Akansha Patel DS6A 2003 Assignment 3

```
!git clone https://github.com/makhan010385/Soybean-.git

     Cloning into 'Soybean-'...
     remote: Enumerating objects: 404, done.
     remote: Counting objects: 100% (57/57), done.
     remote: Compressing objects: 100% (56/56), done.
     remote: Total 404 (delta 1), reused 53 (delta 0), pack-reused 347
     Receiving objects: 100% (404/404), 1.75 GiB | 26.84 MiB/s, done.
     Resolving deltas: 100% (10/10), done.
     Updating files: 100% (355/355), done.
```

LeNet

Load and Preprocess Data

```
import numpy as np
import os
from keras.preprocessing.image import img_to_array, load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define the diseases
diseases = ['Bacterial Pustule', 'Frogeye Leaf Spot', 'Healthy', 'Rust', 'Sudden Death Syndrome', 'Target Leaf Spot'

# Initialize lists to store image data and labels
images = []
labels = []

# Define data augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Load and augment images
for idx, disease in enumerate(diseases):
    disease_path = os.path.join('/content/Soybean-/Soybean Leaf Dataset for Disease Classification', disease)  # Rep
    image_files = [os.path.join(disease_path, f) for f in os.listdir(disease_path) if f.endswith('.jpg')]

    # Split the images into training and validation sets (40 images) and testing set (10 images)
    train_val_images = image_files[:40]
    test_images = image_files[40:]

    for image_file in train_val_images:
        image = load_img(image_file, target_size=(128, 128))  # Resize image to 224x224
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)  # Expand dimensions to (1, height, width, channels) for flow() methoc

        # Generate augmented images
        for batch in datagen.flow(image, batch_size=1):
            augmented_image = batch[0]
            images.append(augmented_image)
            labels.append(idx)
            break  # Break the loop after one augmentation to avoid infinite loop

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Shuffle the data
shuffle_idx = np.random.permutation(len(images))
images = images[shuffle_idx]
labels = labels[shuffle_idx]

# Normalize the images
images = images / 255.0

# Split the dataset into training, validation, and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def create_lenet_model(input_shape):
    model = Sequential()

    # First Convolutional Layer
    model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Second Convolutional Layer
    model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(120, activation='relu'))
    model.add(Dense(84, activation='relu'))
    model.add(Dense(len(diseases), activation='softmax'))

    return model

input_shape = images[0].shape
model = create_lenet_model(input_shape)
```

Compile Keras Model

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

Fit Keras Model

```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stop]
)
```

```
Epoch 1/100
6/6 [==============================] - 3s 399ms/step - loss: 1.8364 - accuracy: 0.2123 - val_loss: 1.7957 - val_
Epoch 2/100
6/6 [==============================] - 2s 377ms/step - loss: 1.6462 - accuracy: 0.3966 - val_loss: 1.7574 - val_
Epoch 3/100
6/6 [==============================] - 2s 360ms/step - loss: 1.5314 - accuracy: 0.3855 - val_loss: 1.7699 - val_
Epoch 4/100
6/6 [==============================] - 3s 485ms/step - loss: 1.4054 - accuracy: 0.4302 - val_loss: 1.7336 - val_
Epoch 5/100
6/6 [==============================] - 3s 466ms/step - loss: 1.2502 - accuracy: 0.5754 - val_loss: 1.6790 - val_
Epoch 6/100
6/6 [==============================] - 2s 353ms/step - loss: 1.0469 - accuracy: 0.6592 - val_loss: 1.8903 - val_
Epoch 7/100
6/6 [==============================] - 2s 320ms/step - loss: 0.9777 - accuracy: 0.6480 - val_loss: 1.7301 - val_
Epoch 8/100
6/6 [==============================] - 2s 325ms/step - loss: 0.7418 - accuracy: 0.7877 - val_loss: 1.6919 - val_
```

```
Epoch 9/100
6/6 [==============================] - 3s 558ms/step - loss: 0.5738 - accuracy: 0.8436 - val_loss: 1.7037 - val_
Epoch 10/100
6/6 [==============================] - 2s 324ms/step - loss: 0.4280 - accuracy: 0.8771 - val_loss: 2.2888 - val_
Epoch 11/100
6/6 [==============================] - 2s 324ms/step - loss: 0.4662 - accuracy: 0.8547 - val_loss: 1.9910 - val_
Epoch 12/100
6/6 [==============================] - 2s 328ms/step - loss: 0.2552 - accuracy: 0.9721 - val_loss: 1.9355 - val_
Epoch 13/100
6/6 [==============================] - 2s 325ms/step - loss: 0.1954 - accuracy: 0.9665 - val_loss: 2.1439 - val_
Epoch 14/100
6/6 [==============================] - 2s 356ms/step - loss: 0.0899 - accuracy: 0.9944 - val_loss: 2.1121 - val_
Epoch 15/100
6/6 [==============================] - 3s 494ms/step - loss: 0.0709 - accuracy: 1.0000 - val_loss: 2.2127 - val_
Epoch 16/100
6/6 [==============================] - 2s 323ms/step - loss: 0.0318 - accuracy: 0.9944 - val_loss: 2.2966 - val_
Epoch 17/100
6/6 [==============================] - 2s 356ms/step - loss: 0.0219 - accuracy: 1.0000 - val_loss: 2.3034 - val_
Epoch 18/100
6/6 [==============================] - 2s 323ms/step - loss: 0.0125 - accuracy: 1.0000 - val_loss: 2.4118 - val_
Epoch 19/100
6/6 [==============================] - 2s 324ms/step - loss: 0.0106 - accuracy: 1.0000 - val_loss: 2.5138 - val_
Epoch 20/100
6/6 [==============================] - 2s 360ms/step - loss: 0.0084 - accuracy: 1.0000 - val_loss: 2.6053 - val_
Epoch 21/100
6/6 [==============================] - 3s 452ms/step - loss: 0.0060 - accuracy: 1.0000 - val_loss: 2.5022 - val_
Epoch 22/100
6/6 [==============================] - 2s 323ms/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 2.5817 - val_
Epoch 23/100
6/6 [==============================] - 2s 327ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 2.6828 - val_
Epoch 24/100
6/6 [==============================] - 2s 328ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 2.6486 - val_
Epoch 25/100
6/6 [==============================] - 2s 328ms/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 2.6254 - val_
```

Evaluate Keras Model

```
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 0s 57ms/step - loss: 1.6790 - accuracy: 0.3778
Validation Loss: 1.6790
Validation Accuracy: 37.78%
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 0s 82ms/step - loss: 1.2678 - accuracy: 0.4464
Test Loss: 1.2678
Test Accuracy: 44.64%
```

Identify Disease from Test Images

```
def predict_disease(image_path):
    image = load_img(image_path, target_size=(128, 128))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0) / 255.0
    prediction = model.predict(image)
    predicted_label = np.argmax(prediction)
    return diseases[predicted_label]

# Example usage
image_path = '/content/Soybean-/Soybean Leaf Dataset for Disease Classification/Target Leaf Spot/Target LS (48).jpg'
predicted_disease = predict_disease(image_path)
print(f"Predicted Disease: {predicted_disease}")
```

```
1/1 [==============================] - 0s 113ms/step
Predicted Disease: Target Leaf Spot
```

```
folder_path = '/content/Soybean-/Soybean Leaf Dataset for Disease Classification/Rust'  # Replace with the path to y

# List all files in the folder
files = os.listdir(folder_path)

# Count the number of files
num_files = len(files)
print(num_files)
```

```
51
```

AlexNet

Load and Preprocess Data

```
import numpy as np
import os
from keras.preprocessing.image import img_to_array, load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define the diseases
diseases = ['Bacterial Pustule', 'Frogeye Leaf Spot', 'Healthy', 'Rust', 'Sudden Death Syndrome', 'Target Leaf Spot'

# Initialize lists to store image data and labels
images = []
labels = []

# Define data augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Load and augment images
for idx, disease in enumerate(diseases):
    disease_path = os.path.join('/content/Soybean-/Soybean Leaf Dataset for Disease Classification', disease)  # Rep
    image_files = [os.path.join(disease_path, f) for f in os.listdir(disease_path) if f.endswith('.jpg')]

    # Split the images into training and validation sets (40 images) and testing set (10 images)
    train_val_images = image_files[:40]
    test_images = image_files[40:]

    for image_file in train_val_images:
        image = load_img(image_file, target_size=(227, 227))  # Resize image to 227x227
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)  # Expand dimensions to (1, height, width, channels) for flow() methoc

        # Generate augmented images
        for batch in datagen.flow(image, batch_size=1):
            augmented_image = batch[0]
            images.append(augmented_image)
            labels.append(idx)
            break  # Break the loop after one augmentation to avoid infinite loop

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Shuffle the data
shuffle_idx = np.random.permutation(len(images))
images = images[shuffle_idx]
labels = labels[shuffle_idx]

# Normalize the images
images = images / 255.0

# Split the dataset into training, validation, and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)
```

Define AlexNet Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

def create_alexnet_model(input_shape):
    model = Sequential()

    # First Convolutional Layer
    model.add(Conv2D(96, kernel_size=(11, 11), strides=(4, 4), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Second Convolutional Layer
    model.add(Conv2D(256, kernel_size=(5, 5), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Third Convolutional Layer
    model.add(Conv2D(384, kernel_size=(3, 3), padding='same', activation='relu'))

    # Fourth Convolutional Layer
    model.add(Conv2D(384, kernel_size=(3, 3), padding='same', activation='relu'))

    # Fifth Convolutional Layer
    model.add(Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(len(diseases), activation='softmax'))

    return model

input_shape = images[0].shape
model = create_alexnet_model(input_shape)
```

Compile Keras Model

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

Fit Keras Model

```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stop]
)
```

```
6/6 [==============================] - 30s 5s/step - loss: 1.8914 - accuracy: 0.2570 - val_loss: 1.7926 - val
Epoch 14/100
6/6 [==============================] - 29s 5s/step - loss: 1.7442 - accuracy: 0.2905 - val_loss: 1.7202 - val
Epoch 15/100
6/6 [==============================] - 30s 5s/step - loss: 1.6873 - accuracy: 0.3520 - val_loss: 1.7077 - val
Epoch 16/100
6/6 [==============================] - 28s 5s/step - loss: 1.6852 - accuracy: 0.3352 - val_loss: 1.7404 - val
Epoch 17/100
6/6 [==============================] - 30s 5s/step - loss: 1.6149 - accuracy: 0.3743 - val_loss: 1.6879 - val
Epoch 18/100
6/6 [==============================] - 31s 5s/step - loss: 1.6005 - accuracy: 0.3240 - val_loss: 1.8183 - val
Epoch 19/100
6/6 [==============================] - 29s 5s/step - loss: 1.5554 - accuracy: 0.3408 - val_loss: 1.9721 - val
Epoch 20/100
6/6 [==============================] - 30s 5s/step - loss: 1.6377 - accuracy: 0.3240 - val_loss: 1.8768 - val
Epoch 21/100
6/6 [==============================] - 29s 5s/step - loss: 1.6160 - accuracy: 0.3408 - val_loss: 1.5419 - val
Epoch 22/100
6/6 [==============================] - 29s 5s/step - loss: 1.5450 - accuracy: 0.3687 - val_loss: 1.5816 - val
Epoch 23/100
6/6 [==============================] - 34s 6s/step - loss: 1.5394 - accuracy: 0.4078 - val_loss: 1.7308 - val
Epoch 24/100
6/6 [==============================] - 30s 5s/step - loss: 1.5232 - accuracy: 0.3799 - val_loss: 1.7509 - val
Epoch 25/100
6/6 [==============================] - 31s 5s/step - loss: 1.5154 - accuracy: 0.3631 - val_loss: 1.8983 - val
Epoch 26/100
6/6 [==============================] - 31s 5s/step - loss: 1.5122 - accuracy: 0.3799 - val_loss: 1.7012 - val
Epoch 27/100
6/6 [==============================] - 29s 5s/step - loss: 1.4713 - accuracy: 0.3687 - val_loss: 1.5790 - val
Epoch 28/100
6/6 [==============================] - 30s 5s/step - loss: 1.4688 - accuracy: 0.4358 - val_loss: 1.4864 - val
Epoch 29/100
6/6 [==============================] - 29s 5s/step - loss: 1.3825 - accuracy: 0.4525 - val_loss: 1.5450 - val
Epoch 30/100
6/6 [==============================] - 29s 5s/step - loss: 1.3333 - accuracy: 0.4804 - val_loss: 1.4787 - val
Epoch 31/100
6/6 [==============================] - 31s 5s/step - loss: 1.2975 - accuracy: 0.4860 - val_loss: 1.6942 - val
Epoch 32/100
6/6 [==============================] - 28s 5s/step - loss: 1.2647 - accuracy: 0.4637 - val_loss: 1.6136 - val
Epoch 33/100
6/6 [==============================] - 33s 5s/step - loss: 1.4319 - accuracy: 0.4078 - val_loss: 1.5651 - val
Epoch 34/100
6/6 [==============================] - 28s 5s/step - loss: 1.6390 - accuracy: 0.3464 - val_loss: 2.2615 - val
Epoch 35/100
6/6 [==============================] - 28s 5s/step - loss: 1.4786 - accuracy: 0.3855 - val_loss: 1.6407 - val
Epoch 36/100
6/6 [==============================] - 29s 5s/step - loss: 1.3459 - accuracy: 0.4246 - val_loss: 1.5510 - val
Epoch 37/100
6/6 [==============================] - 30s 5s/step - loss: 1.3510 - accuracy: 0.4358 - val_loss: 1.6445 - val
Epoch 38/100
6/6 [==============================] - 29s 5s/step - loss: 1.2898 - accuracy: 0.4916 - val_loss: 1.6080 - val
Epoch 39/100
6/6 [==============================] - 29s 5s/step - loss: 1.2679 - accuracy: 0.4525 - val_loss: 1.6885 - val
Epoch 40/100
6/6 [==============================] - 30s 5s/step - loss: 1.2418 - accuracy: 0.5307 - val_loss: 2.1034 - val
```

Evaluate Keras Model

```
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 2s 446ms/step - loss: 1.4787 - accuracy: 0.3778
Validation Loss: 1.4787
Validation Accuracy: 37.78%
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 2s 798ms/step - loss: 1.6631 - accuracy: 0.3036
Test Loss: 1.6631
Test Accuracy: 30.36%
```

Identify Disease from Test Images

```
def predict_disease(image_path):
    image = load_img(image_path, target_size=(227, 227))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0) / 255.0
    prediction = model.predict(image)
    predicted_label = np.argmax(prediction)
    return diseases[predicted_label]

# Example usage
test_image_path = '/content/Soybean-/Soybean Leaf Dataset for Disease Classification/Rust/Rust (46).jpg'  # Replace
predicted_disease = predict_disease(test_image_path)
print(f"Predicted Disease: {predicted_disease}")
```

```
1/1 [==============================] - 0s 158ms/step
Predicted Disease: Yellow Mosaic
```

VGG

Load and Preprocess Data

```
import numpy as np
import os
from keras.preprocessing.image import img_to_array, load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define the diseases
diseases = ['Bacterial Pustule', 'Frogeye Leaf Spot', 'Healthy', 'Rust', 'Sudden Death Syndrome', 'Target Leaf Spot'

# Initialize lists to store image data and labels
images = []
labels = []

# Define data augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Load and augment images
for idx, disease in enumerate(diseases):
    disease_path = os.path.join('/content/Soybean-/Soybean Leaf Dataset for Disease Classification', disease)  # Rep
    image_files = [os.path.join(disease_path, f) for f in os.listdir(disease_path) if f.endswith('.jpg')]

    # Split the images into training and validation sets (40 images) and testing set (10 images)
    train_val_images = image_files[:40]
    test_images = image_files[40:]

    for image_file in train_val_images:
        image = load_img(image_file, target_size=(224, 224))  # Resize image to 224x224
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)  # Expand dimensions to (1, height, width, channels) for flow() methoc

        # Generate augmented images
        for batch in datagen.flow(image, batch_size=1):
            augmented_image = batch[0]
            images.append(augmented_image)
            labels.append(idx)
            break  # Break the loop after one augmentation to avoid infinite loop

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Shuffle the data
shuffle_idx = np.random.permutation(len(images))
images = images[shuffle_idx]
labels = labels[shuffle_idx]

# Normalize the images
images = images / 255.0

# Split the dataset into training, validation, and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)
```

Define VGG16 Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.applications import VGG16

def create_vgg16_model(input_shape):
    vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    model = Sequential()
    model.add(vgg16_model)

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(len(diseases), activation='softmax'))

    return model

input_shape = (224, 224, 3)  # VGG16 input shape
model = create_vgg16_model(input_shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_or
58889256/58889256 [==============================] - 0s 0us/step
```

Compile Keras Model

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

Fit Keras Model

```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stop]
)
```

```
Epoch 1/100
6/6 [==============================] - 376s 61s/step - loss: 40.8600 - accuracy: 0.1620 - val_loss: 12.2682 - va
Epoch 2/100
6/6 [==============================] - 372s 63s/step - loss: 12.1440 - accuracy: 0.1006 - val_loss: 2.0095 - val
Epoch 3/100
6/6 [==============================] - 367s 62s/step - loss: 2.0218 - accuracy: 0.1508 - val_loss: 1.9501 - val_
Epoch 4/100
6/6 [==============================] - 352s 59s/step - loss: 2.0544 - accuracy: 0.1564 - val_loss: 1.9459 - val_
Epoch 5/100
6/6 [==============================] - 366s 61s/step - loss: 2.0276 - accuracy: 0.1844 - val_loss: 1.9569 - val_
Epoch 6/100
6/6 [==============================] - 370s 61s/step - loss: 1.9413 - accuracy: 0.1732 - val_loss: 1.9497 - val_
Epoch 7/100
6/6 [==============================] - 368s 62s/step - loss: 1.9461 - accuracy: 0.1453 - val_loss: 1.9515 - val_
Epoch 8/100
```

```
6/6 [==============================] - 366s 61s/step - loss: 1.9459 - accuracy: 0.1453 - val_loss: 1.9519 - val_
Epoch 9/100
6/6 [==============================] - 367s 61s/step - loss: 1.9455 - accuracy: 0.1453 - val_loss: 1.9527 - val_
Epoch 10/100
6/6 [==============================] - 367s 62s/step - loss: 1.9456 - accuracy: 0.1453 - val_loss: 1.9541 - val_
Epoch 11/100
6/6 [==============================] - 367s 62s/step - loss: 1.9454 - accuracy: 0.1453 - val_loss: 1.9540 - val_
Epoch 12/100
6/6 [==============================] - 364s 62s/step - loss: 1.9451 - accuracy: 0.1453 - val_loss: 1.9542 - val_
Epoch 13/100
6/6 [==============================] - 363s 61s/step - loss: 1.9453 - accuracy: 0.1453 - val_loss: 1.9547 - val_
Epoch 14/100
6/6 [==============================] - 364s 62s/step - loss: 1.9450 - accuracy: 0.1453 - val_loss: 1.9545 - val_
```

Evaluate Keras Model

```
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 24s 8s/step - loss: 1.9459 - accuracy: 0.2000
Validation Loss: 1.9459
Validation Accuracy: 20.00%
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
2/2 [==============================] - 28s 12s/step - loss: 1.9465 - accuracy: 0.1250
Test Loss: 1.9465
Test Accuracy: 12.50%
```

Identify Disease from Test Images

```
def predict_disease(image_path):
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0) / 255.0
    prediction = model.predict(image)
    predicted_label = np.argmax(prediction)
    return diseases[predicted_label]

# Example usage
test_image_path = '/content/Soybean-/Soybean Leaf Dataset for Disease Classification/Rust/Rust (45).jpg'  # Replace
predicted_disease = predict_disease(test_image_path)
print(f"Predicted Disease: {predicted_disease}")
```

```
1/1 [==============================] - 1s 646ms/step
Predicted Disease: Sudden Death Syndrome
```

GoogLeNet

Load and Preprocess Data

```
import numpy as np
import os
from keras.preprocessing.image import img_to_array, load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define the diseases
diseases = ['Bacterial Pustule', 'Frogeye Leaf Spot', 'Healthy', 'Rust', 'Sudden Death Syndrome', 'Target Leaf Spot'

# Initialize lists to store image data and labels
images = []
labels = []

# Load 40 images from each disease for training and validation
# and 10 images for testing
for idx, disease in enumerate(diseases):
    disease_path = os.path.join('/content/Soybean-/Soybean Leaf Dataset for Disease Classification', disease)  # Rep
    image_files = [os.path.join(disease_path, f) for f in os.listdir(disease_path) if f.endswith('.jpg')]

    # Split the images into training and validation sets (40 images) and testing set (10 images)
    train_val_images = image_files[:40]
    test_images = image_files[40:]

    for image_file in train_val_images:
        image = load_img(image_file, target_size=(224, 224))  # Resize image to 224x224
        image = img_to_array(image)
        images.append(image)
        labels.append(idx)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Generate augmented images
augmented_images = []
augmented_labels = []
for image, label in zip(images, labels):
    image = np.expand_dims(image, axis=0)  # Expand dimensions to (1, height, width, channels) for flow() method
    for batch in datagen.flow(image, batch_size=1):
        augmented_image = batch[0]
        augmented_images.append(augmented_image)
        augmented_labels.append(label)
        break  # Break the loop after one augmentation to avoid infinite loop

# Convert lists to numpy arrays
augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# Combine original and augmented data
X_train_val = np.concatenate((images, augmented_images), axis=0)
y_train_val = np.concatenate((labels, augmented_labels), axis=0)

# Shuffle the data
```

```
shuffle_idx = np.random.permutation(len(X_train_val))
X_train_val = X_train_val[shuffle_idx]
y_train_val = y_train_val[shuffle_idx]

# Normalize the images
X_train_val = X_train_val / 255.0

# Split the dataset into training, validation, and testing sets
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)
X_test, y_test = np.array([]), np.array([])  # No test data in this approach
```

Define GoogLeNet Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.applications import InceptionV3

def create_googlenet_model(input_shape):
    googlenet_model = InceptionV3(weights='imagenet', include_top=False, input_shape=input_shape)

    model = Sequential()
    model.add(googlenet_model)

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(len(diseases), activation='softmax'))

    return model

input_shape = (224, 224, 3)  # GoogLeNet input shape
model = create_googlenet_model(input_shape)
```

Compile Keras Model

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

Fit Keras Model

```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stop]
)
```

```
Epoch 1/100
14/14 [==============================] - 290s 19s/step - loss: 17.3340 - accuracy: 0.1629 - val_loss: 3635.5405
Epoch 2/100
14/14 [==============================] - 267s 19s/step - loss: 3.1142 - accuracy: 0.1384 - val_loss: 809.7969 -
Epoch 3/100
14/14 [==============================] - 267s 19s/step - loss: 2.1801 - accuracy: 0.1763 - val_loss: 9024.6191 -
Epoch 4/100
14/14 [==============================] - 272s 20s/step - loss: 2.2013 - accuracy: 0.2388 - val_loss: 458.3075 -
Epoch 5/100
14/14 [==============================] - 285s 20s/step - loss: 2.0348 - accuracy: 0.1763 - val_loss: 49.4507 - v
Epoch 6/100
14/14 [==============================] - 287s 21s/step - loss: 2.1347 - accuracy: 0.2031 - val_loss: 15.0587 - v
Epoch 7/100
14/14 [==============================] - 294s 21s/step - loss: 2.1853 - accuracy: 0.2210 - val_loss: 6.7742 - va
Epoch 8/100
14/14 [==============================] - 286s 21s/step - loss: 2.2336 - accuracy: 0.2076 - val_loss: 4695.0093 -
Epoch 9/100
14/14 [==============================] - 282s 20s/step - loss: 2.2095 - accuracy: 0.2098 - val_loss: 8583.4541 -
Epoch 10/100
14/14 [==============================] - 278s 20s/step - loss: 1.8529 - accuracy: 0.2522 - val_loss: 2870.6790 -
Epoch 11/100
14/14 [==============================] - 283s 20s/step - loss: 1.7525 - accuracy: 0.2567 - val_loss: 678.3629 -
Epoch 12/100
14/14 [==============================] - 281s 20s/step - loss: 1.6317 - accuracy: 0.2857 - val_loss: 129.9949 -
Epoch 13/100
14/14 [==============================] - 283s 20s/step - loss: 1.7700 - accuracy: 0.2924 - val_loss: 46.4299 - v
Epoch 14/100
14/14 [==============================] - 263s 19s/step - loss: 1.9008 - accuracy: 0.2254 - val_loss: 20.9467 - v
Epoch 15/100
14/14 [==============================] - 280s 20s/step - loss: 1.6456 - accuracy: 0.2679 - val_loss: 6.3133 - va
Epoch 16/100
14/14 [==============================] - 295s 21s/step - loss: 1.6184 - accuracy: 0.2857 - val_loss: 2.0771 - va
Epoch 17/100
14/14 [==============================] - 261s 19s/step - loss: 1.7291 - accuracy: 0.2589 - val_loss: 270.7296 -
Epoch 18/100
14/14 [==============================] - 267s 19s/step - loss: 1.8616 - accuracy: 0.2277 - val_loss: 14268.4688
Epoch 19/100
14/14 [==============================] - 263s 19s/step - loss: 2.0438 - accuracy: 0.2478 - val_loss: 3008.6350 -
Epoch 20/100
14/14 [==============================] - 252s 18s/step - loss: 1.6801 - accuracy: 0.2545 - val_loss: 23104.1113
Epoch 21/100
14/14 [==============================] - 261s 19s/step - loss: 1.6504 - accuracy: 0.2746 - val_loss: 8032.3037 -
Epoch 22/100
14/14 [==============================] - 260s 19s/step - loss: 1.6144 - accuracy: 0.2478 - val_loss: 2292.0510 -
Epoch 23/100
14/14 [==============================] - 269s 19s/step - loss: 1.6486 - accuracy: 0.2612 - val_loss: 1144.6288 -
Epoch 24/100
14/14 [==============================] - 257s 18s/step - loss: 1.6413 - accuracy: 0.2478 - val_loss: 385.1186 -
Epoch 25/100
14/14 [==============================] - 298s 21s/step - loss: 1.6223 - accuracy: 0.2656 - val_loss: 91.5850 - v
Epoch 26/100
14/14 [==============================] - 288s 21s/step - loss: 1.6461 - accuracy: 0.2455 - val_loss: 23.5310 - v
```

Evaluate Keras Model

```
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
4/4 [==============================] - 18s 5s/step - loss: 2.0771 - accuracy: 0.2857
Validation Loss: 2.0771
Validation Accuracy: 28.57%
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Loss: {loss:.4f}")
print(f"Accuracy: {accuracy*100:.2f}%")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-8cbad85e0377> in <cell line: 1>()
----> 1 loss, accuracy = model.evaluate(X_test, y_test)
      2 print(f"Loss: {loss:.4f}")
      3 print(f"Accuracy: {accuracy*100:.2f}%")

                              1 frames
/usr/local/lib/python3.10/dist-packages/keras/src/engine/data_adapter.py in __init__(self, x, y, sample_weight, batch_size, steps_per_epoch, initial_epoch, epochs, shuffle, class_weight, max_queue_size, workers, use_multiprocessing, model, steps_per_execution, distribute, pss_evaluation_shards)
   1317 
   1318         if self._inferred_steps == 0:
-> 1319             raise ValueError("Expected input data to be non-empty.")
   1320 
   1321     def _configure_dataset_and_inferred_steps(

ValueError: Expected input data to be non-empty.
```

Next steps: Explain error

Identify Disease from Test Images

```
def predict_disease(image_path):
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0) / 255.0
    prediction = model.predict(image)
    predicted_label = np.argmax(prediction)
    return diseases[predicted_label]


test_image_path = '/content/Soybean-/Soybean Leaf Dataset for Disease Classification/Yellow Mosaic/YM (48).jpg'  # F
predicted_disease = predict_disease(test_image_path)
print(f"Predicted Disease: {predicted_disease}")
```

```
1/1 [==============================] - 0s 216ms/step
Predicted Disease: Yellow Mosaic
```

RESNET

Load and Preprocess Data

```
import numpy as np
import os
from keras.preprocessing.image import img_to_array, load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define the diseases
diseases = ['Bacterial Pustule', 'Frogeye Leaf Spot', 'Healthy', 'Rust', 'Sudden Death Syndrome', 'Target Leaf Spo

# Initialize lists to store image data and labels
images = []
labels = []

# Load 40 images from each disease for training and validation
# and 10 images for testing
for idx, disease in enumerate(diseases):
    disease_path = os.path.join('/content/Soybean-/Soybean Leaf Dataset for Disease Classification', disease)  # Re
    image_files = [os.path.join(disease_path, f) for f in os.listdir(disease_path) if f.endswith('.jpg')]

    # Split the images into training and validation sets (40 images) and testing set (10 images)
    train_val_images = image_files[:40]
    test_images = image_files[40:]

    for image_file in train_val_images:
        image = load_img(image_file, target_size=(224, 224))  # Resize image to 224x224
        image = img_to_array(image)
        images.append(image)
        labels.append(idx)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,  # Randomly rotate images by 20 degrees
    width_shift_range=0.2,  # Randomly shift images horizontally by 20% of the width
    height_shift_range=0.2,  # Randomly shift images vertically by 20% of the height
    shear_range=0.2,  # Shear intensity (shear angle in radians)
    zoom_range=0.2,  # Randomly zoom images by 20%
    horizontal_flip=True,  # Randomly flip images horizontally
    fill_mode='nearest'  # Fill mode for filling in newly created pixels
)

# Generate augmented images
augmented_images = []
augmented_labels = []
for image, label in zip(images, labels):
    image = np.expand_dims(image, axis=0)  # Expand dimensions to (1, height, width, channels) for flow() method
    for batch in datagen.flow(image, batch_size=1):
        augmented_image = batch[0]
        augmented_images.append(augmented_image)
        augmented_labels.append(label)
        break  # Break the loop after one augmentation to avoid infinite loop

# Convert lists to numpy arrays
augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# Combine original and augmented data
X_train_val = np.concatenate((images, augmented_images), axis=0)
y_train_val = np.concatenate((labels, augmented_labels), axis=0)

# Shuffle the data
```

```
shuffle_idx = np.random.permutation(len(X_train_val))
```

Define ResNet50 Model

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout
from keras.applications import ResNet50
from keras.optimizers import Adam

def create_resnet50_model(input_shape):
    resnet50_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
```