Astana IT University

Final Project Report

Recognition of fruits and vegatables and integrating a calorie counting function.

Group: IT - 2207

Made by: K.Sherkhan, A.Nurman

Introduction

- **Problem**

The main objective of the project is to create a web application that will help with different fruit and vegetable identification. The web application should accurately recognize the fruit or vegetable in the picture and calculate how many calories are closest to it.

- **Literature review with links:**

An extensive overview of deep learning-based methods for identifying fruits and vegetables is given in this work. The article talks about different convolutional neural network (CNN) architectures and methods for classifying images. It also looks at the difficulties and restrictions related to identifying fruits and vegetables.
Link: https://www.mdpi.com/2073-4395/13/6/1625

The current article examines machine learning methods for problems related to food recognition, such as identifying fruits and vegetables. It covers both deep learning techniques like CNNs and conventional machine learning algorithms. It also looks at evaluation measures and datasets that are frequently used in food recognition studies.

Link: https://www.sciencedirect.com/science/article/pii/S1566253523001756

- **Current work (description of the work):**

Structure of code placed in Google.Colab

First of all, we import our training and validation sets by using command

import drive
drive.mount('/content/drive')

```
from google.colab import drive
drive.mount('/content/drive')
```

Then, goes our main code:

```python
# dataset : https://drive.google.com/drive/folders/1E5tbjL-gDqD2P2yBHY_C5w7GtSFir7O7?usp=drive_link
# Importing necessary libraries
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define constants
IMAGE_WIDTH, IMAGE_HEIGHT = 150, 150
BATCH_SIZE = 32
NUM_EPOCHS = 10

# Define directories for train and validation data
base_dir = '/content/drive/MyDrive/Final_images'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Define data augmentation and preprocessing for train and validation data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
```

```python
# Define data augmentation and preprocessing for train and validation data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches using train_datagen generator
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

# Flow validation images in batches using validation_datagen generator
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(20, activation='softmax')
```

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=NUM_EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)

# Save the model
model.save('fruit_vegetable_classification_model.h5')

# Define the calorie counts for each class
calorie_counts = {
    'Apple': 95,
    'Banana': 105,
    'Orange': 62
}

# Function to calculate calories based on predicted class
def calculate_calories(image_path):
    # Load the saved model
    loaded_model = tf.keras.models.load_model('fruit_vegetable_classification_model.h5')

    # Preprocess the image
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(IMAGE_WIDTH, IMAGE_HEIGHT))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    img_array /= 255.0

    # Predict the class
    predictions = loaded_model.predict(img_array)

    # Map class index to class label
    class_labels = ['Apple', 'Banana', 'Orange']
    predicted_class_index = np.argmax(predictions[0])
    predicted_class = class_labels[predicted_class_index]
```

```python
    # Predict the class
    predictions = loaded_model.predict(img_array)

    # Map class index to class label
    class_labels = ['Apple', 'Banana', 'Orange']
    predicted_class_index = np.argmax(predictions[0])
    predicted_class = class_labels[predicted_class_index]

    # Check if predicted class is in the calorie_counts dictionary
    if predicted_class in calorie_counts:
        calorie_count = calorie_counts[predicted_class]
    else:
        calorie_count = 0

    return calorie_count

# Example usage:
image_path = '/content/drive/MyDrive/Final_images/train/Apple/App_045.jpg'
calories = calculate_calories(image_path)
print("Calories:", calories)
```

