CS6P05 - Project

[Applied Machine Learning to Detection of Steganography.]
Project Report - Artefact Submission (Software Development Type Project)

Name: Jamie Hoang
ID Number: 19009101
Date: 10/03/2022
First Supervisor: Mohamed Chahine Ghanem
Second Supervisor: Alexandros Chrysikos

Summary/Abstract

Within this document is a complete compilation of the efforts of this student's final project during the Late Winter and Early Spring seasons of 2021. The document is structured into five sections.

It should be stated that this project is a Software Development Type Project. In which the overall goal is to create an end product that attempts to resolve an outstanding issue and innovate within the field of forensics.

As the title of this document states "Applied Machine Learning to Detection of Steganography". The end goal of this project is to create a proof of concept to showcase that the application of Machine Learning can be applied to labour-intensive processes such as the Detection of Steganography. Machine Learning has been known to allow for the automation of tasks usually done by humans. As it is a sub-branch of Artificial Intelligence.

This document aims to also showcase the student's skills in various fields. Primarily skills such as Programming, Design, Project Management and other academic skills are needed to be conveyed. In addition, this document also aims to convey the competence and quality of the potential product presented by the student.

Table of Contents

Software / Hardware Requirements

1.1 - Hardware / Software Specifications

The primary program that this project is centred around will be hosted on Google Colaboratory. This is to alleviate any hardware or software issues that can arise from configuration, hardware limitations or even any possible physical limitations. This ensures that there can be consistency across any platforms. In addition, machine learning can be a taxing process for hardware CPU/GPUs. Since the program is hosted on Google's data servers. It would provide the necessary resources whilst preserving the student's personal resources.

In addition, a vast amount of data will need to be downloaded to the host's computer. It is estimated that the sample data could reach into Gigabytes. Therefore, another reason for the program to be hosted on Google's platform would enable greater accessibility to the assessors. Lastly, whilst an offline and compiled version of the program could be completed. It would be considered to be redundant to the needs of the student.

In short, the only requirement is a standard computer with a monitor, mouse and keyboard. That has access to the internet via any means and that computer is able to support a modern web browser.

Currently, the student uses Google Chrome during the development of this project. Therefore, this Artefact refers to Google's Support Page in which it details its minimum requirements for it's Google Chrome Browser.

The requirements stated by Google are as follows. (Google, 2022)[1]

- Windows 7, Windows 8, Windows 8.1, Windows 10 or later
- An Intel Pentium 4 processor or later that's SSE3 capable

In addition, the standard computer in question. Should also have Python installed. With the capability to support one particular library which is the "os library".

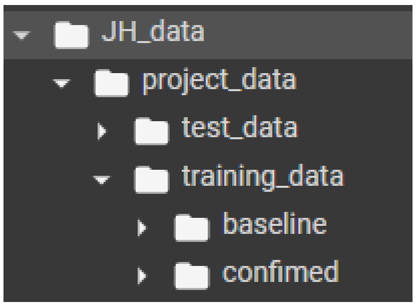
Technical Breakdown

2.1 - Artefact listing

As of the written submission of this document, there are two programs that surround this project. Which aims to be the final submission of this project as declared by the student.

- SampleCreator.py - This program aims to create sample data that is standardised and proper format that would be accepted by the Machine Learning Algorithm.
- Google Collaborate: Machine Learning Algorithm - Hosted on Google's platform "Collaborate". Which facilitates the training, development of the model and produces the predictions for possible files containing steganography.
- JH_data.gz - Stored under a .gz file format and further as a .tar file format. This houses the sample data needed for the program.

The file structure of JH_data can be seen in Figure 2AA



```
JH_data
  project_data
    test_data
    training_data
      baseline
      confimed
```

Figure 2AA - Screenshot of File Structure of JH_data

Further explanation of both the Sample Creator.py and Machine Learning Algorithm can be viewed under the Appendix section. As under the heading of "Code Breakdown".

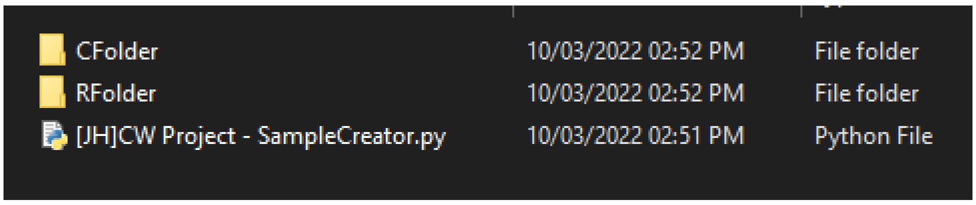
User Manual

3.1 - Sample data creation - SampleCreator.py

The program is made available for download under this URL:
https://drive.google.com/file/d/1KnIaf3F08yyxUs21s8H79X33UH5Rnd7M/view?usp=sharing

As stated prior, this program exists to "create sample data that is in a correct and proper format that would be accepted by the Machine Learning Algorithm"

The program should be run within a separate and empty folder. As files will be modified within it. The file structure should be present next to where the py file is located. As shown in Figure 3A.



CFolder	10/03/2022 02:52 PM	File folder
RFolder	10/03/2022 02:52 PM	File folder
[JH]CW Project - SampleCreator.py	10/03/2022 02:51 PM	Python File

Figure 3A - Screenshot of File Structure next to SampleCreator.py

Whilst it is advised that these folders are created before the application is run. The program can create the folders for the user if needed to.

Within these two folders. They are designated as "CFolder" which is shorthand for "Conversation Folder" and "RFolder" which is shorthand for "Results Folder". The operation of the program is simple. Simply store all of the files that are required to be converted into samples within CFolder and the resulting output will be written to RFolder.

[WARNING NOTE, THIS MUST BE READ.] : Any files submitted to this program are expected to increase via a factor of 3 times its original size. Therefore, users of the program must proceed with caution and bear in mind the file size of CFolder. As minor testing has shown there are possibilities for system instability at file sizes beyond 500 GB (almost similar to a zip bomb). In addition, damage to the hard drive is possible as they weren't intended for strenuous use.

The user should also keep in mind what they are converting. As it is up to the user to make sure they keep track of, if they are converting files that have confirmed steganographic measures or baseline files that can be used as part of the training data.

To operate the program simply run the program in IDLE or in the Command Line Interface with the required files in their CFolder. There aren't any additional inputs needed.

3.1.A - Creation of Custom sample data.

Should the assessor desire to create their own sample data to be used by the program. The following step by step guide must be followed.

Step 1:

The following file structure must be created. Shown in Figure 3B.

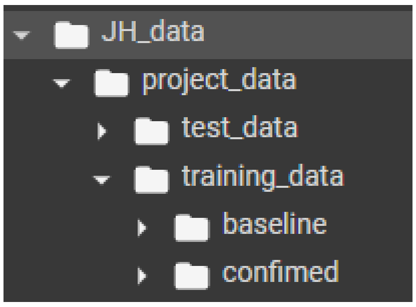


Figure 3B - JH_data file structure

Step 2:

The program "SampleCreator.py" should be used. Please refer to "**3.1 -** Sample data creation - SampleCreator.py" for further information.

The relevant txt file produced by the program must be placed into the relevant folder. Such as, if a specific sample is desired to be used as test data then the relevant txt file must be placed in that folder. I,e "test_data"

In addition, the training data folder is further separated into "baseline" and "confirmed"

Note: No samples should be placed within the "training_data". They should be placed either in "baseline" or "confirmed"

baseline - These are samples that are confirmed not to have steganographic measures. For example, these can be simply images to provide a baseline for the model to differentiate between files that do have steganographic measures.

confirmed - These are samples that are confirmed to have steganographic measures.

Step 3:

The user can upload the following file structure to the Google Colaboratory platform. This can be done by pressing the upload button on the left hand side. This can be seen in Figure 3C.

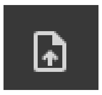


Figure 3C - Screenshot of the File Upload Button

However, should the user wish to upload the file structure to file hosting service/website. They must compress the file structure as an .tar file. Then compress that .tar file as an .gz file.

An example of this can be downloaded from the link listed below.

https://www.mediafire.com/file/wrcp31t4qcvauq3/JH_data.gz/file

3.2 - Final Program, User Manual for Assessors.

The use of Google's platform "Colaboratory" is rather simple and initiative. Within this section of the documentation will be a step-by-step process in operating the program.

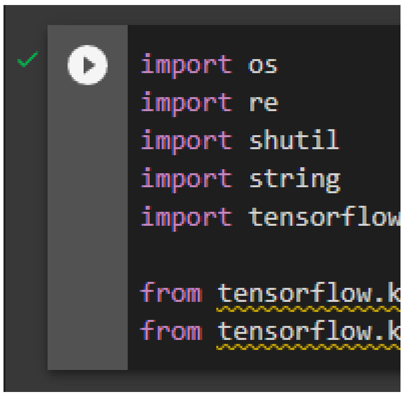
Step 1:

The program can be accessed via the link listed below. The only requirement is that the assessor has a functioning computer with an internet connection and a modern web browser.

https://colab.research.google.com/drive/1vZ_fp8tJz1rxSXv4DuQC4r0ZEslG3dTX

Step 2:

The program is divided into code blocks. As shown in Figure 3D. Each code block can be executed via pressing the associated play button on the left hand side. A green checkmark will then appear on the left hand side. Confirming that the code block was executed successfully without any errors.

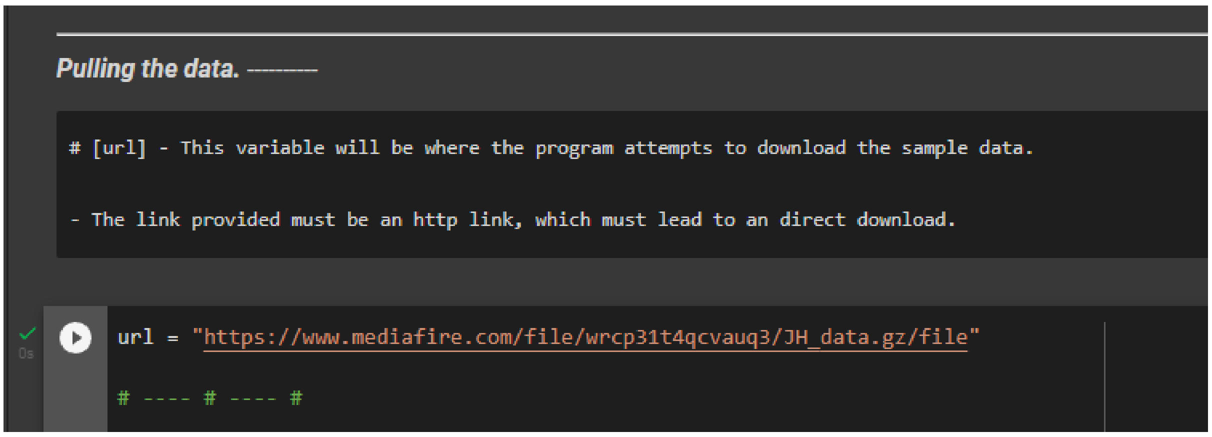


```
import os
import re
import shutil
import string
import tensorflow

from tensorflow.k
from tensorflow.k
```

Figure 3D - Screenshot of a Code Block

Step 3:

Starting from the top, working their way towards the bottom. The assessor should press the corresponding play button on each code block. Each code block will have a text block above each code block which will describe the functions of the code block. In addition, the code block is further divided into configurable elements. Which the assessor can edit to configure parts of the program. This can be seen in Figure 3E below.



Pulling the data. --------

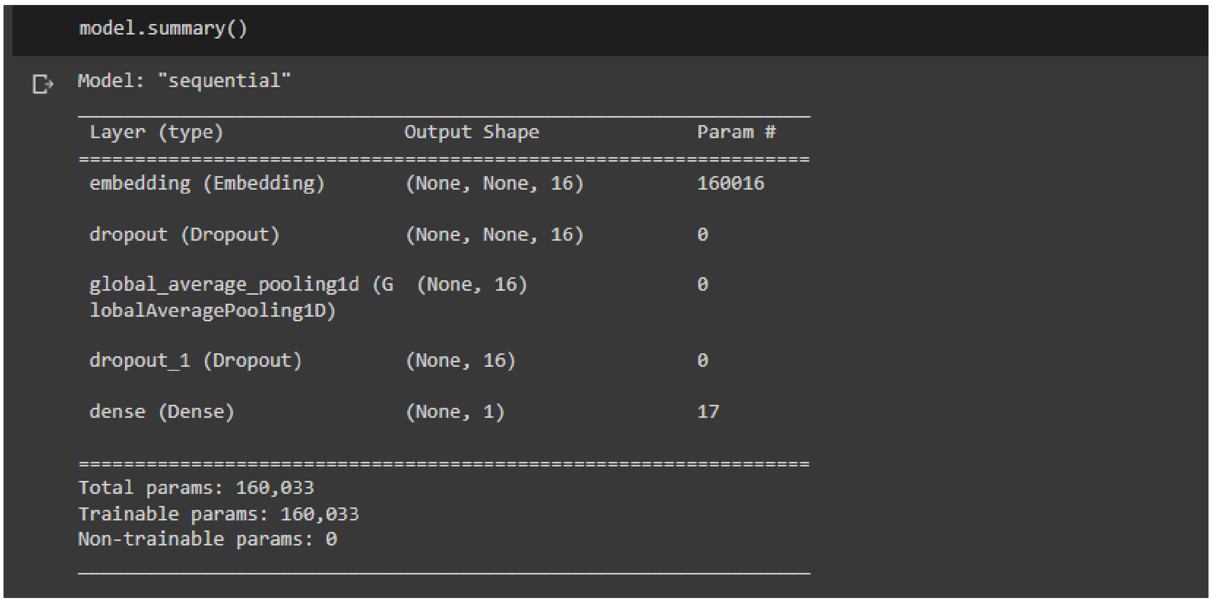
```
# [url] - This variable will be where the program attempts to download the sample data.

- The link provided must be an http link, which must lead to an direct download.
```

```
url = "https://www.mediafire.com/file/wrcp31t4qcvauq3/JH_data.gz/file"

# ---- # ---- #
```

Figure 3E - Screenshot of the Configurable Elements

Step 4:

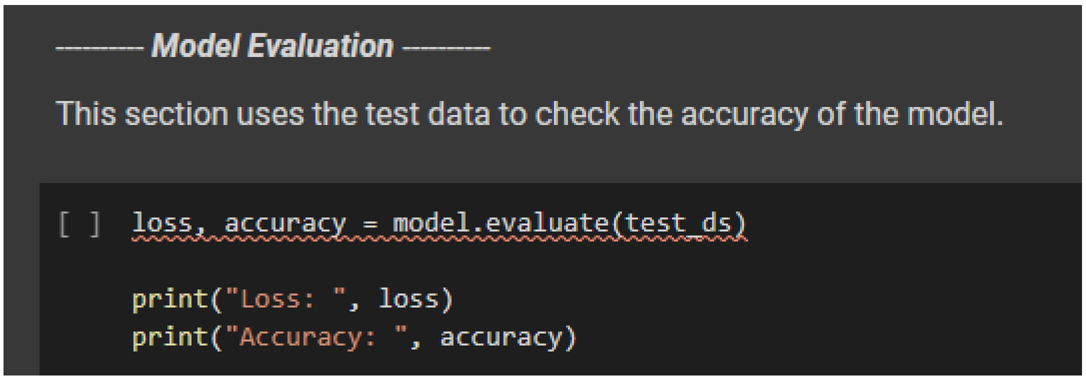
Lastly, the assessor can view the contents of the output from each code block. As it will be displayed under it. This can also be viewed when testing the model and the contents of the code block itself. This can be seen in Figure 3F Below.



```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 16)          160016

 dropout (Dropout)           (None, None, 16)          0

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```

Figure 3F - Screenshot of a code block output

If the assessor wishes to evaluate the model/use the model for predictions. They can refer to the Model Evaluation and Model Prediction section respectively. As shown in Figure 3EA and 3EB.

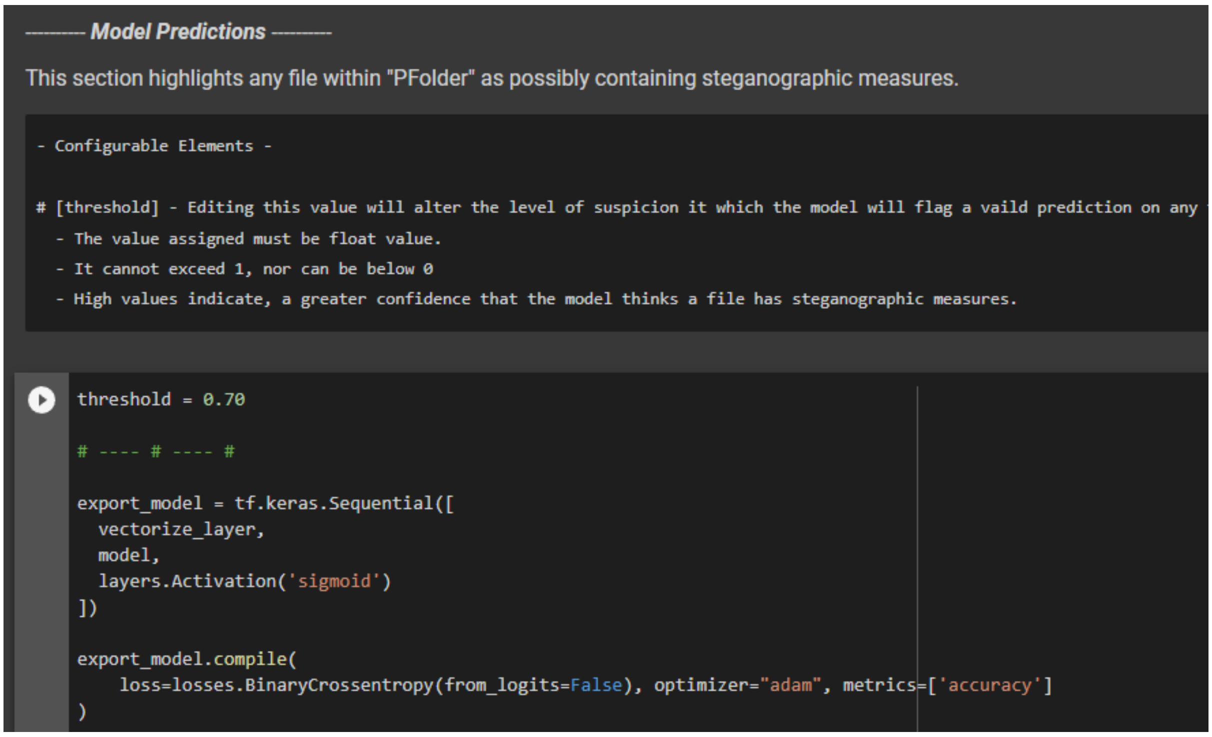


---------- *Model Evaluation* ----------

This section uses the test data to check the accuracy of the model.

```
[ ]  loss, accuracy = model.evaluate(test_ds)

     print("Loss: ", loss)
     print("Accuracy: ", accuracy)
```

Figure 3EA - Screenshot of Model Evaluation section



---------- *Model Predictions* ----------

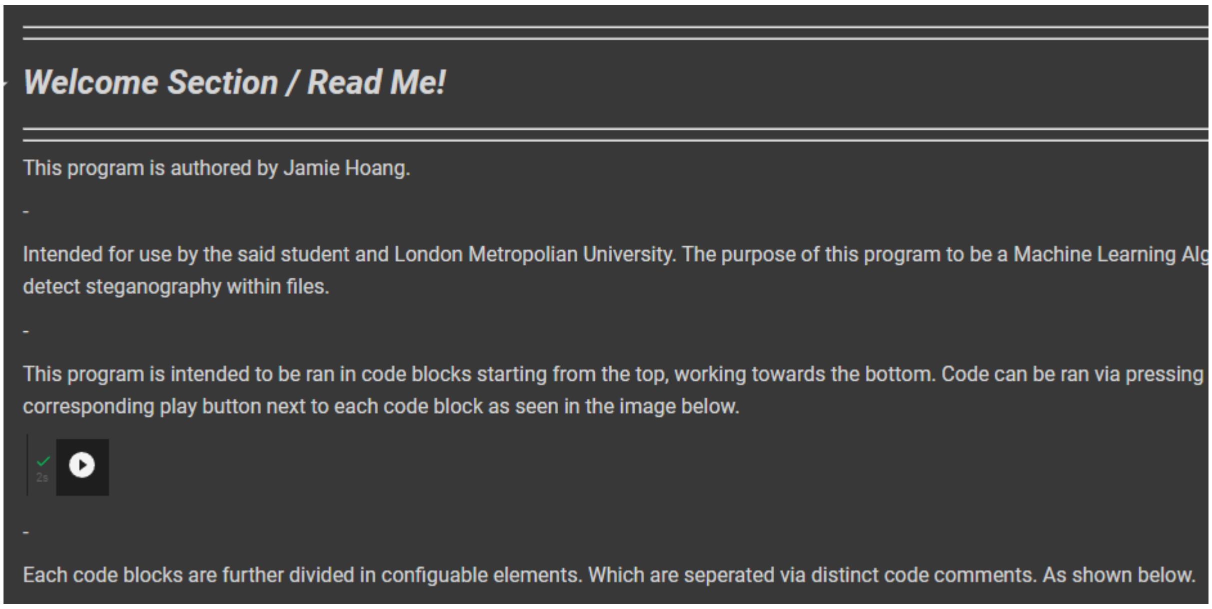
This section highlights any file within "PFolder" as possibly containing steganographic measures.

```
- Configurable Elements -

# [threshold] - Editing this value will alter the level of suspicion it which the model will flag a vaild prediction on any
  - The value assigned must be float value.
  - It cannot exceed 1, nor can be below 0
  - High values indicate, a greater confidence that the model thinks a file has steganographic measures.
```

```
threshold = 0.70

# ---- # ---- #

export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam", metrics=['accuracy']
)
```

Figure 3EB - Screenshot of Model Prediction section

If the assessor needs any further assistance/guidance. There will be text blocks above each code block which explains the functions of the code. As shown in Figure 3F below. Which is an example of the welcome section which describes what was written in the user guide.



Welcome Section / Read Me!

This program is authored by Jamie Hoang.

-

Intended for use by the said student and London Metropolian University. The purpose of this program to be a Machine Learning Alg
detect steganography within files.

-

This program is intended to be ran in code blocks starting from the top, working towards the bottom. Code can be ran via pressing
corresponding play button next to each code block as seen in the image below.

-

Each code blocks are further divided in configuable elements. Which are seperated via distinct code comments. As shown below.

Figure 3F - Screenshot of the Welcome Section

Progress Report / Conclusion

4.1 - Short Summary

In short, the remaining tasks that are yet to be completed are as listed from the Interim Report Submission.

- Collect and process samples
- Document Samples
- Train software
- Software Evaluation
- Final Report

As of the written submission of this report. All of the core features have been implemented. What is listed below are desired features and objectives.

- Offline use of the Machine Learning Algorithm, compiling the source code to create an executable file.
- Modern Graphical User Interface or a basic Command Line Interface.
- Further training and development to increase the accuracy rate.

What are defined as core features are as follows.

- Hardware and Libraries requirements for the project.
- File Hosting Provider paid for by the student to host the sample data.
- Core Functionalities within the machine learning algorithm. Such as
    - ❖ Sample data retrieval from an external or specified file path.
    - ❖ Pre-processing of the sample data in formatted data recognized by the algorithm
    - ❖ Organisation of the sample data in sets so that the program can differentiate between sets of differing purposes.
    - ❖ The creation and structure of the model/machine learning algorithm.
    - ❖ The process of feeding sample data into the model and training the algorithm with said data.
    - ❖ Processes intended to gauge, evaluate and determine the progress of the algorithm
    - ❖ The ability to make predictions about a file, if it has steganography measures.
    - ❖ Save the state of the model and load it into the program to continue training at another instance
- A program that automates the process of creating samples for the program.

4.2 - Extended Explanation

As of the writing of this report. The programs listed within this Artefact Submission will be considered in their final form and aren't expected to drastically change functionality wise. It should be stated that there were delays in production of sample data due to a catastrophic error with the storage of said samples. However, as stated within the Interim report. "It should be stated that the months of April and May 2022 are designated as buffer months in the event of the project deviating from the schedule."

Due to personal circumstances outside of the student's control. The entire of January and February couldn't be dedicated to the creation of sample data. However, due to the ingenuity of the student and a five hour session. An automation tool was able to be created. Namely the SampleCreator.py as stated above. It is of the student's prediction that a manual approach to creating sample data would have taken the two months as predicted. With the automated approach it is estimated to take a week at most then training and evaluation of the model can proceed.

A revised gantt chart is shown below in Figure 4A. It should be stressed that the personal circumstances surrounding the student have been resolved and the project should be back on track. Whilst previously stated within the document "the automated approach it is estimated to take a week at most". A month is budgeted as a fallback. In addition, the training and evaluation process are also budgeted as a week at most. This is to account for possible errors, sudden circumstances and events that could interfere with the process. It is of the student's opinion that both the training and evaluation process shouldn't take more than a 12 hour session.

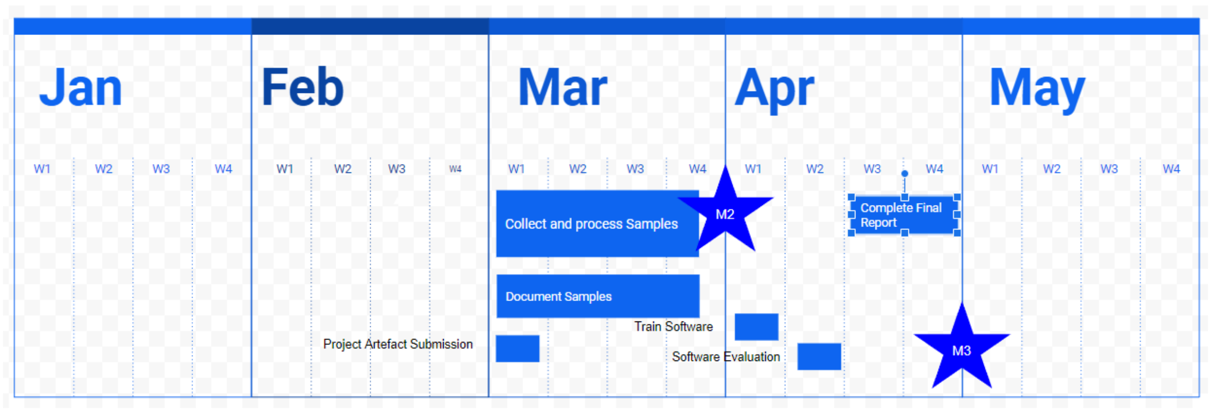


Figure 4A - Revised Gantt Chart.

References/Bibliography

[1] https://support.google.com/chrome/a/answer/7100626?hl=en

Appendix

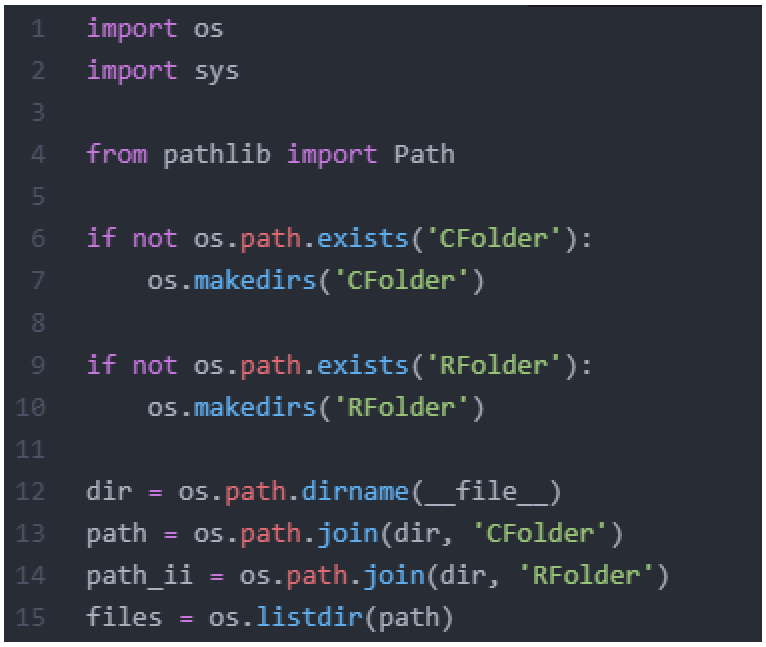
Code Breakdown

SampleCreator.py

As shown in Figure 2A below. We see various syntaxes, statements and variables. Currently the program import the following libraries which are “os”, “sys” and “pathlib”. This grants the functionality for the program to read, write and modify files. In addition, variables related to file management are initialised here.

“Dir” - Which simply gathers the current directory that the py file is contained within. Using the “dirname” syntax.
“Path” - Appends a specific folder to the current directory. Using the “join” syntax
“Files” - Gathers a list of all files contained within a directory listed in the file. Using the “listdir” syntax.

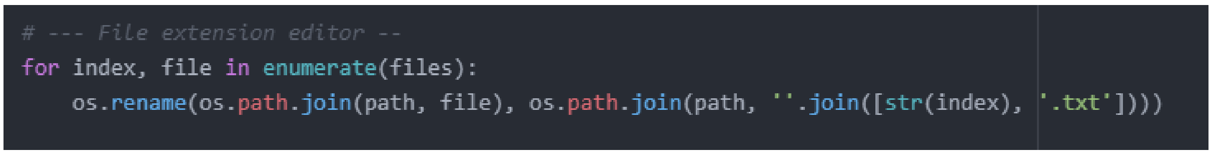
In addition, an IF statement is run where it checks if a folder “CFolder " and “RFolder” is present. If it is not, then it is created using “makedirs” syntax. This snippet of code ensures that the following functionality named relative files is present in the program. This allows for files in proximity to the py file to be stored to be used rather than absolute file paths. I.e “D:\file\folder”. This program in particular targets the CFolder which should be created next to the py file. I.e “D:\[Where the .pyfile is stored]\CFolder”



```python
import os
import sys

from pathlib import Path

if not os.path.exists('CFolder'):
    os.makedirs('CFolder')

if not os.path.exists('RFolder'):
    os.makedirs('RFolder')

dir = os.path.dirname(__file__)
path = os.path.join(dir, 'CFolder')
path_ii = os.path.join(dir, 'RFolder')
files = os.listdir(path)
```

Figure 2A - Screenshot #1 of Code Snippet from SampleCreator.py

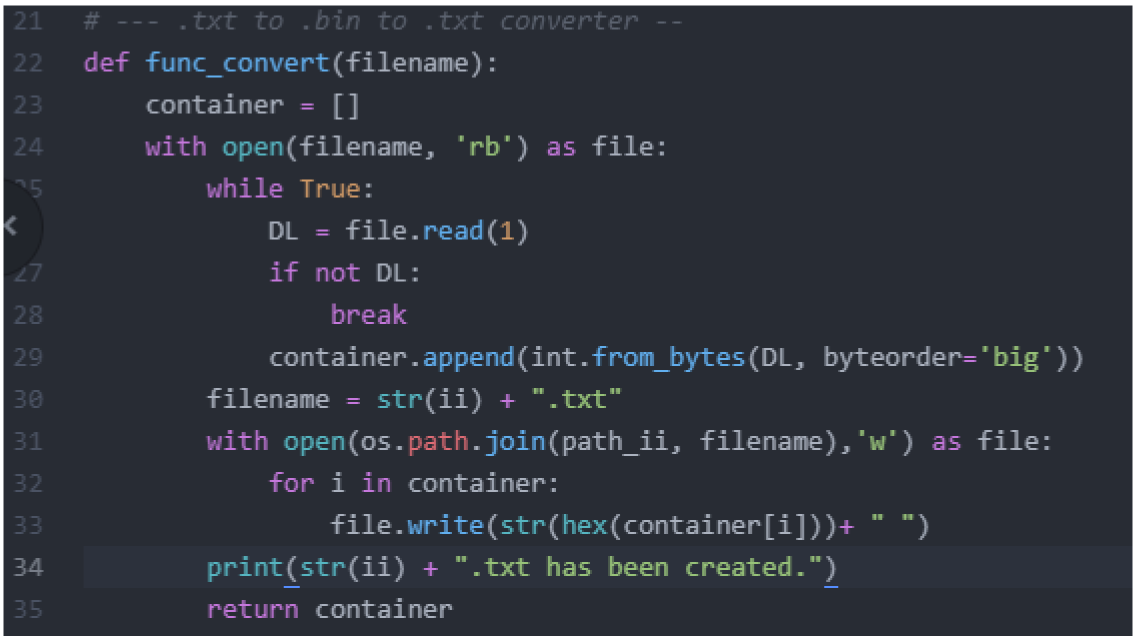
As shown in Figure 2B below. This snippet of code simply runs a FOR loop. In which it iterates through all the files and changes their extension to a .txt. It simply lists all of the required directories and then renames all files listed within that list using the following syntaxes of “join” and “rename”. When files are converted to the extension “txt”. It provides the raw binary data of the file. However, it varies how the file is rendered depending upon which application opens the said file. For example, if the file was opened via Windows standard “Notepad” it would render each character as if it was represented via UNICODE or what keyboard standard is present on said computer. Using specialist software, it would present raw binary data in human readable format.



```python
# --- File extension editor --
for index, file in enumerate(files):
    os.rename(os.path.join(path, file), os.path.join(path, ''.join([str(index), '.txt'])))
```

Figure 2B - Screenshot #2 of Code Snippet from SampleCreator.py

As shown in Figure 2C below. This snippet of code is a defined function where it reads an inputted txt file. Then converts all of it's contents from the raw binary data to a human readable hex value. It would then be formatted and spaced out accordingly. This is done so that the algorithm would interpret raw hex values. Not the characters assigned via UNICODE. Upon a successful conversation, a print statement is issued.



```python
# --- .txt to .bin to .txt converter --
def func_convert(filename):
    container = []
    with open(filename, 'rb') as file:
        while True:
            DL = file.read(1)
            if not DL:
                break
            container.append(int.from_bytes(DL, byteorder='big'))
        filename = str(ii) + ".txt"
        with open(os.path.join(path_ii, filename),'w') as file:
            for i in container:
                file.write(str(hex(container[i]))+ " ")
        print(str(ii) + ".txt has been created.")
        return container
```

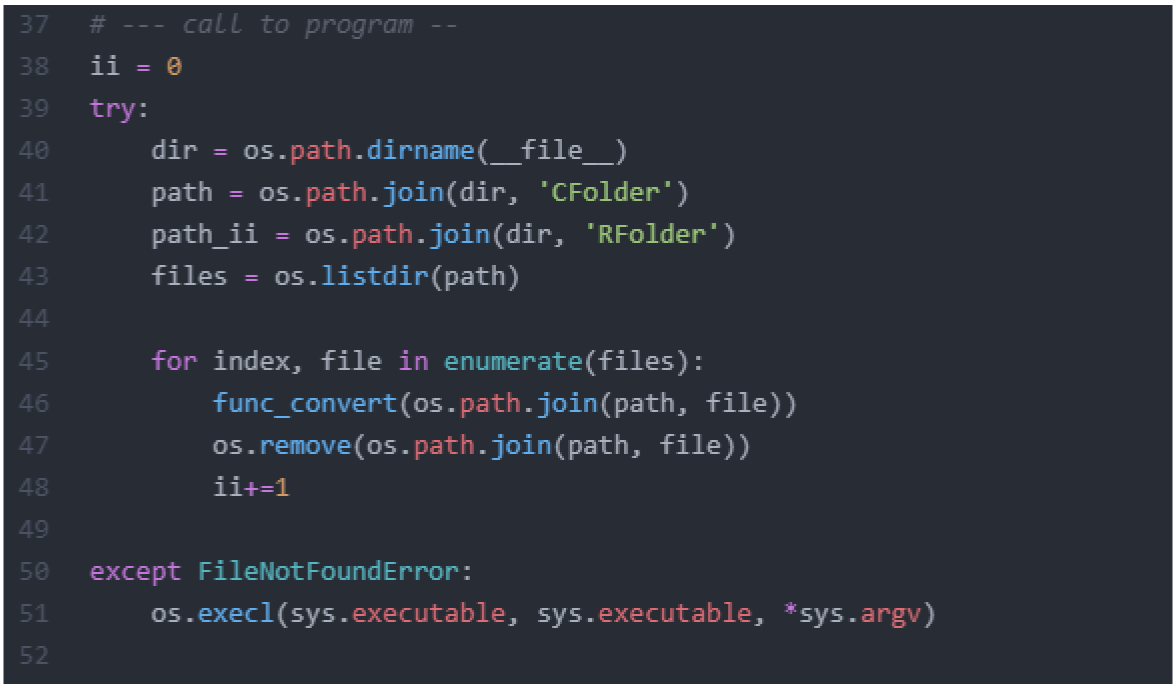
Figure 2C - Screenshot #3 of Code Snippet from SampleCreator.py

As shown in Figure 2D below. This snippet of code actually calls upon the functions of the program.

First off, it refreshes current file paths next to the program. Loading them into variables.

In which a FOR loop is run. It would go through every file contained within "CFolder" and output towards "RFolder". Upon the successful completion of the converted file. The ii variable is iterated by 1 and the process is repeated until all files are converted. Then it deletes files that was converted in order to save space. Then the program concludes and shuts down.

It is written within a Try/Catch block. As an error in which files weren't identified due to the fact that they were modified meant technically the file didn't exist. In which the program is commanded to simply restart the program to load the proper file paths.

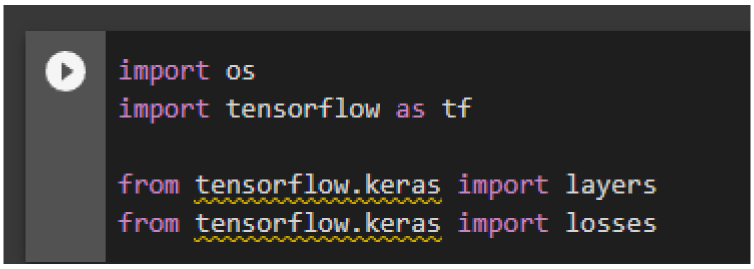


```python
# --- call to program --
ii = 0
try:
    dir = os.path.dirname(__file__)
    path = os.path.join(dir, 'CFolder')
    path_ii = os.path.join(dir, 'RFolder')
    files = os.listdir(path)

    for index, file in enumerate(files):
        func_convert(os.path.join(path, file))
        os.remove(os.path.join(path, file))
        ii+=1

except FileNotFoundError:
    os.execl(sys.executable, sys.executable, *sys.argv)
```

Figure 2D - Screenshot #4 of Code Snippet from SampleCreator.py

Google Collaborate: Machine Learning Algorithm

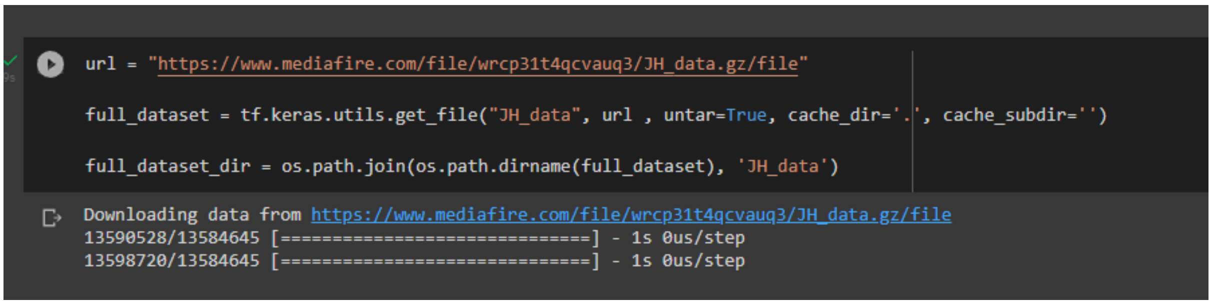
As shown within Figure 2E, within this code snippet are merely the libraries that are required for this program. Which includes "os" and "tensorflow".



```python
import os
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import losses
```

Figure 2E - Screenshot #1

As shown within Figure 2F. This code block enables for the downloading of sample data from a hosted website. Firstly, it loads the url as a variable. Then the syntax "utills.get_file" downloads the file and applies it to the folder name "JH_data". It is then applied to another variable.

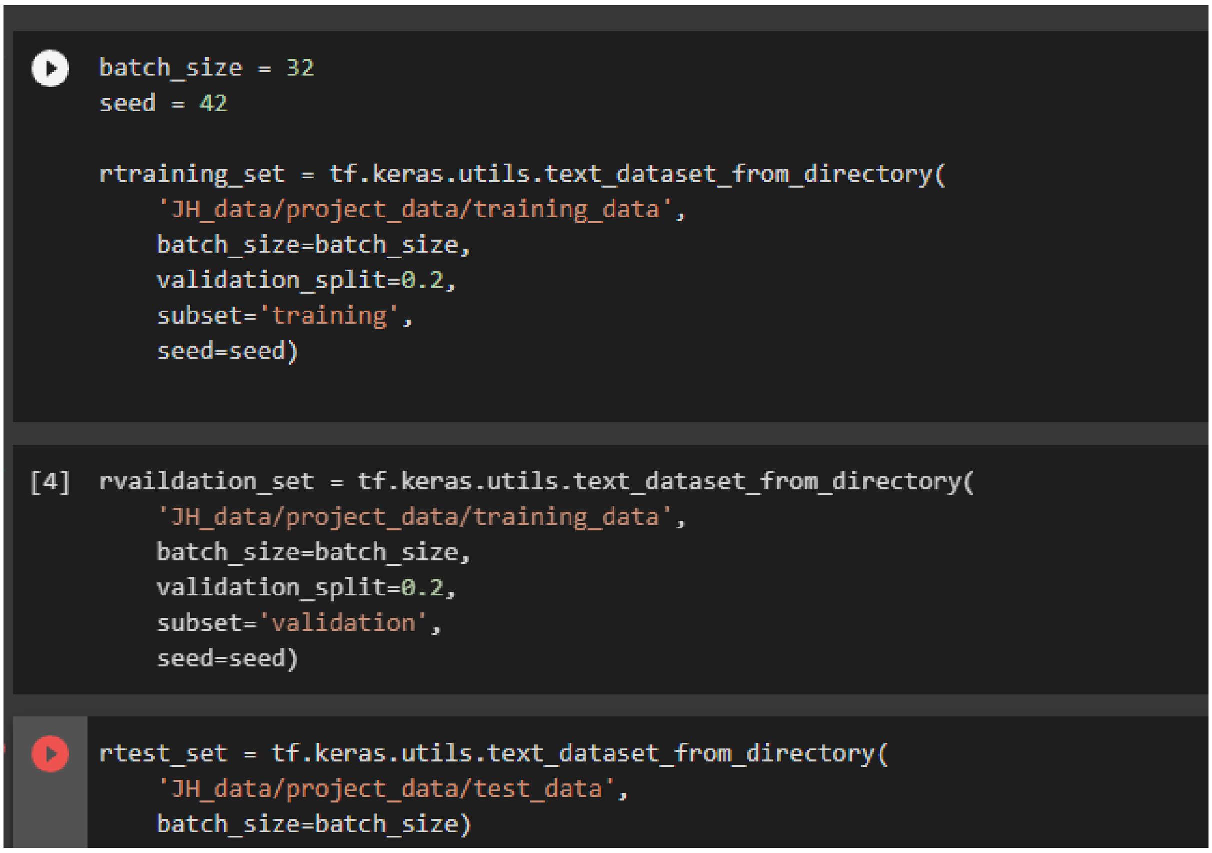


```
url = "https://www.mediafire.com/file/wrcp31t4qcvauq3/JH_data.gz/file"

full_dataset = tf.keras.utils.get_file("JH_data", url , untar=True, cache_dir='.', cache_subdir='')

full_dataset_dir = os.path.join(os.path.dirname(full_dataset), 'JH_data')
```

```
Downloading data from https://www.mediafire.com/file/wrcp31t4qcvauq3/JH_data.gz/file
13590528/13584645 [==============================] - 1s 0us/step
13598720/13584645 [==============================] - 1s 0us/step
```

Figure 2F - Screenshot #1

As shown within Figure 2G. Using folder structure, variables are assigned to specific folders. A specific syntax is used to configure how the sample data is used within the program. Such as, “validation_split” which cuts specific amounts of the sample data from being used. As well as, “batch_size” which determines how much sets of samples are fed to the program. This can be seen under “rtest_set”, where it is defined as the set of sample data that is being used for testing purposes. There aren’t any specific configurations as we would want to use the entire set without obstructions.



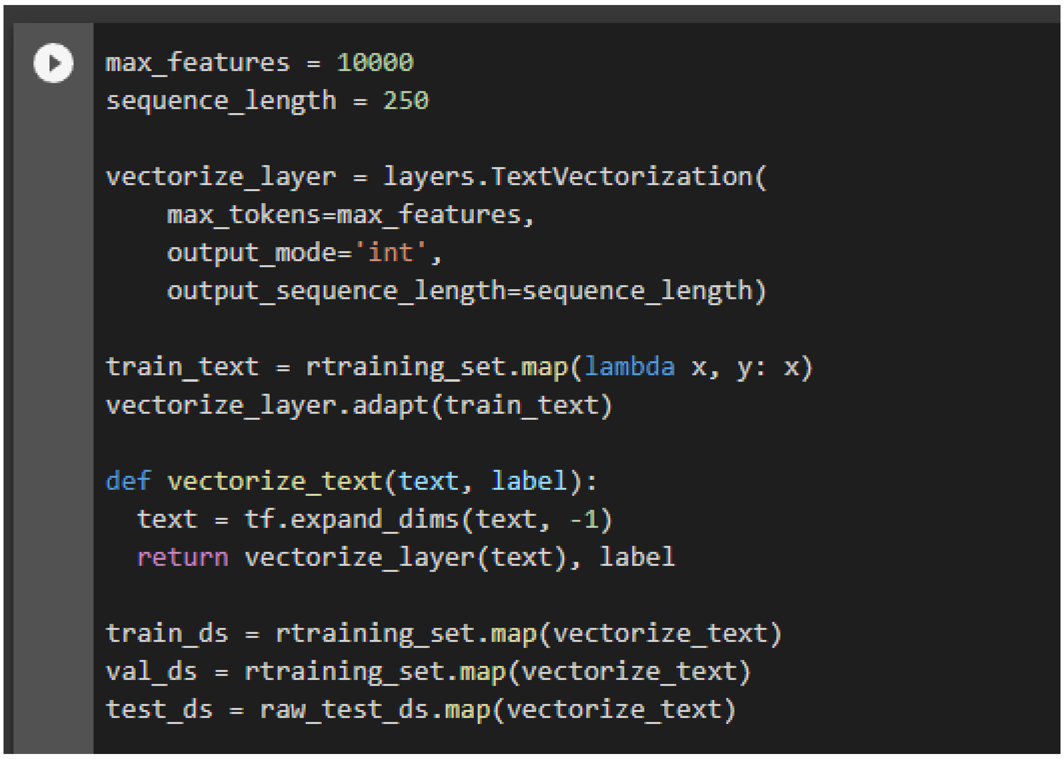
```
batch_size = 32
seed = 42

rtraining_set = tf.keras.utils.text_dataset_from_directory(
    'JH_data/project_data/training_data',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
```

```
[4] rvaildation_set = tf.keras.utils.text_dataset_from_directory(
    'JH_data/project_data/training_data',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
rtest_set = tf.keras.utils.text_dataset_from_directory(
    'JH_data/project_data/test_data',
    batch_size=batch_size)
```

Figure 2G - Screenshot #2

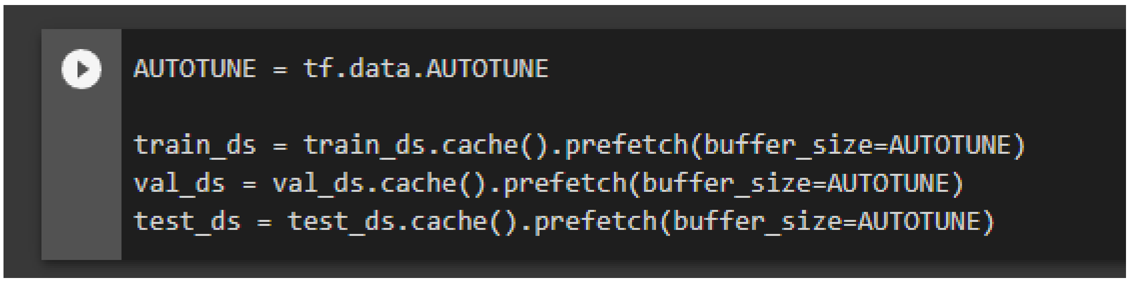
As shown within Figure 2H. It is a further step to processing the data. In which the sample data is “vectorized”. Essentially, it can be considered the connection between allowing the program to interpret the sample data properly and to train the program. First, the sample data is given definitions that can categorise the sample data. Then, the program is given the training set to aid in the creation of the program’s algorithm via the “adapt” syntax. Lastly, the relevant defined functions are called and applied to the sets of sample data.



```python
max_features = 10000
sequence_length = 250

vectorize_layer = layers.TextVectorization(
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)

train_text = rtraining_set.map(lambda x, y: x)
vectorize_layer.adapt(train_text)

def vectorize_text(text, label):
  text = tf.expand_dims(text, -1)
  return vectorize_layer(text), label

train_ds = rtraining_set.map(vectorize_text)
val_ds = rtraining_set.map(vectorize_text)
test_ds = raw_test_ds.map(vectorize_text)
```

Figure 2H - Screenshot #3

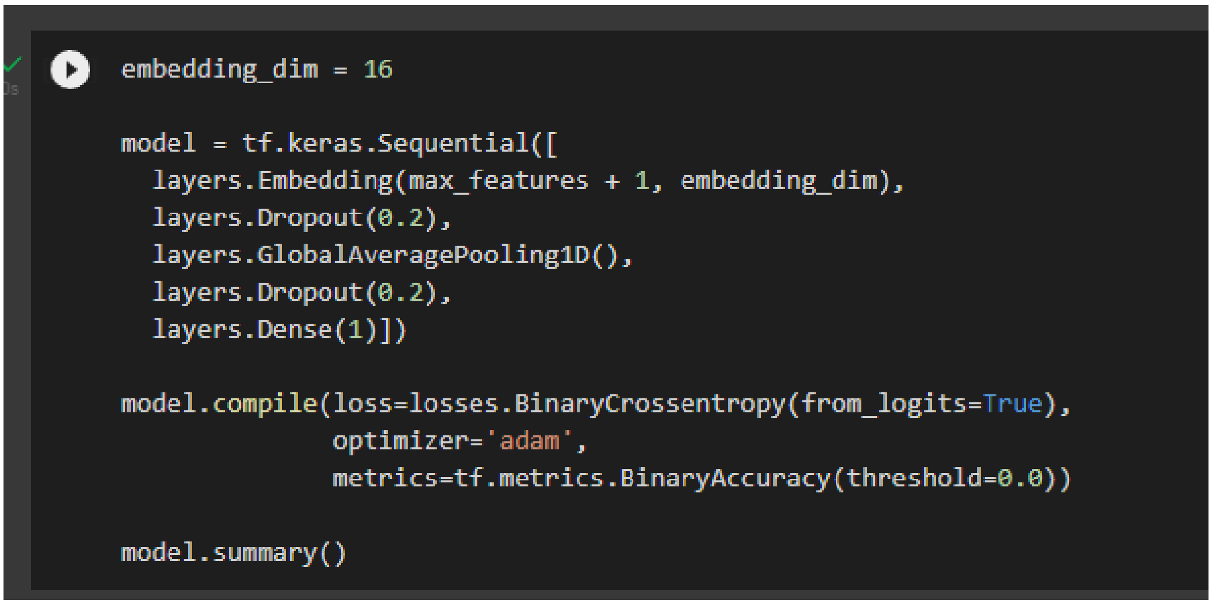
As shown within Figure 2I. This code block simply keeps the sample data loaded in memory rather than reading it off the storage medium that the sample data is hosted on. In addition, it dynamically changes the value of the amount of data fed towards the model.



```python
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Figure 2I - Screenshot #4

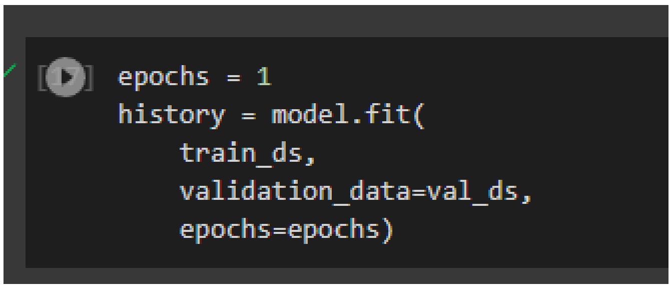
As shown within Figure 2J. This is where the model is structurally created. Various configurations are listed which define how layers are created functionally and it then calls for the model to be created and then printed as a summary.



```
embedding_dim = 16

model = tf.keras.Sequential([
  layers.Embedding(max_features + 1, embedding_dim),
  layers.Dropout(0.2),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(1)])

model.compile(loss=losses.BinaryCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=tf.metrics.BinaryAccuracy(threshold=0.0))

model.summary()
```

Figure 2J - Screenshot #5

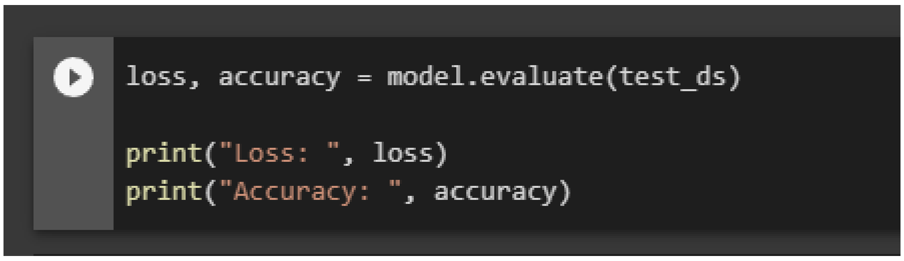
As shown within Figure 2K. The model is then called to train with the training data and validate its results using the test data. "epochs" is the variable which defines how many rounds the model should train and iterate upon the sample data. The results of this is printed out as an output of the code block.



```
epochs = 1
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs)
```

Figure 2K - Screenshot #6

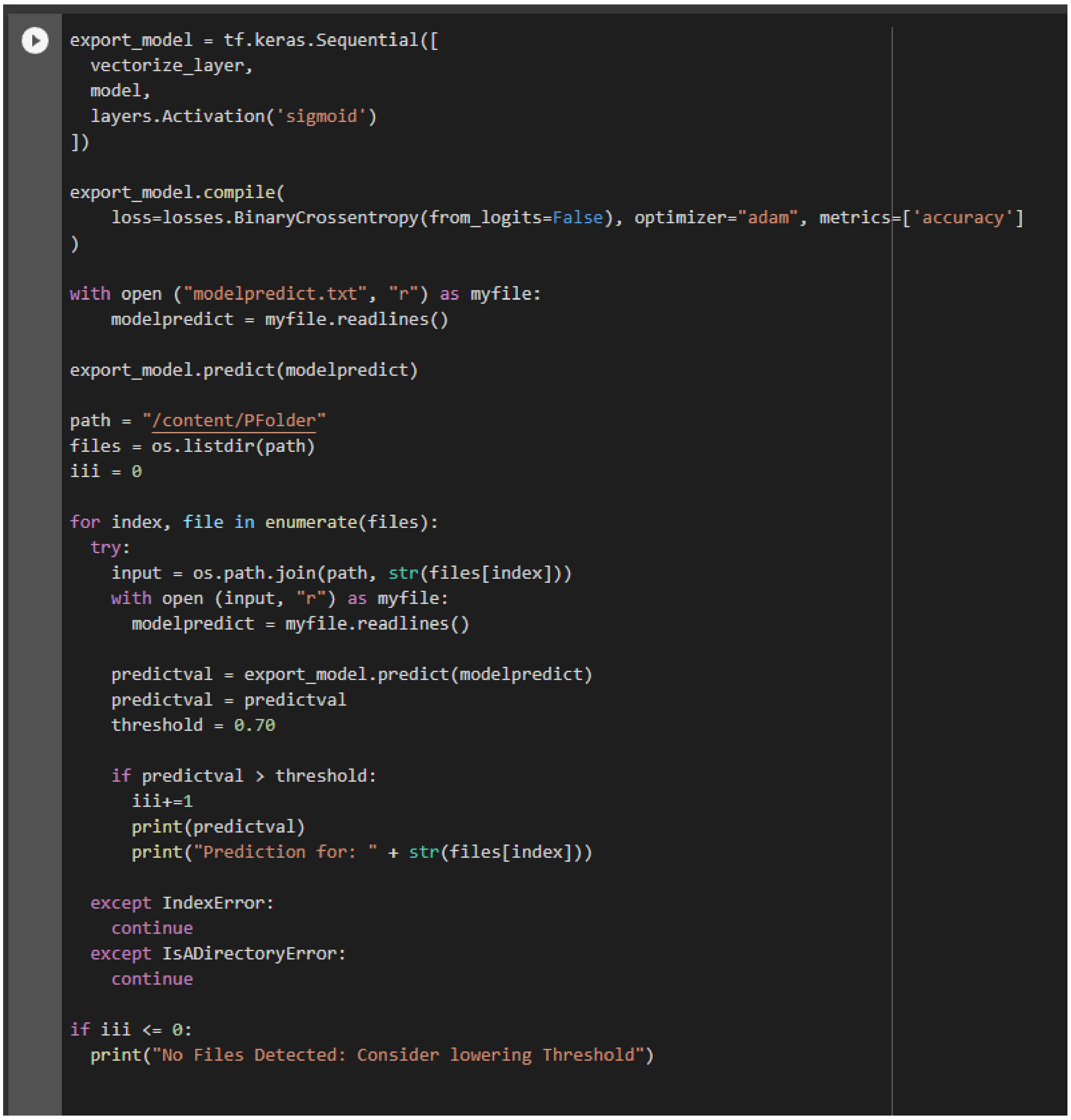
As shown within Figure 2M. The model is then fed the test data. In which, the models will test with the data to see if the model can predict if the samples have steganography measures or not. The results of this are outputted.



```
loss, accuracy = model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

Figure 2M - Screenshot #7

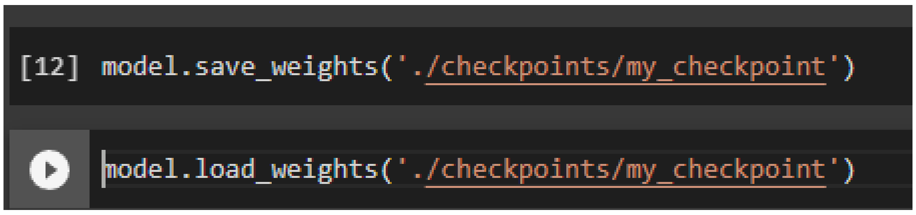
As shown within Figure 2N. The model is then called upon to go through the contents of "/PFolder". In which, it will highlight any file it suspects of steganographic measures.



```
export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam", metrics=['accuracy']
)

with open ("modelpredict.txt", "r") as myfile:
    modelpredict = myfile.readlines()

export_model.predict(modelpredict)

path = "/content/PFolder"
files = os.listdir(path)
iii = 0

for index, file in enumerate(files):
  try:
    input = os.path.join(path, str(files[index]))
    with open (input, "r") as myfile:
      modelpredict = myfile.readlines()

    predictval = export_model.predict(modelpredict)
    predictval = predictval
    threshold = 0.70

    if predictval > threshold:
      iii+=1
      print(predictval)
      print("Prediction for: " + str(files[index]))

  except IndexError:
    continue
  except IsADirectoryError:
    continue

if iii <= 0:
  print("No Files Detected: Consider lowering Threshold")
```

Figure 2N - Screenshot #8

Lastly, As shown within Figure 2N. This code snippet allows for the algorithm itself to be saved as a "checkpoint". In which the user can download this folder and reupload it to the program. If they desire to continue development of the model.



```
[12] model.save_weights('./checkpoints/my_checkpoint')

model.load_weights('./checkpoints/my_checkpoint')
```

Figure 2O - Screenshot #9