The University of Azad Jammu and Kashmir, Muzaffarabad

Department of Software Engineering

Semester Project

Automatic Machine Fault Detection and Recognition using Computer Vision

Submitted To:	Engr. Ahmad Khawaja
Submitted By:	Noor e Muneeba (2020-SE-10) Basma Yasmine (2020-SE-07) Minahil Sijjad (2020-SE-35)
Course Title:	Computer Vision

Table of Contents:

List of Figures:

Installing Required Libraries

```
!pip install librosa resampy
```

Code Explanation:

The command installs the Python package called "**py7zr**" into the current Python environment. This package provides functionality for working with 7z archives in Python.

```
pip install py7zr
```

Code Explanation:

The command installs the Python package called "**py7zr**" into the current Python environment. This package provides functionality for working with 7z archives in Python.

Importing Required Libraries

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, roc_curve, auc, confusion_matrix, classification_report,
precision_recall_curve, average_precision_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import py7zr
import librosa
import os
```

This code imports several Python libraries and modules:

1. **`import numpy as np`:** Imports the NumPy library with the alias `np`, commonly used for numerical computing in Python.

2. **`import tensorflow as tf`:** Imports the TensorFlow library with the alias `tf`, an open-source machine learning framework.

3. **`from tensorflow.keras import layers, models`:** Imports specific modules (`layers` and `models`) from TensorFlow's Keras API for building neural networks.

4. **`from tensorflow.keras.layers import Dropout`:** Imports the `Dropout` layer from TensorFlow's Keras API, used for regularization in neural networks.

5. **`from sklearn.model_selection import train_test_split`**: Imports the `train_test_split` function from scikit-learn, used to split data into training and testing sets.

6. **`from sklearn.metrics import f1_score, roc_curve, auc, confusion_matrix, classification_report, precision_recall_curve, average_precision_score`:** Imports various evaluation metrics from scikit-learn, including F1 score, ROC curve, AUC, confusion matrix, classification report, precision-recall curve, and average precision score.

7. **`from sklearn.preprocessing import LabelEncoder`**: Imports the `LabelEncoder` class from scikit-learn, used for encoding categorical labels as integers.

8. **`import matplotlib.pyplot as plt`:** Imports the `pyplot` module from Matplotlib, commonly used for creating visualizations in Python.

9. **`import py7zr`:** Imports the `py7zr` module, a Python library for reading and writing 7z format archives.

10. **`import librosa`:** Imports the `librosa` library, used for audio processing tasks such as feature extraction and beat tracking.

11. **`import os`:** Imports the `os` module, providing functions for interacting with the operating system, such as working with files and directories.

File Extraction

```
# Path to your .7z file
file_path = r'C:\Users\Hp\Documents\Semester 7\Computer Vision\samples.7z'
```

```
# Directory to extract the contents of the .7z file
extracted_dir = r'C:\Users\Hp\Documents\Semester 7\Computer Vision'
```

```
# Create the directory if it doesn't exist
os.makedirs(extracted_dir, exist_ok=True)
```

```
# Extract the contents of the .7z file
with py7zr.SevenZipFile(file_path, mode='r') as z:
    z.extractall(path=extracted_dir)

print("Extraction completed.")
```

Output:

The output of the executed code is:

```
Extraction completed.
```

Figure 1- Extraction

Code Explanation:

1. **`file_path = r'C:\Users\Hp\Documents\Semester 7\Computer Vision\samples.7z'`:** Specifies the path to the .7z file containing the dataset.

2. **`extracted_dir = r'C:\Users\Hp\Documents\Semester 7\Computer Vision'`:** Specifies the directory where the contents of the .7z file will be extracted.

3. **`os.makedirs(extracted_dir, exist_ok=True)`:** Creates the directory specified by `extracted_dir` if it doesn't already exist, ensuring that the extraction destination is available.

4. **`with py7zr.SevenZipFile(file_path, mode='r') as z:`:** Opens the .7z file specified by `file_path` in read mode using the `py7zr` library, allowing its contents to be extracted.

5. **`z.extractall(path=extracted_dir)`:** Extracts all contents of the .7z file (`z`) to the directory specified by `extracted_dir`, effectively decompressing the archive and storing its contents in the designated directory.

6. **`print("Extraction completed.")`:** Prints a message to indicate that the extraction process has been completed successfully.

Loading and Dividing Dataset:

```
def load_data(test_size=0.2, chunk_duration=1):
    x, y = [], []
    for file in os.listdir('/content/drive/MyDrive/cv_project/samples'):
        # Load audio file
            audio, sample_rate = librosa.load('/content/drive/MyDrive/cv_project/samples/' + file,
res_type='kaiser_fast')
```

```
        # Calculate number of chunks
        num_chunks = int(np.ceil(len(audio) / (sample_rate * chunk_duration)))
```

```
        # Extract features from each chunk
        for i in range(num_chunks):
            start = int(i * sample_rate * chunk_duration)
            end = min(len(audio), int((i + 1) * sample_rate * chunk_duration))
            chunk_audio = audio[start:end]
```

```
    # Extract features from audio chunk
            feature = extract_features_from_audio(chunk_audio, sample_rate)
            x.append(feature)
```

```
            # Extract class label from the file name
            class_label = file.split('(')[0]  # Assuming the class label is before the first '-'
            y.append(class_label)
```

```
    # Encode the labels
    encoder = LabelEncoder()
    y = encoder.fit_transform(y)

    return train_test_split(np.array(x), y, test_size=test_size, random_state=42)
```

Code Explanation:

This function loads and preprocesses the audio data:

1. **Function Definition:** Defines a function named `load_data` with optional parameters `test_size` and `chunk_duration`.

2. **Empty Lists:** Initializes two empty lists `x` and `y` to store features and labels, respectively.

3. **Iterate Over Files:** Iterates through each file in the directory specified by the path `r'C:\Users\Hp\Documents\Semester 7\Computer Vision\samples'`.

4. **Load Audio:** Loads each audio file using `librosa.load()` and specifies a resampling method (`res_type='kaiser_fast'`). The audio data and its sample rate are assigned to the variables `audio` and `sample_rate`, respectively.

5. **Calculate Number of Chunks:** Calculates the number of chunks needed to split the audio file based on the specified `chunk_duration`.

6. **Extract Features:** Iterates through each chunk of the audio file, extracts features using a helper function `extract_features_from_audio()`, and appends them to the list `x`.

7. **Extract Class Label:** Extracts the class label from the file name. Assumes that the class label is located before the first `'('` character in the file name.

8. **Encode Labels:** Uses `LabelEncoder()` from scikit-learn to encode the class labels (`y`) into integer values.

9. **Train-Test Split:** Splits the data into training and testing sets using `train_test_split()` from scikit-learn. Returns the training and testing data along with their corresponding labels.

Features Extraction

```
def extract_features_from_audio(audio, sample_rate, mfcc=True, chroma=True, mel=True, zero_crossing_rate=True, spectral_bandwidth=True, statistic='square_root_sum'):
    result = []

    if mfcc:
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40, n_fft=1024)
        if statistic == 'square_root_sum':
            mfccs = np.sqrt(np.sum(np.square(mfccs), axis=1))
        # Add other cases for different statistics if needed
        result.append(mfccs)

    if chroma:
        stft = np.abs(librosa.stft(audio, n_fft=1024))
        chroma = librosa.feature.chroma_stft(S=stft, sr=sample_rate)
        if statistic == 'square_root_sum':
            chroma = np.sqrt(np.sum(np.square(chroma), axis=1))
```

```
        # Add other cases for different statistics if needed
        result.append(chroma)

    if mel:
        mel = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_fft=1024)
        if statistic == 'square_root_sum':
            mel = np.sqrt(np.sum(np.square(mel), axis=1))
```

```
# Add other cases for different statistics if needed

        result.append(mel)

    if zero_crossing_rate:
        zcr = librosa.feature.zero_crossing_rate(audio)
        if statistic == 'square_root_sum':
            zcr = np.sqrt(np.sum(np.square(zcr), axis=1))
```

```
        # Add other cases for different statistics if needed

        result.append(zcr)

    if spectral_bandwidth:
        spec_bw = librosa.feature.spectral_bandwidth(y=audio, sr=sample_rate)
        if statistic == 'square_root_sum':
            spec_bw = np.sqrt(np.sum(np.square(spec_bw), axis=1))
```

```
        # Add other cases for different statistics if needed
        result.append(spec_bw)

    return np.hstack(result)
```

Output:

Returns a concatenated array of the extracted features.

Code Explanation:

This function extracts features from audio data:

1. **Function Definition:** Defines a function named `extract_features_from_audio` that takes `audio`, `sample_rate`, and several optional parameters as inputs.

2. **Feature Extraction:** Depending on the specified parameters (`mfcc`, `chroma`, `mel`, `zero_crossing_rate`, `spectral_bandwidth`), different audio features are extracted using functions from the `librosa` library.

3. **MFCC (Mel-frequency cepstral coefficients):** If `mfcc` is `True`, it computes MFCC features using `librosa.feature.mfcc()` with specified parameters like the number of MFCCs (`n_mfcc`) and the length of the FFT window (`n_fft`). It then applies a statistical operation (e.g., square root sum) if specified.

4. **Chroma:** If `chroma` is `True`, it computes chroma features using `librosa.feature.chroma_stft()`. Similar to MFCC, it applies a statistical operation if specified.

5. **Mel Spectrogram:** If `mel` is `True`, it computes the Mel spectrogram using `librosa.feature.melspectrogram()`. Once again, it applies a statistical operation if specified.

6. **Zero Crossing Rate:** If `zero_crossing_rate` is `True`, it computes the zero-crossing rate using `librosa.feature.zero_crossing_rate()` and applies a statistical operation if specified.

7. **Spectral Bandwidth:** If `spectral_bandwidth` is `True`, it computes the spectral bandwidth using `librosa.feature.spectral_bandwidth()` and applies a statistical operation if specified.

8. **Result:** It returns a concatenated array of all the extracted features.

Model Building

```python
def create_model_with_dropout(input_shape, num_classes, dropout_rate=0.5):
    model = models.Sequential()
    model.add(layers.Conv1D(64, 3, activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(128, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(256, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(256, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(Dropout(dropout_rate))  # Adding dropout layer
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
```

Code Explanation:

This function creates a convolutional neural network (CNN) model with dropout regularization.

- **Parameters**
  - input_shape: Specifies the shape of the input data.
  - num_classes: Number of output classes in the classification task.
  - dropout_rate: Dropout rate, a hyperparameter controlling the dropout rate.

- **Architecture**

The architecture of the model is as follows:

1. **Conv1D(64, 3, activation='relu'):** 1D convolutional layer with 64 filters of size 3 and ReLU activation function.
2. **MaxPooling1D(2):** Max pooling layer with pool size 2.
3. **Conv1D(128, 3, activation='relu'):** Another convolutional layer with 128 filters of size 3 and ReLU activation.
4. **MaxPooling1D(2):** Max pooling layer.
5. **Conv1D(256, 3, activation='relu'):** Another convolutional layer with 256 filters of size 3 and ReLU activation.
6. **MaxPooling1D(2):** Max pooling layer.

7. **Conv1D(256, 3, activation='relu'):** Another convolutional layer with 256 filters of size 3 and ReLU activation.
8. **MaxPooling1D(2):** Max pooling layer.
9. **Flatten():** Flattens the output of the convolutional layers.
10. **Dense(512, activation='relu'):** Fully connected layer with 512 units and ReLU activation.
11. **Dropout(dropout_rate):** Dropout layer with the specified dropout rate.
12. **Dense(num_classes, activation='softmax'):** Output layer with softmax activation function for multi-class classification.

Model loading, Compiling and Training

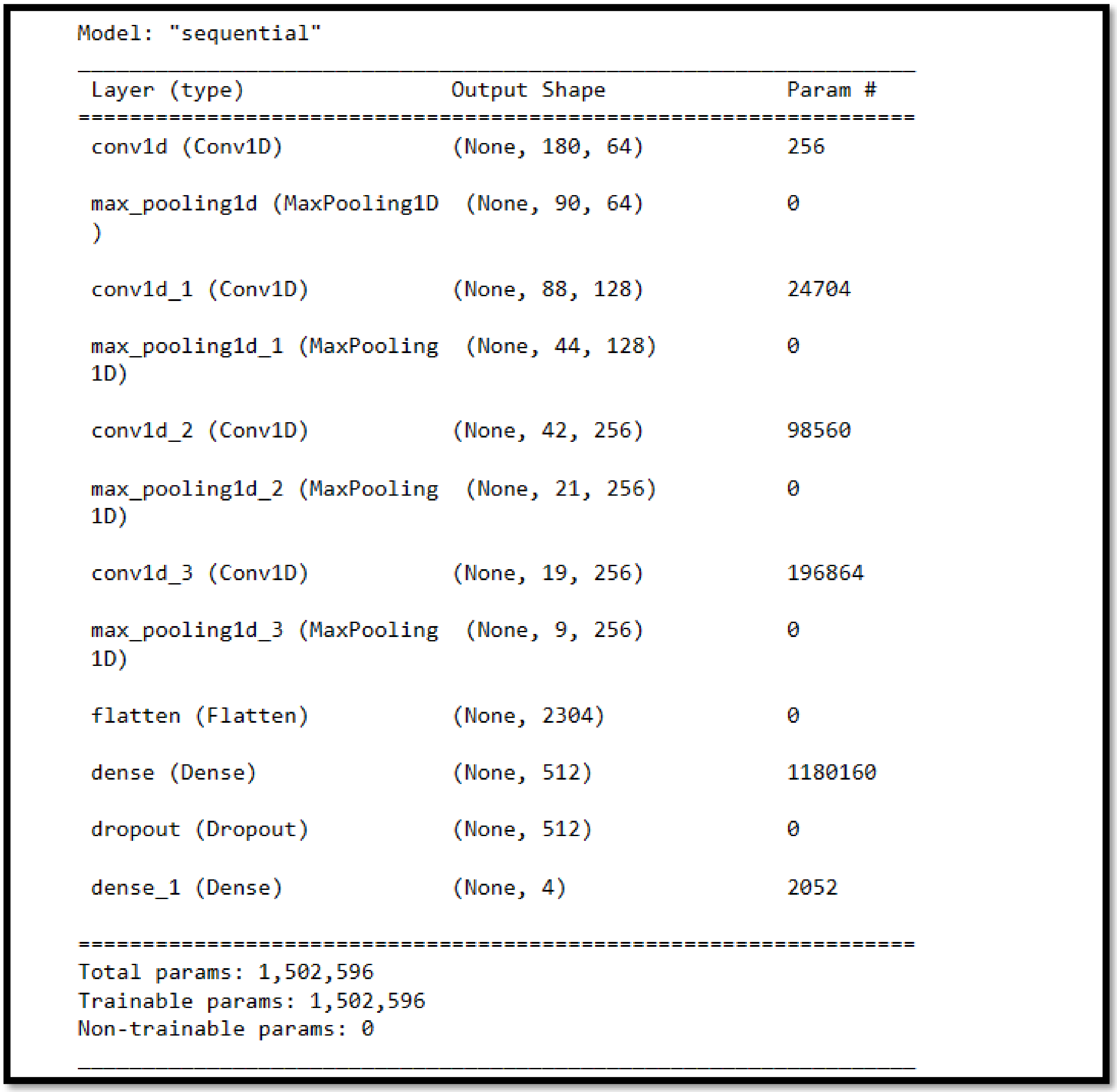
```
# Load and split the dataset
X_train, X_test, y_train, y_test = load_data(test_size=0.2)
```

This line of code loads the dataset and splits it into training and testing sets using the `**load_data**` function defined earlier. The `**test_size**` parameter specifies the proportion of the dataset to include in the test split, which is set to 20% in this case.

```
# Update input_shape to have 2 dimensions (samples, time_steps)
input_shape = (X_train.shape[1], 1)
num_classes=4
# Create the model with dropout
model_with_dropout = create_model_with_dropout(input_shape=(X_train.shape[1], 1),
num_classes=num_classes)

# Compile the model
model_with_dropout.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
# Display model summary
model_with_dropout.summary()
```

Output:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 180, 64)           256

 max_pooling1d (MaxPooling1D  (None, 90, 64)           0
 )

 conv1d_1 (Conv1D)           (None, 88, 128)           24704

 max_pooling1d_1 (MaxPooling  (None, 44, 128)          0
 1D)

 conv1d_2 (Conv1D)           (None, 42, 256)           98560

 max_pooling1d_2 (MaxPooling  (None, 21, 256)          0
 1D)

 conv1d_3 (Conv1D)           (None, 19, 256)           196864

 max_pooling1d_3 (MaxPooling  (None, 9, 256)           0
 1D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 1,502,596
Trainable params: 1,502,596
Non-trainable params: 0
_________________________________________________________________
```

Figure 2- Model Summary

Code Explanation:

In these lines:

1. **`input_shape = (X_train.shape[1], 1)`:** Reshapes the input shape to have 2 dimensions `(samples, time_steps)` by adding a third dimension of size 1.

2. **`num_classes=4`:** Defines the number of classes in the classification problem.

3. **`model_with_dropout = create_model_with_dropout(input_shape=(X_train.shape[1], 1), num_classes=num_classes)`:** Creates a convolutional neural network model with dropout layers using the specified input shape and number of classes.

4. **`model_with_dropout.compile(...)`:** Compiles the model with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy metric.

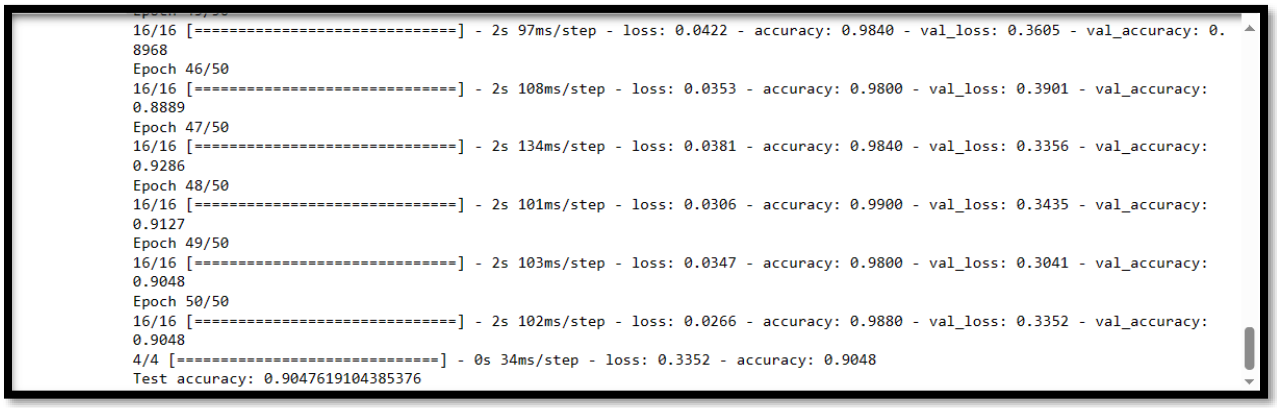
5. **`model_with_dropout.summary()`:** Displays a summary of the model architecture, including the layers, output shapes, and trainable parameters.

Model Training and Evaluation:

```
# Train the model
model_with_dropout.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model_with_dropout.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

Output:



```
16/16 [==============================] - 2s 97ms/step - loss: 0.0422 - accuracy: 0.9840 - val_loss: 0.3605 - val_accuracy: 0.
8968
Epoch 46/50
16/16 [==============================] - 2s 108ms/step - loss: 0.0353 - accuracy: 0.9800 - val_loss: 0.3901 - val_accuracy:
0.8889
Epoch 47/50
16/16 [==============================] - 2s 134ms/step - loss: 0.0381 - accuracy: 0.9840 - val_loss: 0.3356 - val_accuracy:
0.9286
Epoch 48/50
16/16 [==============================] - 2s 101ms/step - loss: 0.0306 - accuracy: 0.9900 - val_loss: 0.3435 - val_accuracy:
0.9127
Epoch 49/50
16/16 [==============================] - 2s 103ms/step - loss: 0.0347 - accuracy: 0.9800 - val_loss: 0.3041 - val_accuracy:
0.9048
Epoch 50/50
16/16 [==============================] - 2s 102ms/step - loss: 0.0266 - accuracy: 0.9880 - val_loss: 0.3352 - val_accuracy:
0.9048
4/4 [==============================] - 0s 34ms/step - loss: 0.3352 - accuracy: 0.9048
Test accuracy: 0.9047619104385376
```

Figure 3- Test accuracy

Code Explanation:

In these lines:

1. **`model_with_dropout.fit(...)`:** Trains the model on the training data (`X_train` and `y_train`) for 50 epochs with a batch size of 32. The validation data (`X_test` and `y_test`) is used for evaluating the model's performance during training.

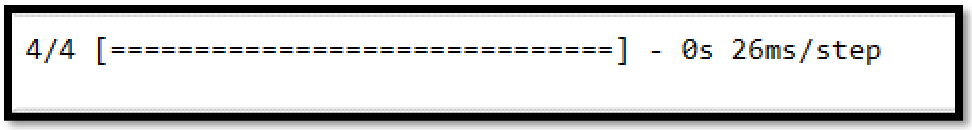
2. **`test_loss, test_acc = model_with_dropout.evaluate(X_test, y_test)`:** Evaluates the trained model on the test data (`X_test` and `y_test`) to calculate the test loss and accuracy.

3. **`print('Test accuracy:', test_acc)`:** Prints the test accuracy obtained from evaluating the model on the test dataset.

Model Evaluation and Visualization

```
# Obtain predictions from the trained model
y_pred_probabilities = model_with_dropout.predict(X_test)
y_pred_labels = np.argmax(y_pred_probabilities, axis=1)
```

Output:



```
4/4 [==============================] - 0s 26ms/step
```

Figure 4- y_pred_label

Code Explanation:

These lines perform the following tasks:

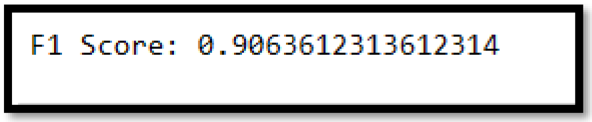
1. **`model_with_dropout.predict(X_test)`:** Generates predictions on the test data (`X_test`) using the trained model (`model_with_dropout`), resulting in predicted probabilities for each class.

2. **`np.argmax(y_pred_probabilities, axis=1)`:** Converts the predicted probabilities (`y_pred_probabilities`) into class labels by selecting the index with the highest probability along the axis 1 (column-wise). These predicted class labels are stored in `y_pred_labels`.

F1 Score:

```
# Calculate F1 score
f1 = f1_score(y_test, y_pred_labels, average='weighted')
print('F1 Score:', f1)
```

Output:



```
F1 Score: 0.9063612313612314
```

Figure 5- F1 Score

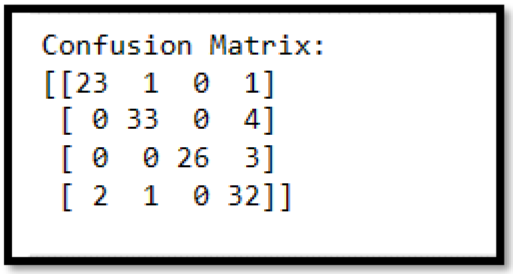
Code Explanation:

This code calculates the F1 score, which is a measure of a model's accuracy, using the true labels **(`y_test`)** and the predicted labels **(`y_pred_labels`).** The F1 score is computed with the `f1_score` function from scikit-learn, specifying the **`average='weighted'`** parameter to compute the weighted average of the F1 score across all classes. Finally, it prints the computed F1 score.

Confusion Matrix:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_labels)
print('Confusion Matrix:')
print(conf_matrix)
```

Output:



```
Confusion Matrix:
[[23  1  0  1]
 [ 0 33  0  4]
 [ 0  0 26  3]
 [ 2  1  0 32]]
```

Figure 6- Confusion Matrix

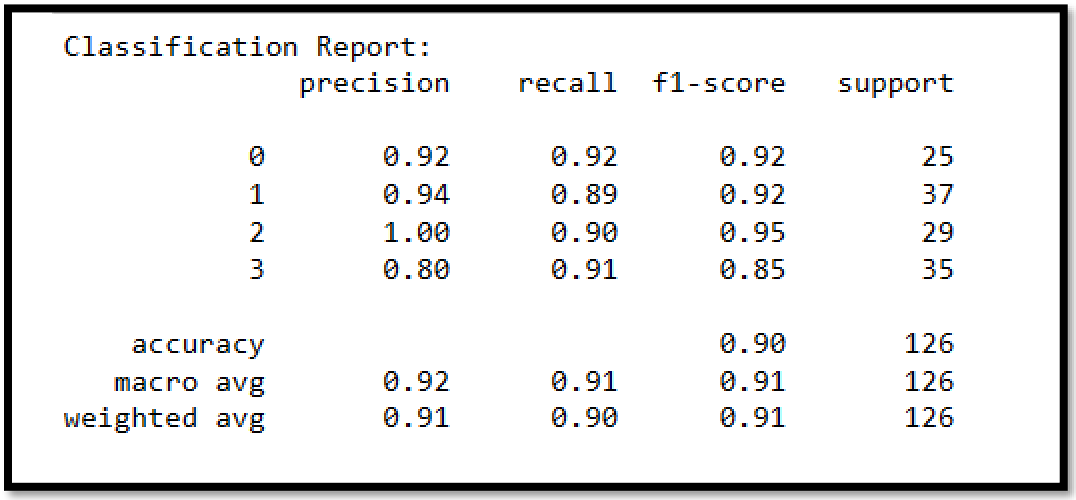
Code Explanation:

This code generates a confusion matrix to evaluate the performance of a classification model. It uses the **`confusion_matrix`** function from scikit-learn, providing the true labels **(`y_test`)** and the predicted labels **(`y_pred_labels`).** Then, it prints the confusion matrix to the console.

Classification Report:

```
# Display classification report
print('Classification Report:')
print(classification_report(y_test, y_pred_labels))
```

Output:



```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        25
           1       0.94      0.89      0.92        37
           2       1.00      0.90      0.95        29
           3       0.80      0.91      0.85        35

    accuracy                           0.90       126
   macro avg       0.92      0.91      0.91       126
weighted avg       0.91      0.90      0.91       126
```

Figure 7- Classification Report

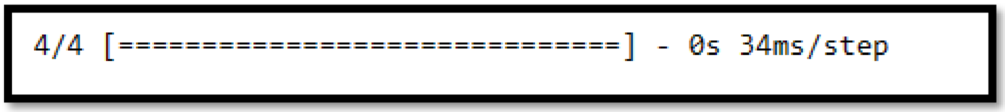
Code Explanation:

This code prints a classification report, which includes precision, recall, F1-score, and support for each class in the classification model. It utilizes the **`classification_report`** function from scikit-learn, providing the true labels **(`y_test`)** and the predicted labels **(`y_pred_labels`)**.

ROC curve

```
# Obtain predicted probabilities for each class from the trained model
y_pred_logits = model_with_dropout.predict(X_test)
y_pred_proba = tf.nn.softmax(y_pred_logits).numpy()
```

Output:



```
4/4 [==============================] - 0s 34ms/step
```

Figure 8- ROC Curve

```
# Plot ROC curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    if np.sum(y_test == i) > 0:  # Check if the class has positive samples
        fpr, tpr, _ = roc_curve(y_test == i, y_pred_proba[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

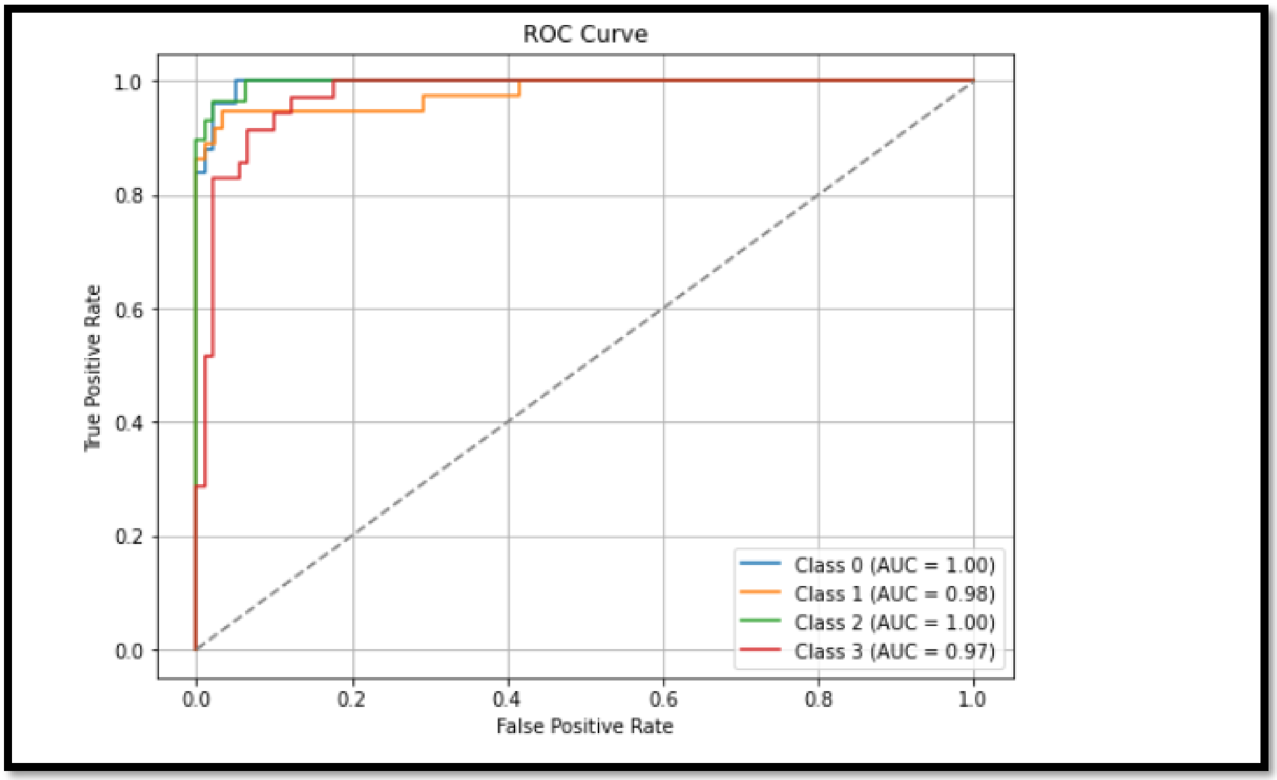


Figure 9- ROC Curve Graph

Code Explanation:

This code plots the Receiver Operating Characteristic (ROC) curve for each class. It iterates over each class index **(`i`)** and checks if there are positive samples for that class in the test data. For each class with positive samples, it computes the False Positive Rate (FPR) and True Positive Rate

(TPR) using the **`roc_curve`** function from scikit-learn, based on the true labels **(`y_test`)** and predicted probabilities **(`y_pred_proba`)** for that class. It calculates the Area Under the Curve (AUC) for the ROC curve and plots it with a label indicating the class and its corresponding AUC value. Finally, it plots the diagonal dashed line (representing random guessing) and displays the plot with labels, title, legend, and grid.

Precision-Recall Curve

```
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    precision, recall, _ = precision_recall_curve(y_test == i, y_pred_proba[:, i])
    average_precision = average_precision_score(y_test == i, y_pred_proba[:, i])
    plt.plot(recall, precision, label=f'Class {i} (AP = {average_precision:.2f})')
```

Output:

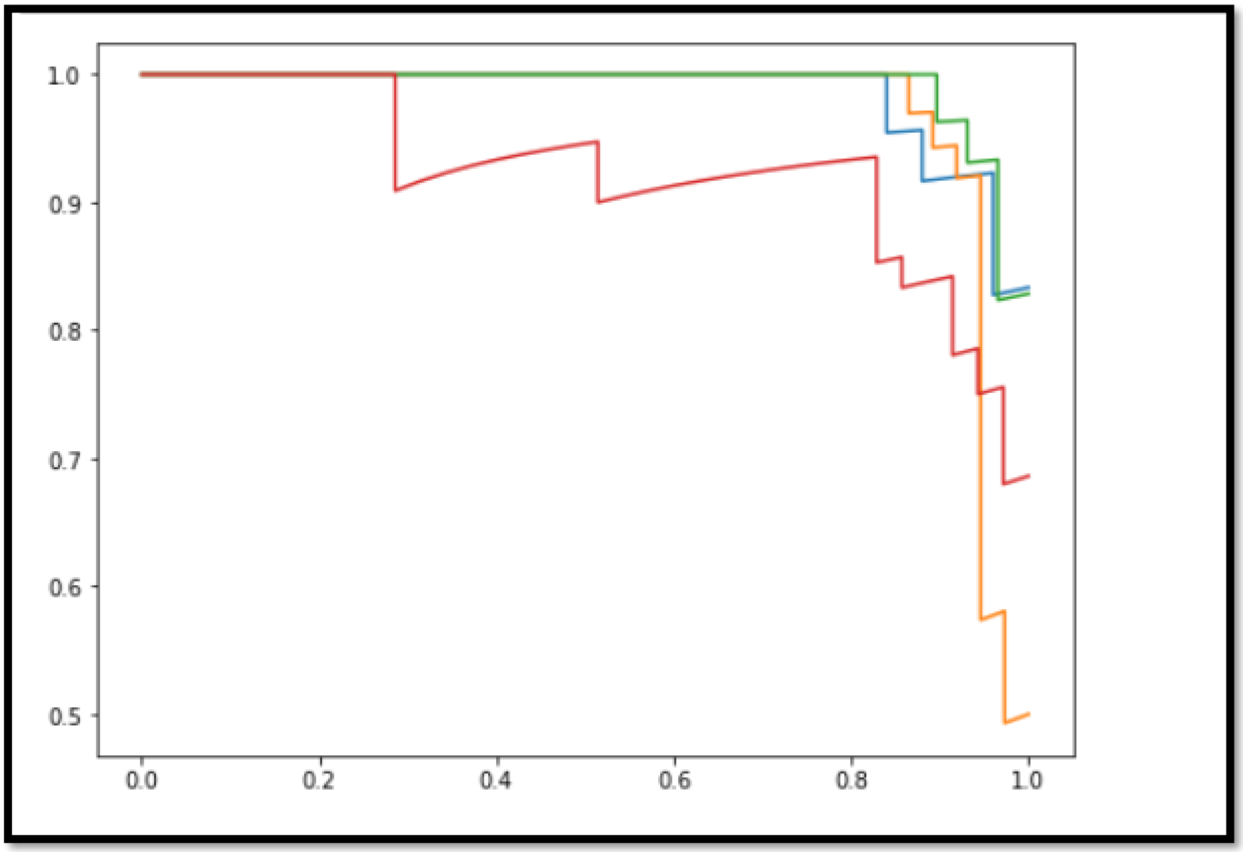


Figure 10- Precision Recall Curve

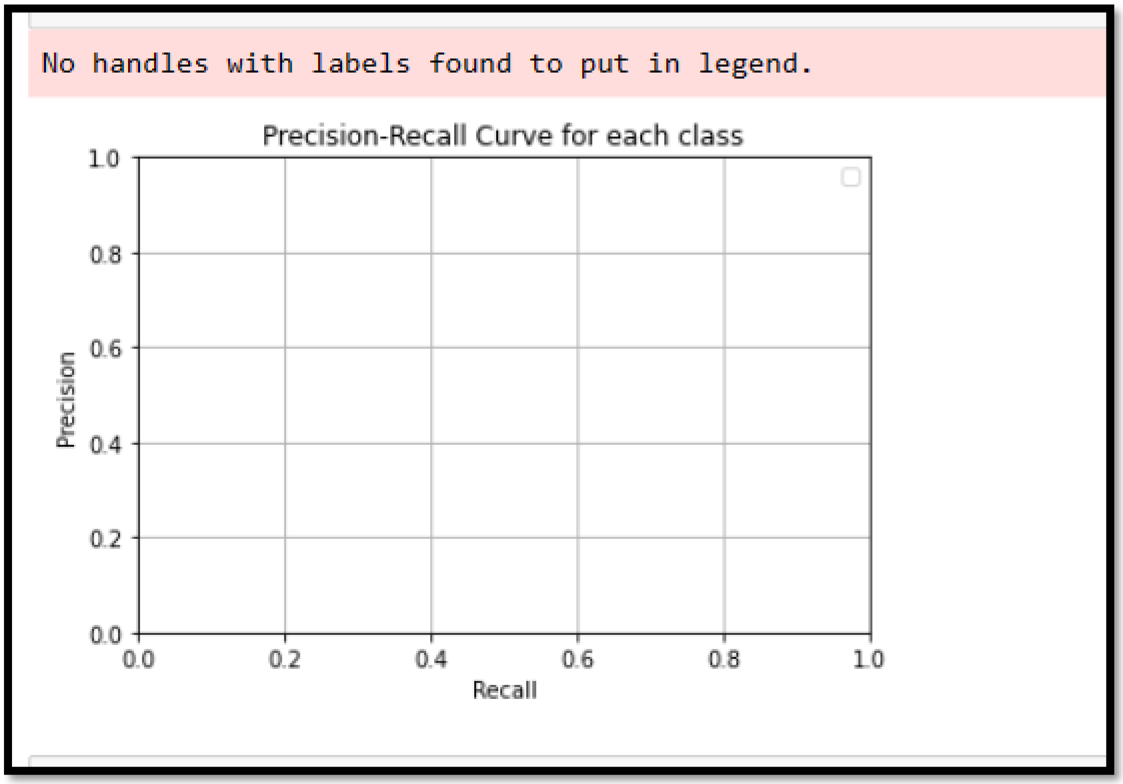
Code Explanation:

This code plots the Precision-Recall curve for each class. It iterates over each class index **(`i`)** and computes the precision and recall values using the **`precision_recall_curve`** function from scikit-learn based on the true labels **(`y_test`)** and predicted probabilities **(`y_pred_proba`)** for that class. It also calculates the Average Precision (AP) score using the **`average_precision_score`** function. Then, it plots the precision-recall curve for each class with a label indicating the class and its corresponding AP value. Finally, it displays the plot with a specified figure size.

Precision-Recall Curve

```
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for each class')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



```
No handles with labels found to put in legend.
```

Figure 11- Precision Recall Curve for each class

Code Explanation:

This code adds labels to the x-axis **(`Recall`)** and y-axis **(`Precision`)**, sets the title of the plot to **`'Precision-Recall Curve for each class'`,** adds a legend to the plot, enables the grid for better visualization, and finally displays the plot.