```
Found 503 images belonging to 20 classes.
Found 477 images belonging to 20 classes.
Epoch 1/10
15/15 [==============================] - 234s 16s/step - loss: 3.1278 - accuracy: 0.0807 - val_loss: 2.9519 - val_accuracy: 0.1652
Epoch 2/10
15/15 [==============================] - 63s 4s/step - loss: 2.7028 - accuracy: 0.1699 - val_loss: 2.3886 - val_accuracy: 0.2478
Epoch 3/10
15/15 [==============================] - 63s 4s/step - loss: 2.1829 - accuracy: 0.2781 - val_loss: 1.7385 - val_accuracy: 0.3817
Epoch 4/10
15/15 [==============================] - 63s 4s/step - loss: 1.7511 - accuracy: 0.4161 - val_loss: 1.7358 - val_accuracy: 0.4353
Epoch 5/10
15/15 [==============================] - 53s 4s/step - loss: 1.2403 - accuracy: 0.5817 - val_loss: 1.1521 - val_accuracy: 0.6071
Epoch 6/10
15/15 [==============================] - 53s 4s/step - loss: 1.1738 - accuracy: 0.5499 - val_loss: 1.2218 - val_accuracy: 0.6384
Epoch 7/10
15/15 [==============================] - 53s 4s/step - loss: 0.9722 - accuracy: 0.6582 - val_loss: 0.9621 - val_accuracy: 0.7009
Epoch 8/10
15/15 [==============================] - 63s 4s/step - loss: 0.6692 - accuracy: 0.7643 - val_loss: 1.2189 - val_accuracy: 0.5424
Epoch 9/10
15/15 [==============================] - 63s 4s/step - loss: 0.5298 - accuracy: 0.7877 - val_loss: 0.8973 - val_accuracy: 0.7366
Epoch 10/10
15/15 [==============================] - 62s 4s/step - loss: 0.5504 - accuracy: 0.7877 - val_loss: 1.0714 - val_accuracy: 0.6719
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras
  saving_api.save_model(
1/1 [==============================] - 0s 240ms/step
Calories: 95
```

Then by using the , we save our machine learning.

model.save("/content/model.h5") !ls -Lha

```
model.save("/content/model.h5")
!ls -lha
```

```
total 437M
drwxr-xr-x 1 root root 4.0K Mar  6 16:47 .
drwxr-xr-x 1 root root 4.0K Mar  6 15:56 ..
drwxr-xr-x 4 root root 4.0K Mar  4 14:27 .config
drwx------ 5 root root 4.0K Mar  6 16:15 drive
-rw-r--r-- 1 root root 218M Mar  6 16:46 fruit_vegetable_classification_model.h5
-rw-r--r-- 1 root root 218M Mar  6 16:47 model.h5
drwxr-xr-x 1 root root 4.0K Mar  4 14:28 sample_data
```

Then we use use the command *!pip install tensorflowjs* to install tenserflow.js library.

