Speech Recognition Project

Name: Nouran Mohamed Mohamed Hemdan

ID: 20221321932

Department: AI

Audio Classification of Cats and Dogs

Objective

1-The provided code is designed to set up an environment for downloading, extracting, and processing an audio dataset of cat and dog sounds. The ultimate goal appears to be building a machine learning model using TensorFlow to classify audio data into categories—likely identifying whether an audio file is of a cat or a dog:

```
zip_file_path = '/content/audio-cats-and-dogs.zip'
extracted_path = '/content/extracted_dataset'
audio_path = os.path.join(extracted_path, 'cats_dogs')
os.makedirs(extracted_path, exist_ok=True)
```

2-The provided code is a function designed to visualize the distribution of labels within a dataset using a bar plot. This is crucial for understanding the balance of classes in a classification task, such as the one involving audio data of cats and dogs:

```
def plot_data_distribution(labels):
    sns.countplot(x=labels)
    plt.title('Distribution of Labels')
    plt.xlabel('Label')
    plt.ylabel('Count')
    plt.show()
plot_data_distribution(y)
```

3-The provided code defines a function, preprocess_data, which is designed to preprocess features and labels for use in a machine learning model. The function performs label encoding, one-hot encoding, and splits the data into training and testing sets. This preprocessing is essential for preparing data before feeding it into a deep learning model.

```
def preprocess_data(features, labels):
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(labels)
    y_categorical = to_categorical(y_encoded)

    X_train, X_test, y_train, y_test = train_test_split(features,
y_categorical, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test, label_encoder

X_train, X_test, y_train, y_test, label_encoder = preprocess_data(X, y)
```

4-The provided code defines two functions: extract_features and load_data, which are designed to process audio files from a specified directory, extract meaningful features, and prepare the data for use in a machine learning model. The goal is to convert raw audio data into a structured format that can be fed into a model for tasks like classification.

```
def extract_features(file_path):
    try:
        y, sr = librosa.load(file_path, sr=None)
        mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
        return np.mean(mfccs.T, axis=0)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
        return None

def load_data(audio_path):
    labels = []
    features = []
```

```
    for file_name in os.listdir(audio_path):
        file_path = os.path.join(audio_path, file_name)
        if file_name.startswith('cat'):
            label = 'cat'
        elif file_name.startswith('dog'):
            label = 'dog'
        else:
            continue

        feature = extract_features(file_path)
        if feature is not None:
            features.append(feature)
            labels.append(label)

    return np.array(features), np.array(labels)
X, y = load_data(audio_path)
```

5-The provided code defines a function, augment_audio, which is designed to augment audio data by adding random noise. Audio augmentation is a common technique used to increase the diversity of the training data, helping the machine learning model to generalize better.

```
def augment_audio(y, noise_factor=0.005):
    noise = np.random.randn(len(y))
    augmented_y = y + noise_factor * noise
    return augmented_y
```

6-The provided code defines a function, build_model, which is designed to create and compile a neural network model for a binary classification task. This model is likely intended for classifying audio data, such as distinguishing between "cat" and "dog" sounds.

```
def build_model(input_shape):
    model = Sequential([
        Input(shape=input_shape),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(2, activation='softmax')
    ])
    model.compile(optimizer='adam',
```

```
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
model = build_model(X_train.shape[1:])
```

7-The provided code defines a function, evaluate_model, which evaluates the performance of a trained neural network model on a test dataset. This function calculates the test loss and accuracy, and also generates a confusion matrix to visually assess the model's classification performance.

```
def evaluate_model(model, X_test, y_test):
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f'Test Loss: {loss:.4f}')
    print(f'Test Accuracy: {accuracy:.4f}')

    y_pred = model.predict(X_test)
    cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred,
axis=1))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_encoder.classes_)
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()

evaluate_model(model, X_test, y_test)
```

Comparing Models:

1-The provided code defines a function, train_and_evaluate_model, which trains a neural network model on a training dataset and then evaluates its performance on a test dataset. The function returns the training history, test loss, and test accuracy, which can be used to compare different models.

```
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2, verbose=0)
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    return history, loss, accuracy
```

2-The compare_models function is designed to train, evaluate, and compare multiple models on the same dataset. It provides a comprehensive comparison of model performance by calculating test accuracy, displaying confusion matrices, and visualizing model accuracy in a bar plot.

```
def compare_models(models, names, X_train, y_train, X_test, y_test,
label_encoder):
    results = []

    for model, name in zip(models, names):
        print(f"Training and evaluating {name}...")
        history, loss, accuracy = train_and_evaluate_model(model, X_train,
y_train, X_test, y_test)
        results.append((name, accuracy))
        print(f"{name} - Test Loss: {loss:.4f}, Test Accuracy:
{accuracy:.4f}")
        y_pred = model.predict(X_test)
        cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred,
axis=1))
        disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_encoder.classes_)
        disp.plot(cmap=plt.cm.Blues)
        plt.title(f'Confusion Matrix - {name}')
        plt.show()
    names, accuracies = zip(*results)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=names, y=accuracies, palette='viridis')
    plt.title('Model Accuracy Comparison')
    plt.xlabel('Model')
    plt.ylabel('Accuracy')
    plt.ylim(0, 1)
    plt.show()
```

3-The provided code snippet initializes two instances of a neural network model using the build_model function, assigns names to these models, and prepares them for comparison. This setup is crucial for evaluating and comparing the performance of different model configurations or instances.

```
model_1 = build_model(X_train.shape[1:])
```

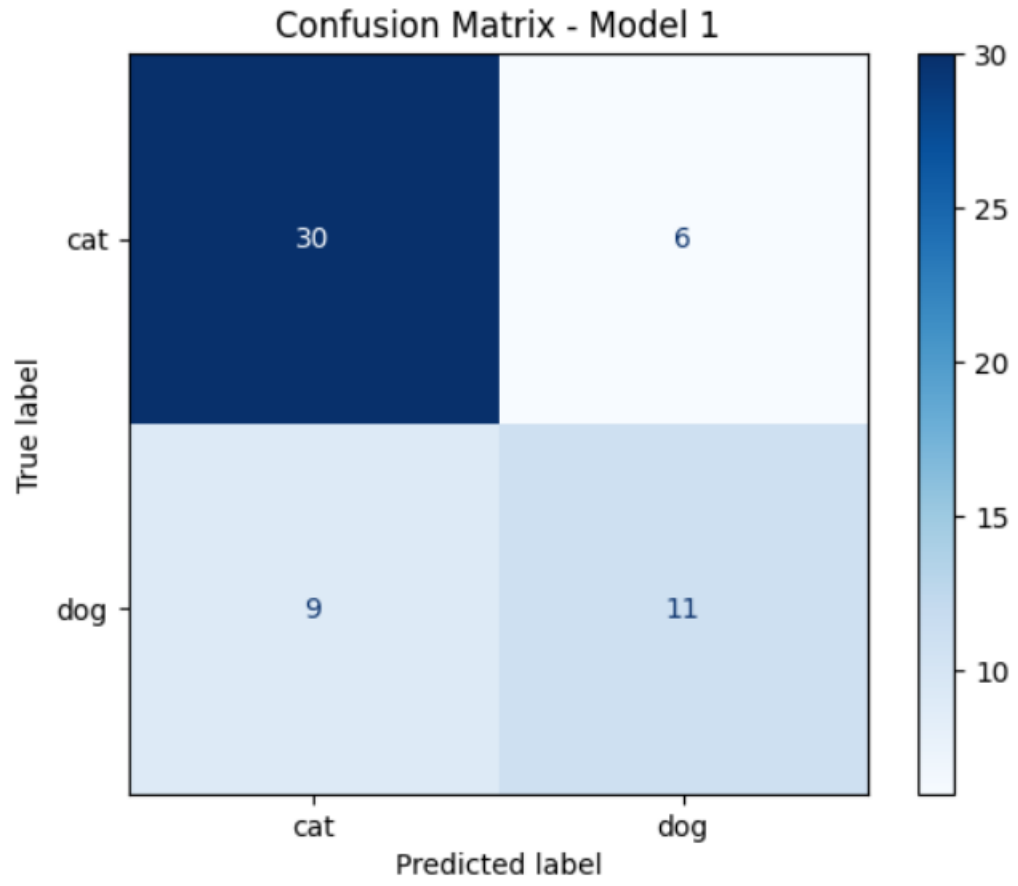
```
model_2 = build_model(X_train.shape[1:])
models = [model_1, model_2]
names = ['Model 1', 'Model 2']
```

4-The provided code snippet executes the compare_models function, which trains and evaluates multiple neural network models, compares their performance, and visualizes the results. This process is used to identify the best-performing model among the ones tested.

```
compare_models(models, names, X_train, y_train, X_test, y_test,
label_encoder)
```

output:

```
Training and evaluating Model 1...
Model 1 - Test Loss: 3.8791, Test Accuracy: 0.7321
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
```



```
Training and evaluating Model 2...
Model 2 - Test Loss: 2.7787, Test Accuracy: 0.6250
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```