Deep Learning Project

COURSE CODE	CSE485
COURSE NAME	Deep Learning
SEMESTER	Fall 2024
DATE OF SUBMISSION	12/25/2024

Submitted By:

Kareem Mohamed Shoaib	**20P5553**
Adham Amr	**20P5249**
Mazen Tayseer	**20P7460**

Pre-Processing

```
numeric_columns = train_df.select_dtypes(include=['float64', 'int64']).columns
train_df[numeric_columns] = train_df[numeric_columns].fillna(train_df[numeric_columns].mean())

categorical_columns = train_df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    train_df[col].fillna(train_df[col].mode()[0], inplace=True)
```

Handle Missing Values: Ensures that no NaN values remain in the dataset, which could otherwise cause errors during model training or data processing.

```
train_df.drop_duplicates(inplace=True)
```

Drop duplicates Removes duplicate rows from the train_df DataFrame.
inplace=True ensures that the changes are applied directly to the train_df object without needing to reassign it.

```
label_encoder = LabelEncoder()
train_df['species'] = label_encoder.fit_transform(train_df['species'])
label_mapping = dict(zip(label_encoder.transform(label_encoder.classes_), label_encoder.classes_))
```

Encoding:
Encoding the categorical column species into numerical labels, which are easier for machine learning algorithms to process.

For example:

- 'Acer_Palmatum' → 0
- 'Quercus_Rubra' → 1
- 'Tilia_Tomentosa' → 2

Loading Images

```python
def load_images(image_folder, image_names, target_size=(128, 128)):
    images = []
    for name in image_names:
        img_path = os.path.join(image_folder, name)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, target_size)
            images.append(img)
    return np.array(images)

image_folder = 'images'
train_images = load_images(image_folder, train_df['id'].astype(str) + '.jpg')
train_images = train_images / 255.0
```

Parameters:

- image_folder: The directory containing the images.
- image_names: A list of image file names to load (e.g., ['1.jpg', '2.jpg', ...]).
- target_size: The desired dimensions of the images after resizing (default is (128, 128)).

Process:

- Step 1: Initialize an empty list images to store loaded and processed images.
- Step 2: Iterate over each image file name in image_names:
  - Construct File Path: Combine image_folder and name using os.path.join to create the full file path for each image.
  - Load Image: Use cv2.imread() to load the image from the file path.
    - If the image is successfully loaded (not None):
      - Resize Image: Resize the image to the specified target_size using cv2.resize().
      - Append to List: Add the resized image to the images list.
- Step 3: Convert the images list into a NumPy array using np.array(). This ensures compatibility with TensorFlow and machine learning frameworks.

Train-Test-Split

```python
train_df.drop('id', axis=1, inplace=True)

X_train, X_test, y_train, y_test = train_test_split(
    train_images, train_df['species'].values, test_size=0.2, stratify=train_df['species'].values, random_state=42
)

num_classes = len(np.unique(y_train))
```

- **train_df.drop('id', axis=1, inplace=True):**
    - Removes the id column from the train_df DataFrame because it is no longer required for training or testing.
    - axis=1: Specifies that the column (not rows) should be dropped.
    - inplace=True: Modifies the DataFrame directly without creating a copy.
- **train_test_split Function:**
    - Splits the dataset into training and testing sets.

Arguments:

- train_images: Array of preprocessed images (features).
- train_df['species'].values: The target labels (species) corresponding to each image.
- test_size=0.2: Specifies that 20% of the dataset will be used for testing, and the remaining 80% for training.
- stratify=train_df['species'].values:
    - Ensures that the proportion of each class (species) in the training and testing sets matches that of the original dataset.
    - This is important when the dataset has imbalanced classes.

Display Images

Display Sample Images

```python
num_samples = 5
plt.figure(figsize=(15, 5))
for i in range(num_samples):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(X_train[i])
    plt.title(f"Label: {label_mapping[y_train[i]]}")
    plt.axis('off')
plt.show()
```

[16] ✓ 0.3s Python

Grid Search

```
param_grid = {
    'num_layers': [2],
    'dropout_rate': [0.5 , 0.3],
    'optimizer_name': ['adam', 'sgd', 'rmsprop'],
    'weight_decay': [0.01 , 0.001],
    'initial_lr': [0.001 , 0.0001],
    'lr_scheduler': [
        None,
        'step_decay',
        'ReduceLROnPlateau'
    ],
}

param_combinations = list(product(
    param_grid['num_layers'],
    param_grid['dropout_rate'],
    param_grid['optimizer_name'],
    param_grid['weight_decay'],
    param_grid['initial_lr'],
    param_grid['lr_scheduler']
))

best_accuracy = 0
best_params = None
best_model_path = "leaf_classification_cnn_model.keras"
results = []
```

Training Function

```python
for params in param_combinations:
    num_layers, dropout_rate, optimizer_name, weight_decay, initial_lr, lr_scheduler = params
    print(f"\nTraining with params: {params}")

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3),
                     kernel_regularizer=l2(weight_decay)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    for _ in range(num_layers - 1):
        model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(weight_decay)))
        model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(y_train.max() + 1, activation='softmax', kernel_regularizer=l2(weight_decay)))

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=initial_lr)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate=initial_lr, momentum=0.9)
    elif optimizer_name == 'rmsprop':
        optimizer = RMSprop(learning_rate=initial_lr, decay=1e-6, rho=0.9)
    else:
        raise ValueError("Invalid optimizer. Choose 'adam', 'sgd', or 'rmsprop'.")
```

```python
        callbacks.append(ReduceLROnPlateau(monitor='val_loss', factor=0.5, pat

    history = model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        epochs=20,
        batch_size=32,
        callbacks=callbacks,
        verbose=1
    )
```

```python
    history = model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        epochs=20,
        batch_size=32,
        callbacks=callbacks,
        verbose=1
    )

    _, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

    results.append((params, test_accuracy))

    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy
        best_params = params
        model.save(best_model_path)
        print(f"New best model saved to {best_model_path}")

print("\nBest Parameters:")
print(best_params)
print(f"Best Accuracy: {best_accuracy * 100:.2f}%")
print(f"Best model saved to: {best_model_path}")

print("\nAll Results:")
for params, accuracy in results:
    print(f"Params: {params}, Accuracy: {accuracy * 100:.2f}%")
```

Results

Standardized data

```
Training with params: (2, 0.5, 'rmsprop', 0.01, 0.001, 'ReduceLROnPlateau')
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 397ms/step - accuracy: 0.0120 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 0.0010
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 10s 378ms/step - accuracy: 0.0033 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 0.0010
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 10s 392ms/step - accuracy: 0.0115 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 0.0010
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 551ms/step - accuracy: 0.0138 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 5.0000e
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 477ms/step - accuracy: 0.0069 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 5.0000e
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 11s 458ms/step - accuracy: 0.0050 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 5.0000e
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 467ms/step - accuracy: 0.0099 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 2.5000e
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 477ms/step - accuracy: 0.0090 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 2.5000e
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 470ms/step - accuracy: 0.0074 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 2.5000e
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 492ms/step - accuracy: 0.0111 - loss: nan - val_accuracy: 0.0101 - val_loss: nan - learning_rate: 1.2500e
Test Accuracy: 1.01%
New best model saved to leaf_classification_cnn_model.keras
```

Accuracy with standardized data by computing the mean and standard deviation for each feature dimension using the training set only, then subtracting the mean and dividing by the stdev for each feature and each sample. Produced very bad accuracies and result , so we disposed it

Best Results after grid-search

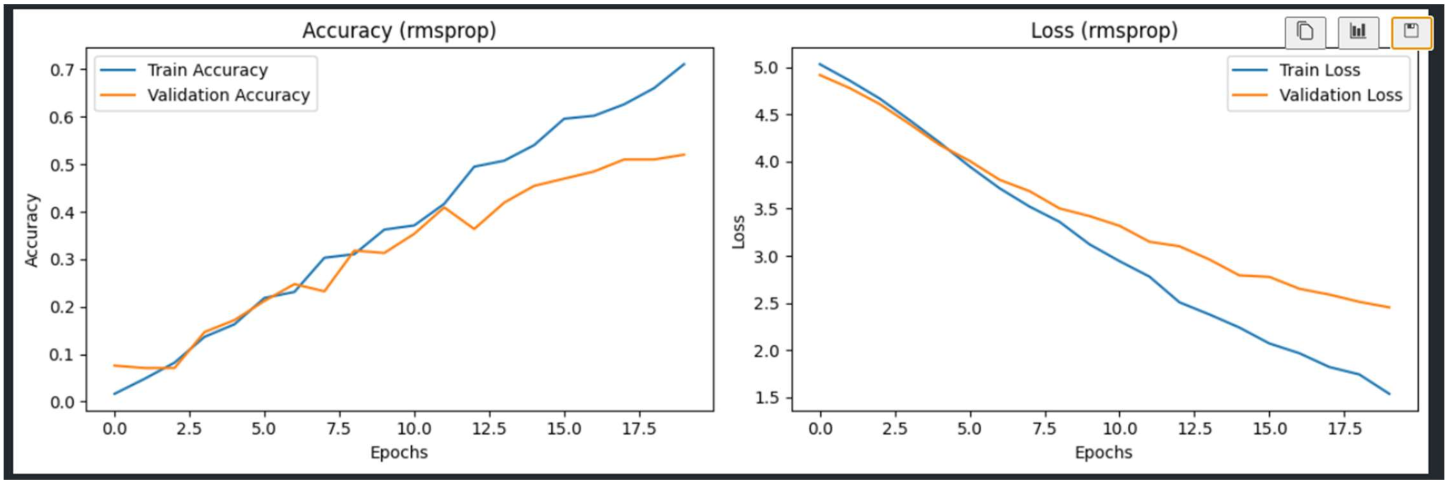
```
Best Parameters:
(2, 0.5, 'rmsprop', 0.01, 0.001, 'step_decay')
Best Accuracy: 61.11%
Best model saved to: leaf_classification_cnn_model.keras
```

Evaluation

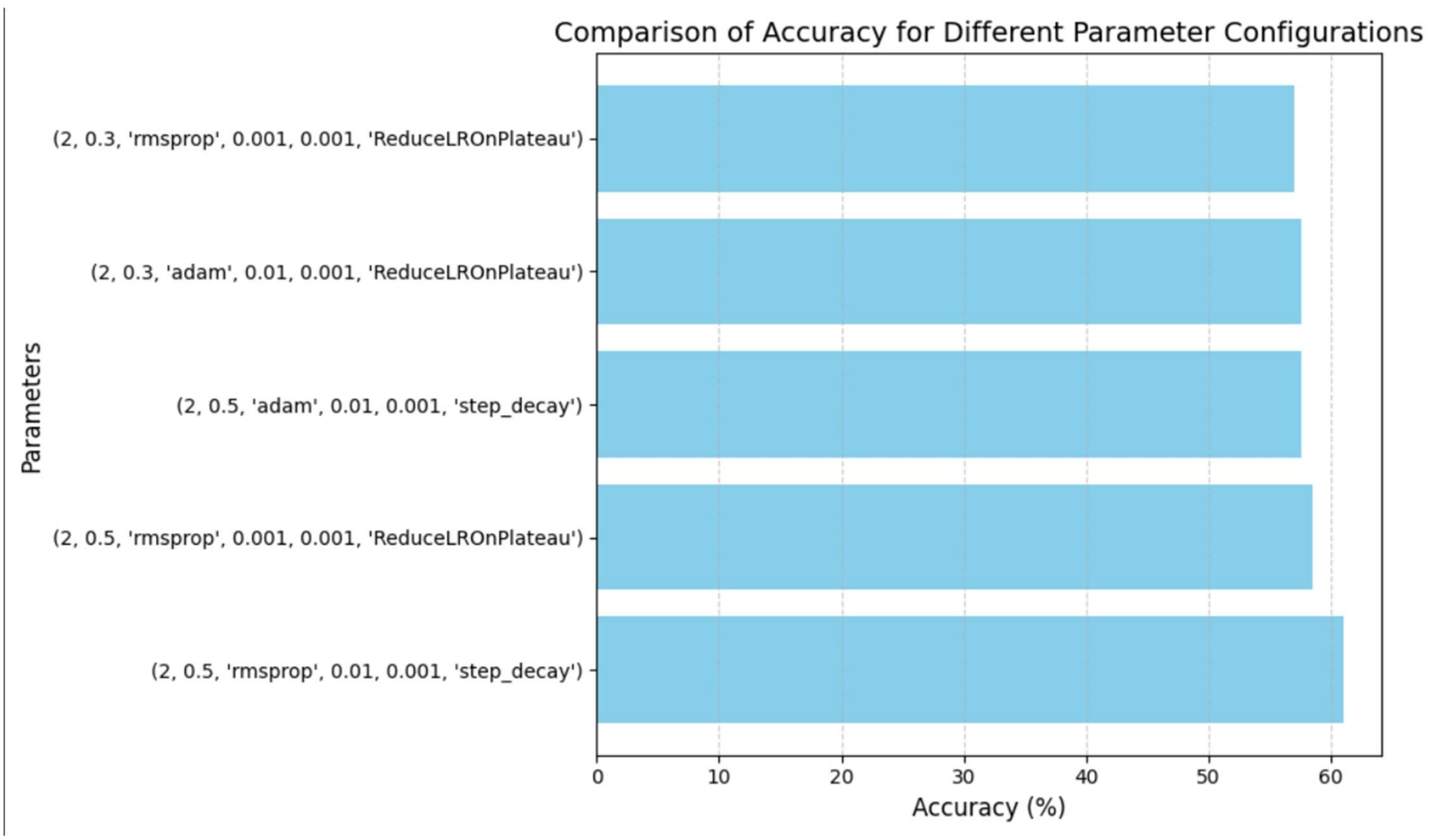
```python
model_path = 'leaf_classification_cnn_model.keras'

model = tf.keras.models.load_model(model_path)
print(f"\nModel loaded from {model_path}")

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

print("\nEvaluating on Training Set:")
train_loss, train_accuracy = model.evaluate(X_train, y_train, verbose=0)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")

print("\nEvaluating on Test Set:")
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

print("\nClassification Report (Test Set):")
print(classification_report(y_test, y_pred_classes))
```

```
Model loaded from leaf_classification_cnn_model.keras

Evaluating on Training Set:
Training Accuracy: 98.23%

Evaluating on Test Set:
Test Accuracy: 61.11%
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```

Training history plot



Comparing with 5 best results



Classification of images

```python
model = tf.keras.models.load_model(model_path)
print(f"\nModel loaded from {model_path}")

def preprocess_image(img_path, target_size=(128, 128)):
    img = cv2.imread(img_path)
    if img is not None:
        img = cv2.resize(img, target_size)
        img = img / 255.0
        return img
    return None

images = []
valid_image_names = []

test_images_folder = 'test_images'
test_images = ['1.jpg', '2.jpg', '3.jpg','40.jpg']
for name in test_images:
    img_path = os.path.join(test_images_folder, name)
    img = preprocess_image(img_path)
    if img is not None:
        images.append(img)
        valid_image_names.append(name)
    else:
        print(f"Warning: Could not load image {name}. Skipping.")

if not images:
    print("No valid images to classify.")
else:
    images = np.array(images)
    predictions = model.predict(images)
    predicted_classes = np.argmax(predictions, axis=1)

    for i, name in enumerate(valid_image_names):
        print(f"Image: {name} -> Predicted Label: {label_mapping[predicted_classes[i]]}")
```

```
Model loaded from leaf_classification_cnn_model.keras
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
Image: 1.jpg -> Predicted Label: Acer_Opalus
Image: 2.jpg -> Predicted Label: Pterocarya_Stenoptera
Image: 3.jpg -> Predicted Label: Quercus_Hartwissiana
Image: 40.jpg -> Predicted Label: Quercus_Phillyraeoides
```