```
import tensorflow as tf
from google.colab import drive
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow.keras import layers, models
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import accuracy_score
import seaborn as sns
import random

# Mount Google Drive
drive.mount('/content/drive')

# Dataset directories and file paths
image_base_path = "/content/drive/MyDrive/Image_dataset"
csv_file_path = "/content/drive/MyDrive/Capstone_csv/crop_recommend.csv"

# Load CSV file for state-crop mapping
crop_data = pd.read_csv(csv_file_path)
class_names = os.listdir(image_base_path)

# Load dataset and split
ds_train = tf.keras.preprocessing.image_dataset_from_directory(
    image_base_path,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(224, 224),
    batch_size=32
)

ds_validation = tf.keras.preprocessing.image_dataset_from_directory(
    image_base_path,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(224, 224),
    batch_size=32
)

# Data augmentation
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2)
])

ds_train = ds_train.map(lambda x, y: (data_augmentation(x, training=True), y))

# EfficientNet model with fine-tuning
num_classes = len(class_names)
base_model = EfficientNetB0(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

# Fine-tune some layers
for layer in base_model.layers[-20:]:
    layer.trainable = True

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),  # Increased layer size
    layers.Dropout(0.5),  # Increased dropout
    layers.BatchNormalization(),
    layers.Dense(num_classes, activation='softmax')
])

# Compile model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Early stopping and learning rate reduction
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

# Training the model with callbacks
```

```
history = model.fit(
    ds_train,
    validation_data=ds_validation,
    epochs=100,  # Increased epochs for better training
    callbacks=[early_stopping, reduce_lr]
)

# Testing and evaluation
ds_test = tf.keras.preprocessing.image_dataset_from_directory(
    image_base_path,
    image_size=(224, 224),
    batch_size=32
)

test_loss, test_accuracy = model.evaluate(ds_test)
test_results = {'loss': test_loss, 'accuracy': test_accuracy}

# Calculate and plot crop accuracy per state
def calculate_and_plot_crop_accuracy(crop_data):
    accuracy_data = []
    for (state, crop), group in crop_data.groupby(['State_name', 'Crop_name']):
        total_recommendations = len(group)
        correct_recommendations = group[group['Crop_name'] == crop].count()['Crop_name']
        accuracy_percentage = (correct_recommendations / total_recommendations) * 100 if total_recommendations > 0 else 0
        accuracy_data.append({'State': state, 'Crop': crop, 'Accuracy': accuracy_percentage})

    accuracy_df = pd.DataFrame(accuracy_data)

    # Plotting crop accuracy
    plt.figure(figsize=(12, 8))
    for state in accuracy_df['State'].unique():
        state_data = accuracy_df[accuracy_df['State'] == state]
        plt.bar(state_data['Crop'], state_data['Accuracy'], label=state, alpha=0.7)
    plt.title("Crop Recommendation Accuracy per State")
    plt.xlabel("Crop")
    plt.ylabel("Accuracy (%)")
    plt.xticks(rotation=45)
    plt.legend(title='State', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()

# Plotting function for training, validation, and testing history
def plot_training_validation_testing_history(history, test_results=None):
    plt.figure(figsize=(14, 6))

    # Plot Loss
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    if test_results:
        plt.axhline(y=test_results['loss'], color='r', linestyle='--', label='Test Loss')
    plt.title('Training, Validation, and Test Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # Plot Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    if test_results:
        plt.axhline(y=test_results['accuracy'], color='r', linestyle='--', label='Test Accuracy')
    plt.title('Training, Validation, and Test Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Plot training, validation, and test accuracy/loss
plot_training_validation_testing_history(history, test_results)

# Sample and display random images with crop names
def get_random_images_with_crop_names(base_path, crop_data, num_images=5):
    state_folders = os.listdir(base_path)
    state_crop_images = {}

    for state in state_folders:
        state_path = os.path.join(base_path, state)
        if os.path.isdir(state_path):
```

```
                images = os.listdir(state_path)
                random_images = random.sample(images, min(num_images, len(images)))

                state_crop = crop_data[crop_data['State_name'] == state]['Crop_name'].unique()
                crop_name = state_crop[0] if len(state_crop) > 0 else "Unknown Crop"

                state_crop_images[state] = {'images': random_images, 'crop': crop_name}

        return state_crop_images

    def display_images_with_crop_names(state_crop_images, base_path):
        plt.figure(figsize=(12, 12))

        for i, (state, data) in enumerate(state_crop_images.items()):
            for j, img_name in enumerate(data['images']):
                img_path = os.path.join(base_path, state, img_name)
                img = Image.open(img_path).resize((150, 150))
                plt.subplot(len(state_crop_images), len(data['images']), i * len(data['images']) + j + 1)
                plt.imshow(img)
                plt.axis('off')
                plt.title(f"{state} - {data['crop']}", fontsize=8)

        plt.tight_layout()
        plt.show()

    # Fetch and display random images with crop names
    state_crop_images = get_random_images_with_crop_names(image_base_path, crop_data, num_images=3)
    display_images_with_crop_names(state_crop_images, image_base_path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
Found 1302 files belonging to 10 classes.
Using 1042 files for training.
Found 1302 files belonging to 10 classes.
Using 260 files for validation.
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 453s 11s/step - accuracy: 0.1361 - loss: 2.9019 - val_accuracy: 0.1654 - val_loss: 2.2458 - learning
Epoch 2/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 604ms/step - accuracy: 0.3027 - loss: 2.1872 - val_accuracy: 0.3115 - val_loss: 2.0405 - learnir
Epoch 3/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 22s 644ms/step - accuracy: 0.4605 - loss: 1.6428 - val_accuracy: 0.4462 - val_loss: 1.7787 - learnir
Epoch 4/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 41s 646ms/step - accuracy: 0.5265 - loss: 1.4632 - val_accuracy: 0.5769 - val_loss: 1.4672 - learnir
Epoch 5/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 602ms/step - accuracy: 0.6036 - loss: 1.1843 - val_accuracy: 0.6654 - val_loss: 1.2060 - learnir
Epoch 6/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 22s 629ms/step - accuracy: 0.6689 - loss: 1.0166 - val_accuracy: 0.7462 - val_loss: 0.9293 - learnir
Epoch 7/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 611ms/step - accuracy: 0.7071 - loss: 0.9431 - val_accuracy: 0.7769 - val_loss: 0.7770 - learnir
Epoch 8/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 23s 660ms/step - accuracy: 0.7346 - loss: 0.8580 - val_accuracy: 0.7885 - val_loss: 0.6761 - learnir
Epoch 9/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 40s 638ms/step - accuracy: 0.7593 - loss: 0.7275 - val_accuracy: 0.8192 - val_loss: 0.5913 - learnir
Epoch 10/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 631ms/step - accuracy: 0.7896 - loss: 0.6408 - val_accuracy: 0.8000 - val_loss: 0.5447 - learnir
Epoch 11/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 42s 674ms/step - accuracy: 0.7889 - loss: 0.5816 - val_accuracy: 0.8038 - val_loss: 0.5461 - learnir
Epoch 12/100
```

```
print(test_results)
print(accuracy_score(y_true, y_pred))
```

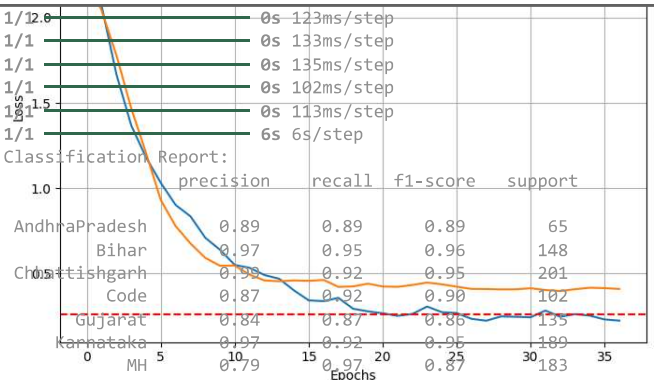
```
{'loss': 0.2574582099914551, 'accuracy': 0.9201228618621826}
0.9201228878648233
```

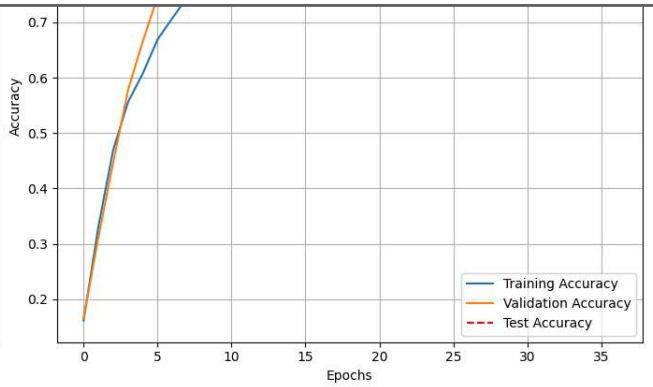
```
Epoch 14/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 628ms/step - accuracy: 0.8329 - loss: 0.4701 - val_accuracy: 0.8346 - val_loss: 0.4540 - learnir
Epoch 15/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 22s 641ms/step - accuracy: 0.8658 - loss: 0.3874 - val_accuracy: 0.8346 - val_loss: 0.4589 - learnir
```

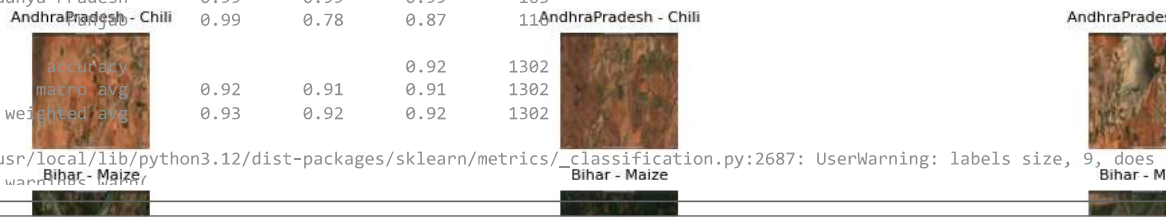
```
from sklearn.metrics import classification_report

# Get true labels and predictions for the test dataset
y_true = []
y_pred = []

for images, labels in ds_test:
    predictions = model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(tf.argmax(predictions, axis=1).numpy())

# Get unique classes from predictions and true labels
unique_classes = sorted(list(set(y_true + y_pred)))

# Generate the classification report using the unique classes as labels
report = classification_report(y_true, y_pred, labels=unique_classes, target_names=ds_test.class_names)
print("Classification Report:\n", report)
```

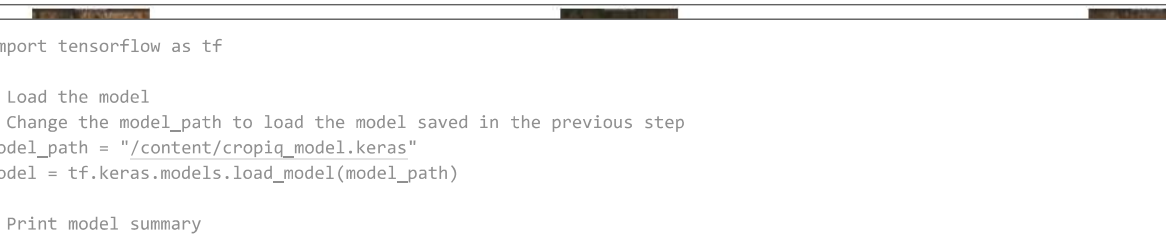
```
Epoch 26/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 600ms/step - accuracy: 0.9302 - loss: 0.2451 - val_accuracy: 0.8808 - val_loss: 0.4222 - learnir
Epoch 27/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 22s 629ms/step - accuracy: 0.9253 - loss: 0.2263 - val_accuracy: 0.8846 - val_loss: 0.4092 - learnir
Epoch 28/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 600ms/step - accuracy: 0.9184 - loss: 0.2252 - val_accuracy: 0.8846 - val_loss: 0.4084 - learnir
Epoch 29/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 612ms/step - accuracy: 0.9201 - loss: 0.2735 - val_accuracy: 0.8846 - val_loss: 0.4063 - learnir
Epoch 30/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 41s 623ms/step - accuracy: 0.9125 - loss: 0.2752 - val_accuracy: 0.8846 - val_loss: 0.4067 - learnir
Epoch 31/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 636ms/step - accuracy: 0.8938 - loss: 0.2969 - val_accuracy: 0.8769 - val_loss: 0.4130 - learnir
Epoch 32/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 41s 637ms/step - accuracy: 0.9126 - loss: 0.2793 - val_accuracy: 0.8962 - val_loss: 0.4026 - learnir
Epoch 33/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 587ms/step - accuracy: 0.9177 - loss: 0.2419 - val_accuracy: 0.8923 - val_loss: 0.3966 - learnir
Epoch 34/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 21s 636ms/step - accuracy: 0.9260 - loss: 0.2465 - val_accuracy: 0.8808 - val_loss: 0.4073 - learnir
Epoch 35/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 42s 664ms/step - accuracy: 0.9252 - loss: 0.2479 - val_accuracy: 0.8769 - val_loss: 0.4160 - learnir
Epoch 36/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 20s 609ms/step - accuracy: 0.9121 - loss: 0.2392 - val_accuracy: 0.8769 - val_loss: 0.4138 - learnir
Epoch 37/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 22s 639ms/step - accuracy: 0.9284 - loss: 0.2177 - val_accuracy: 0.8808 - val_loss: 0.4089 - learnir
Found 1302 files belonging to 10 classes.
41/41 ━━━━━━━━━━━━━━━━━━━━ 19s 452ms/step - accuracy: 0.9214 - loss: 0.2640
```

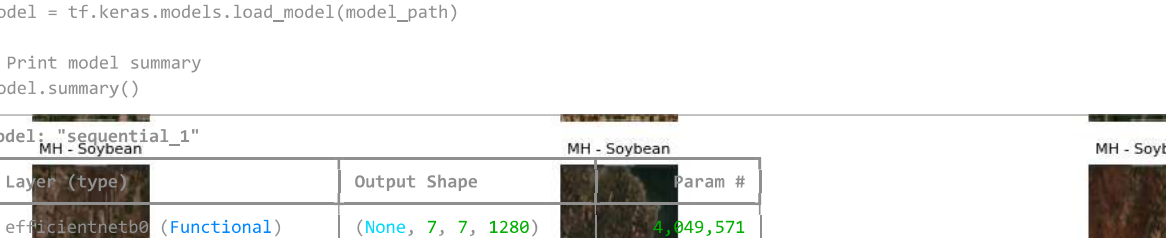
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step
Classification Report:
                precision    recall  f1-score   support

 AndhraPradesh       0.89      0.89      0.89        65
         Bihar       0.97      0.95      0.96       148
  Chhattishgarh      0.99      0.92      0.95       201
          Code       0.87      0.92      0.90       102
       Gujarat       0.84      0.87      0.86       135
     Karnataka       0.97      0.92      0.95       189
            MH       0.79      0.97      0.87       183
Madhya Pradesh       0.99      0.99      0.99       163
        Punjab       0.99      0.78      0.87       116

      accuracy                           0.92      1302
     macro avg       0.92      0.91      0.91      1302
  weighted avg       0.93      0.92      0.92      1302

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2687: UserWarning: labels size, 9, does not match si
  warnings.warn(
```

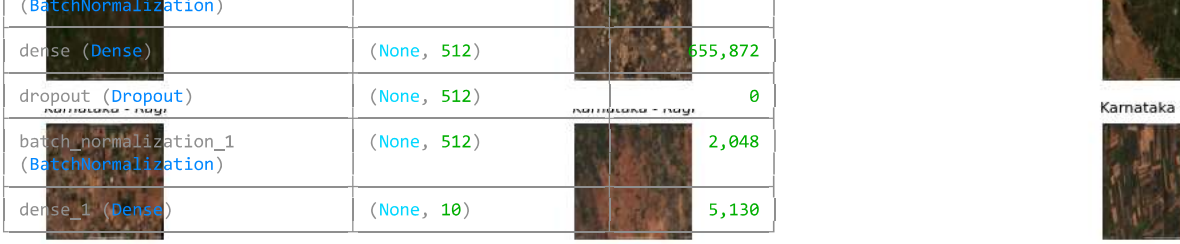













```
import keras.saving

# Save the full model
model.save('/content/cropiq_model.keras')
```

```
import tensorflow as tf

# Load the model
# Change the model_path to load the model saved in the previous step
model_path = "/content/cropiq_model.keras"
model = tf.keras.models.load_model(model_path)

# Print model summary
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
batch_normalization (BatchNormalization)	(None, 1280)	5,120
dense (Dense)	(None, 512)	655,872
dropout (Dropout)	(None, 512)	0
batch_normalization_1 (BatchNormalization)	(None, 512)	2,048
dense_1 (Dense)	(None, 10)	5,130

```
Total params: 14,062,011 (53.64 MB)
Trainable params: 4,672,134 (17.82 MB)
```

```
import tensorflow as tf

# Load the trained model, provide the full path if it's not in the current directory
model_path = "/content/cropiq_model.keras"  # Update with the actual path if needed
model = tf.keras.models.load_model(model_path)

# Convert to TFLite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TFLite model
with open("model.tflite", "wb") as f:
    f.write(tflite_model)
```