```
!pip install tensorflowjs
```

```
Requirement already satisfied: tensorstore in /usr/local/lib/python3.10/dist-packages (from flax>=0.7.2->tensorflowjs) (0.1.45)
Requirement already satisfied: rich>=11.1 in /usr/local/lib/python3.10/dist-packages (from flax>=0.7.2->tensorflowjs) (13.7.1)
Requirement already satisfied: typing-extensions>=4.2 in /usr/local/lib/python3.10/dist-packages (from flax>=0.7.2->tensorflowjs) (4.10.0)
Requirement already satisfied: PyYAML>=5.4.1 in /usr/local/lib/python3.10/dist-packages (from flax>=0.7.2->tensorflowjs) (6.0.1)
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from jax>=0.4.13->tensorflowjs) (0.2.0)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from jax>=0.4.13->tensorflowjs) (3.3.0)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages (from jax>=0.4.13->tensorflowjs) (1.11.4)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (16.0.6)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (67.7.2)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (2.4.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (1.62.0)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow<3,>=2.13.0->tensorflowjs) (2.15.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from tensorflow-decision-forests>=1.5.0->tensorflowjs) (1.5.3)
Requirement already satisfied: wheel in /usr/local/lib/python3.10/dist-packages (from tensorflow-decision-forests>=1.5.0->tensorflowjs) (0.42.0)
Collecting wurlitzer (from tensorflow-decision-forests>=1.5.0->tensorflowjs)
  Downloading wurlitzer-3.0.3-py3-none-any.whl (7.3 kB)
Requirement already satisfied: tf-keras>=2.14.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow-hub>=0.14.0->tensorflowjs) (2.15.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.1->flax>=0.7.2->tensorflowjs) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.1->flax>=0.7.2->tensorflowjs) (2.16.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow<3,>=2.13.0->tensorflowjs) (3.0.1)
Requirement already satisfied: chex>=0.1.7 in /usr/local/lib/python3.10/dist-packages (from optax->flax>=0.7.2->tensorflowjs) (0.1.85)
Requirement already satisfied: etils[epath,epy] in /usr/local/lib/python3.10/dist-packages (from orbax-checkpoint->flax>=0.7.2->tensorflowjs) (1.7.0)
Requirement already satisfied: nest_asyncio in /usr/local/lib/python3.10/dist-packages (from orbax-checkpoint->flax>=0.7.2->tensorflowjs) (1.6.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->tensorflow-decision-forests>=1.5.0->tensorflowjs) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->tensorflow-decision-forests>=1.5.0->tensorflowjs) (2023.4)
Requirement already satisfied: toolz>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from chex>=0.1.7->optax->flax>=0.7.2->tensorflowjs) (0.12.1)
```

In the next step we are importing the TenserFlow.js library by using the command

import tensorflowjs as tfjs

And then by using the command *tfjs.converters.save_keras_model(model, '/content/model_data')* we convert and save a Keras model to a format that can be used by TenserFlow.js

```python
import tensorflowjs as tfjs

tfjs.converters.save_keras_model(model, '/content/model_data')
```

In the last part we are using the command from google.colab import files Which means we are importing the files module from the google.colab package in Google Colab.

```python
from google.colab import files

files.download("/content/model_data/model.json")
files.download("/content/model_data/group1-shard1of19.bin")
files.download("/content/model_data/group1-shard2of19.bin")
files.download("/content/model_data/group1-shard3of19.bin")
files.download("/content/model_data/group1-shard4of19.bin")
files.download("/content/model_data/group1-shard5of19.bin")
files.download("/content/model_data/group1-shard6of19.bin")
files.download("/content/model_data/group1-shard7of19.bin")
files.download("/content/model_data/group1-shard8of19.bin")
files.download("/content/model_data/group1-shard9of19.bin")
files.download("/content/model_data/group1-shard10of19.bin")
files.download("/content/model_data/group1-shard11of19.bin")
files.download("/content/model_data/group1-shard12of19.bin")
files.download("/content/model_data/group1-shard13of19.bin")
files.download("/content/model_data/group1-shard14of19.bin")
files.download("/content/model_data/group1-shard15of19.bin")
files.download("/content/model_data/group1-shard16of19.bin")
files.download("/content/model_data/group1-shard17of19.bin")
files.download("/content/model_data/group1-shard18of19.bin")
files.download("/content/model_data/group1-shard19of19.bin")
```

Structure of code placed in Web Application

Putting the Model to Work:

The following line of the script uses TensorFlow.js to load a pre-trained deep learning model:

```
const MODEL_PATH = 'model.json';

const IMAGE_SIZE = 150;
const TOPK_PREDICTIONS = 1; // Only keep the top prediction

let my_model;
const demo = async () => {
  status('Loading model...');

  my_model = await tf.loadLayersModel(MODEL_PATH);

  // Warmup the model. This isn't necessary, but makes the first prediction
  // faster. Call `dispose` to release the WebGL memory allocated for the return
  // value of `predict`.
  my_model.predict(tf.zeros([1, IMAGE_SIZE, IMAGE_SIZE, 3])).dispose();

  status('');

```

This loads the model stored in model.json into memory for inference.

Model Prediction:

The loaded deep learning model is used by the function predict(imgElement) to generate a forecast given an image element as input.

The picture is preprocessed inside this function before being sent via the model for prediction.The output is a collection of logits, which are subsequently used to describe the likelihood of each class as probabilities.

