CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

One of the most pivotal part in the program is the pre-processing of image from the dataset before being used as an input for the model. Because the process gives hint to help the model learn the input image. The pre-processing function used in the model is as follows.

```
1 def imgPreProcess(img):
2     gbImg = gaborFilter(img)
3     cvImg = chanveseFilter(img)
4     resultImg = blendImage(np.asarray(cvImg[1]),gbImg)
5     resultImg = 2.2 * resultImg + 75
6     return resultImg
```

As seen above the function is very simple , it only call another individual function to do the actual pre-processing like in line number 2 where it calls for Gabor filter function and just like that line number 3 and 4 calls for Chan-Vese segmentation and combining function respectively. While line number 5 is used to increase contrast and brightness of the combined feature , because from what I saw the combined feature tend to have a rather low contrast and it makes me worried that it might cause the model to have a difficulty at learning the image.

However as simple as it looks the pre-processing function is very pivotal for the program. In previous section , I explained that the most fatal error that makes the model does not work as intended is located in the pre-processing part , to be more specific the error was located in the combining function.

```
1 def blendImage(imgGabor,imgCV):
2     imgCV = 255 - imgCV
3     blendImg = cv2.subtract(imgCV, imgGabor, dtype=cv2.CV_8U)
4     return blendImg
```

The codes right above this section is the combining or blending function. In line number 2 the output of Chan-Vese Segmentation was reversed , becaused if it was not reversed the combined feature will not work and only gives a blank white image as an output. Right below it , in line number 3 is what was used to combine the output of Gabor and Chan-Vese model. And as a reminder without dtype parameter the combining function will not work properly.

The functions above are key parts of the whole preparation for the dataset before they were sent to the ResNet50 neural network to be studied. And to take a closer look at the preparation codes below can give an idea how it works.

```
1. def loadData(DATA_DIR):
2.     DATA_Images = []
3.     progress = 1
4.     for imgFile in os.listdir(DATA_DIR):
5.         print('Image File Name : ',imgFile)
6.         tempHolder1 = cv2.imread(os.path.join(DATA_DIR,imgFile))
7.         tempHolder1 = cv2.cvtColor(tempHolder1, cv2.COLOR_BGR2YCR_CB)
8.           tempHolder2            =           cv2.resize(tempHolder1[:,:,1
           (200,200),interpolation=cv2.INTER_AREA)
9.         print("  Success Reading Image ...")
10.        imgProcess = imgPreProcess(tempHolder2)
11.        print("  Success Processing Image ...(",progress,")")
12.        progress += 1
13.        if 'au' in imgFile.lower():
14.            DATA_Images.append([imgProcess,1])
15.        elif 'tp' in imgFile.lower():
16.            DATA_Images.append([imgProcess,0])
17.    TRAIN_img, TEST_img = train_test_split(DATA_Images, test_size=0.2,
       random_state=25)
18.    train_set_x = []
19.    train_set_y = []
20.    test_set_x = []
21.    test_set_y = []
22.    for xTr,yTr in TRAIN_img:
23.        train_set_x.append(xTr)
24.        train_set_y.append(yTr)
25.    for xTe,yTe in TEST_img:
26.        test_set_x.append(xTe)
27.        test_set_y.append(yTe)
28.    return train_set_x, train_set_y, test_set_x, test_set_y
```

The load data fuction above start from reading images from directory as seen in line number 4 to 6. From file names in list directory from the addresed directory , each image was read by the the program before being sent to the next process. The next process , in line number 7 , the image obtained from the directory which in RGB format was then converted into YcbCr format for further processing , but before that the image must be resized to fit the input setting of ResNet50 just as seen in line number 8 , by taking the red chroma or Cr channel of YcbCr channel as input. In line number 10 , the resized single channel image from Cr channel was then used as input for pre-process function , where it will be used as input for combined feature method. Then in line 13 to 16 , based on the file name of the image that have just been processed , we conclude the ground truth as coded in the name. If the image have “au” in their file name , it means that the image is authentic which will be represented by the number 1 , while images with

“tp” in their name means that they are forged images which will be represented by the number 0. Only after we have processed all of the images from the directory and make a dataset , in line number 17 , we split the dataset for training and testing purposes with 20% of the dataset used for testing. Then in line 22 to 27 , the training and testing datesets were then divided into separate arrays for images and the ground truths in arrays we have prepared in line 18 to 21. Then finally in line 28 we return the results of the function which are 2 image arrays and 2 ground truth arrays one each for training and testing purposes.

