Inception V3 Model

Oluwadarasimi Emmanuel Falade

December 2024

1 Introduction

The aim of this project/model was to analyse coral images and classify them into two classes, which are ”Bleached corals” and ”Healthy corals.” This project contributes towards the preservation of life below water, which is SDG 14. Applying the use of AI/ML in this project, will help tackle the challenges the marine ecosystem is currently facing.

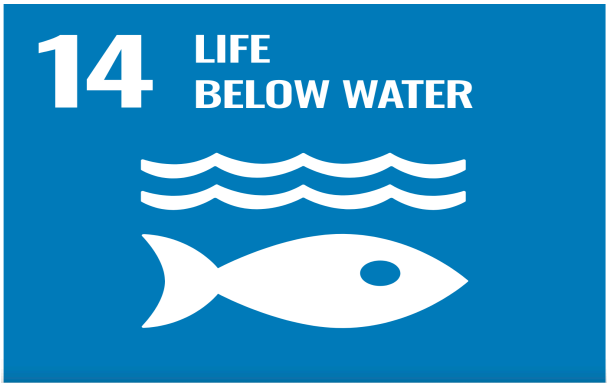

Figure 1: SDG 17

The model used in this report is Inception V3. Inception V3 is a deep convolutional network developed by Google. Inception V3 uses techniques like aggressive regularizations, factorized convolutions, and strategic balance of model depth and width to improve the performance and reduce computational demand.

2 Imports and Code Setup

In this section, we will be looking at the imports used in the code.

2.1 Imports

2.1.1 OS import

OS gives a way of using an operating system-dependent functionality like reading or writing to the filesystem.

2.1.2 Numpy

Numpy is a fundamental package for data science computing with Python. It provides support for arrays, matrices, and other functions.

2.1.3 PIL

The PIL (Python Imaging Library) is used for opening, manipulating, and saving many different image file formats.

2.1.4 Matplotlib

Matplotlib is a library used for plotting, creating static, animated, and interactive visualizations in Python.

2.1.5 Scikit-learn train_test_split

The train_test_split is a Scikit-learn module used for splitting arrays or matrices into random train and test subsets.

2.1.6 sklearn.preprocessing LabelEncoder

The Label Encoder is another module within Sckit-learn; it is used for converting categorical labels into numeric forms.

2.1.7 sklearn.metrics accuracyscore, classificationreport, confusionmatrix

These are tools to evaluate the performance of the classification model.

2.2 Seaborn

Seaborn is a data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.

2.2.1 Tensorflow

TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building and deploying machine learning models, including deep learning models.

2.2.2 Tensorflow Keras

Keras is a high-level neural network API written in Python and capable of running on top of TensorFlow.

2.2.3 tensorflow.keras.layers GlobalAveragePooling2D, Dense, Dropout

These are different types of layers used in building neural networks.

2.2.4 tensorflow.keras.callbacks EarlyStopping, ReduceLROnPlateau

Callbacks are used to control the training process (e.g., early stopping to prevent overfitting, reducing the learning rate on a plateau).

2.2.5 tensorflowkeras, mixedprecision

This module is used for mixed precision training to improve the performance of the model.

2.3 Setup

2.3.1 GPU Configuration

In order the optimize the GPU usage during the model training, the following code below was used;

```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
```

```
except RuntimeError as e:
    print(e)
```

2.3.2 Mixed Precision

For the setup of mixed precision,

```
    policy = mixed_precision.Policy('mixed_float16')
mixed_precision.set_global_policy(policy)
```

This enabled setting the model policy to mixed precision, which uses both 16-bit and 32-bit floating point types to speed up training on supported hardware.

3 Data Overview

3.1 Paths

```
bleached_path = "./data/bleached_corals"
healthy_path = "./data/healthy_corals"
```

The code above specifies the directories where images of bleached and healthy corals are stored.

3.2 Functions to path

```
def get_image_paths(folder_path):
return [os.path.join(folder_path, f)
for f in os.listdir(folder_path) if f.endswith('.jpg')]
```

The code above defines a function to retrieve the file paths of all .jpg images in a given directory.

3.3 Retrieve Image Paths

```
bleached_image_paths = get_image_paths(bleached_path)
healthy_image_paths = get_image_paths(healthy_path)
```

This code calls the get_image_paths function to get the file paths of healthy and bleached corals.

3.4 Function to Load and Resize Images

```
def load_and_resize_images(image_paths, size=(224, 224)):
    images = []
    for img_path in image_paths:
        with Image.open(img_path) as img:
            img = img.resize(size)
            images.append(np.array(img))
    return np.array(images)
```

This function is used to load and resize images to a specified size (224x224 pixels in this case).

```
bleached_images = load_and_resize_images(bleached_image_paths)
healthy_images = load_and_resize_images(healthy_image_paths)
```

This calls for the load and resize function.

3.5 Labeling the DataSet

```
bleached_labels = ['bleached'] * len(bleached_images)
healthy_labels = ['healthy'] * len(healthy_images)
```

This creates labels for each image, marking them as either 'bleached' or 'healthy'.

3.6 Combining Data and Labels

```
data = np.vstack((bleached_images, healthy_images))
labels = np.array(bleached_labels + healthy_labels)
```

The code above shows the combination of the images and their corresponding labels into a single dataset.

3.7 Label Encoding

```
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```

This uses the LabelEncoder module to convert the string labels ('bleached' and 'healthy') into numeric form.

This section shows how the dataset was prepared by loading, resizing, and labeling images of bleached and healthy corals, making it ready for further processing and model training.

4 Split in Training and Data Testing

4.1 Data Splitting

```
X_train, X_test, y_train, y_test =
train_test_split(data, labels_encoded, test_size=0.3, random_state=42)
```

In this part of the splitting data, The train_test_split from scikit-learn was used to split the data into training and testing sets.
Data shows the combined dataset of images.
Labelsencoded shows the encoded labels corresponding to the images. test_size=0.3 specifies that 30% (percent) of the data should be used for testing and 70% (percent) for training.
Randomstate=42 ensures reproducibility by setting a seed for the random number generator.

4.2 Normalization

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

This normalizes the pixel values of the images to the range [0, 1] by dividing by 255.

5 Data Augmentation

5.1 Preprocessing Function

```
def preprocess_image(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
    return image, label
```

The preprocessing function defines a function preprocess_image to apply data augmentation techniques:
tf.image.random_flip_left_right(image): Randomly flips the image horizontally.
tf.image.random_flip_up_down(image): Randomly flips the image vertically.

tf.image.random_brightness(image, max_delta=0.1): Randomly adjusts the brightness.
tf.image.random_contrast(image, lower=0.9, upper=1.1): Randomly adjusts the contrast.

5.2 Creating Datasets

```
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_dataset = train_dataset.map
(preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.batch(16).prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_dataset = test_dataset.batch(16).prefetch(tf.data.AUTOTUNE)
```

For the training dataset:
i. It converts the training data to a TensorFlow dataset.
ii. Applies the preprocess_image function to each image-label pair.
iii. Batches the dataset with a batch size of 16.
iv. Pre-fetches data for optimal performance.
For the testing dataset:
i. Converts testing data to a TensorFlow dataset.
ii. Batches the dataset with a batch size of 16.
iii. Pre-fetches data for optimal performance.

6 Transfer Learning

6.1 Input Tensor

```
input_tensor = tf.keras.layers.Input(shape=(224, 224, 3))
```

This defines the input shape for the model, which is (224, 224, 3) corresponding to the image dimensions and RGB color channels.

6.2 Inception V3 Base Model

```
base_model = InceptionV3(weights='imagenet',
include_top=False, input_tensor=input_tensor)
```

This part shows the loading of the InceptionV3 model pre-trained on the ImageNet dataset without the top fully connected layers (includetop=False). Uses the defined input_tensor as the input to this model.

6.2.1 Addition of Custom Layers

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid', dtype='float32')(x)
```

i. **base_model.output**: Accesses the output of the base model.
GlobalAveragePooling2D(): Adds a global average pooling layer to reduce the spatial dimensions.
ii. **Dense(512, activation='relu')**: Adds a dense (fully connected) layer with 512 units and ReLU activation.
iii. **Dropout(0.5)**: Adds a dropout layer with a 50% dropout rate to prevent overfitting.
iv. **Dense(1, activation='sigmoid', dtype='float32')**: Adds the final dense layer with a single unit and sigmoid activation for binary classification.

6.3 Definition of Model

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

This defintion combines the base model and the custom layers into a single model.

6.4 Unfreezing Layers for Fine tuning

```
for layer in base_model.layers[-50:]:
layer.trainable = True
```

This unfreezes the last 50 layers of the base model to allow fine-tuning during the training process.

6.5 Compilation of the Model

```
model.compile(optimizer=keras.optimizers.Adam
(learning_rate=0.00001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

i. Compiling the model with the Adam optimizer and a very low learning rate (0.00001).
ii. Implementing the use of binary cross-entropy as the loss function and tracks accuracy as a metric.

6.6 Early Stopping and Rate Callbacks

```
early_stopping = EarlyStopping
(monitor='val_loss', patience=5, restore_best_weights=True)
```

i. **Monitoring**: It watches the validation loss (val_loss).
ii. **Patience**: It stops the training if the validation loss does not improve for 5 consecutive epochs.
iii. **Restore Best Weights**: It restores the model weights from the epoch with the best validation loss.

6.7 ReductionLROn Plateau Callback

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.2, patience=3, min_lr=0.00001)
```

i. **Monitoring**: It watches the validation loss (val_loss).
ii. **Factor**: It reduces the learning rate by a factor of 0.2 if the validation loss does not improve.
iii. **Patience**: It observes and waits for 3 epochs without improvement before reducing the learning rate.
iv. **Min_Lr**: It ensures the learning rate does not go below 0.00001.
Using these callbacks help prevent overfitting and ensure efficient training by stopping the training early if no improvement is seen and reducing the learning rate when progress slows.

This section shows how the transfer learning was setup by leveraging the pre-trained InceptionV3 model, adding custom layers on top, and preparing the model for fine-tuning to adapt it for the specific task of classifying coral images.

7 Model Evaluation

```
loss, accuracy = model.evaluate(test_dataset)
print(f'Validation␣accuracy:␣{accuracy␣*␣100:.2f}%')
```

i. **Evaluate Method**: The model.evaluate method is used to evaluate the model's performance on the test dataset.
ii. **Output**: It returns the loss and accuracy of the model on the test data.
iii. **Print Statement**: The print statement displays the validation accuracy as a percentage (%).

8 Confusion Matrix

```
y_pred = (model.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)
```

i. **Predictions**: The model.predict(X_test) generates predictions for the test data, and (model.predict(X_test) > 0.5).astype("int32") converts the predictions to binary values (0 or 1).
ii. **Confusion Matrix**: The confusion_matrix(y_test, y_pred) computes the confusion matrix using the true labels (y_test) and predicted labels (y_pred).
A confusion matrix is a table/graphical representation used to evaluate the performance of a classification model. It shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions.

8.1 Displaying The Confusion Matrix

```
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion␣Matrix')
plt.show()
```

i. **cm**: This is the confusion matrix's data.

ii. **annot=True**: Annotates each cell with the numeric value. iii. **fmt='d'**: Formats the annotations as integers.
iv. **cmap='Blues'**: It uses the 'Blues' colormap for coloring the heatmap.
i. **Plotting**: By using Seaborn's heatmap to visualize the confusion matrix.
ii. **Heatmap Function**: The Seaborn heatmap function is used to visualize data in matrix form with color coding, useful for displaying confusion matrices.
iii. **Labels and Title**: This adds axis labels and a title for clarity.

9 Findings

After multiple training processes on 50 epochs, applying a learning rate of 0.00001, using the binary cross entropy loss function and the Adam optimizer, the model was consistent at a range of 76% to 77% and it had a low loss. But the model once reached an accuracy of 80% after training. This is a moderate-decent result for the model.

9.1 Confusion Matrix

i. **True Positive**: From the confusion matrix, the model correctly classified 119 instances/images as positive. ii. **False Negative**: The model incorrectly predicted/classified 35 images as false.
iii. **False Positive**: The model incorrectly predicted 30 images as positive.
iv. **True Negative**: The model correctly classified 93 images as negative.
From the results above, we can see that the model performed decently with the classifications.

9.2 Classification Report

In the classification report, the Healthy corals are represented as 0 and the Bleached corals are represented as 1.
From the classification report, for the **Healthy Corals (0)** we can see that the the precision was high at about 0.80, the recall at **0.77**, the f1-score at 0.79 and a support of **154** (which means that there are 154 true occurences of the healthy corals in the ground truth.
For the **Bleached Corals (1)**, we can see that the the precision was decent at about **0.73**, the recall at **0.76**, the f1-score at **0.74** and a support of **123** (which means that there are 123 true occurences of the healthy corals in the ground truth.
From this performance, it shows that the model was very good at classifying the healthy corals, and it was moderately good at classifying the bleached corals.

Figure 2: Training

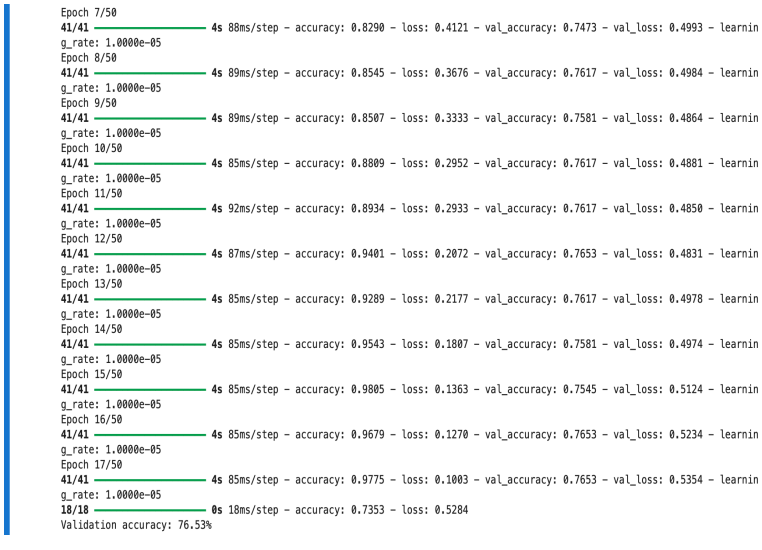

Figure 3: Training

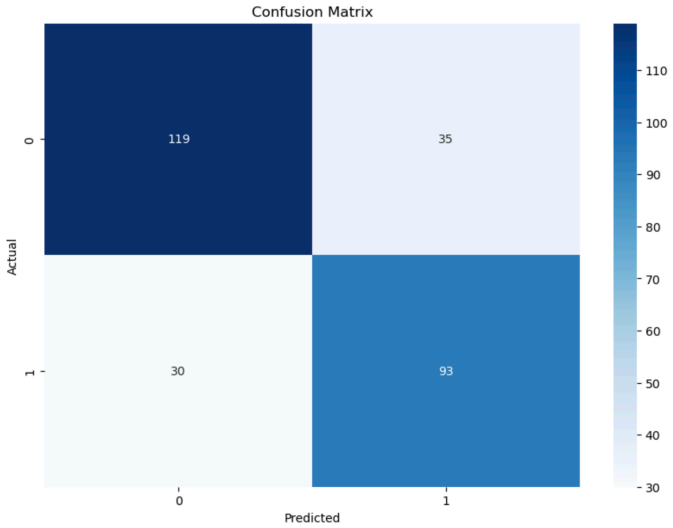

Figure 4: Confusion Matrix

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.77	0.79	154
1	0.73	0.76	0.74	123
accuracy			0.77	277
macro avg	0.76	0.76	0.76	277
weighted avg	0.77	0.77	0.77	277

Figure 5: Classification Report

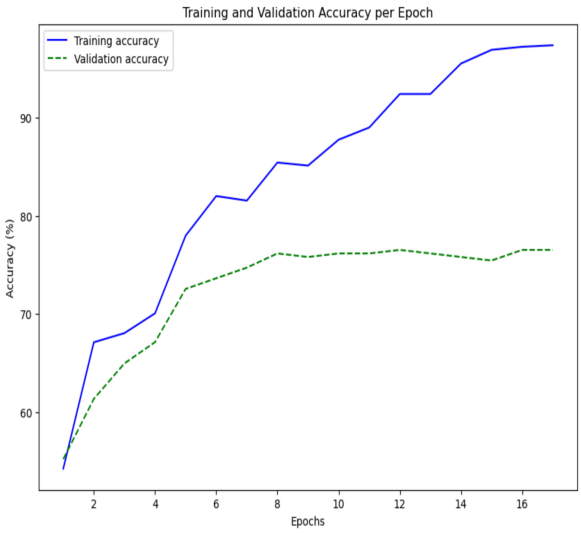

Figure 6: Training and Validation Accuracy

10 Limitations

i. **Resizing images**: Resizing the dimensions of the image proved difficult initially.
ii. **Limited Dataset**
iii. **Class Imbalance**: Due to unequal distribution of images.
iv. **Mixed Precision Issues**: This sometimes led to numerical instability.
v. **GPU Memory Issues**: This made the code run slowly without GPU optimizers.
vi. **Binary Classification Metrics**: The use of accuracy alone sometimes cannot help determine or evaluate the whole performance of the model.

11 Contributions/Impacts To SDG 14

This model contributes to the SDG 14 by;
Coral Health Monitoring Ocean Habitats Studying: Analysis of ocean habitats like corals can help access the impacts of pollution or acidification in marine ecosystems.

12 Future Improvements or Alternatives

There are different methods or techniques that can be used to improve the functionality of the code and the model.
The methods that can be used are:
i. **Enhancing the Model Architecture[1]**: By using other pre-trained models [2]or ensemble methods which combine predictions from multiple models to improve accuracy and robustness, the accuracy and efficiency of the model can be improved[3].
ii. **Learning Rate Scheduling**: By implementing more advanced learning rate schedules, such as ”Cyclical Learning Rates” the model’s accuracy can be improved[4].
iii. **Gradient Accumulation**: By using gradient accumulation, training models with larger batch sizes can be done on non-demanding hardwares[5].
iv. **Advanced Regularization Techniques**: By ensuring the proper user of dropout [6] and batch normalization, overfitting can be prevented[7]. Also, by using weight decay or l2 regularization, large weights can be penalized or removed[8].
v. **Hyperparameter Tuning**: By using tools like ”Optuna” or ”Hyperopt” automated hyperparameter tuning can be carried out [9].
vi. **Cross-Validation**: To better assess the model’s performance, the k-fold cross validation technique can be implemented[10].
vii. **Loss Function**: Using focal loss can help handle class imbalance by focusing more on the images that are hard to classify.
viii. **Evaluation Metrics**: To evaluate the model better, ”F1 score”[11] or ”AUROC” can be used to get more comprehensive asseessment of the model’s performance[12].

References

[1] T. E. W. Zhengwei Wang Qi She. “Generative adversarial networks in computer vision: A survey and taxonomy.” (), [Online]. Available: https://arxiv.org/abs/1906.01529.

[2] Q. V. L. Mingxing Tan. “Efficientnet: Rethinking model scaling for convolutional neural networks.” (), [Online]. Available: https://arxiv.org/abs/1905.11946.

[3] H. Springer Berlin. “Ensemble methods in machine learning.” (), [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45014-9_1.

[4] L. N. Smith. “Cyclical learning rates for training neural networks.” (), [Online]. Available: https://arxiv.org/abs/1506.01186.

[5] P. G. P. D. R. G. P. N. L. W. A. K. A. T. Y. J. K. He. “Accurate, large minibatch sgd: Training imagenet in 1 hour.” (), [Online]. Available: https://arxiv.org/abs/1706.02677.

[6] N. S. G. H. A. K. I. S. R. Salakhutdinov. “Dropout: A simple way to prevent neural networks from overfitting.” (), [Online]. Available: https://jmlr.org/papers/v15/srivastava14a.html.

[7] C. S. Sergey Ioffe. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” (), [Online]. Available: https://arxiv.org/abs/1502.03167.

[8] F. H. Ilya Loshchilov. “Decoupled weight decay regularization.” (), [Online]. Available: https://arxiv.org/abs/1711.05101.

[9] T. A. S. S. T. Y. T. O. M. Koyama. “Optuna: A next-generation hyperparameter optimization framework.” (), [Online]. Available: https://arxiv.org/abs/1907.10902.

[10] J. Brownlee. “A gentle introduction to k-fold cross-validation.” (), [Online]. Available: https://machinelearningmastery.com/k-fold-cross-validation/.

[11] J. D. Raphaël Cerf. “The quasispecies distribution.” (), [Online]. Available: https://arxiv.org/abs/1609.05738.

[12] R. J. T. David J. Hand. “A simple generalisation of the area under the roc curve for multiple class classification problems.” (), [Online]. Available: https://link.springer.com/article/10.1023/A:1010920819831.