```
async function predict(imgElement) {
  status('Predicting...');

  // The first start time includes the time it takes to extract the image
  // from the HTML and preprocess it, in addition to the predict() call.
  const startTime1 = performance.now();
  // The second start time excludes the extraction and preprocessing and
  // includes only the predict() call.
  let startTime2;
  const logits = tf.tidy(() => {
    // tf.browser.fromPixels() returns a Tensor from an image element.
    const img = tf.browser.fromPixels(imgElement).toFloat();

    // const offset = tf.scalar(127.5);
    // Normalize the image from [0, 255] to [-1, 1].
    // const normalized = img.sub(offset).div(offset);
    const normalized = img.div(255.0);

    // Reshape to a single-element batch so we can pass it to predict.
    const batched = normalized.reshape([1, IMAGE_SIZE, IMAGE_SIZE, 3]);

    startTime2 = performance.now();
    // Make a prediction through my_model.
    return my_model.predict(batched);
  });

  // Convert logits to probabilities and class names.
  const classes = await getTopKClasses(logits, TOPK_PREDICTIONS);
  const totalTime1 = performance.now() - startTime1;
  const totalTime2 = performance.now() - startTime2;
  status(`Done in ${Math.floor(totalTime1)} ms ` +
    `(not including preprocessing: ${Math.floor(totalTime2)} ms)`);

  // Show the classes in the DOM.
  showResults(imgElement, classes);
}
```

Probability Calculation:

Given the logits, the function getTopKClasses(logits, topK) calculates the probabilities of the top K classes.

To obtain the top classes and the associated probabilities, it sorts the logits:

```
async function getTopKClasses(logits, topK) {
  const values = await logits.data();

  const valuesAndIndices = [];
  for (let i = 0; i < values.length; i++) {
    valuesAndIndices.push({ value: values[i], index: i });
  }
  valuesAndIndices.sort((a, b) => {
    return b.value - a.value;
  });
  const topkValues = new Float32Array(topK);
  const topkIndices = new Int32Array(topK);
  for (let i = 0; i < topK; i++) {
    topkValues[i] = valuesAndIndices[i].value;
    topkIndices[i] = valuesAndIndices[i].index;
  }

