```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
aryashah2k_soybean_seedsclassification_dataset_path = kagglehub.dataset_download('aryashah2k/soybean-seedsclassification-dataset')

print('Data source import complete.')
```

```
#import ing lib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

Loading Data Set

```
data_dir = '/kaggle/input/soybean-seedsclassification-dataset'
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create the Image Data Generator for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # Use 20% for validation
)


# Create the training generator
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=True,
    subset='training'  # Set as training data
)

# Create the validation generator
validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False,
    subset='validation'  # Set as validation data
)
```

```
Found 4412 images belonging to 5 classes.
Found 1101 images belonging to 5 classes.
```

```
import os

# List the directories in train_data_dir
class_labels=os.listdir(data_dir)
print(class_labels)
```

```
['Immature soybeans', 'Intact soybeans', 'Skin-damaged soybeans', 'Spotted soybeans', 'Broken soybeans']
```

Model

Research Paper based code

```
import os
import cv2
import shutil
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Original and new dataset directories
data_dir = '/kaggle/input/soybean-seedsclassification-dataset'  # Original dataset
processed_data_dir = '/kaggle/working/processed_soybean_dataset'  # Processed dataset

# Create processed directory if it doesn't exist
if not os.path.exists(processed_data_dir):
    os.makedirs(processed_data_dir)

# Create subdirectories for each class and copy images
for class_dir in os.listdir(data_dir):
    class_path = os.path.join(processed_data_dir, class_dir)
    if not os.path.exists(class_path):
        os.makedirs(class_path)
#         print(f"Created directory: {class_path}")

    # Copy images to respective class directories
    for img_file in os.listdir(os.path.join(data_dir, class_dir)):
        shutil.copy(os.path.join(data_dir, class_dir, img_file), class_path)
#         print(f"Copied image: {img_file} to {class_path}")


# Function to apply CLAHE to a single image
def apply_clahe(image):
    """
    Apply CLAHE to the input image.
    Args:
    image: Input image in RGB format.

    Returns:
    Image with CLAHE applied.
    """
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    l = np.uint8(l)
    cl = clahe.apply(l)
    merged_image = cv2.merge((cl, a, b))
    return cv2.cvtColor(merged_image, cv2.COLOR_LAB2RGB)

# Custom generator to apply CLAHE during training
class CustomImageDataGenerator(ImageDataGenerator):
    def flow_from_directory(self, directory, **kwargs):
        generator = super().flow_from_directory(directory, **kwargs)
        while True:
            batch_x, batch_y = next(generator)
            # Apply CLAHE to each image in the batch
            for i in range(len(batch_x)):
                batch_x[i] = apply_clahe(batch_x[i])
            yield batch_x, batch_y
```

```
import os
import cv2
import shutil
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Add, BatchNormalization, Activation, Input, MaxPooling2D, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model

# Original and new dataset directories
data_dir = '/kaggle/input/soybean-seedsclassification-dataset'  # Original dataset
processed_data_dir = '/kaggle/working/processed_soybean_dataset'  # Processed dataset

# Create processed directory if it doesn't exist
if not os.path.exists(processed_data_dir):
    os.makedirs(processed_data_dir)

# Create subdirectories for each class and copy images
for class_dir in os.listdir(data_dir):
    class_path = os.path.join(processed_data_dir, class_dir)
    if not os.path.exists(class_path):
        os.makedirs(class_path)
    # Copy images to respective class directories
    for img_file in os.listdir(os.path.join(data_dir, class_dir)):
        shutil.copy(os.path.join(data_dir, class_dir, img_file), class_path)
```

```python
# Initialize ImageDataGenerator
train_datagen = ImageDataGenerator(validation_split=0.2)

# Create the training and validation generators for the processed dataset
train_generator = train_datagen.flow_from_directory(
    processed_data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=True,
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    processed_data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False,
    subset='validation'
)

# Print class indices to confirm they're available
print("Class Indices:", train_generator.class_indices)

# Get the number of classes from the training generator
num_classes = len(train_generator.class_indices)  # Get the number of classes

# Example usage of the modified residual block
input_shape = (150, 150, 3)
inputs = Input(shape=input_shape)

# Initial Conv2D layer
x = Conv2D(64, (7, 7), padding='same', strides=(2, 2))(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

# Residual Blocks
def residual_block(x, filters):
    shortcut = x
    x = Conv2D(filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    if shortcut.shape[-1] != filters:
        shortcut = Conv2D(filters, (1, 1), padding='same')(shortcut)
        shortcut = BatchNormalization()(shortcut)
    x = Add()([x, shortcut])
    x = Activation('relu')(x)
    return x

x = residual_block(x, 64)   # Output shape will be (38, 38, 64)
x = residual_block(x, 128)  # Output shape will be (38, 38, 128)
x = residual_block(x, 256)  # Output shape will be (38, 38, 256)

# Global Average Pooling and Output
x = GlobalAveragePooling2D()(x)
x = Dense(num_classes, activation='softmax')(x)  # Use the number of classes obtained from the training generator

# Create the model
model = Model(inputs, x)

# Summary of the model
model.summary()
```

```
Found 4412 images belonging to 5 classes.
Found 1101 images belonging to 5 classes.
Class Indices: {'Broken soybeans': 0, 'Immature soybeans': 1, 'Intact soybeans': 2, 'Skin-damaged soybeans': 3, 'Spotted soybeans
Model: "functional_2"
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_1       │ (None, 150, 150,  │          0 │ -                 │
│ (InputLayer)        │ 3)                │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_3 (Conv2D)   │ (None, 75, 75,    │      9,472 │ input_layer_1[0]… │
│                     │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalization │ (None, 75, 75,    │        256 │ conv2d_3[0][0]    │
│ (BatchNormalizatio… │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation          │ (None, 75, 75,    │          0 │ batch_normalizat… │
│ (Activation)        │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ max_pooling2d_3     │ (None, 38, 38,    │          0 │ activation[0][0]  │
│ (MaxPooling2D)      │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_4 (Conv2D)   │ (None, 38, 38,    │     36,928 │ max_pooling2d_3[… │
│                     │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │        256 │ conv2d_4[0][0]    │
│ (BatchNormalizatio… │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation_1        │ (None, 38, 38,    │          0 │ batch_normalizat… │
│ (Activation)        │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_5 (Conv2D)   │ (None, 38, 38,    │     36,928 │ activation_1[0][… │
│                     │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │        256 │ conv2d_5[0][0]    │
│ (BatchNormalizatio… │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add (Add)           │ (None, 38, 38,    │          0 │ batch_normalizat… │
│                     │ 64)               │            │ max_pooling2d_3[… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation_2        │ (None, 38, 38,    │          0 │ add[0][0]         │
│ (Activation)        │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_6 (Conv2D)   │ (None, 38, 38,    │     73,856 │ activation_2[0][… │
│                     │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │        512 │ conv2d_6[0][0]    │
│ (BatchNormalizatio… │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation_3        │ (None, 38, 38,    │          0 │ batch_normalizat… │
│ (Activation)        │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_7 (Conv2D)   │ (None, 38, 38,    │    147,584 │ activation_3[0][… │
│                     │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_8 (Conv2D)   │ (None, 38, 38,    │      8,320 │ activation_2[0][… │
│                     │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │        512 │ conv2d_7[0][0]    │
│ (BatchNormalizatio… │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │        512 │ conv2d_8[0][0]    │
│ (BatchNormalizatio… │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add_1 (Add)         │ (None, 38, 38,    │          0 │ batch_normalizat… │
│                     │ 128)              │            │ batch_normalizat… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation_4        │ (None, 38, 38,    │          0 │ add_1[0][0]       │
│ (Activation)        │ 128)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_9 (Conv2D)   │ (None, 38, 38,    │    295,168 │ activation_4[0][… │
│                     │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │      1,024 │ conv2d_9[0][0]    │
│ (BatchNormalizatio… │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ activation_5        │ (None, 38, 38,    │          0 │ batch_normalizat… │
│ (Activation)        │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_10 (Conv2D)  │ (None, 38, 38,    │    590,080 │ activation_5[0][… │
│                     │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_11 (Conv2D)  │ (None, 38, 38,    │     33,024 │ activation_4[0][… │
│                     │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │      1,024 │ conv2d_10[0][0]   │
│ (BatchNormalizatio… │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 38, 38,    │      1,024 │ conv2d_11[0][0]   │
│ (BatchNormalizatio… │ 256)              │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add_2 (Add)         │ (None, 38, 38,    │          0 │ batch_normalizat… │
│                         │ 256)                    │                │ batch_normalizat… │
├─────────────────────────┼─────────────────────────┼────────────────┼───────────────────┤
│ activation_6            │ (None, 38, 38,          │              0 │ add_2[0][0]       │
│ (Activation)            │ 256)                    │                │                   │
├─────────────────────────┼─────────────────────────┼────────────────┼───────────────────┤
│ global_average_poo…     │ (None, 256)             │              0 │ activation_6[0][… │
│ (GlobalAveragePool…     │                         │                │                   │
├─────────────────────────┼─────────────────────────┼────────────────┼───────────────────┤
│ dense_2 (Dense)         │ (None, 5)               │          1,285 │ global_average_p… │
└─────────────────────────┴─────────────────────────┴────────────────┴───────────────────┘
 Total params: 1,238,021 (4.72 MB)
 Trainable params: 1,235,333 (4.71 MB)
 Non-trainable params: 2,688 (10.50 KB)
```

Compile the Model

```
# Compile the model
from tensorflow.keras.optimizers import SGD
model.compile(
    optimizer=SGD(learning_rate=0.01, momentum=0.9),  # Adjust learning rate and momentum as needed
    loss='categorical_crossentropy',  # Use categorical cross-entropy for multi-class classification
    metrics=['accuracy']  # Track accuracy during training
)


from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define EarlyStopping callback to prevent overfitting
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,         # Stop training after 10 epochs without improvement
    restore_best_weights=True  # Restore the model weights from the best epoch
)

# Define ModelCheckpoint callback to save the best model
model_checkpoint = ModelCheckpoint(
    'soybean_drcnn_clahe_best_model.keras',  # Save model to this file with `.keras` extension
    monitor='val_loss',                      # Monitor validation loss
    save_best_only=True,                     # Save only the best model
    verbose=1                                # Verbose output
)


# Train the model
history = model.fit(
    train_generator,                   # Training data generator
    validation_data=validation_generator,  # Validation data generator
    epochs=10,                         # Number of epochs to train
    callbacks=[early_stopping, model_checkpoint],  # Callbacks for early stopping and model checkpointing
    verbose=1                          # Print training progress
)
```

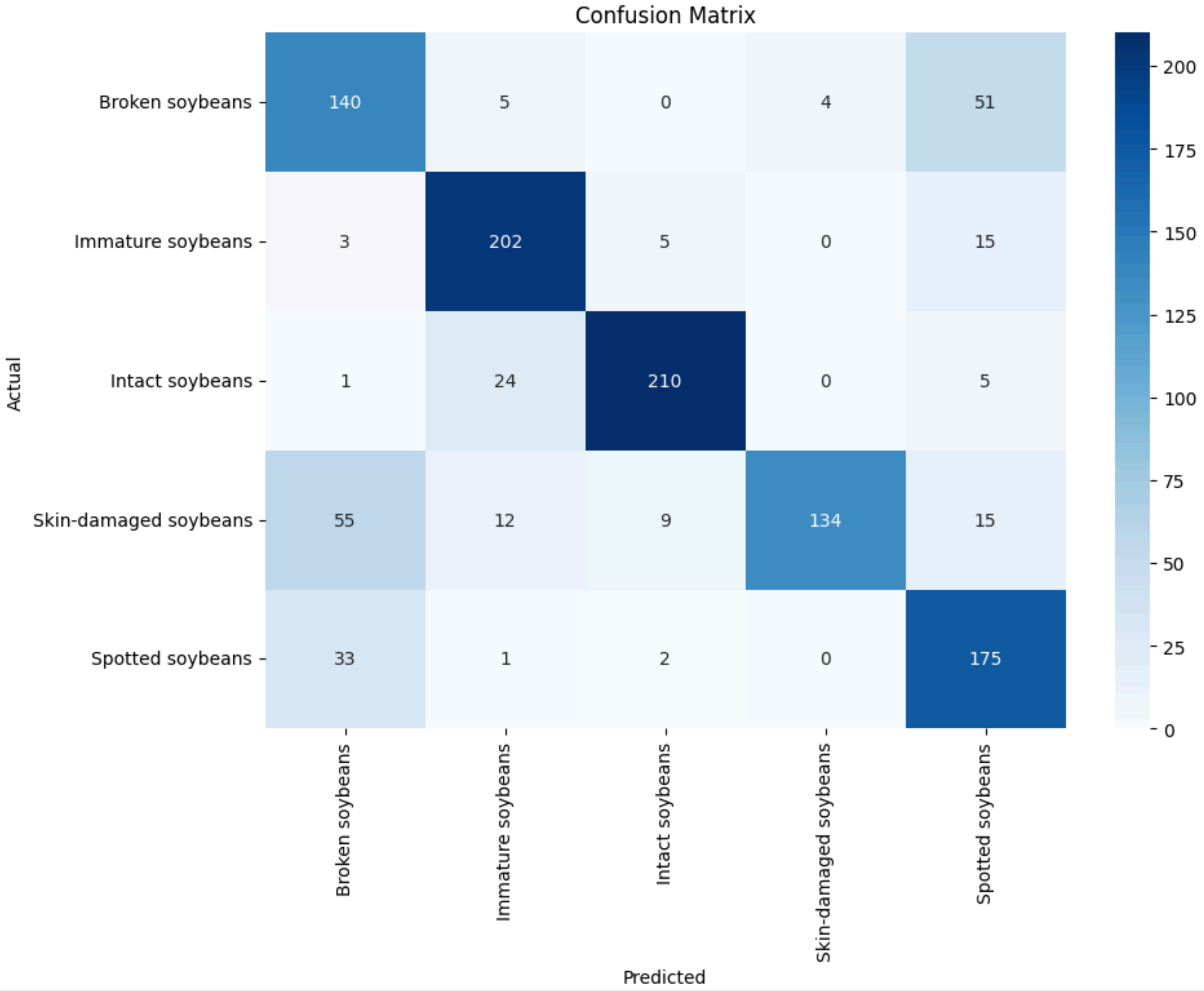
```
Epoch 1/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8103 - loss: 0.5159
Epoch 1: val_loss did not improve from 1.85139
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 82ms/step - accuracy: 0.8104 - loss: 0.5157 - val_accuracy: 0.2234 - val_loss: 6.1157
Epoch 2/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.8256 - loss: 0.4777
Epoch 2: val_loss improved from 1.85139 to 1.12160, saving model to soybean_drcnn_clahe_best_model.keras
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 84ms/step - accuracy: 0.8256 - loss: 0.4777 - val_accuracy: 0.6312 - val_loss: 1.1216
Epoch 3/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.8446 - loss: 0.4169
Epoch 3: val_loss did not improve from 1.12160
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 85ms/step - accuracy: 0.8447 - loss: 0.4169 - val_accuracy: 0.5949 - val_loss: 1.8926
Epoch 4/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.8504 - loss: 0.4117
Epoch 4: val_loss improved from 1.12160 to 0.82485, saving model to soybean_drcnn_clahe_best_model.keras
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 85ms/step - accuracy: 0.8504 - loss: 0.4117 - val_accuracy: 0.7203 - val_loss: 0.8249
Epoch 5/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.8451 - loss: 0.4180
Epoch 5: val_loss improved from 0.82485 to 0.70009, saving model to soybean_drcnn_clahe_best_model.keras
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 84ms/step - accuracy: 0.8452 - loss: 0.4179 - val_accuracy: 0.7511 - val_loss: 0.7001
Epoch 6/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.8649 - loss: 0.3700
Epoch 6: val_loss did not improve from 0.70009
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 82ms/step - accuracy: 0.8649 - loss: 0.3700 - val_accuracy: 0.6485 - val_loss: 1.2385
Epoch 7/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8681 - loss: 0.3556
Epoch 7: val_loss did not improve from 0.70009
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 82ms/step - accuracy: 0.8682 - loss: 0.3556 - val_accuracy: 0.2698 - val_loss: 9.0727
Epoch 8/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8836 - loss: 0.3246
Epoch 8: val_loss did not improve from 0.70009
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - accuracy: 0.8835 - loss: 0.3246 - val_accuracy: 0.2643 - val_loss: 7.5641
Epoch 9/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8850 - loss: 0.3086
```

```
Epoch 9: val_loss did not improve from 0.70009
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 82ms/step - accuracy: 0.8849 - loss: 0.3087 - val_accuracy: 0.5377 - val_loss: 1.5919
Epoch 10/10
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.8836 - loss: 0.3168
Epoch 10: val_loss improved from 0.70009 to 0.62159, saving model to soybean_drcnn_clahe_best_model.keras
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 83ms/step - accuracy: 0.8836 - loss: 0.3168 - val_accuracy: 0.7820 - val_loss: 0.6216
```

Evaluate the Model

```
import matplotlib.pyplot as plt
import seaborn as sns

# Evaluate the model on the validation set
val_loss, val_accuracy = model.evaluate(validation_generator, verbose=1)
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy}')

# Get predictions
predictions = model.predict(validation_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Get true classes
true_classes = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Generate classification report
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(true_classes, predicted_classes, target_names=class_labels))

# Confusion Matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

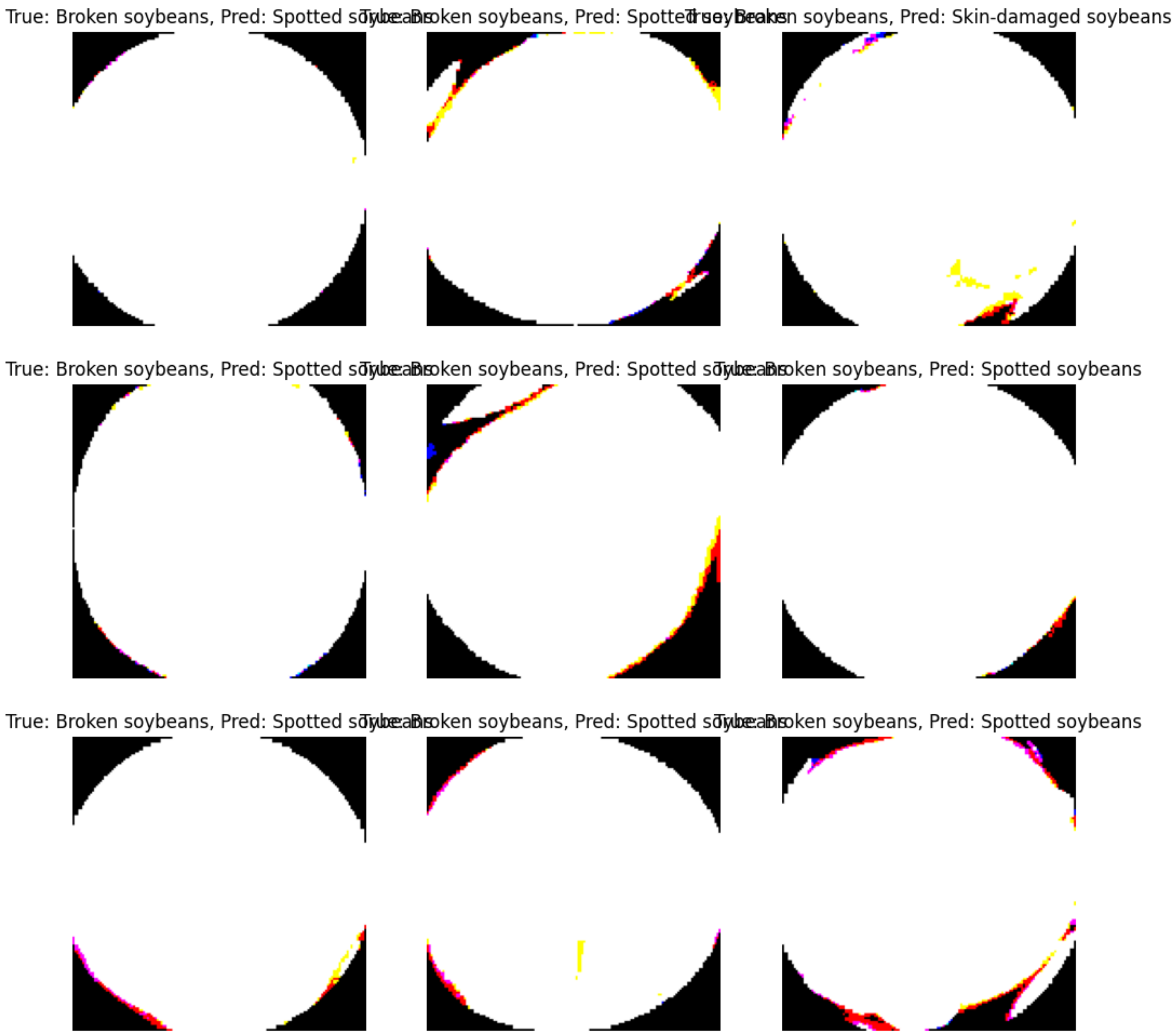
```
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.7685 - loss: 0.5997
Validation Loss: 0.6215949058532715, Validation Accuracy: 0.7820163369178772
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step
                        precision    recall  f1-score   support

      Broken soybeans       0.60      0.70      0.65       200
    Immature soybeans       0.83      0.90      0.86       225
      Intact soybeans       0.93      0.88      0.90       240
Skin-damaged soybeans       0.97      0.60      0.74       225
     Spotted soybeans       0.67      0.83      0.74       211

             accuracy                           0.78      1101
            macro avg       0.80      0.78      0.78      1101
         weighted avg       0.81      0.78      0.78      1101
```



	Broken soybeans	Immature soybeans	Intact soybeans	Skin-damaged soybeans	Spotted soybeans
Broken soybeans	140	5	0	4	51
Immature soybeans	3	202	5	0	15
Intact soybeans	1	24	210	0	5
Skin-damaged soybeans	55	12	9	134	15
Spotted soybeans	33	1	2	0	175

Analyze Errors

```
misclassified_indices = np.where(predicted_classes != true_classes)[0]
plt.figure(figsize=(12, 12))
for i, idx in enumerate(misclassified_indices[:9]):  # Display 9 misclassifications
    plt.subplot(3, 3, i + 1)
    plt.imshow(validation_generator[idx][0][0])  # Accessing the image
    plt.title(f'True: {class_labels[true_classes[idx]]}, Pred: {class_labels[predicted_classes[idx]]}')
    plt.axis('off')
plt.show()
```


True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Skin-damaged soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans
True: Broken soybeans, Pred: Spotted soybeans

Hyperparameter Tuning

```
from keras_tuner import RandomSearch
from keras.models import Sequential
from keras.layers import Flatten, Dense, Conv2D,MaxPooling2D
import tensorflow as tf

def build_model(lr):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(150, 150, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(len(class_labels), activation='softmax'))
    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = RandomSearch(build_model,
                    objective='val_accuracy',
                    max_trials=5,
                    executions_per_trial=3,
                    directory='my_dir',
                    project_name='soybean_classification')

tuner.search(train_generator, validation_data=validation_generator, epochs=10)
```

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

```
Trial 1 Complete [00h 01m 54s]
val_accuracy: 0.20920374989509583

Best val_accuracy So Far: 0.20920374989509583
Total elapsed time: 00h 01m 54s
```

Save and Deploy the Model

```
# Define a function to save the model in both formats
def save_model(model):
    try:
        # Save the model in HDF5 format
        model.save('soybean_classifier.h5')
        print("Model saved successfully as 'soybean_classifier.h5'")
    except Exception as e:
        print(f"Error saving model as HDF5: {e}")

    try:
        # Save the model in Keras format
        model.save('soybean_classifier.keras')
        print("Model saved successfully as 'soybean_classifier.keras'")
    except Exception as e:
        print(f"Error saving model as Keras: {e}")

# Train the model (assuming you have already trained your model)
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=7,
    callbacks=[early_stopping, model_checkpoint],
    verbose=1
)

# Call the function to save the model
save_model(model)

# Move the model files to the output directory for download
import shutil

try:
    shutil.move('soybean_classifier.h5', '/kaggle/working/soybean_classifier.h5')
    print("Model file moved to /kaggle/working directory for download.")
except Exception as e:
    print(f"Error moving HDF 5 model file: {e}")

try:
    shutil.move('soybean_classifier.keras', '/kaggle/working/soybean_classifier.keras')
    print("Model file moved to /kaggle/working directory for download.")
except Exception as e:
    print(f"Error moving Keras model file: {e}")
```

```
Epoch 1/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8907 - loss: 0.2951
Epoch 1: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 83ms/step - accuracy: 0.8906 - loss: 0.2951 - val_accuracy: 0.6440 - val_loss: 1.1919
Epoch 2/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.9016 - loss: 0.2770
Epoch 2: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 84ms/step - accuracy: 0.9015 - loss: 0.2772 - val_accuracy: 0.3288 - val_loss: 3.5901
Epoch 3/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.9023 - loss: 0.2756
Epoch 3: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 85ms/step - accuracy: 0.9023 - loss: 0.2756 - val_accuracy: 0.2216 - val_loss: 9.6521
Epoch 4/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.9011 - loss: 0.2761
Epoch 4: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 84ms/step - accuracy: 0.9011 - loss: 0.2761 - val_accuracy: 0.6948 - val_loss: 1.0079
Epoch 5/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.9237 - loss: 0.2283
Epoch 5: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 12s 82ms/step - accuracy: 0.9237 - loss: 0.2285 - val_accuracy: 0.7702 - val_loss: 0.6586
Epoch 6/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.9153 - loss: 0.2377
Epoch 6: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - accuracy: 0.9153 - loss: 0.2377 - val_accuracy: 0.4142 - val_loss: 3.6588
Epoch 7/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.9232 - loss: 0.2156
Epoch 7: val_loss did not improve from 0.62159
138/138 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - accuracy: 0.9231 - loss: 0.2157 - val_accuracy: 0.7130 - val_loss: 0.9840
Epoch 8/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.9119 - loss: 0.2326
```