As said in the previous section , in the later of the development , I tried to change the input of the dateset and to do that not only I need to change the parameters of ResNet50 , a change in dataset preparation is also needed. In other word a change in load data function must be done.

```
1. def loadData(DATA_DIR):
2.     DATA_Images = []
3.     progress = 1
4.     for imgFile in os.listdir(DATA_DIR):
5.         print('Image File Name : ',imgFile)
6.         tempHolder = cv2.imread(os.path.join(DATA_DIR,imgFile))
7.         tempHolder1 = cv2.cvtColor(tempHolder, cv2.COLOR_BGR2YCR_CB)
8.             tempHolder2              =              cv2.resize(tempHolder1
           (100,100),interpolation=cv2.INTER_AREA)
9.         print("  Success Reading Image ...")
10.        imgProcessCBlue = imgPreProcess(tempHolder2[:,:,2])
11.        imgProcessCRed = imgPreProcess(tempHolder2[:,:,1])
12.        imgProcess = tempHolder2
13.        imgProcess[:,:,2] = imgProcessCBlue
14.        imgProcess[:,:,1] = imgProcessCRed
15.        imgProcess[:,:,0]  =  4  *  (cv2.resize(cv2.cvtColor(tempHolder,
           cv2.COLOR_BGR2GRAY),(100,100),interpolation=cv2.INTER_AREA))
16.        print("  Success Processing Image ...(",progress,")")
17.        progress += 1
18.        if 'au' in imgFile.lower():
19.            DATA_Images.append([imgProcess,1])
20.        elif 'tp' in imgFile.lower():
21.            DATA_Images.append([imgProcess,0])
22.    TRAIN_img, TEST_img = train_test_split(DATA_Images, test_size=0.2,
        random_state=25)
23.    train_set_x = []
24.    train_set_y = []
25.    test_set_x = []
26.    test_set_y = []
27.    for xTr,yTr in TRAIN_img:
28.        train_set_x.append(xTr)
29.        train_set_y.append(yTr)
30.    for xTe,yTe in TEST_img:
31.        test_set_x.append(xTe)
32.        test_set_y.append(yTe)
```

```
33.     return train_set_x, train_set_y, test_set_x, test_set_y
```

In the new load data function , everything from reading images to spliting the datasets are the same except for the pre-processing step that happen in line 8 and 10 to 15. In line 8 unlike the previous load data function where they only resize Cr channel because the input is only a single channel image , in the current load data function the entire YcbCr format image is the that get resized. Which lead to line number 10 and 11 where blue and red chroma channel were individually processed by combined feature method. Then in line 13 and 14 , the results of line 10 and 11 were respectively put back together as Cb and Cr channel. However for the lumination or Y channel , instead of being replaced by the combined feature processed Y channel , it were replaced by a greyscale format of the original image that were increased in contrast by multiplying the array value by 4 , as seen in line number 15. One thing to note the new load data function above is the one used in the second new input method that use image with 3 channel which were high-contrast greyscale , blue chroma and red chroma channels that have been processed by combined feature , the first new input method use image with 3 channel consisting of red , green, and blue channel that have been processed by combined feature. The difference between the 2 new method in their load data function is that in the first method all 3 original color channel were processed by the combined feature were used in contrast with the second method that only use two , and another difference is that the first method does not convert the image into YcbCr and keep it in RGB format.

```
1. ROWS = 200
2. COLS = 200
3. CHANNELS = 1
4. CLASSES = 2
5.
6. DATA_DIRECT = "C:\\Users\\...\\ProjectDocument\\TrainValData\\"
7. train_set_x, train_set_y, test_set_x, test_set_y = loadData(DATA_DIRECT)
8. train_set_x = np.array(train_set_x,dtype="uint8")
9. train_set_y = np.array(train_set_y,dtype="uint8")
10.test_set_x = np.array(test_set_x,dtype="uint8")
11.test_set_y = np.array(test_set_y,dtype="uint8")
12.X_train = train_set_x/255
13.X_test = test_set_x/255
14.Y_train = convert_to_one_hot(train_set_y, CLASSES).T
15.Y_test = convert_to_one_hot(test_set_y, CLASSES).T
```