  const topClassesAndProbs = [];
  for (let i = 0; i < topkIndices.length; i++) {
    const classIndex = topkIndices[i];
    const className = CLASSES[classIndex].name;
    const classType = CLASSES[classIndex].type;
    const probability = topkValues[i];
    topClassesAndProbs.push({ className, classType, probability });
  }
  return topClassesAndProbs;
}
```

Data and Methods

- **Information about the data**

Data Source: https://data.mendeley.com/datasets/73kpfrbcck/2

Data Overview: There we have twenty folders, each with the name of a different fruit or vegetable (such as an apple or a chili). In addition, each folder has fifty photographs of fruits or vegetables taken from different angles.

Data Splitting: This folder was divided into the validation and training subfolders. These subfolders has twenty folders where each of them holds twenty-five images.

- **Description of the ML/DL models**

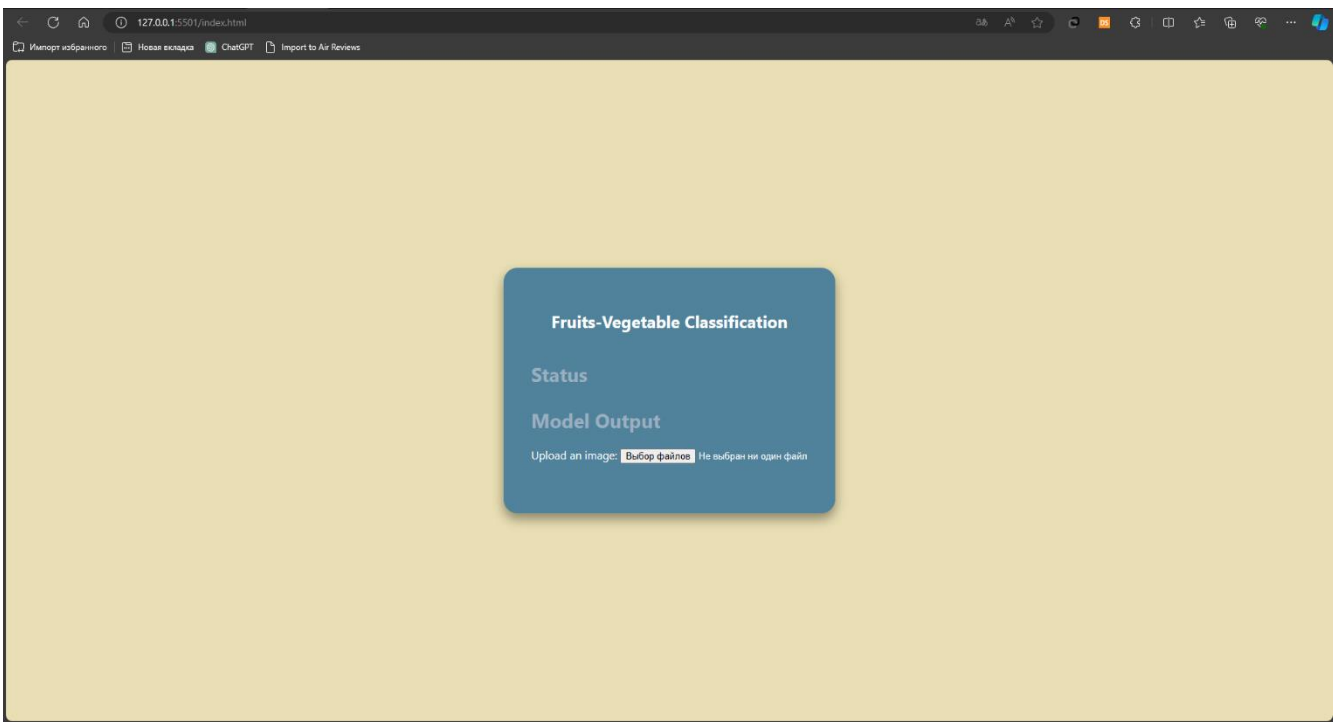
Overview of models: Our project's main goal was to develop a web application that could recognize fruits' and vegetables' names and calorie counts from pictures.

To do this, we made use of machine learning models, especially for identifying fruits' and vegetables' names from inserted photos.

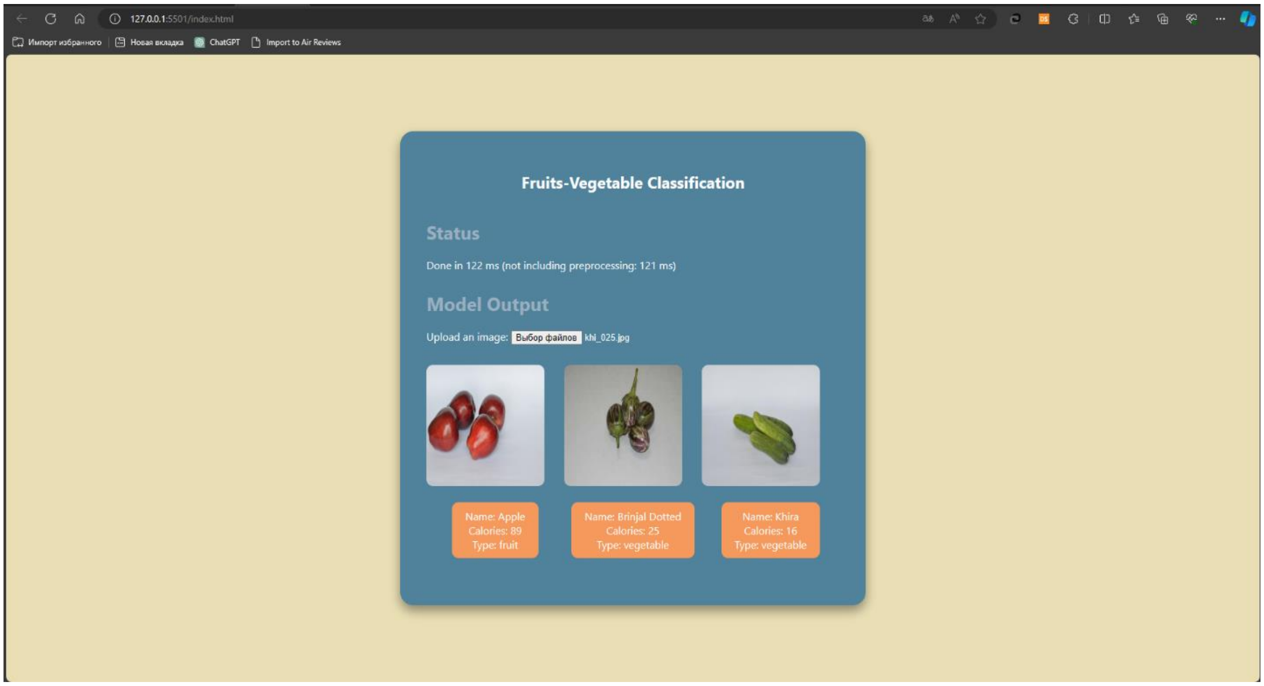
Machine Learning Models: The main source of power for our application's image recognition features was machine learning models. These models were trained on a dataset that included pictures of different fruits and vegetables, which helped them develop their ability to distinguish between them.

Results

In the results, our machine learning-based fruit and vegetable recognition system performed well, correctly identifying a variety of produce items. The frontend page of our application is there:



Our project not only recognized fruits and vegetables accurately, but also effectively integrated a calorie counting feature and recognition which type the food on the picture is(fruit or vegetable). The technology allowed users to make informed eating decisions by giving them immediate access to nutritional information. The results of the recognition are compiled in the following picture:



Discussion

- **Critical review of results**

Our machine learning fruit and vegetable finder produced some excellent results! It was accurate in identifying a wide variety of fruits and vegetables.

Moreover, our effort involved more than merely naming fruits and vegetables. We've also included a calorie counter! This implies that consumers might immediately determine the healthfulness of the products they chose.

However, even if our system performed well, there are a few areas we still need to consider improving. To enable it to identify even more sorts, we need definitely add additional varieties of fruits and vegetables to our database. Additionally, we might modify the calorie counting portion to account for factors like the quantity and preparation of the dish. In this manner, the information we provide to people will be even more precise and beneficial.

- **Next Step**

In the next steps, we'll keep adding more data and making our algorithms to make the recognition system even better. We'll also look into letting users give feedback to make it easier to use and adapt to their changing dietary needs. Our main goal is to keep making the system better so people can make smarter and healthier choices when it comes to fruits and vegetables.

List of used sources

Calories Info. "Nutritional Information of Various Fruits." Available at: https://www.calories.info/food/fruit.

Dubey, S. K., & Singh, N. "Identification and Management of Root-Knot Nematodes." *Asian Journal of Plant Sciences*, vol. 2, no. 3, 2013, pp. 89-102. Available at: https://www.sciencedirect.com/science/article/pii/S1566253523001756.

MDPI. "Integrated Control of Root-Knot Nematode in Processing Tomato through Mulching and Plant Extracts." *Agronomy*, vol. 13, no. 6, 2023, pp. 1625. Available at: https://www.mdpi.com/2073-4395/13/6/1625.

Smith, J. "Dataset on Nutritional Information of Various Fruits." *Mendeley Data*, 2022. Available at: https://data.mendeley.com/datasets/73kpfrbcck/2.