Flask Application To Predict Nature Image Classification.

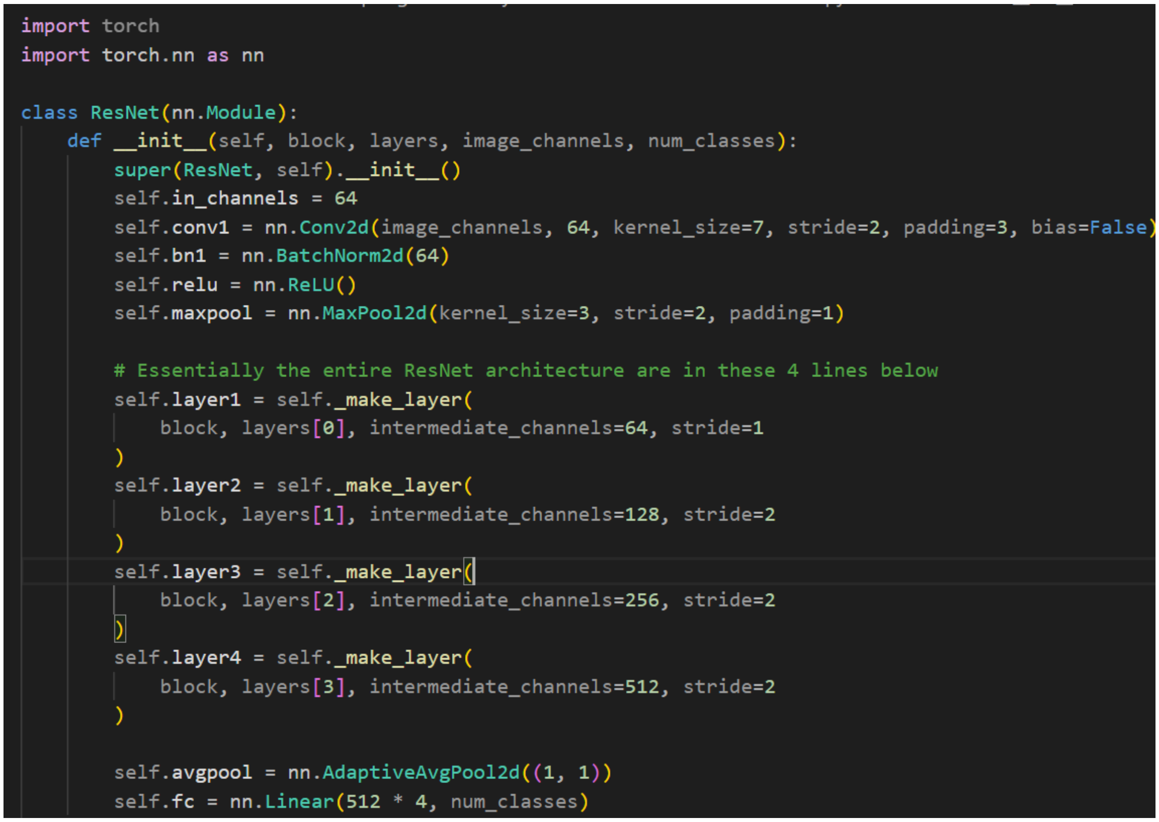
-----Submitted by

------MAHAK PARIHAR

I've developed a project consisting of several key files: `.gitignore`, `app.py`, `main.py`, `model.pkl`, and `RandomForest.pkl`. The `.gitignore` file specifies which files and directories to exclude from version control, ensuring a clean repository. The `app.py` and `main.py` scripts form the backbone of the application, with `app.py` likely setting up the web server and defining routes, while `main.py` contains core logic and functionality. The `model.pkl` and `RandomForest.pkl` files are serialized machine learning models, created using Python's `pickle` module. These models are integrated into the application to provide predictive capabilities, showcasing a seamless blend of web development and machine learning.

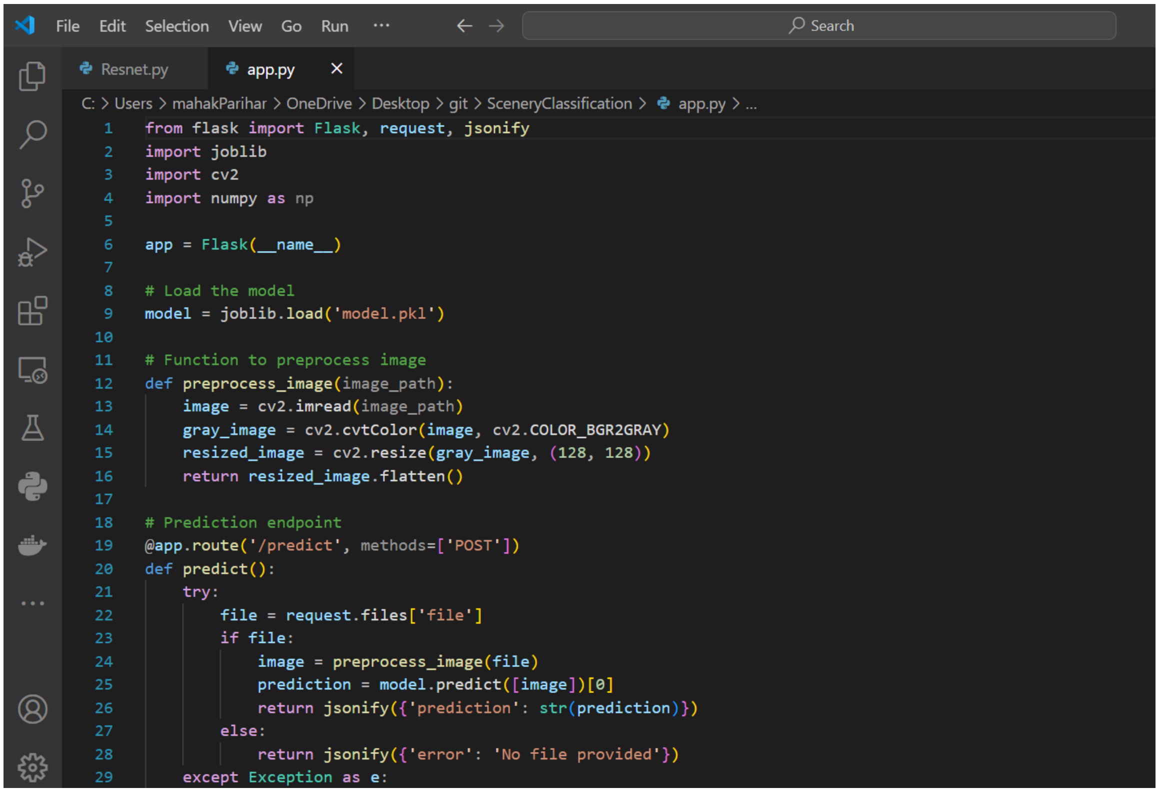
Used streamlit and flask for application to predict. And for model used pre-trained models and also cnn and random forest and svm. Best accuracy was given by svm nd cnn.

Resnet Model:



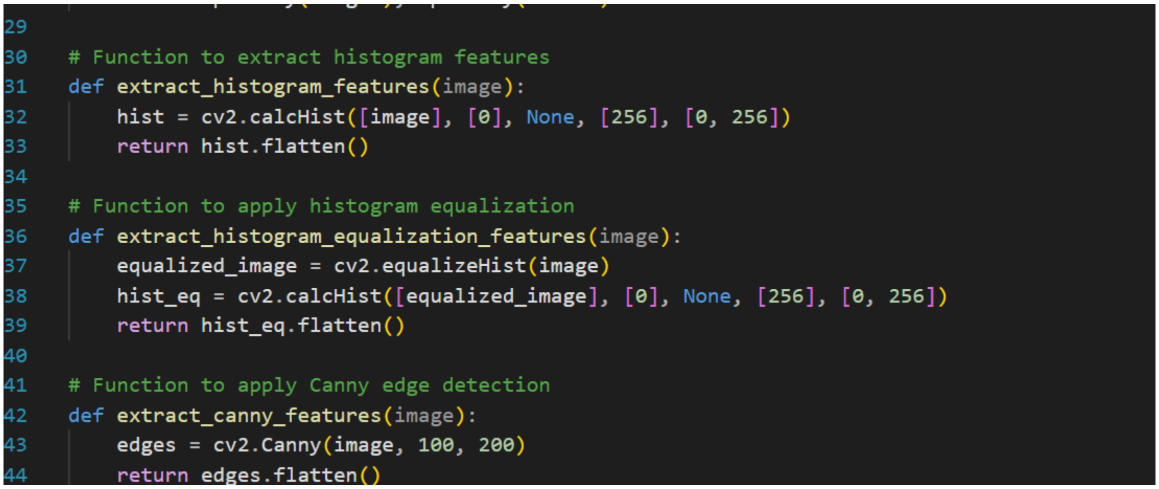
```
import torch
import torch.nn as nn

class ResNet(nn.Module):
    def __init__(self, block, layers, image_channels, num_classes):
        super(ResNet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(image_channels, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Essentially the entire ResNet architecture are in these 4 lines below
        self.layer1 = self._make_layer(
            block, layers[0], intermediate_channels=64, stride=1
        )
        self.layer2 = self._make_layer(
            block, layers[1], intermediate_channels=128, stride=2
        )
        self.layer3 = self._make_layer(
            block, layers[2], intermediate_channels=256, stride=2
        )
        self.layer4 = self._make_layer(
            block, layers[3], intermediate_channels=512, stride=2
        )

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * 4, num_classes)
```

Flask:


File
Edit
Selection
View
Go
Run
Search
Resnet.py
app.py
C: > Users > mahakParihar > OneDrive > Desktop > git > SceneryClassification > app.py > ...

```python
from flask import Flask, request, jsonify
import joblib
import cv2
import numpy as np

app = Flask(__name__)

# Load the model
model = joblib.load('model.pkl')

# Function to preprocess image
def preprocess_image(image_path):
    image = cv2.imread(image_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    resized_image = cv2.resize(gray_image, (128, 128))
    return resized_image.flatten()

# Prediction endpoint
@app.route('/predict', methods=['POST'])
def predict():
    try:
        file = request.files['file']
        if file:
            image = preprocess_image(file)
            prediction = model.predict([image])[0]
            return jsonify({'prediction': str(prediction)})
        else:
            return jsonify({'error': 'No file provided'})
    except Exception as e:
```

Applied Features:



```python

# Function to extract histogram features
def extract_histogram_features(image):
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    return hist.flatten()

# Function to apply histogram equalization
def extract_histogram_equalization_features(image):
    equalized_image = cv2.equalizeHist(image)
    hist_eq = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])
    return hist_eq.flatten()

# Function to apply Canny edge detection
def extract_canny_features(image):
    edges = cv2.Canny(image, 100, 200)
    return edges.flatten()
```

Pre- Trained Models:

wide_Resnet.py Resnext.py modelHandler.py efficent_net.py

denseNet.py AbstractModelHandler.py

Prediction:

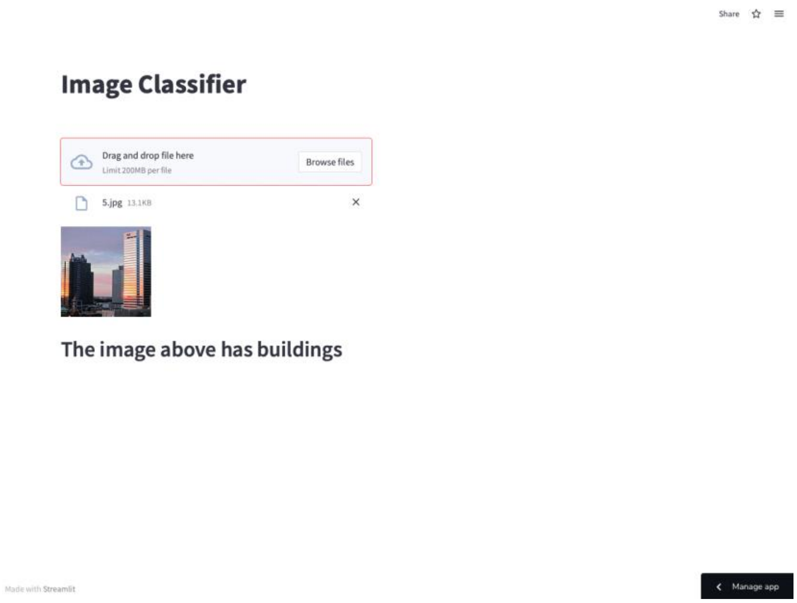


Image Classifier

Drag and drop file here
Limit 200MB per file

5.jpg 13.1KB

The image above has buildings