Configuration Manual

MSc Research Project
Cyber Security

Hardik Sawant
Student ID: x21232105

School of Computing
National College of Ireland

Supervisor: Evgeniia Jayasekara.

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Hardik Sudhir Sawant
Student ID:	x21232105
Programme:	MSc Cyber Security **Year:** 2022-2023
Module:	MSc Research Project.
Lecturer:	Evgeniia Jayasekara
Submission Due Date:	Evgeniia Jayasekara
Project Title:	Enhancing Digital Image Forensics in Cybersecurity Using Machine Learning.
Word Count:	**Page:** 18 **Count:** 705

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Hardik Sudhir Sawant ……………………………………………………………………………………………………

Date: 13-08-2023 ………………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	Hardik Sudhir Sawant
Date:	13-08-2023
Penalty Applied (if applicable):	

Configuration Manual

Hardik Sawant
Student ID: x21232105

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and its effects on a specific setting. A glimpse of the source for Data Importing & Image Preprocessing and after that Image augmentation while taking into consideration about class balancing ṁṇ, all the created algorithms, and Evaluations is also supplied, together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating environment configuration provided in Section 2.
Information about data gathering is detailed in Section 3. Image pre-processing included in Section 4's information extraction section. In section 5, the Image Augmentation is described. Details well about models that were created and tested are provided in Section 6. How the results are calculated and shown is described in Section 7.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. MacOs M1 Chip, macOS 10.15.x (Catalilna) operating system, 8GB RAM, 256GB Storage, 24'' Display.

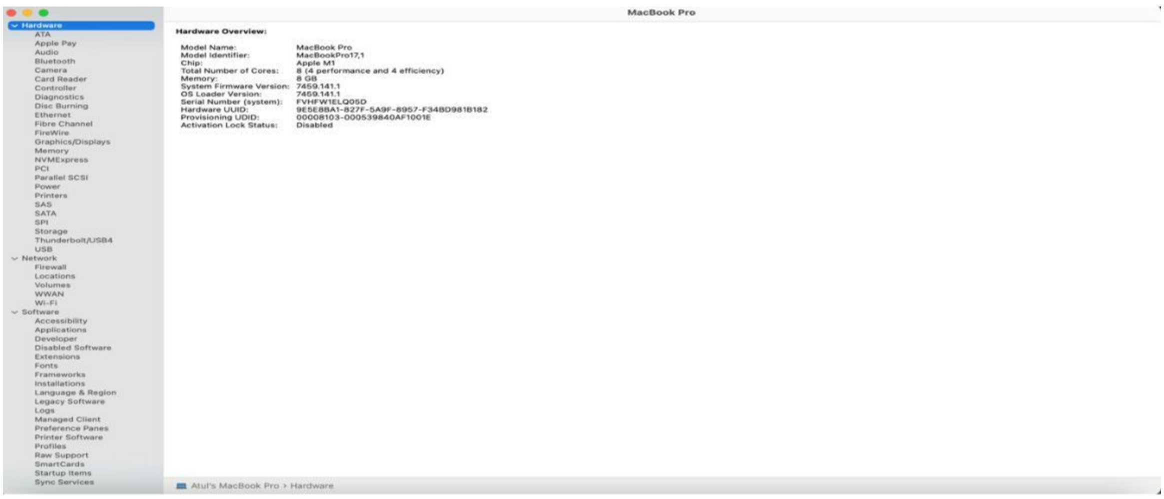

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and Run all cells.

3 Data Collection

The dataset is taken from Kaggle public repository from the link https://www.kaggle.com/c/alaska2-image-steganalysis. The dataset contains a large number of unaltered images, called the Cover image, as well as corresponding examples in which information has been hidden using one of three steganography algorithms (JMiPOD, JUNIWARD, UERD).

4 Data Exploration

Figure 2 includes a list of every Python library necessary to complete the project.

```
import glob, random, re
import os, sys
import pandas as pd
import shutil
import cv2
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
import pandas as pd
import warnings
from skimage.io import imshow, imread
from skimage.color import rgb2yuv, rgb2hsv, rgb2gray, yuv2rgb, hsv2rgb
from scipy.signal import convolve2d
warnings.filterwarnings("ignore")
from sklearn.decomposition import PCA
from collections import Counter
from imblearn.over_sampling import SMOTE
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.svm import SVC
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Layer, Input, Dropout, GlobalAveragePooling2D
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = True
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
```

Figure 2: Necessary Python libraries

The Figure 3 represents the block of code to make a list of the images in their respective categories.

```
JMiPOD = glob.glob("G:/Steganography/Alaska2/data/JMiPOD/*.jpg"
print ("Total of %d  images.\nFirst 5 filenames:" % len(JMiPOD)
print ('\n'.join(JMiPOD[:10]))
```

```
JUNIWARD = glob.glob("G:/Steganography/Alaska2/data/JUNIWARD/*.jpg")
print ("Total of %d  images.\nFirst 5 filenames:" % len(JUNIWARD))
print ('\n'.join(JUNIWARD[:10]))
```

```
UERD = glob.glob("G:/Steganography/Alaska2/data/UERD/*.jpg")
print ("Total of %d  images.\nFirst 5 filenames:" % len(UERD))
print ('\n'.join(UERD[:10]))
```

```
Cover = glob.glob("G:/Steganography/Alaska2/data/Cover/*.jpg")
print ("Total of %d  images.\nFirst 5 filenames:" % len(Cover))
print ('\n'.join(Cover[:10]))
```

```
Total of 4695  images.
First 5 filenames:
G:/Steganography/Alaska2/data/Cover\00001.jpg
G:/Steganography/Alaska2/data/Cover\00002.jpg
G:/Steganography/Alaska2/data/Cover\00003.jpg
G:/Steganography/Alaska2/data/Cover\00004.jpg
G:/Steganography/Alaska2/data/Cover\00005.jpg
G:/Steganography/Alaska2/data/Cover\00007.jpg
G:/Steganography/Alaska2/data/Cover\00008.jpg
G:/Steganography/Alaska2/data/Cover\00009.jpg
G:/Steganography/Alaska2/data/Cover\00010.jpg
G:/Steganography/Alaska2/data/Cover\00011.jpg
```

Figure 3: EDA for Checking Data Information

As seen in Figure 4, the block of code to set global variables of the train and test folder path, image size and list of categories.

```
trainDir = 'G:/Steganography/Alaska2/data/train'
testDir = 'G:/Steganography/Alaska2/data/test'
data_dir = 'G:/Steganography/Alaska2/data'
```

```
categories=['Cover', 'Steganography']
```

```
IMG_SIZE=100
```

Figure 4: Global Variables

In figure 5, the code to check for sharpening array for images and creating category dictionary.

```
sharpen = np.array([[0, -1, 0],
                    [-1, 5, -1],
                    [0, -1, 0]])
```

```
categories=sorted(categories)
print(categories)
```

```
['Cover', 'Steganography']
```

```
labels=[i for i in range(len(categories))]
labels
```

```
[0, 1]
```

```
label_dict=dict(zip(categories, labels))
label_dict
```

```
{'Cover': 0, 'Steganography': 1}
```

```
data_list=[] #data_list- storing the images
labels_list=[] #label_list - storing the class label:
```

Figure 5: Sharpening and Label Dictionary

The Figure 6, illustrate the code to functions and implementation of image convolver doing image sharpening.

```
def multi_convolver(image, kernel, iterations):
    for i in range(iterations):
        image = convolve2d(image, kernel, 'same', boundary = 'fill',
                           fillvalue = 0)
    return image
```

```
def convolver_rgb(image, kernel, iterations = 1):
    img_yuv = rgb2yuv(image)
    img_yuv[:,:,0] = multi_convolver(img_yuv[:,:,0], kernel,
                                     iterations)
    final_image = yuv2rgb(img_yuv)
    return final_image
```

```
final_image = convolver_rgb(og_image, sharpen, iterations = 1)
imshow(final_image);
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
```

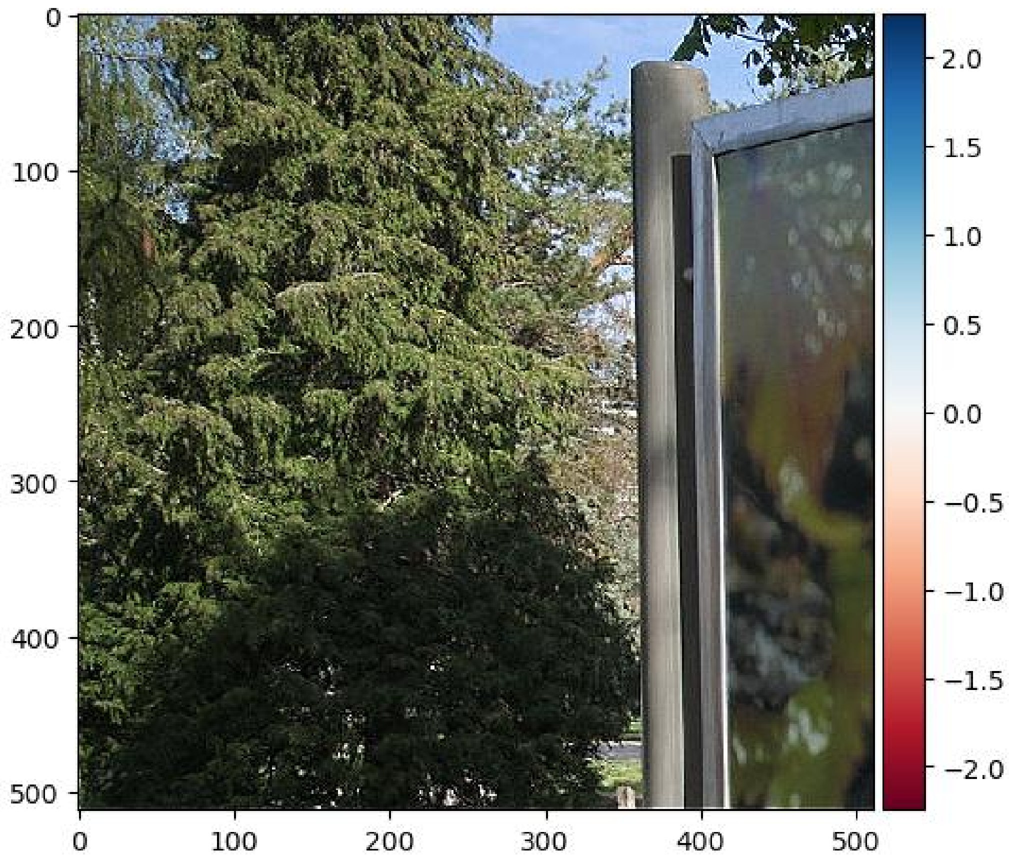

Figure 6: Multi Convolver

5 Image Augmentation

The Figure 7, illustrate the code for image augmentation.

```python
def plot_image(img, cmap='gray'):
    fig = plt.figure(figsize=(8,8))
    axes = fig.add_subplot(111)
    axes.imshow(img, cmap=cmap)
```

```python
# read image
img = cv2.imread(JMiPOD[2])

# blur
blur = cv2.GaussianBlur(img, (3,3), 0)

# convert to hsv and get saturation channel
sat = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)[:,:,1]

# threshold saturation channel
thresh = cv2.threshold(sat, 100, 255, cv2.THRESH_BINARY)[1]

# apply morphology close and open to make mask
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9,9))
morph = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)
mask = cv2.morphologyEx(morph, cv2.MORPH_OPEN, kernel, iterations=1)

# do OTSU threshold to get circuit image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
otsu = cv2.threshold(gray, 255, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# write black to otsu image where mask is black
otsu_result = otsu.copy()
otsu_result[mask==1] = 0

# write black to input image where mask is black
img_result = img.copy()
img_result[mask==0] = 0



# display it
plot_image(img)
plot_image(sat)
plot_image(mask)
plot_image(otsu)
plot_image(otsu_result)
plot_image(img_result)
```

*

Figure 7: Image Augmentation

Figures 8 show the code used to read a image and plot it in RGB and gray scale.

```
img1 = cv2.imread(JMiPOD[1])
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plot_image(img1)
width, height, dimension = img1.shape
print(f'Width RGB = {width}')
print(f'Height RGB = {height}')
print(f'Dimension RGB = {dimension}')
```

```
Width RGB = 512
Height RGB = 512
Dimension RGB = 3
```

```
img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
plot_image(img1_gray)
width, height = img1_gray.shape
print(f'Width Grayscale = {width}')
print(f'Height Grayscale = {height}')
print(f'Image Shape Grayscale {img1_gray.shape}')
```

```
Width Grayscale = 512
Height Grayscale = 512
Image Shape Grayscale (512, 512)
```

Figure 8: RBG and Gray Scale Image

The Figure 9, illustrate the code to generate threshold contour of the image using adaptive thresholding.

```
img1_gray = cv2.adaptiveThreshold(img1_gray,5,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,11,3)
plot_image(img1_gray)
```

```
contours = cv2.findContours(img1_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
contours = sorted(contours, key=cv2.contourArea, reverse=True)
for c in contours:
    x,y,w,h = cv2.boundingRect(c)
    img1_ROI = img1[y:y+h, x:x+w]
    break
plot_image(img1_ROI)
width, height, dimension = img1_ROI.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 310
Height = 512
Dimension = 3
```

Figure 9: Image threshold contouring

The Figure 10, illustrate the code for to generate folders for categories and move the images to their respective category folders.

```
try:
    for category in categories:
        path = os.path.join(trainDir, category)
        os.makedirs(path)
        path = os.path.join(testDir, category)
        os.makedirs(path)
    print("Folders created")
except:
    print("Folders already created")
```

```
Folders already created
```

```
def generateData(lst,fnm):
    for i in range(4695):
        if(i<=4230):
            destination=trainDir+ '/'+fnm
        else:
            destination=testDir+'/'+fnm
        shutil.copy(lst[i], destination)
try:
    generateData(Cover,"Cover")
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
```

```
def generateData(lst):
    for i in range(1565):
        if(i<=1410):
            destination=trainDir+ '/Steganography'
        else:
            destination=testDir+'/Steganography'
        shutil.copy(lst[i], destination)
try:
    generateData(JMiPOD)
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
try:
    generateData(JUNIWARD)
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
try:
    generateData(UERD)
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
Images set in training and testing folders
Images set in training and testing folders
```

Figure 10: Generate folders and data

Figures 11 show the code to read the images from their category folders and creating training data.

```
def create_data(loc):
    data=[]
    for category in categories:
        path=os.path.join(loc, category)
        class_num=categories.index(category)
        for img in os.listdir(path):
            try:
                img=cv2.imread(os.path.join(path,img))
                img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
                new_array=cv2.resize(img,(IMG_SIZE,IMG_SIZE)
                data.append([new_array,class_num])
            except Exception as e:
                pass
    return data
```

```
training_data= create_data(trainDir)
len(training_data)
```

```
5642
```

```
lenofimage = len(training_data)
X_train=[]
y_train=[]

for category, label in training_data:
    X_train.append(category)
    y_train.append(label)
```

```
X_train= np.array(X_train).reshape(lenofimage,-1)
```

```
X_train = X_train/255.0
X_train.shape
```

```
(5642, 10000)
```

```
y_train=np.array(y_train)
y_train.shape
```

```
(5642,)
```

Figure 11: Generate training data

Figures 12 show the code to read the images from their category folders and creating testing data.

```
: testing_data = create_data(testDir)
  len(testing_data)

: 618

: lenofimage = len(testing_data)
  X_test=[]
  y_test=[]

  for category, label in testing_data:
      X_test.append(category)
      y_test.append(label)

: X_test= np.array(X_test).reshape(lenofimage,-1)

: X_test = X_test/255.0
  print(X_test.shape)

  (618, 10000)

: y_test=np.array(y_test)
  y_test.shape

: (618,)
```

Figure 12: Generate testing data

The Figure 13, illustrate the code to select features using Principal Component Analysis.

```
pca = PCA(n_components=100)

# fit and transform data
X_train = pca.fit_transform(X_train)
X_train.shape

(5642, 100)

X_test = pca.transform(X_test)
X_test.shape

(618, 100)
```

Figure 13: Image feature selection

The Figure 14, illustrate the code to build training and testing data for neural network based model using Image Data Generator.

```
gen = ImageDataGenerator(rescale=1./255)
train = gen.flow_from_directory(directory=trainDir, target_size=(IMG_SIZE,IMG_SIZE),  class_mode='sparse'
```

```
Found 5642 images belonging to 2 classes.
```

```
gen = ImageDataGenerator(rescale=1./255)
test = gen.flow_from_directory(directory=testDir, target_size=(IMG_SIZE,IMG_SIZE), class_mode='sparse')
```

```
Found 618 images belonging to 2 classes.
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed i
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()


augmented_images = [train[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

Figure 14: Image Data Generator

6 Machine Learning Models

6.1 InceptioNEt

```
base_model = InceptionV3(weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='tanh')(x)
x = Dense(32, activation='tanh')(x)
x = Dropout(0.5)(x)
x = Dense(16, activation='tanh')(x)
x = Dropout(0.5)(x)
x = Dense(8, activation='tanh')(x)
predictions = Dense(1, activation='sigmoid')(x)
```

```
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])
es = EarlyStopping(monitor='loss', mode='min', verbose=1)
model.fit(train, epochs=10, callbacks=[es])
```

```
Epoch 1/10
177/177 [==============================] - 255s 1s/step - loss: 0.0000e+00 - accuracy: 0.7464
Epoch 2/10
177/177 [==============================] - 235s 1s/step - loss: 0.0000e+00 - accuracy: 0.7499
Epoch 2: early stopping
```

```
<keras.callbacks.History at 0x270cf7a9490>
```

```
LOSS, accuracyInc = model.evaluate(test)
accuracyInc*100
```

```
20/20 [==============================] - 6s 224ms/step - loss: 0.0000e+00 - accuracy: 0.7508
```

```
75.08090734481812
```

Figure 15: Implementation of InceptionV3

6.2 CNN

```
cnnModel = Sequential()
cnnModel.add(Conv2D(32, kernel_size=(3, 3),activation='tanh',padding = 'Same',input_shape=input_shape))
cnnModel.add(GlobalAveragePooling2D())
cnnModel.add(Dropout(0.25))
cnnModel.add(Dense(128, activation='tanh'))
cnnModel.add(Dropout(0.5))
cnnModel.add(Dense(1, activation='sigmoid'))
cnnModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_94 (Conv2D)          (None, 100, 100, 32)      896

 global_average_pooling2d_1   (None, 32)               0
 (GlobalAveragePooling2D)

 dropout_2 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 128)               4224

 dropout_3 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 1)                 129

=================================================================
Total params: 5,249
Trainable params: 5,249
Non-trainable params: 0
_________________________________________________________________
```

```
cnnModel.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
```

```
model.fit(train, epochs=10, callbacks=[es])
```

```
Epoch 1/10
177/177 [==============================] - 236s 1s/step - loss: 0.0000e+00 - accuracy: 0.7499
Epoch 2/10
177/177 [==============================] - 239s 1s/step - loss: 0.0000e+00 - accuracy: 0.7499
Epoch 2: early stopping

<keras.callbacks.History at 0x27080f0d0d0>
```

```
LOSS, accuracyCNN = cnnModel.evaluate(test)
accuracyCNN*100
```

```
20/20 [==============================] - 4s 167ms/step - loss: 0.0000e+00 - accuracy: 0.2508

25.080907344818115
```

Figure 16: Implementation of CNN

6.3 SVM

```
: svm = SVC(kernel='linear', C=0.5, gamma=10, random_state= 232)

: svm.fit(X_train, y_train)

: SVC(C=0.5, gamma=10, kernel='linear', random_state=232)

: #Predict the response for test dataset
  predSVM = svm.predict(X_test)

: accuracySVM = accuracy_score(y_test,predSVM)
  accuracySVM

: 0.7508090614886731
```

Figure 17: Implementation of SVM

7 Model result

This section explains the performance of the models.

7.1 Model Scores

	Model	Accuracy
)	InceptionNet	75.0
I	CNN	25.0
2	SVM	75.0

Figure 18: Model Performance

7.2 Accuracy

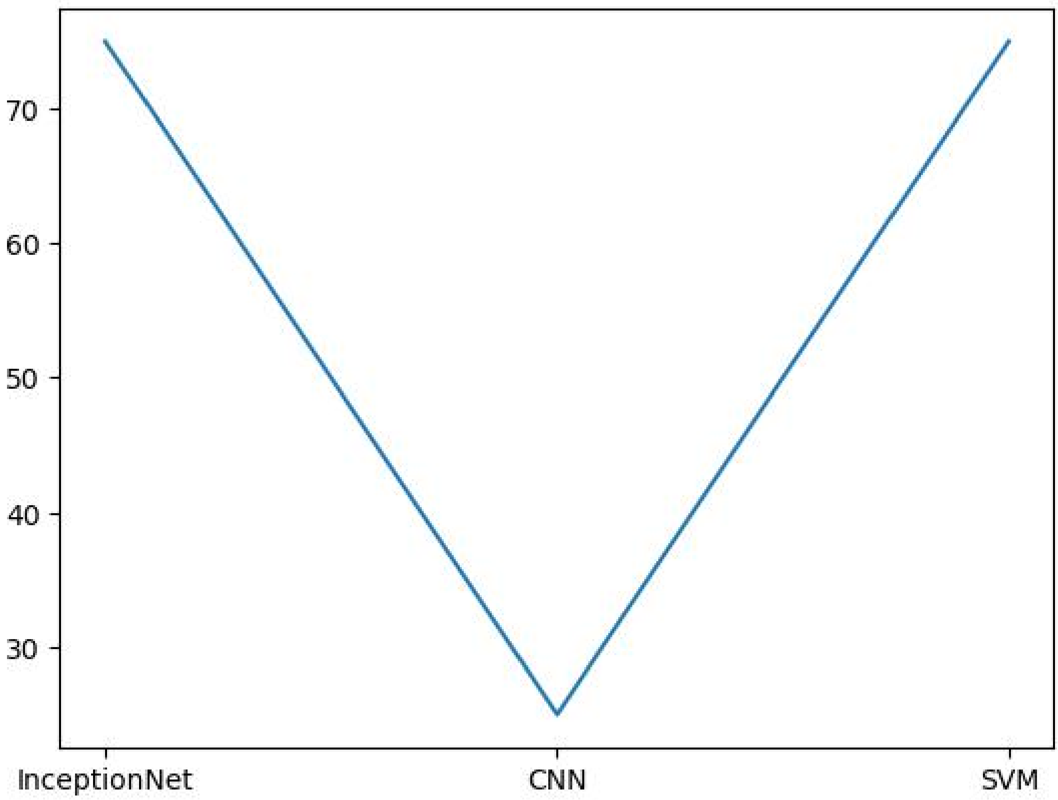

Figure 19: Accuracy

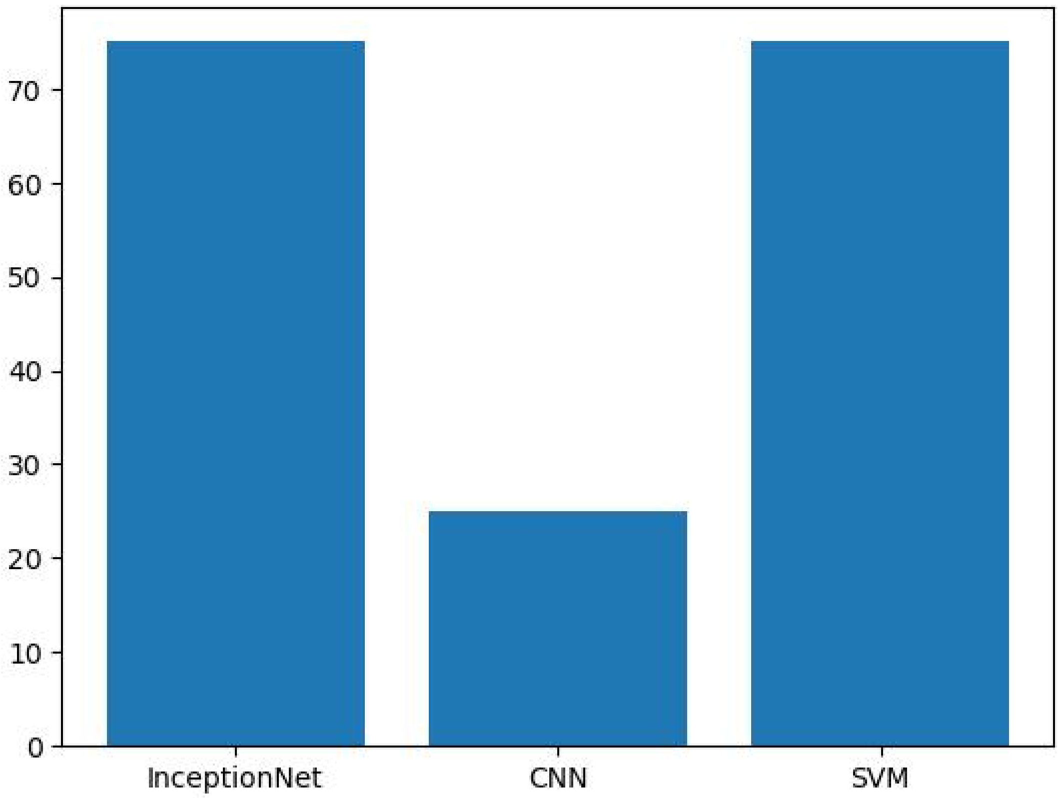

Figure 20: Accuracy

References

https://www.kaggle.com/c/alaska2-image-steganalysis

OpenCV: OpenCV modules

tf.keras.preprocessing.image.ImageDataGenerator | TensorFlow v2.13.0

Convolutional Neural Network (CNN) | TensorFlow Core

tf.keras.applications.inception_v3.InceptionV3 | TensorFlow v2.13.0

1.4. Support Vector Machines — scikit-learn 1.3.0 documentation