crop classification completed (1)

November 6, 2024

1 1. Install Dependencies and Setup

[ ]:
```
%pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

[ ]:
```
!pip list
```

[ ]:
```
import tensorflow as tf
import os
```

[ ]:
```
# Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

[ ]:
```
tf.config.list_physical_devices('GPU')
```

2 2. Remove dodgy images

[ ]:
```
import cv2
import imghdr
```

[ ]:
```
data_dir ="Users\eakes\Documents\Applicative␣
 ↪Project\flask_crop_classifier\data\maize"
```

[ ]:
```
import os

y_path = r"C:\Users\eakes\Documents\Applicative␣
 ↪Project\flask_crop_classifier\data\maize"
print(os.listdir(y_path))
```

[ ]:
```
image_exts = ['jpeg','jpg', 'bmp', 'png']
```

[ ]:
```
import os
import cv2
import imghdr
```

```
# Define your data directory and acceptable image formats
data_dir = r"C:\Users\eakes\Documents\Applicative␣
 ↪Project\flask_crop_classifier\data\maize"
image_exts = ['jpeg', 'jpg', 'png', 'gif', 'bmp']  # Add any other formats you␣
 ↪want to include

# Loop through each class subdirectory
for image_class in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, image_class)

    if os.path.isdir(class_dir):  # Ensure it's a directory
        for image in os.listdir(class_dir):
            image_path = os.path.join(class_dir, image)
            try:
                img = cv2.imread(image_path)  # Try to read the image
                tip = imghdr.what(image_path)  # Get the image type

                if tip not in image_exts:
                    print(f'Image not in ext list: {image_path}')
                    os.remove(image_path)  # Remove the image if not valid
            except Exception as e:
                print(f'Issue with image {image_path}: {e}')  # Print the␣
 ↪exception message
                # Optionally, you can uncomment the next line to remove the␣
 ↪problematic image
                # os.remove(image_path)
```

3 3. Load Data

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
```

[ ]:
```
data = tf.keras.utils.image_dataset_from_directory(r"C:
 ↪\Users\eakes\Documents\Applicative Project\flask_crop_classifier\data")
```

[ ]:
```
data_iterator = data.as_numpy_iterator()
```

[ ]:
```
batch = data_iterator.next()
```

[ ]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```

4 4. Scale Data

[ ]:
```
data = data.map(lambda x,y: (x/255, y))
```

[ ]:
```
data.as_numpy_iterator().next()
```

5 5. Split Data

[ ]:
```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
```

[ ]:
```
train_size
```

[ ]:
```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

6 6. Build Deep Learning Model

[ ]:
```
train
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,␣
↪Dropout
```

[ ]:
```
model = Sequential()
```

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

# Initialize the model
model = Sequential()

# Add layers
model.add(Conv2D(16, (3, 3), strides=1, activation='relu', input_shape=(256,␣
↪256, 3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3, 3), strides=1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Adjust based on your output needs
```

[ ]:
```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

[ ]:
```
model.summary()
```

7 7. Train

[ ]:
```
logdir='logs'
```

[ ]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

[ ]:
```
hist = model.fit(train, epochs=10, validation_data=val,␣
↪callbacks=[tensorboard_callback])
```

8 8. Plot Performance

[ ]:
```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

[ ]:
```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

9 9. Evaluate

[ ]:
```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

[ ]:
```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

[ ]:
```
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y, yhat)
    re.update_state(y, yhat)
    acc.update_state(y, yhat)
```

[ ]:
```
print(pre.result(), re.result(), acc.result())
```

10 10. Test

[ ]:
```
import cv2
```

[ ]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
# Ensure the correct path to the image is provided
img = cv2.imread(r"C:\Users\eakes\Documents\Applicative␣
 ↪Project\flask_crop_classifier\data\maize\20200612_104246_jpg.rf.
 ↪f8aaf88d286d20c00dd3a6cb4ebf0993.jpg")
if img is not None:
    plt.imshow(img)
    plt.show()
else:
    print("Error: Image not found or cannot be opened.")
```

[ ]:
```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

[ ]:
```
import cv2
import matplotlib.pyplot as plt

# Convert the image from BGR to RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the image using Matplotlib
plt.imshow(img_rgb)
plt.axis('off')  # Optional: Turn off axis
plt.show()
```

[ ]:
```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

[ ]:
```
yhat
```

[ ]:
```
if yhat > 0.5:
    print(f'Predicted class is maize')
else:
    print(f'Predicted class is chilli')
```

11 11. Save the Model

[ ]:
```
from tensorflow.keras.models import load_model
```

[ ]:
```
import os

if not os.path.exists('models'):
    os.makedirs('models')
```

[ ]:
```
model.save(os.path.join('models','imageclassifier.keras'))
```

[ ]:
```
new_model = load_model(os.path.join('models','imageclassifier.keras'))
```

[ ]:
```
yhatnew = new_model.predict(np.expand_dims(resize/255,0))
```

[ ]:
```
model.export(os.path.join('models', 'imageclassifier_tf'))
```

[ ]:
```
if yhat > 0.5:
    print(f'Predicted class is maize')
else:
    print(f'Predicted class is chilli')
```

[ ]:
```
from tensorflow.keras.models import load_model
import numpy as np
import os

# Ensure the directory exists
if not os.path.exists('models'):
    os.makedirs('models')

# Save the model in the models directory
model.save(os.path.join('models', 'imageclassifier.keras'))

# Load the model
new_model = load_model(os.path.join('models', 'imageclassifier.keras'))

# Predict with a sample image (replace 'resize' with the processed image array)
# Assuming 'resize' is an image array
yhatnew = new_model.predict(np.expand_dims(resize / 255.0, axis=0))

# Interpretation of the prediction
if yhatnew > 0.5:
    print('Predicted class is maize')
else:
    print('Predicted class is chilli')
```

[ ]:
```
import zipfile
import os

# Path to your saved model
model_path = 'models/imageclassifier.h5'
```

```
zip_path = 'models/imageclassifier.zip'

# Zip the model file
with zipfile.ZipFile(zip_path, 'w') as zipf:
    zipf.write(model_path, os.path.basename(model_path))

print(f"Model zipped at {zip_path}")
```