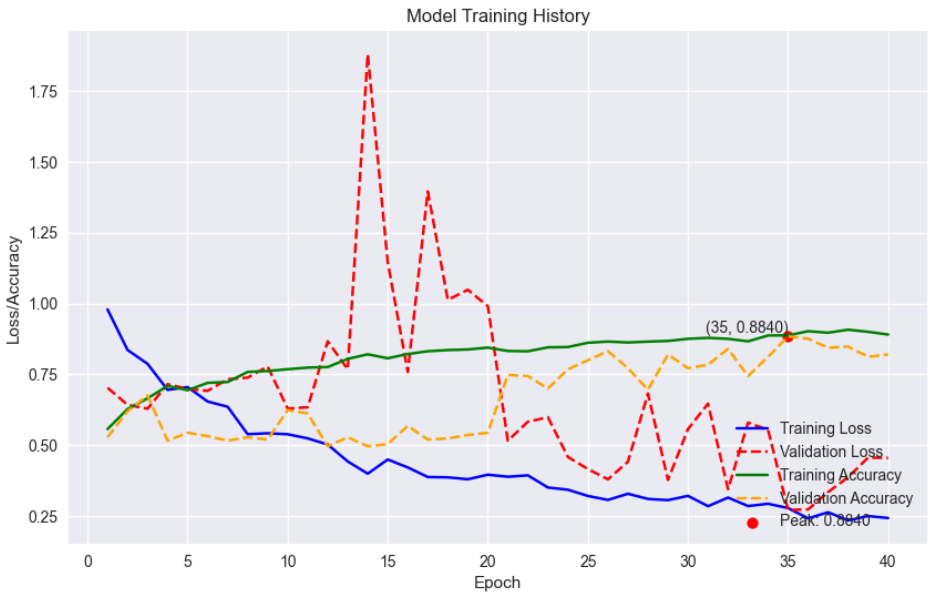
- 1246 zdjęciach 50:50 (men/women)
- Liczba epok: 40
- Learning rate: 0.001
- Batch size: 64
- IMG_WIDTH: 96
- IMG_HEIGHT: 96
- IMG_DEPTH: 3 (RGB)

Model 1:

```
model = Sequential([
    Conv2D(32, (3, 3), padding="same", input_shape=inputShape),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024),
    Activation("relu"),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("sigmoid")
])
```

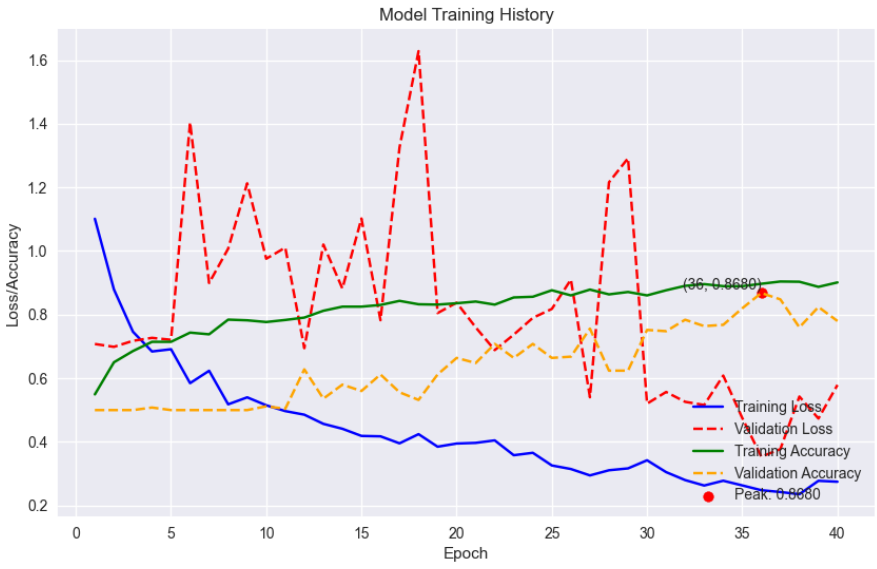
Model 2:

```
model = Sequential([
    Conv2D(32, (3, 3), padding="same", input_shape=inputShape),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(64, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),

    Conv2D(64, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(512),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(256),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("softmax")
])
```



Model 3:

```python
model = Sequential([
    Conv2D(32, (3, 3), padding="same", input_shape=inputShape),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(256, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(512, (3, 3), padding="same"),  # Dodatkowa warstwa konwolucyjna
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024),
    Activation("relu"),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("sigmoid")
])
```



Model 4:

```python
model = Sequential([
    Conv2D(32, (3, 3), padding="same", input_shape=inputShape),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(64, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),

    Conv2D(128, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(256, (3, 3), padding="same"),
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(512, (3, 3), padding="same"),  # Dodatkowa warstwa konwolucyjna
    Activation("relu"),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024),
    Activation("relu"),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("sigmoid")
])
```

Model 5:

```python
model = Sequential([
    Conv2D(64, (3, 3), padding="same", input_shape=inputShape),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(256, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),

    Conv2D(256, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(512, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(512),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("softmax")
])
```

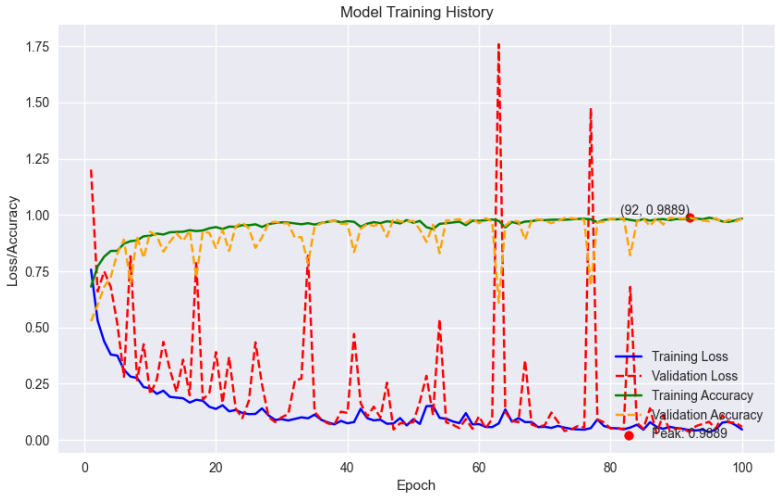
Model 6:

```python
model = Sequential([
    Conv2D(64, (3, 3), padding="same", input_shape=inputShape),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    Dropout(0.25),

    Conv2D(128, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(256, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    Dropout(0.25),

    Conv2D(256, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(512, (3, 3), padding="same"),
    LeakyReLU(alpha=0.1),
    BatchNormalization(axis=chanDim),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(512),
    LeakyReLU(alpha=0.1),
    BatchNormalization(),
    Dropout(0.5),

    Dense(2),
    Activation("softmax")
])
```

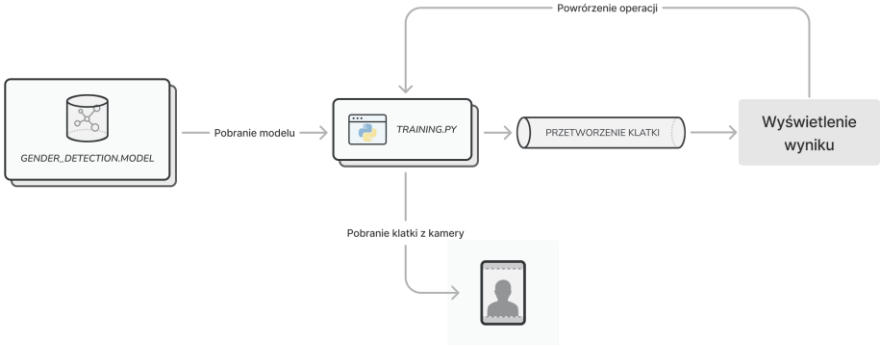
6.Model 1 nauczony na większej bazy danych

1. 7689 zdjęciach 50:50 (men/women)
2. Liczba epok: 40
3. Learning rate: 0.001
4. Batch size: 64
5. IMG_WIDTH: 96
6. IMG_HEIGHT: 96
7. IMG_DEPTH: 3 (RGB)

Przebieg treningu:



7.Proces wykrywania płci



1. Wczytanie wytrenowanego modelu za pomocą funkcji load_model().
2. Inicjalizacja kamery za pomocą funkcji cv2.VideoCapture().
3. W pętli while, odbywa się odczytywanie klatek z kamery za pomocą funkcji camera.read().
4. Wykrycie twarzy na klatce za pomocą funkcji cv.detect_face(). Zwracane są współrzędne prostokąta obejmującego twarz oraz pewność wykrycia.
5. Dla każdej wykrytej twarzy:
    a. Pobierane są współrzędne wierzchołków prostokąta obejmującego twarz
    b. Narysowanie takowego prostokąta
    c. Przycięcie obszaru z wykrytą twarzą
    d. Zmiana rozmiaru obrazu do 96x96
    e. Przygotowanie przyciętego obrazu dla modelu, normalizacja wartości pikseli.
    f. Wykonywanie predykcji płci na podstawie obrobionych danych. Zwracane są wartości prawdopodobieństwa dla każdej klasy
    g. Pobierany jest indeks o największej wartości predykcji [„men”/”women”]
    h. Narysowanie danych predykcji na prostokącie
    i. Zapis twarzy jeżeli spełnia wymagania (> 99.2% pewności wykrycia)

Wyniki pobrane z aplikacji kamery: