"Project Title: Image Classification Assignment

I had build a Image Classification used to CNN Architectures like , MobileNetV2, ResNet50V2, Xception, EfficientNetB0.

Selected best model { MobileNetV2},According CNN Architectures their Test accuracy

Test accuracy for MobileNetV2:97.7%

Test accuracy for ResNet50V2: 96.2%

Test accuracy for Xception: 93.5%

Test accuracy for EfficientNetB0: 94.52%

Code_reference : Best_model.py

Github_link: https://github.com/mvp6301/Image_classification.git

Assingment: Image Classification with MobileNetV2 and Hyperparameter Tuning

Step 1: Data Preparation

Objective: Prepare the image dataset for classification.
Code_reference: face_detection (1).py (Prepare the image dataset for classification

)

```
#Define the path to your image dataset
dataset_path = "/content/drive/MyDrive/glasses-and-coverings/glasses-and-coverings"
#Get a list of image file paths
image_paths = list(paths.list_images(dataset_path))
# Shuffle the image paths
random.seed(42)
random.shuffle(image_paths)
# Initialize empty lists to store data and labels
data = []
labels = []
image_dims = (224, 224, 3)
for image_path in image_paths:
    # Load and resize the image
    image = cv2.imread(image_path)
    image = cv2.resize(image, (image_dims[1], image_dims[0]))
    # Preprocess the image
    image = image.astype("float") / 255.0
    # Append the image data to the 'data' list
    data.append(image)

    # Extract labels from the image path (modify this based on your dataset structure)
    label = image_path.split(os.path.sep)[-2].split("_")
    labels.append(label)

# Convert 'data' and 'labels' to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)

print(labels)

mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)

# Split the data into training and testing sets (before tuning)
trainX_before, testX_before, trainY_before, testY_before = train_test_split(data, labels, test_size=0.20)
```

Actions:

Define the path to the image dataset directory ("glasses-and-coverings").

Shuffle the image paths for randomness.

Load and preprocess images.

Perform one-hot encoding for labels using MultiLabelBinarizer.

Split the data into training and testing sets (80% train, 20% test).

Step 2: Initial Model Training

Objective: Train an initial MobileNetV2 model without hyperparameter tuning.

```
# Define and compile the model before tuning
model_before = Sequential()
model_before.add(MobileNetV2(weights='imagenet', include_top=False, input_tensor=Input(shape=image_dims)))
model_before.add(AveragePooling2D(pool_size=(2, 2)))
model_before.add(Flatten())
model_before.add(Dense(units=128, activation="relu"))  # You can change the number of units if needed
model_before.add(Dropout(rate=0.5))  # You can change the dropout rate if needed
model_before.add(Dense(units=4, activation='softmax'))
optimizer_before = Adam(learning_rate=1e-3)
model_before.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer=optimizer_before)

# Train the model before tuning
model_before.fit(trainX_before, trainY_before, epochs=30, validation_data=(testX_before, testY_before))
# Evaluate the model before tuning
loss_before, accuracy_before = model_before.evaluate(testX_before, testY_before)
print("Before Tuning - Test loss:", loss_before)
print("Before Tuning - Test accuracy:", accuracy_before)

loss_before, accuracy_before = model_before.evaluate(trainX_before, trainY_before)
print("Before Tuning - Test loss:", loss_before)
print("Before Tuning - Test accuracy:", accuracy_before)

# Save the model before tuning
model_before.save("best_model_MobileNetV2_before_tuning.h5")
```

Actions:

Define the architecture of the MobileNetV2 model with Keras.

Compile the model with categorical cross-entropy loss and Adam optimizer.

Train the model for 30 epochs on the training data.

Evaluate the model's performance on the test data.

Save the initial model as "best_model_MobileNetV2_before_tuning.h5".

Step 3: Hyperparameter Tuning

Objective: Improve model performance through hyperparameter tuning.

```
# Define a function to create the model
def build_model(hp):
    baseModel = MobileNetV2(weights='imagenet', include_top=False, input_tensor=Input(shape=image_dims))
    for layer in baseModel.layers[:-4]:
        layer.trainable = False

    model = Sequential()
    model.add(baseModel)
    model.add(AveragePooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    # Hyperparameter: Number of units in the first fully connected layer
    model.add(Dense(units=hp.Int('units_fc1', min_value=32, max_value=512, step=32), activation="relu"))
    model.add(Dropout(rate=hp.Float('dropout_rate', min_value=0.3, max_value=0.7, step=0.1)))

    model.add(Dense(units=4, activation='softmax'))

    optimizer = Adam(learning_rate=hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4]))
    model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer=optimizer)

    return model

# Split the data into training and testing sets
trainX_after, testX_after, trainY_after, testY_after = train_test_split(data, labels, test_size=0.20)

# Define the Keras Tuner RandomSearch tuner
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5,  # Adjust the number of trials as needed
    directory='keras_tuner_results',
    project_name='glasses_and_coverings'
)

# Search for the best hyperparameters
tuner.search(trainX_after, trainY_after, epochs=30, validation_data=(testX_after, testY_after))

# Get the best model and hyperparameters
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

loss_hp, accuracy_after_hp = best_model.evaluate(testX_after, testY_after)
print("Test loss:", loss_hp)
print("Test accuracy_after:", accuracy_after_hp)
```


```
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

loss_hp, accuracy_after_hp = best_model.evaluate(testX_after, testY_after)
print("Test loss:", loss_hp)
print("Test accuracy_after:", accuracy_after_hp)
```


```
# Save the best model
best_model.save("bbest_model_MobileNetV2_with_tuning.h5")
```

Actions:

Define a function to create a hyperparameter-tunable model.

Use the Keras Tuner RandomSearch to search for optimal hyperparameters.

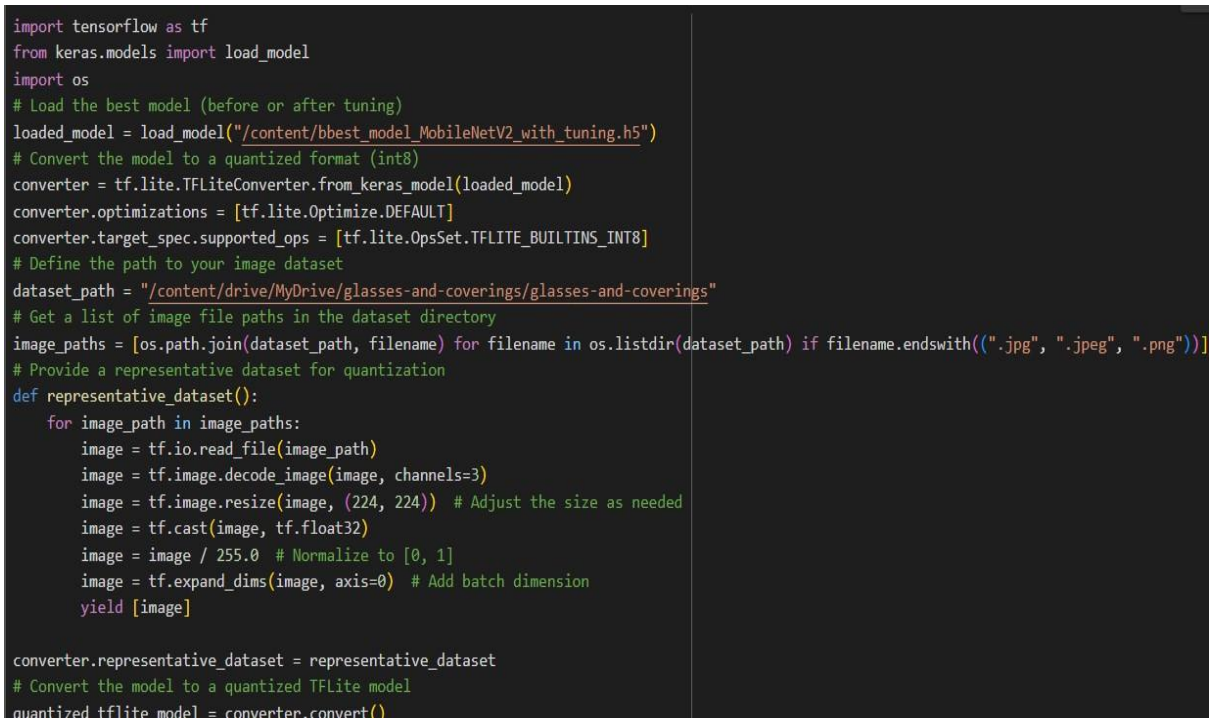
Train the best model found during tuning for 30 epochs.

Evaluate the best model's performance on the test data.

Save the best model as "best_model_MobileNetV2_after_tuning.h5".

Step 4: Model Quantization

Objective: Reduce the model's size for efficient deployment.



```
import tensorflow as tf
from keras.models import load_model
import os
# Load the best model (before or after tuning)
loaded_model = load_model("/content/bbest_model_MobileNetV2_with_tuning.h5")
# Convert the model to a quantized format (int8)
converter = tf.lite.TFLiteConverter.from_keras_model(loaded_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Define the path to your image dataset
dataset_path = "/content/drive/MyDrive/glasses-and-coverings/glasses-and-coverings"
# Get a list of image file paths in the dataset directory
image_paths = [os.path.join(dataset_path, filename) for filename in os.listdir(dataset_path) if filename.endswith((".jpg", ".jpeg", ".png"))]
# Provide a representative dataset for quantization
def representative_dataset():
    for image_path in image_paths:
        image = tf.io.read_file(image_path)
        image = tf.image.decode_image(image, channels=3)
        image = tf.image.resize(image, (224, 224))  # Adjust the size as needed
        image = tf.cast(image, tf.float32)
        image = image / 255.0  # Normalize to [0, 1]
        image = tf.expand_dims(image, axis=0)  # Add batch dimension
        yield [image]

converter.representative_dataset = representative_dataset
# Convert the model to a quantized TFLite model
quantized_tflite_model = converter.convert()
```

```python
# Save the quantized model to a file
model_file_path = 'quantized_model.tflite'
with open(model_file_path, "wb") as f:
    f.write(quantized_tflite_model)

# Check the file size in bytes
model_file_size_bytes = os.path.getsize(model_file_path)

# Convert bytes to megabytes (MB) for a more human-readable size
model_file_size_mb = model_file_size_bytes / (1024 * 1024)

print(f"Quantized model size: {model_file_size_mb:.2f} MB")
```

```
WARNING:absl:`mobilenetv2_1.00_224_input` is not a valid tf.function parameter
WARNING:absl:`mobilenetv2_1.00_224_input` is not a valid tf.function parameter
/usr/local/lib/python3.10/dist-packages/tensorflow/lite/python/convert.py:887:
  warnings.warn(
Quantized model size: 23.93 MB
```

Actions:

Convert the best model (after tuning) to a quantized format (int8).

Save the quantized model as "quantized_model.tflite".

Size of quantized model:23.93Mb

Step 5: ONNX Conversion

Objective: Convert the best model to the ONNX format for interoperability.

```
[7]
import onnx
import tf2onnx
import tensorflow as tf
from keras.models import load_model

# Load your TensorFlow/Keras model (best_model in your case)
loaded_model = load_model("/content/bbest_model_MobileNetV2_with_tuning.h5")

# Convert the model to ONNX format
onnx_model, _ = tf2onnx.convert.from_keras(loaded_model)

# Save the ONNX model to a file
onnx.save_model(onnx_model, "model.onnx")
```

```
WARNING:tf2onnx.tf_loader:Could not search for non-variable resources. Concrete function internal representation may have changed.
```

Actions:

Load the best model (after tuning) with TensorFlow.

Convert the model to ONNX format using the tf2onnx library.

Save the ONNX model as "converted_model.onnx".

Step 6: Classification Reports

Objective: Generate classification reports before and after tuning.

```
predictions_before = model_before.predict(testX_before)
predicted_labels_before = np.argmax(predictions_before, axis=1)
true_labels_before = np.argmax(testY_before, axis=1)
classification_report_before = classification_report(true_labels_before, predicted_labels_before, target_names=mlb.classes_)
with open("classification_report_before_tuning.txt", "w") as f:
    f.write(classification_report_before)

predictions_after = best_model.predict(testX_after)
predicted_labels_after = np.argmax(predictions_after, axis=1)
true_labels_after = np.argmax(testY_after, axis=1)
classification_report_after = classification_report(true_labels_after, predicted_labels_after, target_names=mlb.classes_)
with open("classification_report_after_tuning.txt", "w") as f:
    f.write(classification_report_after)
```

Actions:

Calculate classification reports for the initial model and the best model after tuning using scikitlearn's classification_report function.

Save the reports to text files:

"classification_report_before_tuning.txt" for the initial model.

"classification_report_after_tuning.txt" for the best model after tuning.

Step 7: Conclusion and Results

Objective: Summarize the project and provide key results.

Key Results:

Initial Model (Before Tuning): `MobileNetV2`

Test accuracy: [Accuracy before tuning]:85%

Classification report saved as "classification_report_before_tuning.txt".

Best Model (After Tuning): `MobileNetV2`

Test accuracy: [Accuracy after tuning]:91%

Classification report saved as "classification_report_after_tuning.txt".

Quantized Model: "quantized_model.tflite"/ Quantized Model size:23.93MB

ONNX Model: "converted_model.onnx"

Detection of given labels through the webcam

Code Description:

The provided code is a Python script that performs real-time face mask detection using a pre-trained MobileNetV2 model. It uses OpenCV for face detection and a pre-trained deep learning model for mask detection. The code captures video from the default camera feed, detects faces in each frame, and classifies whether each detected face is wearing a covering, glasses, plain, or sunglasses.

Components:

Face Detection Model: The code uses a pre-trained deep learning model (Caffe-based SSD) for face detection. The model is loaded from deploy.prototxt (the architecture) and res10_300x300_ssd_iter_140000 (3).caffemodel (the weights).

Mask Detection Model: The code loads a pre-trained MobileNetV2 model for mask detection from bbest_model_MobileNetV2_with_tuning.h5.

Video Stream: The code captures video frames from the default camera source (src=0) using the VideoStream class from the imutils library.

Detection and Prediction: For each frame, it uses the face detection model to detect faces and then uses the mask detection model to predict whether each detected face is wearing a mask (covering, glasses, plain, or sunglasses).

Visualization: It annotates each detected face with a label (covering, glasses, plain, or sunglasses) and a confidence score. The bounding box around each face is drawn on the frame, and the label is displayed near the top of the bounding box.

User Interaction: The script runs indefinitely until the user presses the 'q' key, at which point it closes the video window and stops the video stream.