IBM TEAM_ECOPULSE

October 19, 2024

[13]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[ ]:
```
```

[20]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[33]:
```
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20,␣
 ↪width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('train', target_size=(128,␣
 ↪128), batch_size=32, class_mode='binary')
validation_generator = test_datagen.flow_from_directory('val',␣
 ↪target_size=(128, 128), batch_size=32, class_mode='binary')
```

```
Found 656 images belonging to 4 classes.
Found 38 images belonging to 1 classes.
```

[34]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))  #␣
 ↪Adjust input_shape
model.add(MaxPooling2D((2, 2)))
# Add more convolutional and pooling layers as needed
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification
```

[35]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[42]:
```
from PIL import Image
import os

def check_images(directory):
    for filename in os.listdir(directory):
        try:
            img = Image.open(os.path.join(directory, filename))
            img.verify()  # Verify the image is not corrupted
        except (IOError, SyntaxError) as e:
            print(f"Bad file: {filename}")

check_images('train')
```

```
Bad file: training
Bad file: .ipynb_checkpoints
Bad file: bleached
Bad file: unbleached
```

[43]:
```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

[39]:
```
model.fit(train_generator, steps_per_epoch=len(train_generator), epochs=10,␣
↪validation_data=validation_generator,␣
↪validation_steps=len(validation_generator))
```

```
Epoch 1/10
21/21 [==============================] - 28s 1s/step - loss: -1492.2742 -
accuracy: 0.2622 - val_loss: 5446.7334 - val_accuracy: 0.0000e+00
Epoch 2/10
21/21 [==============================] - 23s 1s/step - loss: -14500.7393 -
accuracy: 0.2805 - val_loss: 31789.0625 - val_accuracy: 0.0000e+00
Epoch 3/10
21/21 [==============================] - 24s 1s/step - loss: -60677.3672 -
accuracy: 0.2805 - val_loss: 109299.7422 - val_accuracy: 0.0000e+00
Epoch 4/10
21/21 [==============================] - 19s 866ms/step - loss: -172320.2969 -
accuracy: 0.2805 - val_loss: 278098.2500 - val_accuracy: 0.0000e+00
Epoch 5/10
21/21 [==============================] - 20s 948ms/step - loss: -394388.0000 -
accuracy: 0.2805 - val_loss: 585847.8125 - val_accuracy: 0.0000e+00
Epoch 6/10
21/21 [==============================] - 21s 975ms/step - loss: -764696.0000 -
accuracy: 0.2805 - val_loss: 1084920.8750 - val_accuracy: 0.0000e+00
Epoch 7/10
21/21 [==============================] - 22s 1s/step - loss: -1347185.5000 -
accuracy: 0.2805 - val_loss: 1824000.8750 - val_accuracy: 0.0000e+00
Epoch 8/10
21/21 [==============================] - 20s 919ms/step - loss: -2213935.2500 -
```

```
accuracy: 0.2805 - val_loss: 2854091.7500 - val_accuracy: 0.0000e+00
Epoch 9/10
21/21 [==============================] - 22s 1s/step - loss: -3365112.5000 -
accuracy: 0.2805 - val_loss: 4249814.5000 - val_accuracy: 0.0000e+00
Epoch 10/10
21/21 [==============================] - 31s 1s/step - loss: -4916634.0000 -
accuracy: 0.2805 - val_loss: 6023660.5000 - val_accuracy: 0.0000e+00
```

[39]:
```
<keras.callbacks.History at 0x3ff4c12d3c0>
```

[40]:
```
loss, accuracy = model.evaluate(validation_generator)
print('Validation accuracy:', accuracy)
```

```
2/2 [==============================] - 1s 36ms/step - loss: 6023660.0000 -
accuracy: 0.0000e+00
Validation accuracy: 0.0
```

[4]:

[ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set up data directories
train_dir = 'train/training'
validation_dir = 'val'

# Create data generators with more aggressive augmentation
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=45,
                                   width_shift_range=0.25,
                                   height_shift_range=0.25,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,␣
 ↪target_size=(224, 224), batch_size=32, class_mode='binary')
validation_generator = test_datagen.flow_from_directory(validation_dir,␣
 ↪target_size=(224, 224), batch_size=32, class_mode='binary')

# Define a more complex CNN model with additional layers
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(224, 224, 3)))
```

```
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model with a different optimizer and learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
 ↪loss='binary_crossentropy', metrics=['accuracy'])


# Run the function on your dataset directories
remove_corrupted_images('path_to_train_directory')
remove_corrupted_images('path_to_validation_directory')
# Train the model with more epochs
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# Now, when you run your training, it should load even truncated images
model.fit(train_generator, steps_per_epoch=len(train_generator), epochs=20,␣
 ↪validation_data=validation_generator,␣
 ↪validation_steps=len(validation_generator))
# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print('Validation accuracy:', accuracy)
```

```
Found 156 images belonging to 1 classes.
Found 38 images belonging to 1 classes.
Epoch 1/20
```

[60]:
```
new_image = test_datagen.flow_from_directory(
    'test',
    target_size=(224, 224),  # Use the same dimensions as in the model summary
    batch_size=1,
    class_mode='binary',
    color_mode='rgb'  # If your model uses RGB input
)
```

```
Found 3 images belonging to 1 classes.
```

[61]:
```
for batch in new_image:
    print(batch[0].shape)  # Output should be (1, height, width, channels)
    break
```

```
(1, 224, 224, 3)
```

[64]:
```
# Run the prediction
prediction = model.predict(new_image)

# Since `prediction` is likely an array, access the first element of the array
pred_value = prediction[0][0]  # Assuming the prediction is a batch of one␣
 ↪image with binary output

# Now apply the condition on the scalar value
if pred_value > 0.5:
    print("Healthy coral reef")
else:
    print("Bleached coral reef")
```

```
3/3 [==============================] - 0s 126ms/step
Bleached coral reef
```

[50]:
```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 222, 222, 64)      1792

 max_pooling2d_6 (MaxPooling  (None, 111, 111, 64)     0
 2D)

 conv2d_7 (Conv2D)           (None, 109, 109, 128)     73856

 max_pooling2d_7 (MaxPooling  (None, 54, 54, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 52, 52, 256)       295168

 max_pooling2d_8 (MaxPooling  (None, 26, 26, 256)      0
 2D)

 flatten_6 (Flatten)         (None, 173056)            0

 dense_12 (Dense)            (None, 512)               88605184

 dense_13 (Dense)            (None, 1)                 513

=================================================================
Total params: 88,976,513
Trainable params: 88,976,513
Non-trainable params: 0
```

```
----------------------------------------------------------------
```

[ ]: