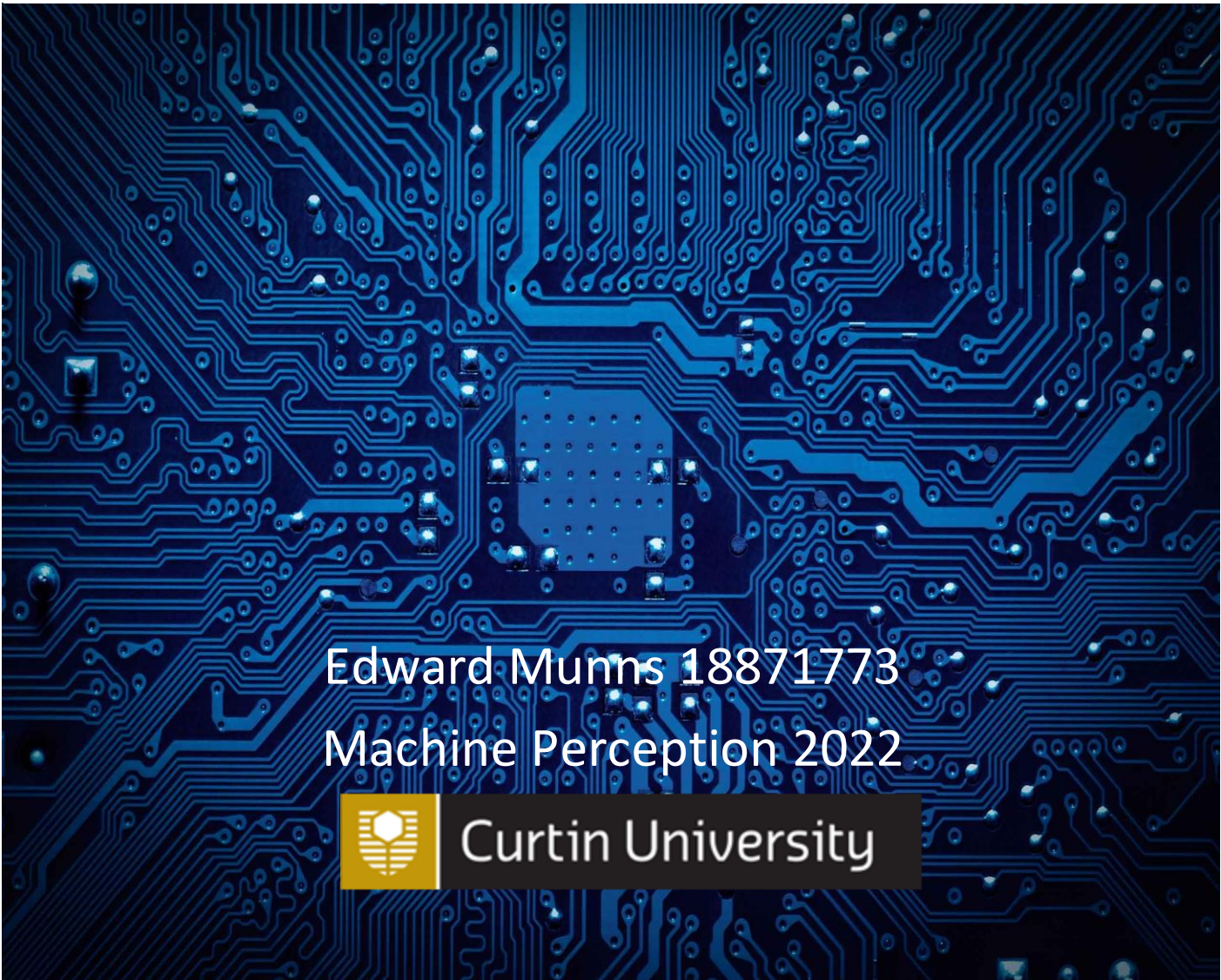




# MACHINE PERCEPTION ASSIGNMENT



Edward Munns 18871773  
Machine Perception 2022



Curtin University

## Contents

Table of Figures:.....	
1. Introduction:.....	1
2. Task 1: Building Sign Digit Extraction.....	1
2.1 Image Pre-Processing.....	1
2.2 Feature Detection and Extraction .....	2
2.4 Results.....	4
3. Task 2: Coral Image Classification .....	5
3.1 Problem Domain Discussion .....	5
3.2 Pre-Processing Approach:.....	5
3.3 Machine learning Model 1: .....	6
3.4 Machine Learning Model 2:.....	8
4. Conclusion:.....	12
Bibliography.....	12
Appendix:.....	13
A1: Task 1 Test Image Results:.....	13
A2: Source Code.....	18
A2.1: Model 1 Trainer .....	18
A2.2: Model 2 Trainer .....	19

## Table of Figures:

Figure 1: From left, B, G, R channels after binary inverse thresholding with threshold value of 160. ..	1
Figure 2: Result of summation of the R, G, B inverse threshold images.....	2
Figure 3: Regions Detected After MSER Algorithm Application. ....	3
Figure 4: Sign and Digit Extracted Areas .....	4
Figure 5: Sample Result of the Applied Algorithm for Task 1. ....	4
Figure 6: Model 1 Layout.....	6
Figure 7: TensorFlow Task 2 Model 1 Summary Output .....	7
Figure 8: Task 2 Model 1 Training Accuracy and Loss vs Epoch.....	7
Figure 9: Model 1 Results .....	8
Figure 10: Model 1 layout .....	9
Figure 11: TensorFlow Task 2 Model 2 Summary Output.....	10
Figure 12: Model 2 Training Accuracy and Loss vs Epoch .....	11
Figure 13: Model 2 Results Over the Testing Image Dataset. ....	11

## 1. Introduction:

This assignment details the process of the development of 3 machine perception agents applicable to 2 different problem domains. The first agent acting upon the first problem domain is required to isolate digits from building signs in images varying in lighting conditions and geometric properties.

The second problem domain requires the classification of images containing coral and not containing coral. For this task 2 classification agents using different machine learning approaches were trained and results will be compared in this report.

## 2. Task 1: Building Sign Digit Extraction

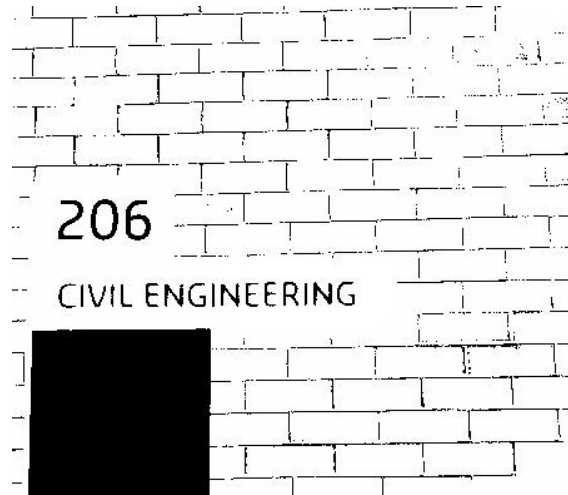
### 2.1 Image Pre-Processing

The pre-processing phase of this task began with a color space examination in the RGB, HSV and LAB color spaces being performed over all the images in the training and validation sets to identify any color channel properties that could aid in region of interest extraction. As the intensity of the white digits in all the images are high in the R, G and B color channels in the RGB color space, binary inverse thresholding was applied to these channels with a threshold value of 160. A sample of the result of this thresholding operation can be seen in figure 1.



Figure 1: From left, B, G, R channels after binary inverse thresholding with threshold value of 160.

The next step in the pre-processing phase was to sum the three binary images to create a single binary image to be used in digit extraction. This operation results in a binary image very similar in intensity levels to the color channel with the most desirable response to the last operation so essentially selects this most desirable response from the three channels as the dominant global color changes. Results of this process applied to the figure 1 image set can be seen in figure 2.



*Figure 2: Result of summation of the R, G, B inverse threshold images.*

After this process the result was deemed suitable to move on to digit and sign detection and extraction.

## 2.2 Feature Detection and Extraction

To detect the digits in the pre-processed image, first the OpenCV simple blob detector class was tested, however results from this algorithm proved unreliable, so this algorithm was swapped out for the OpenCV Maximally Stable Extremal Regions (MSER) Algorithm which provided much better results. This algorithm detects stable regions of a greyscale image by thresholding the image over varying threshold values to find areas of stability through this thresholding process. The algorithm can also apply minimum and maximum area thresholding to remove found regions outside of this area threshold range (OpenCV 2022). After iterative testing of the area thresholds a minimum area of 100 pixels and maximum area of 1000 pixels was found to detect all digits in the training and validation datasets irrespective of digit scale variance. An example of the found regions after this process was applied can be seen in figure 3.



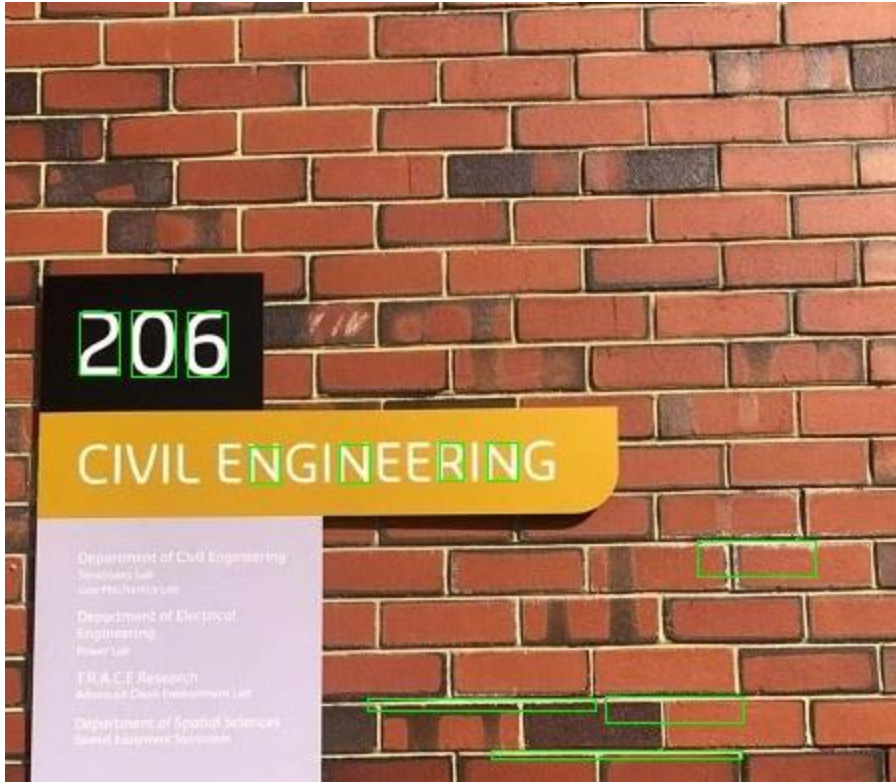


Figure 3: Regions Detected After MSER Algorithm Application.

After this region detection was applied, each detected region was used to create a blob object and these objects were added to a list if the following height to width ratio statement was true:

$$3.5 > \text{height} / \text{width} > 1.2$$

Any found regions internal to other regions were also removed. This list was then subjected to the following sorting/pruning algorithm which was created after assessing the ordering of application of the sorting/pruning methods to achieve the greatest information gain at each step:

1. The list was pruned to 5 elements based on blob area from smallest to largest area if the length of the list is greater than 3. This removed all smaller text blobs present in many of the provided images. The median of this list is now a digit blob.
2. The 0<sup>th</sup>, 1<sup>st</sup>, 4<sup>th</sup> and 5<sup>th</sup> elements in the list are pruned on median blob height with a threshold of 5 pixels if the length of the list is greater than 3.
3. The 0<sup>th</sup>, 1<sup>st</sup>, 4<sup>th</sup> and 5<sup>th</sup> elements in the list are pruned on median blob y co-ordinate with a threshold of 10 pixels if the length of the list is greater than 3.

This process proved successful for all training, validation, and testing images.

Finally, the sign area region was found by first finding the maximum width of the blobs in the blob list, sorting the blob list on ascending x co-ordinate values and calculating the top left and bottom right points of the sign region considering cases where the sign area exceeded the limits of the image. This was done with the following process:

```
Top_left = [max(blob_list[0].x - int(0.75 * max_blob_width), 0),  
max(blob_list[0].y - int(0.5 * blob_list[0].height), 0)]
```

```
Bottom_right = [max(blob_list[2].x + int(1.5 * max_blob_width), image_width),  
max(blob_list[2].y + int(1.5 * blob_list[2].height), image_height)]
```

## 2.4 Results

The above discussed algorithm was run over the provided test image dataset for task 1, a sample of the result of both the detected digit regions and the sign regions can be seen in figure 4 below. All results of the provided test image dataset can be found in section A1 of the appendix of this report.



Figure 4: Sign and Digit Extracted Areas

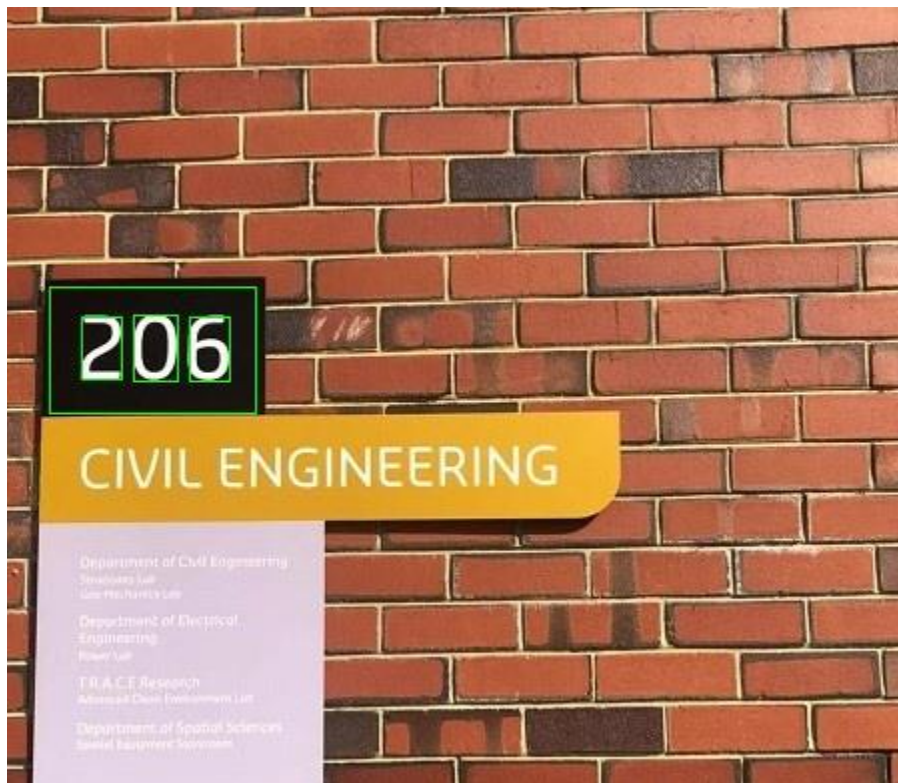


Figure 5: Sample Result of the Applied Algorithm for Task 1.

### 3. Task 2: Coral Image Classification

#### 3.1 Problem Domain Discussion

This task requires the pre-processing and classification of an image dataset of underwater images either containing coral or not containing coral. This task is therefore a binary classification problem with just positive and negative classes. Two different models will be compared on their classification performance over the dataset as well as on the computational complexity of training and running the models. The two models being tested are model 1, a neural network with 1 convolutional input layer and 4 hidden fully connected layers implementing neuron dropout. This model will be compared against model 2, a convolutional neural network (CNN) with a convolutional input layer, 3 hidden convolutional layers and a hidden fully connected layer. These networks were both created and trained in the TensorFlow API with help from the TensorFlow documentation (TensorFlow 2022) and made use of the Keras library to simplify dataset generation and model creation.

As the size of the training image set was relatively small the first step was to augment the training data. Augmentation in this approach was done through flipping and rotation transformations being performed on all images in both the positive and negative training file paths to expand the training data set before model training was commenced. This was achieved with the Python script “image\_manipulator\_script.py” included in the task 2 auxiliary files folder along with the training scripts for both models. This augmentation resulted in a training set of ~7000 images. This same image set was used to train both models in order to keep grounds of comparison as equal as possible.

#### 3.2 Pre-Processing Approach:

Pre-Processing for these models was performed through the TensorFlow Keras API using the `keras.utils Image_dataset_from_directory` function (TensorFlow Documentation 2022). This function reads in a specified directory, infers image class labels from the sub folder names they are contained in and resizes the images to the specified image size to suit the input layer of the model. The image data is shuffled, a batch size is set, and tensors are created by stacking the resized images and their corresponding labels into batches of the batch size depth.

For training of these models, the images were resized to 80 x 80 pixels, the RGB color space was used and a batch size of 32 was used for creating the training image dataset. The same parameters were applied to the validation dataset for use in model evaluation.

### 3.3 Machine learning Model 1:

#### 3.3.1 Model Defining and Compiling:

The initial intention for this model was to create a “traditional” fully connected neural network using only fully connected layers with the input being a vector, however during testing it was found that this approach could not attain the required accuracy. This may be due to the loss of any locality information in the input image due to the flattening process. To overcome this issue a single convolutional layer was added in place of the input layer. This helped considerably with increasing the accuracy of the model to an acceptable level.

The structure of this network was:

1. A convolutional input layer with 80x80x3 input shape and 4 kernels of size 3x3 and ‘reLu’ activation function.
2. Max-Pooling 2x2
3. A flattening layer to collapse the matrix into a vector.
4. A Fully connected (Dense) layer with 512 neurons and ‘reLu’ activation function.
5. A 20% neuron dropout application between these layers during training.
6. 4. A Fully connected layer with 128 neurons and ‘reLu’ activation function.
9. A Fully Connected Binary Output layer with ‘sigmoid’ activation function.

The layout of this model can be seen in figure 6.

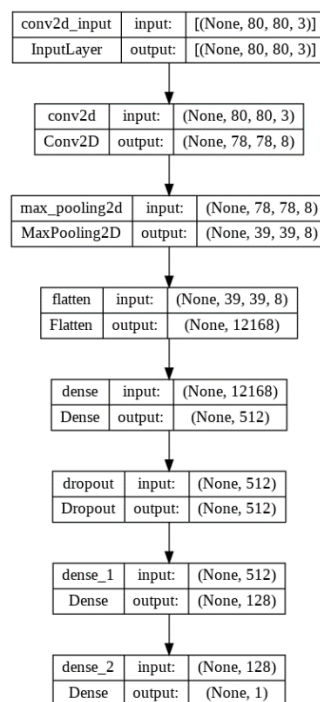


Figure 6: Model 1 Layout.



This model was compiled with binary cross-entropy loss function and Adam optimization with a learning rate of 0.001. Figure 6 shows the TensorFlow training model summary output of this model.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 4)	112
max_pooling2d (MaxPooling2D)	(None, 39, 39, 4)	0
flatten (Flatten)	(None, 6884)	0
dense (Dense)	(None, 512)	3115520
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 1)	65

```

Total params: 3,189,617
Trainable params: 3,189,617
Non-trainable params: 0

```

Figure 7: TensorFlow Task 2 Model 1 Summary Output.

### 3.3.2 Model Training:

This model was trained on the training data over 25 epochs with a batch size of 32 and accuracy was assessed with the validation data throughout this training phase. Figure 9 shows the accuracy curve over this 25 epoch training operation.

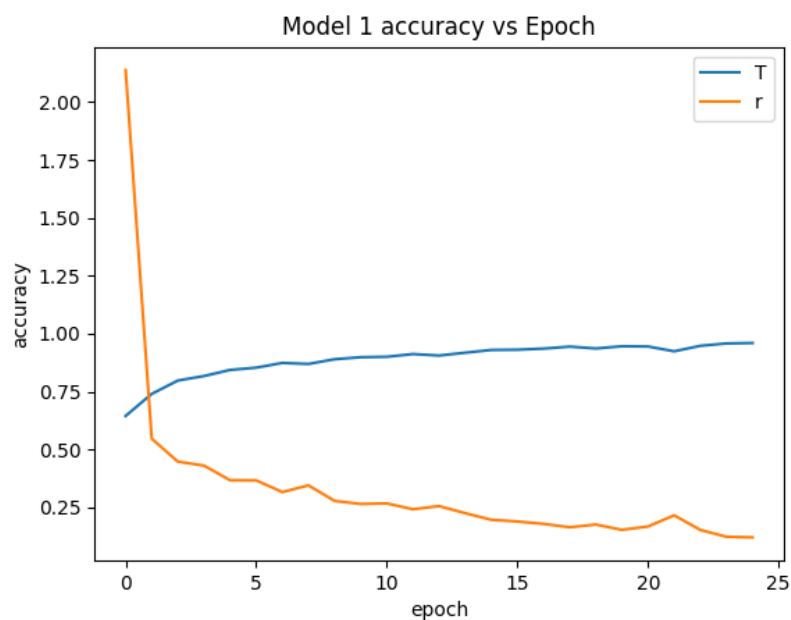


Figure 8: Task 2 Model 1 Training Accuracy and Loss vs Epoch.

### 3.3.3 Model 1 Test Results:

The model was evaluated on the provided test images with a prediction accuracy of 77.00%. Figure 9 shows these results.

```
[44] # create ImageDataGenerator objects
test_generator = ImageDataGenerator()

# load the test data into an image dataset with flow from directory
# resize to 80 x 80 to suit input layer of CNN, batch size of 32, rgb color space,
# binary class mode. As data is nont shuffled sorts alpha-numerically
test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=False,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False)

Found 400 images belonging to 2 classes.

[45] print(test_data.class_indices)

loss, acc = model.evaluate(test_data, verbose=2)

{'Non-Coral Images': 0, 'Coral Images': 1}
13/13 - 3s - loss: 0.6509 - accuracy: 0.7700 - 3s/epoch - 193ms/step
```

Figure 9: Model 1 Results

## 3.4 Machine Learning Model 2:

### 3.4.1 Model Defining and Compiling:

This model is a Convolutional Neural Network consisting of:

1. Convolutional input layer with 8 kernels of 3x3, 80x80x3 input shape to match input images and 'reLu' activation function.
2. Max-Pooling of size 2x2.
3. Convolutional hidden layer with 16 kernels of size 3x3 and 'reLu' activation function.
4. Max-Pooling of size 2x2
5. Convolutional hidden layer with 32 kernels of size 3x3 and 'reLu' activation function.
6. Max-Pooling of size 2x2

7. Convolutional hidden layer with 64 kernels of size 3x3 and 'reLu' activation function.
8. Max-Pooling of size 2x2
9. A flattening layer to collapse the matrix into a vector.
10. A fully connected (Dense) layer with 512 neurons and 'reLu' activation function.
11. A fully connected binary output layer with 'sigmoid' activation function.

The layout of this model can be seen in figure 10.

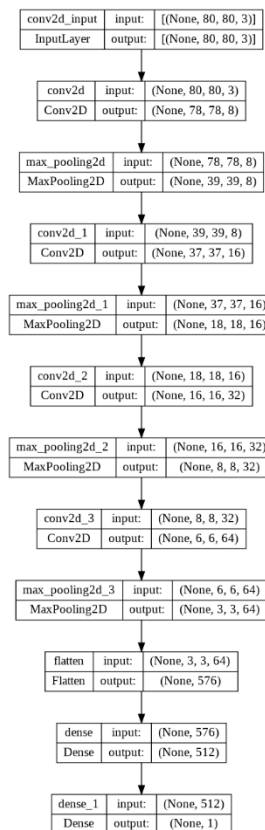


Figure 10: Model 2 layout

This model was compiled with binary cross-entropy loss function and RMSprop optimization with a learning rate of 0.001. The model summary output of the compiled model can be seen in figure 11.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 8)	224
max_pooling2d (MaxPooling2D)	(None, 39, 39, 8)	0
conv2d_1 (Conv2D)	(None, 37, 37, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 18, 18, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_3 (Conv2D)	(None, 6, 6, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 512)	295424
dense_1 (Dense)	(None, 1)	513

```

Total params: 320,465
Trainable params: 320,465
Non-trainable params: 0

```

Figure 11: TensorFlow Task 2 Model 2 Summary Output.

### 3.4.2 Model Training:

This model was trained on the training data over 30 epochs with a batch size of 32 and accuracy was assessed with the validation data throughout this training phase. Figure 12 shows the accuracy curve over this 30 epoch training operation.



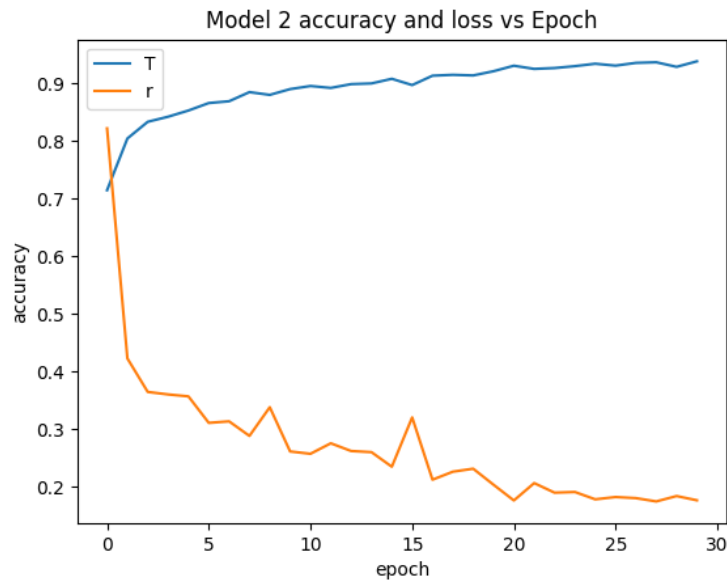


Figure 12: Model 2 Training Accuracy and Loss vs Epoch

### 3.4.3 Model 2 Test Results:

The model was evaluated on the provided test images with a prediction accuracy of 97.75%. Figure 12 shows these results.

```
[53] # create ImageDataGenerator objects
test_generator = ImageDataGenerator()

# load the test data into an image dataset with flow from directory
# resize to 80 x 80 to suit input layer of CNN, batch size of 32, rgb color space,
# binary class mode. As data is nont shuffled sorts alpha-numerically
test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=False,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False)

Found 400 images belonging to 2 classes.

print(test_data.class_indices)

loss, acc = cnn_model.evaluate(test_data, verbose=2)

{'Non-Coral Images': 0, 'Coral Images': 1}
13/13 - 2s - loss: 0.0675 - accuracy: 0.9775 - 2s/epoch - 158ms/step
```

Figure 13: Model 2 Results Over the Testing Image Dataset.

## 4. Conclusion:

In comparing the 2 models provided for task 2 of this assignment the convolutional network showed much better prediction results, particularly when considering the changes that had to be made to the input layer of model 1 to provide a prediction accuracy of greater than 60%. This is likely due to the better fit of 2 dimensional neural networks to image data through the maintaining of spatial information throughout the network.

## Bibliography

OpenCV. 2022. "cv::MSER Class Reference." *OpenCV*.

[https://docs.opencv.org/3.4/d3/d28/classcv\\_1\\_1MSER.html](https://docs.opencv.org/3.4/d3/d28/classcv_1_1MSER.html).

TensorFlow Documentation. 2022. "tf.keras.utils.image\_dataset\_from\_directory." *TensorFlow*.

[https://www.tensorflow.org/api\\_docs/python/tf/keras/utils/image\\_dataset\\_from\\_directory](https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory)

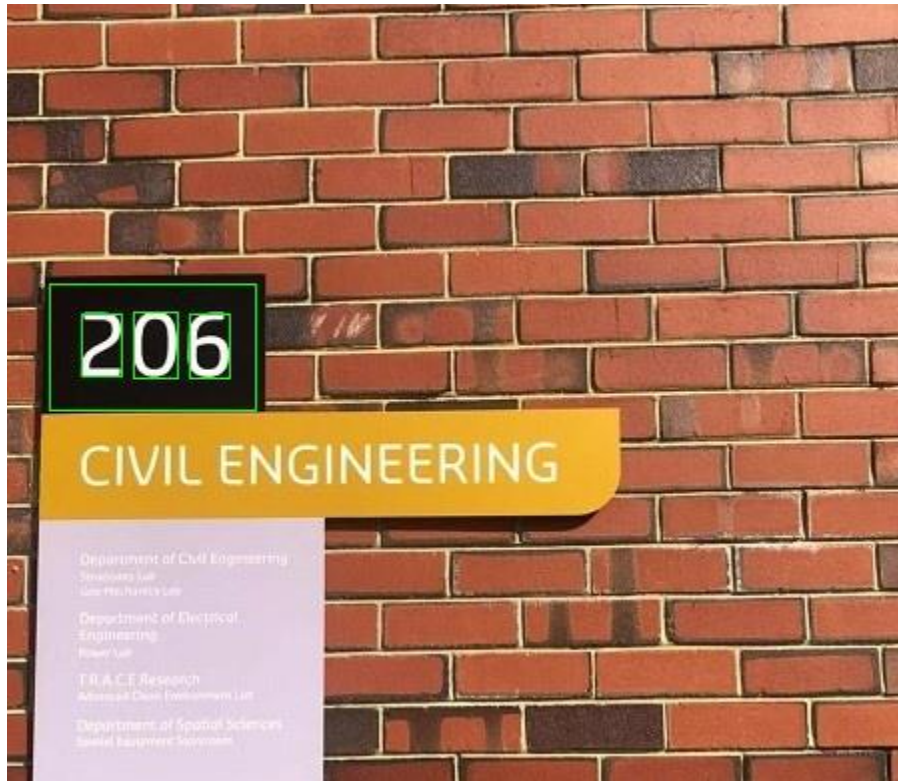
.

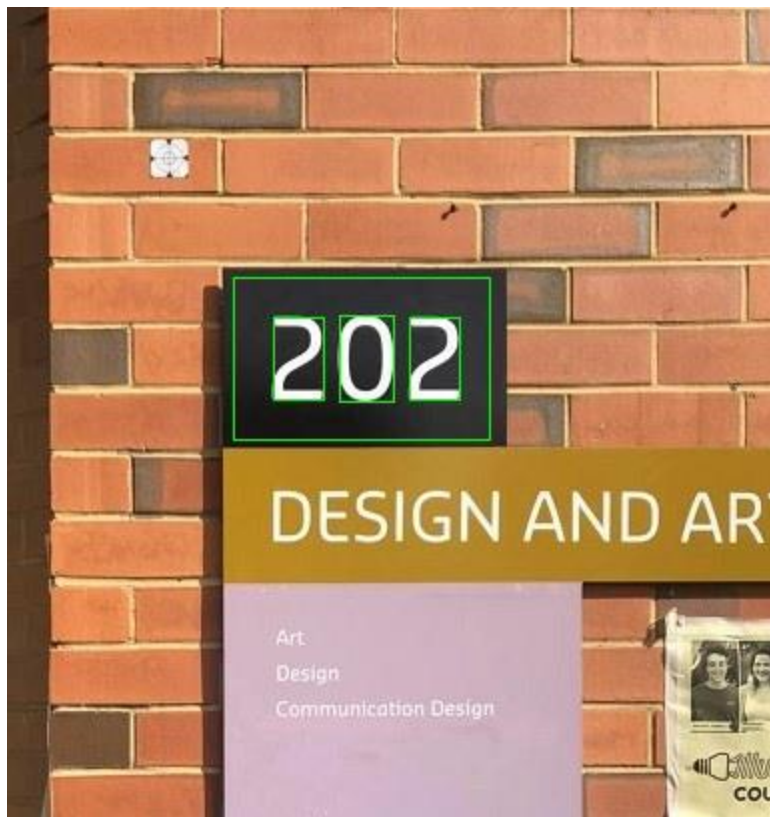
TensorFlow. 2022. "tf.keras.Model." [https://www.tensorflow.org/api\\_docs/python/tf/keras/Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model).

—. 2022. "The Sequential model." [https://www.tensorflow.org/guide/keras/sequential\\_model](https://www.tensorflow.org/guide/keras/sequential_model).

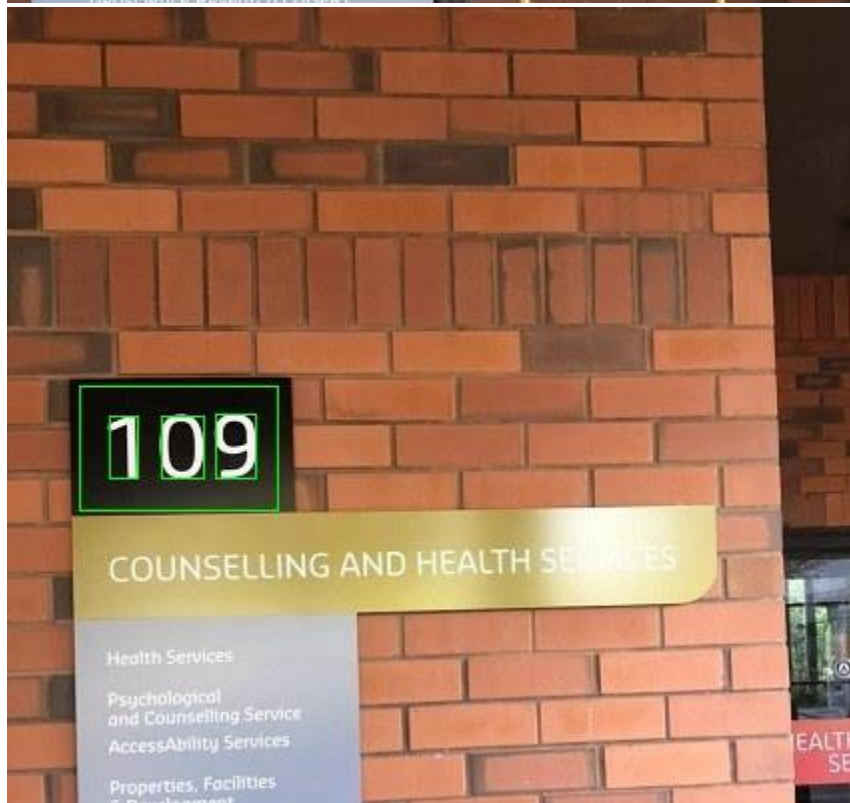
## Appendix:

### A1: Task 1 Test Image Results:















## A2: Source Code

### A2.1: Model 1 Trainer Source Code

```
import keras.optimizers
from keras.preprocessing.image import ImageDataGenerator
import keras
import tensorflow as tf
import matplotlib.pyplot as plt
tf.compat.v1.enable_eager_execution()
import os

# steps:
# 1. read the positive and negative images into python lists, send through pre-processing
# pipeline and save to disk
# 2. read the pre-processed data back in as a Keras image dataset
# 2. define the model
# 3. compile the model
# 4. train the model
# 5. save the model to file

model_save_path = "model1_fully_connected_with_20_dropout"

# create the shuffled training and validation datasets
# with labels inferred from the folder name (positive or negative)
# resize to 80 x 80, use a batch size of 32
# color mode is greyscale
# as there are only 2 classes this will be a binary dataset

# this code created with help from tensorflow docs
# at: https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory
# and dataset tutorial
# at: https://www.tensorflow.org/tutorials/load_data/images

# data paths
model_save_path = "Final_Task_2_Model_1"
train_path = 'New_Data/coral_image_classification/train'
val_path = 'New_Data/coral_image_classification/val'

# create ImageDataGenerator objects
train_generator = ImageDataGenerator()
val_generator = ImageDataGenerator()

# load the training and validation data into the image datasets with flow from directory
# resize to 80 x 80 to suit input layer of Fully connected neural network, batch size of 32, rgb color space,
# binary class mode. Shuffle the data.
train_data = train_generator.flow_from_directory(
    train_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False)

# do the same with the validation dataset
val_data = val_generator.flow_from_directory(
    val_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False)

print(train_data.class_indices)

# this model created with help from TensorFlow tf.keras.Model
# https://www.tensorflow.org/api_docs/python/tf/keras/Model
# and pythonguides:
# https://pythonguides.com/tensorflow-fully-connected-layer/
model = tf.keras.models.Sequential([
    # layer 1 is a 2D convolutional layer with 16 kernels at size 3 x 3
    # relu activation function and input share of image stack size
    # followed by max pooling
    tf.keras.layers.Conv2D(8, (3, 3), activation="relu", input_shape=(80, 80, 3)),
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Flatten(),
```



```

        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1, activation="sigmoid"))

model.compile(loss='binary_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.001),
              steps_per_execution=32,
              metrics='accuracy')

# print the model summary
model.summary()

checkpoint_path = "model_checkpoints"
checkpoint_dir = os.path.dirname(checkpoint_path)

# this callback will save the model weights at each epoch as we train
callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                              save_best_only=True,
                                              verbose=1)

# train the model on the training dataset
# with a validation step at every 2 epochs
# run for 30 epochs
model_train = model.fit(train_data,
                        batch_size=32,
                        validation_data=val_data,
                        validation_freq=1,
                        verbose=2,
                        use_multiprocessing=False,
                        epochs=25)

model.save(model_save_path)

# evaluate the model over all the validation data
# to find final accuracy over the entire validation dataset
model_val = model.evaluate(
    x=val_data,
    y=None,
    verbose='auto',
    sample_weight=None,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    return_dict=False
)

plt.plot(model_train.history['accuracy'])
plt.plot(model_train.history['loss'])
plt.title('Model 1 accuracy vs Epoch')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend('Training')
plt.show()

```

## A2.2: Model 2 Trainer Source Code

```

from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers.optimizer_v2 import rmsprop
import keras
import os
import tensorflow as tf
import matplotlib.pyplot as plt
tf.compat.v1.enable_eager_execution()

# data paths
model_save_path = "Final Task 2 Model 2"
train_path = 'New Data/coral image classification/train'
val_path = 'New Data/coral image classification/val'

# create ImageDataGenerator objects, normalize the images to [0:1]
train_generator = ImageDataGenerator()
val_generator = ImageDataGenerator()

# load the training and validation data into the image datasets with flow from directory
# resize to 80 x 80 to suit input layer of CNN, batch size of 32, rgb color space,
# binary class mode. Shuffle the data.
train_data = train_generator.flow_from_directory(
    train_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,

```

```

interpolation='nearest',
keep_aspect_ratio=False)

# do the same with the validation dataset
val_data = val_generator.flow_from_directory(
    val_path,
    target_size=(80, 80),
    color_mode='rgb',
    classes=['Non-Coral Images', 'Coral Images'],
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False)

print(str(train_data.class_indices))

# define convolutional neural network model structure
# this code created with help from tensorflow docs
# at: https://www.tensorflow.org/api_docs/python/tf/keras/Model
cnn_model = tf.keras.models.Sequential([
    # layer 1 is a 2D convolutional layer with 16 kernels at size 3 x 3
    # reLu activation function and input share of image stack size
    # followed by max pooling
    tf.keras.layers.Conv2D(8, (3, 3), activation="relu", input_shape=(80, 80, 3)),
    tf.keras.layers.MaxPool2D(2, 2),

    # 2nd convolutional layer with 32 kernels at size 3 x 3
    # reLu activation followed by max pooling
    tf.keras.layers.Conv2D(16, (3, 3), activation="relu"),
    tf.keras.layers.MaxPool2D(2, 2),

    # 2nd convolutional layer with 32 kernels at size 3 x 3
    # reLu activation followed by max pooling
    tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
    tf.keras.layers.MaxPool2D(2, 2),

    # 2nd convolutional layer with 32 kernels at size 3 x 3
    # reLu activation followed by max pooling
    tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
    tf.keras.layers.MaxPool2D(2, 2),

    # flatten the matrix to a vector
    tf.keras.layers.Flatten(),

    # Fully connected layer with 512 neurons
    tf.keras.layers.Dense(512, activation="relu"),

    # output layer
    tf.keras.layers.Dense(1, activation="sigmoid")
])

# compile the model using binary cross entropy loss, Adam optimization with learning rate of 0.001
# and a learning rate of 0.001
cnn_model.compile(loss='binary_crossentropy',
                  optimizer=keras.optimizers.RMSprop(learning_rate=0.001),
                  steps_per_execution=32,
                  metrics='accuracy')

# print the model summary
cnn_model.summary()

# save the model checkpoints
# this code created with help from tensorflow docs
# at: https://www.tensorflow.org/tutorials/keras/save_and_load
checkpoint_path = "model_checkpoints"
checkpoint_dir = os.path.dirname(checkpoint_path)

# this callback will save the model weights at each epoch as we train
callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                              save_best_only=True,
                                              verbose=1)

# train the model on the training dataset
# with a validation step at every epoch
# run for 30 epochs
model_train = cnn_model.fit(train_data,
                             shuffle=True,
                             validation_data=val_data,
                             validation_freq=1,
                             batch_size=32,
                             verbose=2,
                             callbacks=callback,
                             use_multiprocessing=False,
                             epochs=30)

cnn_model.save(model_save_path)

# evaluate the model over all the validation data
# to find final accuracy over the entire validation dataset
model_val = cnn_model.evaluate(
    x=val_data,
    y=None,
    verbose='auto',
    sample_weight=None,
    steps=None,

```

```
        callbacks=None,
        max_queue_size=10,
        workers=1,
        use_multiprocessing=False,
        return_dict=False
    )

    plt.plot(model_train.history['accuracy'])
    plt.plot(model_train.history['loss'])
    plt.title('Model 2 accuracy and loss vs Epoch')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend('Training')
    plt.show()
```