|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное автономное образовательное учреждение высшего образования  «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ» |

Кафедра компьютерных технологий и программной инженерии

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ОТЧЁТ ПО ПРАКТИКЕ  ЗАЩИЩЁН С ОЦЕНКОЙ  Руководитель |  | | | |
| Ст. преп. |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | по получению профессиональных умений и опыта профессиональной деятельности | |
| на тему индивидуального задания | | разработка нейросетевой модели определения пола |
| человека по фотографии лица | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Самариным Дмитрием Васильевичем |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | Программная инженерия |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 02 |  | Проектирование программных систем |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4134К |  |  |  | Д. В. Самарин |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2024

Оглавление

[Цель работы 3](#_Toc172326944)

[Теоретический раздел 3](#_Toc172326945)

[Практический раздел 4](#_Toc172326946)

[Результаты 10](#_Toc172326947)

[Приложение А 12](#_Toc172326948)

[Выводы 17](#_Toc172326949)

[Список литературы 18](#_Toc172326950)

# Цель работы

Разработка нейросетевой модели определения пола человека по фотографии лица. В качестве промежуточного преобразования изображения в эмбеддинги использовать:

* FaceNet;
* Любой классификатор (например, машину опорных векторов);
* Датасет – IMBD-WIKI;

На выходе должна получиться обученная модель, которая может распознаться пол человека по лицу.

# Теоретический раздел

Распознавание пола по изображениям является важной задачей в области компьютерного зрения и машинного обучения. Современные достижения в этой области позволяют строить эффективные модели, способные с высокой точностью определять пол человека по изображению его лица. В данном проекте используется комбинация конволюционных нейронных сетей для извлечения признаков (эмбеддингов) лица и метода опорных векторов (SVM) для классификации пола.

FaceNet – это глубокая сверточная нейронная сеть, обученная на задаче верификации лиц. Она преобразует изображения лиц в компактное евклидовое пространство, где расстояние между эмбеддингами различных лиц пропорционально их различию. В проекте используется предобученная модель FaceNet, загружаемая из .pb файла. Извлечение эмбеддингов осуществляется следующим образом:

* Лицо, обнаруженное MTCNN, выравнивается и масштабируется до размеров 160x160 пикселей.
* Лицо нормализуется по среднему значению и стандартному отклонению.
* Полученное изображение передается в модель FaceNet, которая генерирует 512-мерный эмбеддинг.

Для демонстрации работы модели создается веб-приложение с использованием Streamlit. Приложение позволяет пользователю загрузить изображение, на котором система распознает лицо и определяет пол. Основные шаги работы приложения:

* Загрузка изображения пользователем.
* Обнаружение и выравнивание лица на изображении.
* Извлечение эмбеддинга и предсказание пола.
* Отображение результата пользователю.

# Практический раздел

​В данном разделе описываются практические шаги по реализации системы распознавания пола по изображениям лиц. Приведенный код разделен на две части: обучение модели и создание веб-приложения для использования обученной модели.

Обучение модели (train.py):

Первым делом загружаем необходимые библиотеки и моудли:

|  |
| --- |
| import numpy as np  import cv2  import os  import scipy.io  from sklearn.svm import SVC  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score  from mtcnn.mtcnn import MTCNN  import tensorflow as tf  import joblib  from tqdm import tqdm  from concurrent.futures import ThreadPoolExecutor  import multiprocessing |

Настройка использования GPU для TensorFlow:

|  |
| --- |
| # Установка использования GPU для TensorFlow  gpus = tf.config.list\_physical\_devices('GPU')  if gpus:      try:          for gpu in gpus:              tf.config.experimental.set\_memory\_growth(gpu, True)      except RuntimeError as e:          print(e) |

Загрузка модели FaceNet из .pb файла:

|  |
| --- |
| # Путь к .pb файлу  model\_path = '20180402-114759/20180402-114759.pb'  # Загрузка модели  graph = load\_facenet\_model(model\_path)  # Получение тензоров ввода и вывода  input\_tensor = graph.get\_tensor\_by\_name("input:0")  output\_tensor = graph.get\_tensor\_by\_name("embeddings:0")  phase\_train\_tensor = graph.get\_tensor\_by\_name("phase\_train:0") |

Инициализация MTCNN для выравнивания лиц:

|  |
| --- |
| # Инициализация MTCNN для выравнивания лиц  detector = MTCNN() |

Функции для выравнивания лица и извлечения эмбеддингов:

|  |
| --- |
| # Функция для выравнивания лица  def align\_face(image):      results = detector.detect\_faces(image)      if results:          x, y, w, h = results[0]['box']          face = image[y:y + h, x:x + w]          face = cv2.resize(face, (160, 160))          return face      return None  # Функция для извлечения эмбеддингов с помощью FaceNet  def get\_embedding(face\_pixels):      face\_pixels = face\_pixels.astype('float32')      mean, std = face\_pixels.mean(), face\_pixels.std()      face\_pixels = (face\_pixels - mean) / std      samples = np.expand\_dims(face\_pixels, axis=0)      with tf.compat.v1.Session(graph=graph) as sess:          feed\_dict = {input\_tensor: samples, phase\_train\_tensor: False}          embedding = sess.run(output\_tensor, feed\_dict=feed\_dict)      return embedding[0] |

Обработка изображений и создание датасета:

|  |
| --- |
| # Функция для обработки одного изображения  def process\_image(image\_path, gender):      try:          image = cv2.imread(image\_path)          if image is None:              return None, None          face = align\_face(image)          if face is not None:              embedding = get\_embedding(face)              return embedding, int(gender)      except Exception as e:          print(f"Error processing image {image\_path}: {e}")      return None, None  # Функция для создания tf.data.Dataset  def create\_dataset(dataset\_path, metadata\_file, batch\_size=32, fraction=0.1, max\_workers=6):      data = scipy.io.loadmat(metadata\_file)      full\_path = data['imdb'][0][0][2][0]      gender = data['imdb'][0][0][3][0]      # Используем только часть данных      num\_samples = int(len(full\_path) \* fraction)      full\_path = full\_path[:num\_samples]      gender = gender[:num\_samples]      paths\_and\_labels = [(os.path.join(dataset\_path, full\_path[i][0]), gender[i]) for i in range(len(full\_path)) if                          not np.isnan(gender[i])]      def generator():          with ThreadPoolExecutor(max\_workers=max\_workers) as executor:              futures = [executor.submit(process\_image, image\_path, gender) for image\_path, gender in paths\_and\_labels]              for future in tqdm(futures, desc="Processing images"):                  embedding, gender = future.result()                  if embedding is not None:                      yield embedding, gender      dataset = tf.data.Dataset.from\_generator(generator, output\_signature=(          tf.TensorSpec(shape=(512,), dtype=tf.float32),          tf.TensorSpec(shape=(), dtype=tf.int64)      ))      dataset = dataset.batch(batch\_size)      dataset = dataset.prefetch(buffer\_size=tf.data.AUTOTUNE)      return dataset  # Путь к датасету и метаданным  dataset\_path = 'imdb\_crop'  # Путь к папке imdb\_crop  metadata\_file = os.path.join(dataset\_path, 'imdb.mat')  # Путь к файлу imdb.mat  # Создание tf.data.Dataset  print("Creating dataset...")  dataset = create\_dataset(dataset\_path, metadata\_file, batch\_size=32, fraction=0.1,                           max\_workers=6)  print("Dataset created.") |

Загрузка данных и разделение на обучающую и тестовую выборки:

|  |
| --- |
| # Загрузка данных в numpy массивы  X, y = [], []  for batch in tqdm(dataset, desc="Loading data"):      X\_batch, y\_batch = batch      X.extend(X\_batch.numpy())      y.extend(y\_batch.numpy())  X = np.array(X)  y = np.array(y)  # Разделение данных на обучающую и тестовую выборки  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |

Обучение, сохранение модели SVM и оценка модели:

|  |
| --- |
| # Обучение SVM с прогресс-баром  print("Training SVM model...")  model = SVC(kernel='linear', verbose=True)  model.fit(X\_train, y\_train)  print("Model trained.")  # Путь для сохранения модели  model\_save\_path = 'svm\_gender\_classifier.pkl'  # Сохранение модели  joblib.dump(model, model\_save\_path)  print(f"Model saved to {model\_save\_path}")  # Загрузка модели  model = joblib.load(model\_save\_path)  print("Model loaded.")  # Оценка модели с прогресс-баром  print("Evaluating model...")  y\_pred = []  for i in tqdm(range(len(X\_test)), desc="Predicting"):      y\_pred.append(model.predict([X\_test[i]]))  y\_pred = np.array(y\_pred).flatten()  accuracy = accuracy\_score(y\_test, y\_pred)  print(f'Accuracy: {accuracy \* 100:.2f}%') |

На выходе получаем обученную модель svm\_gender\_classifier.pkl. Для того, чтобы проверить результат был разработан веб-интерфейс:

Загружаем необходимые библиотеке и проверяем работу GPU

|  |
| --- |
| import numpy as np  import cv2  import os  from mtcnn.mtcnn import MTCNN  import tensorflow as tf  import joblib  import streamlit as st  # Установка использования GPU для TensorFlow  gpus = tf.config.list\_physical\_devices('GPU')  if gpus:      try:          for gpu in gpus:              tf.config.experimental.set\_memory\_growth(gpu, True)      except RuntimeError as e:          print(e) |

Загрузка модели FaceNet и инициализация MTCNN:

|  |
| --- |
| # Загрузка модели FaceNet из .pb файла  def load\_facenet\_model(model\_path):      if not os.path.exists(model\_path):          raise FileNotFoundError(f"Model file not found: {model\_path}")      with tf.io.gfile.GFile(model\_path, "rb") as f:          graph\_def = tf.compat.v1.GraphDef()          graph\_def.ParseFromString(f.read())          tf.import\_graph\_def(graph\_def, name="")      return tf.compat.v1.get\_default\_graph()  # Путь к .pb файлу  model\_path = '20180402-114759/20180402-114759.pb'  # Загрузка модели  graph = load\_facenet\_model(model\_path)  # Получение тензоров ввода и вывода  input\_tensor = graph.get\_tensor\_by\_name("input:0")  output\_tensor = graph.get\_tensor\_by\_name("embeddings:0")  phase\_train\_tensor = graph.get\_tensor\_by\_name("phase\_train:0")  # Инициализация MTCNN для выравнивания лиц  detector = MTCNN() |

Функции для выравнивания лица и извлечения эмбеддингов:

|  |
| --- |
| # Функция для выравнивания лица  def align\_face(image):      results = detector.detect\_faces(image)      if results:          x, y, w, h = results[0]['box']          face = image[y:y + h, x:x + w]          face = cv2.resize(face, (160, 160))          return face, (x, y, w, h)      return None, None  # Функция для извлечения эмбеддингов с помощью FaceNet  def get\_embedding(face\_pixels):      face\_pixels = face\_pixels.astype('float32')      mean, std = face\_pixels.mean(), face\_pixels.std()      face\_pixels = (face\_pixels - mean) / std      samples = np.expand\_dims(face\_pixels, axis=0)      with tf.compat.v1.Session(graph=graph) as sess:          feed\_dict = {input\_tensor: samples, phase\_train\_tensor: False}          embedding = sess.run(output\_tensor, feed\_dict=feed\_dict)      return embedding[0] |

Загрузка модели SVM и функции для предсказания пола:

|  |
| --- |
| # Загрузка модели SVM  model\_save\_path = 'learned\_model/svm\_gender\_classifier.pkl'  svm\_model = joblib.load(model\_save\_path)  # Функция для предсказания пола на основе изображения  def predict\_gender(image):      face, box = align\_face(image)      if face is not None:          embedding = get\_embedding(face)          prediction = svm\_model.predict([embedding])          gender = "Male" if prediction[0] == 1 else "Female"          return gender, box      else:          raise ValueError("Лицо не разпознано")  # Функция для рисования прямоугольника и надписи  def draw\_label(image, box, gender):      x, y, w, h = box      label = gender      color = (0, 255, 0) if gender == "Male" else (255, 0, 0)      cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)      cv2.putText(image, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2) |

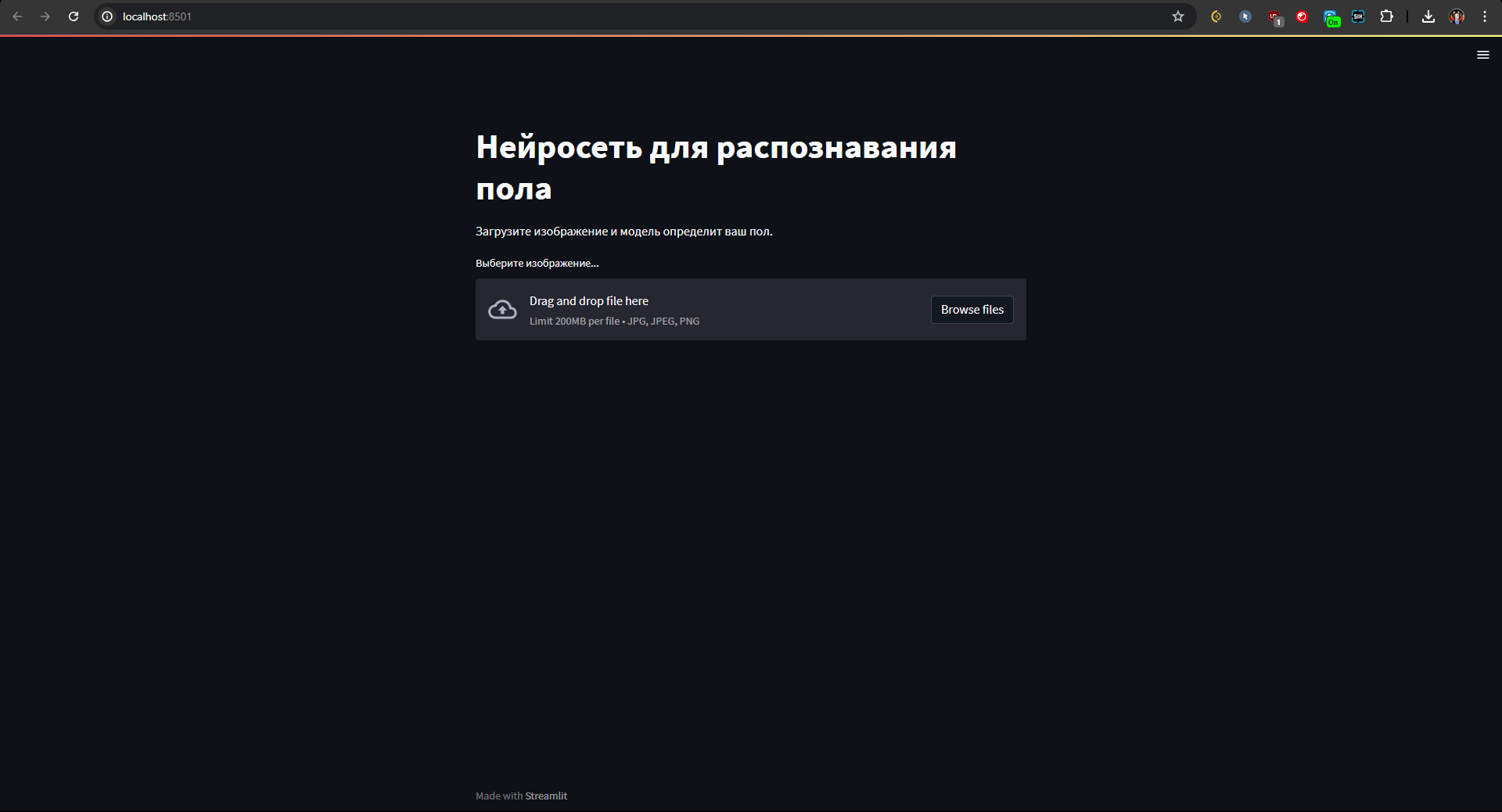
После создаем веб-приложение и запускаем полученный скрипт командой: streamlit run usagePrime.

|  |
| --- |
| # Создание веб-приложения с помощью Streamlit  st.title("Нейросеть для распознавания пола")  st.write("Загрузите изображение и модель определит ваш пол.")  uploaded\_file = st.file\_uploader("Выберите изображение...", type=["jpg", "jpeg", "png"])  if uploaded\_file is not None:      image = np.array(cv2.imdecode(np.frombuffer(uploaded\_file.read(), np.uint8), cv2.IMREAD\_COLOR))      # Преобразование изображения из BGR в RGB      image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)        try:          gender, box = predict\_gender(image)          draw\_label(image\_rgb, box, gender)          st.image(image\_rgb, caption="Ваше изображение", use\_column\_width=True)          gender\_rus = "Мужской пол" if gender == "Male" else "Женский пол"          st.write(f"Гендер человека на основе изображения: {gender\_rus}")      except ValueError as e:          st.image(image\_rgb, caption="Загруженное изображение", use\_column\_width=True)          st.write(f"Error: {str(e)}") |

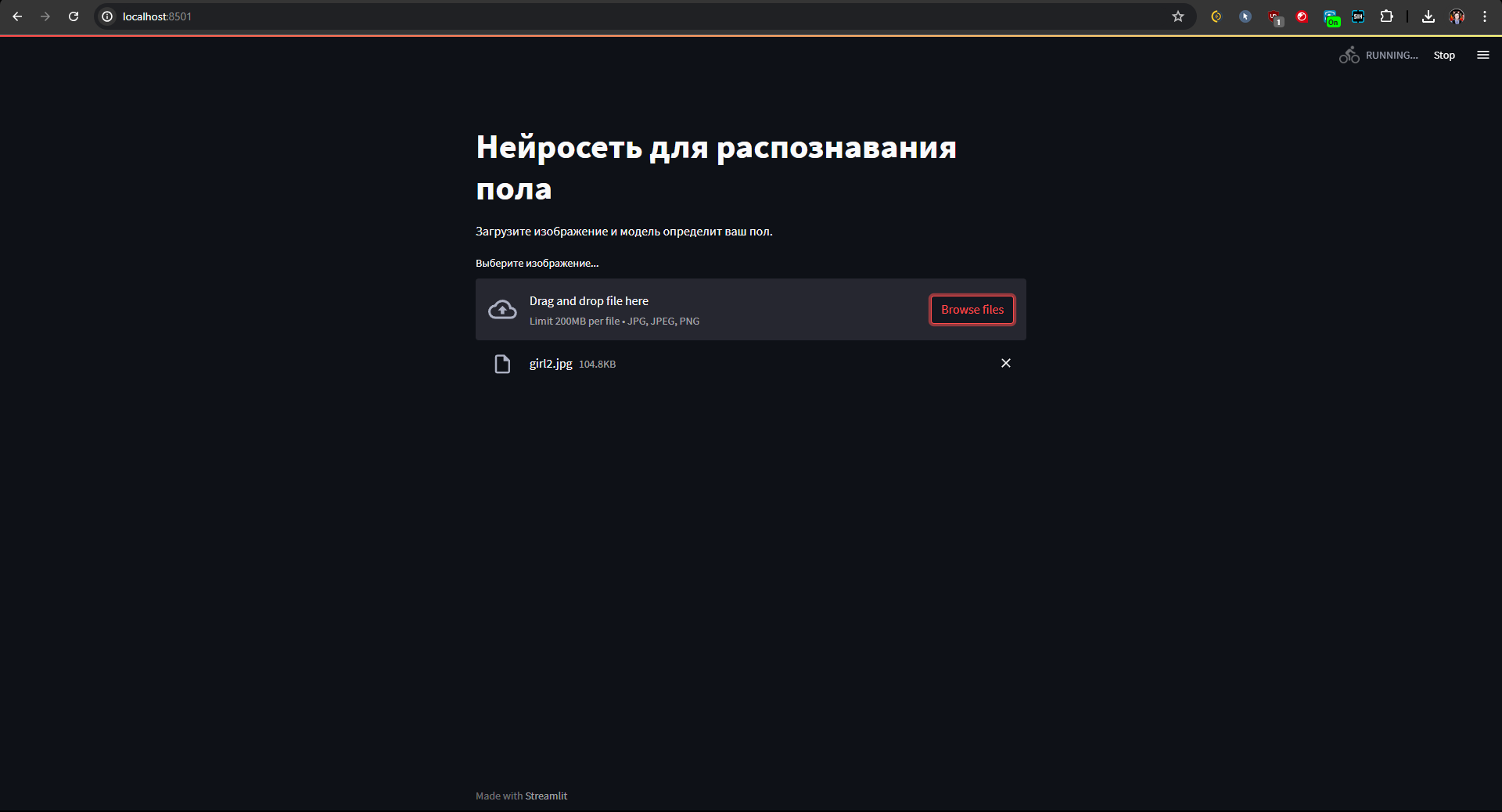
С результатами работы обученной модели можно ознакомиться ниже.

