


```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
import os
import json
import numpy as np
import yaml
from PIL import Image
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, precision_score, recall_score
```

```
# Paths to the dataset
dataset_dir = '/content/drive/MyDrive/Datasets/WeedCrop.v1i.yolov5pytorch'
train_dir = os.path.join(dataset_dir, 'train')
valid_dir = os.path.join(dataset_dir, 'valid')
test_dir = os.path.join(dataset_dir, 'test')
data_yaml = os.path.join(dataset_dir, 'data.yaml')

# Load class names from data.yaml
def load_data(img_dir, label_dir):
    images = []
    labels = []
    for img_name in os.listdir(img_dir):
        if img_name.endswith('.jpg'):
            img_path = os.path.join(img_dir, img_name)
            label_path = os.path.join(label_dir, os.path.splitext(img_name)[0] + '.txt')

            # Check if the annotation file exists
            if not os.path.exists(label_path):
                print(f"Annotation file {label_path} not found, skipping...")
                continue

            # Load image
            img = Image.open(img_path)
            img = img.resize((224, 224))
            images.append(np.array(img) / 255.0) # Normalize to [0, 1]

            # Load annotation and set label (1 for weed, 0 for crop)
            with open(label_path, 'r') as f:
                anns = f.read().strip().split('\n')
                is_weed = False
                for ann in anns:
                    parts = ann.split(' ')
                    if len(parts) > 0 and parts[0].isdigit():
                        cls_id = int(parts[0])
                        if classes[cls_id] == 'weed':
                            is_weed = True
                            break
                labels.append(1 if is_weed else 0)
    return np.array(images), np.array(labels)

# Load the datasets
X_train, y_train = load_data(os.path.join(train_dir, 'images'), os.path.join(train_dir, 'labels'))
X_val, y_val = load_data(os.path.join(valid_dir, 'images'), os.path.join(valid_dir, 'labels'))
X_test, y_test = load_data(os.path.join(test_dir, 'images'), os.path.join(test_dir, 'labels'))
```

 [Show hidden output](#)

```
# Data augmentation for training data
train_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
)

# Data augmentation for validation and test data (only rescaling)
val_datagen = ImageDataGenerator()

# Create data generators
train_generator = train_datagen.flow(X_train, y_train, batch_size=32)
val_generator = val_datagen.flow(X_val, y_val, batch_size=32)
test_generator = val_datagen.flow(X_test, y_test, batch_size=32, shuffle=False)

# Load VGG16 model pre-trained on ImageNet, excluding top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Adding custom top layers for binary classification
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Create the optimizer without weight_decay or other unsupported parameters
optimizer = Adam(learning_rate=0.0001)

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

```
# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('weed_detection_VGG-16_.keras', save_best_only=True, monitor='val_loss')

# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=15,
    callbacks=[early_stopping, model_checkpoint]
)

# Save the model
model.save('weed_detection_VGG-16_.h5')
```

```
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
self._warn_if_super_not_called()
72/72 ━━━━━━━━━━━ 120s 1s/step - accuracy: 0.9049 - loss: 0.2968 - val_accuracy: 0.9702 - val_loss: 0.1225
Epoch 2/15
72/72 ━━━━━━━━━━━ 41s 532ms/step - accuracy: 0.9438 - loss: 0.2111 - val_accuracy: 0.9702 - val_loss: 0.1259
Epoch 3/15
72/72 ━━━━━━━━━━━ 42s 542ms/step - accuracy: 0.9344 - loss: 0.2349 - val_accuracy: 0.9702 - val_loss: 0.1425
Epoch 4/15
72/72 ━━━━━━━━━━━ 81s 543ms/step - accuracy: 0.9395 - loss: 0.2345 - val_accuracy: 0.9702 - val_loss: 0.1240
Epoch 5/15
72/72 ━━━━━━━━━━━ 49s 650ms/step - accuracy: 0.9330 - loss: 0.2338 - val_accuracy: 0.9702 - val_loss: 0.1179
Epoch 6/15
72/72 ━━━━━━━━━━━ 74s 542ms/step - accuracy: 0.9459 - loss: 0.2204 - val_accuracy: 0.9702 - val_loss: 0.1351
Epoch 7/15
72/72 ━━━━━━━━━━━ 42s 549ms/step - accuracy: 0.9521 - loss: 0.1885 - val_accuracy: 0.9702 - val_loss: 0.1240
Epoch 8/15
72/72 ━━━━━━━━━━━ 42s 541ms/step - accuracy: 0.9367 - loss: 0.2222 - val_accuracy: 0.9702 - val_loss: 0.1668
Epoch 9/15
72/72 ━━━━━━━━━━━ 81s 542ms/step - accuracy: 0.9345 - loss: 0.2216 - val_accuracy: 0.9702 - val_loss: 0.1754
Epoch 10/15
72/72 ━━━━━━━━━━━ 42s 542ms/step - accuracy: 0.9476 - loss: 0.1938 - val_accuracy: 0.9702 - val_loss: 0.1512
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
# Load the best model
model = tf.keras.models.load_model('weed_detection_VGG-16_.h5')

# Evaluate on the test set
y_pred = (model.predict(X_test) > 0.5).astype("int32")


# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# F1 Score
f1 = f1_score(y_test, y_pred, average='weighted')
print("F1 Score:", f1)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Precision
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)

# Recall
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)
```

⚠ WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you call `model.compile()`.
 4/4  14s 4s/step

Confusion Matrix:
 [[0 9]
 [0 109]]

F1 Score: 0.8871052042111551
 Accuracy: 0.923728813559322
 Precision: 0.8532749209997127
 Recall: 0.923728813559322

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision is ill-defined ar
 _warn_prf(average, modifier, msg_start, len(result))

✓ Testing

Classification test on an image

```
import numpy as np
from PIL import Image
import tensorflow as tf

# Function to load and preprocess a single image
def load_and_preprocess_image(img_path):
    img = Image.open(img_path)
    img = img.resize((224, 224))
    img = np.array(img) / 255.0 # Normalize to [0, 1]
    img = np.expand_dims(img, axis=0) # Add batch dimension
    return img

# Function to make a prediction on a single image
def predict_image(model, img_path):
    img = load_and_preprocess_image(img_path)
    display(Image.fromarray((img[0] * 255).astype(np.uint8)))
    prediction = model.predict(img)
    return np.argmax(prediction[0])

# An example image
example_img_path = '/content/drive/MyDrive/Datasets/WeedCrop.v1i.yolov5pytorch/test/images/301_jpg.rf.e0a19ebd17c5738934d155d4747cecba.jpg'

# Load the model
model = tf.keras.models.load_model('weed_detection_VGG-16_.h5')

# Make a prediction
prediction = predict_image(model, example_img_path)

# Output the prediction
if prediction > 0.5:
    print(f"The image is predicted to be a weed with a probability of {prediction:.4f}")
else:
    print(f"The image is predicted to be a crop with a probability of {1 - prediction:.4f}")
```

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until ,



1/1 1s 1s/step