# Результаты

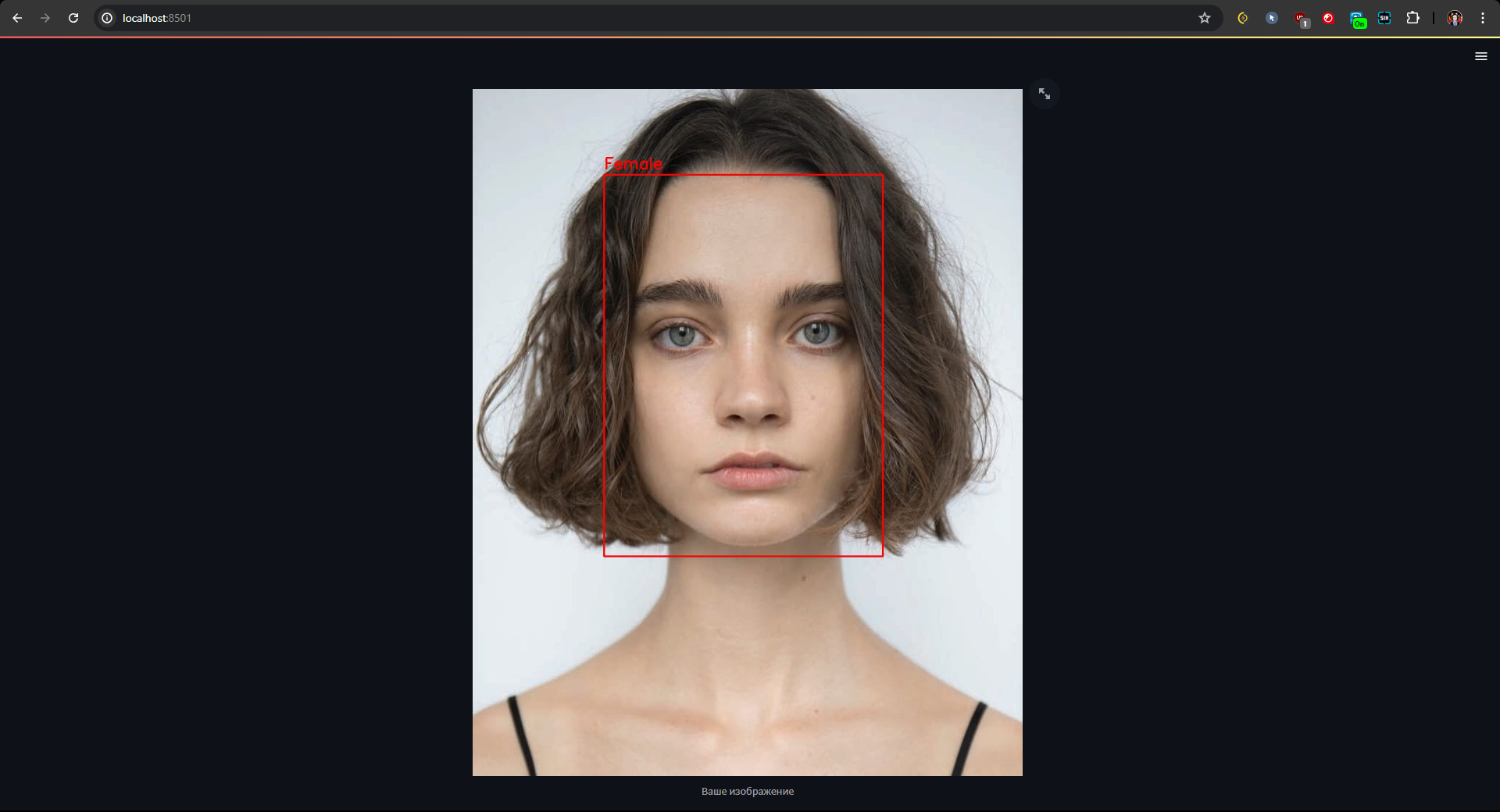
Пользователя встречает главная страница, где тот должен загрузить изображение для определения пола человека:



Выбрав изображение, начинается его обработка (требуется подождать некоторое время):



После обработки изображения моделью высветится результат с гендером человека на фото:



В случае если требуется обработать другое изображение пользователь должен выбрать новую фотографию.

Для ознакомления сделайте клон репозитория:

https://github.com/dYGamma/neuro-face

# Приложение А

Рассмотрим полный листинг кода, включая обучение модели:

train.py

|  |
| --- |
| import numpy as np  import cv2  import os  import scipy.io  from sklearn.svm import SVC  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score  from mtcnn.mtcnn import MTCNN  import tensorflow as tf  import joblib  from tqdm import tqdm  from concurrent.futures import ThreadPoolExecutor  import multiprocessing  # Установка использования GPU для TensorFlow  gpus = tf.config.list\_physical\_devices('GPU')  if gpus:      try:          for gpu in gpus:              tf.config.experimental.set\_memory\_growth(gpu, True)      except RuntimeError as e:          print(e)  # Загрузка модели FaceNet из .pb файла  def load\_facenet\_model(model\_path):      if not os.path.exists(model\_path):          raise FileNotFoundError(f"Model file not found: {model\_path}")      with tf.io.gfile.GFile(model\_path, "rb") as f:          graph\_def = tf.compat.v1.GraphDef()          graph\_def.ParseFromString(f.read())          tf.import\_graph\_def(graph\_def, name="")      return tf.compat.v1.get\_default\_graph()  # Путь к .pb файлу  model\_path = '20180402-114759/20180402-114759.pb'  # Загрузка модели  graph = load\_facenet\_model(model\_path)  # Получение тензоров ввода и вывода  input\_tensor = graph.get\_tensor\_by\_name("input:0")  output\_tensor = graph.get\_tensor\_by\_name("embeddings:0")  phase\_train\_tensor = graph.get\_tensor\_by\_name("phase\_train:0")  # Инициализация MTCNN для выравнивания лиц  detector = MTCNN()  # Функция для выравнивания лица  def align\_face(image):      results = detector.detect\_faces(image)      if results:          x, y, w, h = results[0]['box']          face = image[y:y + h, x:x + w]          face = cv2.resize(face, (160, 160))          return face      return None  # Функция для извлечения эмбеддингов с помощью FaceNet  def get\_embedding(face\_pixels):      face\_pixels = face\_pixels.astype('float32')      mean, std = face\_pixels.mean(), face\_pixels.std()      face\_pixels = (face\_pixels - mean) / std      samples = np.expand\_dims(face\_pixels, axis=0)      with tf.compat.v1.Session(graph=graph) as sess:          feed\_dict = {input\_tensor: samples, phase\_train\_tensor: False}          embedding = sess.run(output\_tensor, feed\_dict=feed\_dict)      return embedding[0]  # Функция для обработки одного изображения  def process\_image(image\_path, gender):      try:          image = cv2.imread(image\_path)          if image is None:              return None, None          face = align\_face(image)          if face is not None:              embedding = get\_embedding(face)              return embedding, int(gender)      except Exception as e:          print(f"Error processing image {image\_path}: {e}")      return None, None  # Функция для создания tf.data.Dataset  def create\_dataset(dataset\_path, metadata\_file, batch\_size=32, fraction=0.1, max\_workers=6):      data = scipy.io.loadmat(metadata\_file)      full\_path = data['imdb'][0][0][2][0]      gender = data['imdb'][0][0][3][0]      # Используем только часть данных      num\_samples = int(len(full\_path) \* fraction)      full\_path = full\_path[:num\_samples]      gender = gender[:num\_samples]      paths\_and\_labels = [(os.path.join(dataset\_path, full\_path[i][0]), gender[i]) for i in range(len(full\_path)) if                          not np.isnan(gender[i])]      def generator():          with ThreadPoolExecutor(max\_workers=max\_workers) as executor:              futures = [executor.submit(process\_image, image\_path, gender) for image\_path, gender in paths\_and\_labels]              for future in tqdm(futures, desc="Processing images"):                  embedding, gender = future.result()                  if embedding is not None:                      yield embedding, gender      dataset = tf.data.Dataset.from\_generator(generator, output\_signature=(          tf.TensorSpec(shape=(512,), dtype=tf.float32),          tf.TensorSpec(shape=(), dtype=tf.int64)      ))      dataset = dataset.batch(batch\_size)      dataset = dataset.prefetch(buffer\_size=tf.data.AUTOTUNE)      return dataset  # Путь к датасету и метаданным  dataset\_path = 'imdb\_crop'  # Путь к папке imdb\_crop  metadata\_file = os.path.join(dataset\_path, 'imdb.mat')  # Путь к файлу imdb.mat  # Создание tf.data.Dataset  print("Creating dataset...")  dataset = create\_dataset(dataset\_path, metadata\_file, batch\_size=32, fraction=0.1,                           max\_workers=6)  print("Dataset created.")  # Загрузка данных в numpy массивы  X, y = [], []  for batch in tqdm(dataset, desc="Loading data"):      X\_batch, y\_batch = batch      X.extend(X\_batch.numpy())      y.extend(y\_batch.numpy())  X = np.array(X)  y = np.array(y)  # Разделение данных на обучающую и тестовую выборки  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Обучение SVM с прогресс-баром  print("Training SVM model...")  model = SVC(kernel='linear', verbose=True)  model.fit(X\_train, y\_train)  print("Model trained.")  # Путь для сохранения модели  model\_save\_path = 'svm\_gender\_classifier.pkl'  # Сохранение модели  joblib.dump(model, model\_save\_path)  print(f"Model saved to {model\_save\_path}")  # Загрузка модели  model = joblib.load(model\_save\_path)  print("Model loaded.")  # Оценка модели с прогресс-баром  print("Evaluating model...")  y\_pred = []  for i in tqdm(range(len(X\_test)), desc="Predicting"):      y\_pred.append(model.predict([X\_test[i]]))  y\_pred = np.array(y\_pred).flatten()  accuracy = accuracy\_score(y\_test, y\_pred)  print(f'Accuracy: {accuracy \* 100:.2f}%') |

usagePrime.py

|  |
| --- |
| import numpy as np  import cv2  import os  from mtcnn.mtcnn import MTCNN  import tensorflow as tf  import joblib  import streamlit as st  # Установка использования GPU для TensorFlow  gpus = tf.config.list\_physical\_devices('GPU')  if gpus:      try:          for gpu in gpus:              tf.config.experimental.set\_memory\_growth(gpu, True)      except RuntimeError as e:          print(e)  # Загрузка модели FaceNet из .pb файла  def load\_facenet\_model(model\_path):      if not os.path.exists(model\_path):          raise FileNotFoundError(f"Model file not found: {model\_path}")      with tf.io.gfile.GFile(model\_path, "rb") as f:          graph\_def = tf.compat.v1.GraphDef()          graph\_def.ParseFromString(f.read())          tf.import\_graph\_def(graph\_def, name="")      return tf.compat.v1.get\_default\_graph()  # Путь к .pb файлу  model\_path = '20180402-114759/20180402-114759.pb'  # Загрузка модели  graph = load\_facenet\_model(model\_path)  # Получение тензоров ввода и вывода  input\_tensor = graph.get\_tensor\_by\_name("input:0")  output\_tensor = graph.get\_tensor\_by\_name("embeddings:0")  phase\_train\_tensor = graph.get\_tensor\_by\_name("phase\_train:0")  # Инициализация MTCNN для выравнивания лиц  detector = MTCNN()  # Функция для выравнивания лица  def align\_face(image):      results = detector.detect\_faces(image)      if results:          x, y, w, h = results[0]['box']          face = image[y:y + h, x:x + w]          face = cv2.resize(face, (160, 160))          return face, (x, y, w, h)      return None, None  # Функция для извлечения эмбеддингов с помощью FaceNet  def get\_embedding(face\_pixels):      face\_pixels = face\_pixels.astype('float32')      mean, std = face\_pixels.mean(), face\_pixels.std()      face\_pixels = (face\_pixels - mean) / std      samples = np.expand\_dims(face\_pixels, axis=0)      with tf.compat.v1.Session(graph=graph) as sess:          feed\_dict = {input\_tensor: samples, phase\_train\_tensor: False}          embedding = sess.run(output\_tensor, feed\_dict=feed\_dict)      return embedding[0]  # Загрузка модели SVM  model\_save\_path = 'learned\_model/svm\_gender\_classifier.pkl'  svm\_model = joblib.load(model\_save\_path)  # Функция для предсказания пола на основе изображения  def predict\_gender(image):      face, box = align\_face(image)      if face is not None:          embedding = get\_embedding(face)          prediction = svm\_model.predict([embedding])          gender = "Male" if prediction[0] == 1 else "Female"          return gender, box      else:          raise ValueError("Лицо не разпознано")  # Функция для рисования прямоугольника и надписи  def draw\_label(image, box, gender):      x, y, w, h = box      label = gender      color = (0, 255, 0) if gender == "Male" else (255, 0, 0)      cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)      cv2.putText(image, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2)  # Создание веб-приложения с помощью Streamlit  st.title("Нейросеть для распознавания пола")  st.write("Загрузите изображение и модель определит ваш пол.")  uploaded\_file = st.file\_uploader("Выберите изображение...", type=["jpg", "jpeg", "png"])  if uploaded\_file is not None:      image = np.array(cv2.imdecode(np.frombuffer(uploaded\_file.read(), np.uint8), cv2.IMREAD\_COLOR))      # Преобразование изображения из BGR в RGB      image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)        try:          gender, box = predict\_gender(image)          draw\_label(image\_rgb, box, gender)          st.image(image\_rgb, caption="Ваше изображение", use\_column\_width=True)          gender\_rus = "Мужской пол" if gender == "Male" else "Женский пол"          st.write(f"Гендер человека на основе изображения: {gender\_rus}")      except ValueError as e:          st.image(image\_rgb, caption="Загруженное изображение", use\_column\_width=True)          st.write(f"Error: {str(e)}") |

# Выводы

Проект продемонстрировал, как современные методы глубокого обучения могут быть использованы для решения задачи распознавания пола по изображениям лиц. Реализация от этапа подготовки данных до создания веб-приложения показала, что данная технология может быть интегрирована в различные приложения и сервисы, предлагая точные и быстрые решения для распознавания пола.

# Список литературы

1. FaceNet: A Unified Embedding for Face Recognition and Clustering - <https://arxiv.org/abs/1503.03832>
2. Support Vector Machines (SVMs) - <https://link.springer.com/article/10.1007/BF00994018>
3. Streamlit: The fastest way to build and share data apps - <https://streamlit.io/>
4. IMDB-WIKI – 500k+ face images with age and gender labels - <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>
5. FaceNet model - <https://github.com/davidsandberg/facenet>