Now that the load data function has been made , further preparation can be made , such as in line number 1 to 4 are parameters prepared for ResNet50. In line number 6 is where the directory filled with images for the dataset is located and then it was used as input for the load

data function as seen in line number 7. Then in line number 8 to 11 , output from the load data function was moved into new arrays with “uint8” data type. Next in line 12 and 13 , image arrays was then normalized which was done by dividing them by 255 , this process was done to reduce the complexity of the datasets. Then the last preparation for the datsets to be made as inputs for machine learning , the ground truth arrays was the made into one-hot encoding to convert categorical to integer data as seen in line 14 and 15.

```
1.  model = ResNet50(input_shape = (ROWS, COLS, CHANNELS), classes =
    CLASSES)
2.
3.    model.compile(optimizer='adam',   loss='categorical_crossentropy',
    metrics=['accuracy'])
4.
5. K.set_value(model.optimizer.learning_rate, 0.0000001)
6.
7. model.fit(X_train, Y_train, epochs = 50, batch_size = 16)
8.
9. preds = model.evaluate(X_test, Y_test)
10.print ("Loss = " + str(preds[0]))
11.print ("Test Accuracy = " + str(preds[1]))
12.
13.model.summary()
14.
15.model.save('ResNet50.h5')
```

Then after preparing the dataset , ResNet50 structure was built with functions preapared as seen in line 1 , then in line 2 using compile we configure the model for training. After changing the learning rate in line 5 , with the training datasets and parameters that have been prepared we train our model in line 7. Then evaluate them in line 9 with testing datasets. Finally , after all said and done , in line 15 , the trained model was then saved as h5 file that can be loaded to predict an input image.

5.2. Results

Some of the recorded results are taken when the model does not work properly. Even worst , one of the problem cause test accuracy to get stuck and the only way we can observe the differnce is by looking at the training epoch. However , even with such shortcoming it can still be useful to answer the question and therefore the results will still be shown here.

One of the question asked is will there be an influence of learning rate to the model. Even when no data have been shown in this section , it can be concluded that the answer of that question is yes. The reason is because learning rate was the very source of problem that cause the

test accuracy to get stuck as it have been said in the previous section where drastically lowering the accuracy fix the problem. But that bring up another question just how low is too low , in other words we know that high learning rate of $1x10^{-2}$ to $1x10^{-3}$ is too high to the point it cause a problem but what kind of problem will arise if the learning rate is too low and what is the lowest learning rate where the model can still work properly? That question must be answered later because there are another question that must be answered.

The next question is will number of epoch infulence the performance of the model? The answer can be seen in the table below that shows training accuracy of 5 last training epochs.

Table 1: Epoch Influence Analysis Table

No	**Training Accuracy**	
	Epoch = 10	***Epoch = 50***
1	0.6360	0.6368
2	0.6336	0.6384
3	0.6258	0.6370
4	0.6357	0.6370
5	0.6348	0.6373

Despite the data was taken when learning rate error has not been taken care of the influence of epoch can still be seen. As seen above training that was done with higher number of epoch have higher results in accuracy. Because higher number of epoch means higher number of chance given to the model to learn from input data. It can be concluded that the answer of the question is yes , number of epoch does influence the model performance.

Now , how about batch size? Will decreasing the batch size change the results? Let us see the accuracy of the last training epoch and the test accuracy of three different attempt where the only change given is the batch size. As a note the data collection was done after fixing the learning rate to $1x10^{-6}$.

Table 2: Batch Size Influence Analysis Table

	Batch Size =16	***Batch Size = 8***	***Batch Size = 4***
Training Accuracy	0.6770	0.6755	0.6664
Test Accuracy	0.6228	0.6215	0.6209

It can be clearly seen from the table , there is a significant difference between each results. Even more it can be seen that decrease in batch size also results in decrease of accuracy , or at least under that scope of batch size. As the reason for such phenomenon is that batch size is the number of images being studied per section before it updates the model , and bigger number of batch size means that the model have more samples to compare before updating the model , so the model didn't just mindlessly update the model every few images considering difference between images in the dataset might be not as obvious in this case. However , the results already answer the question which is will batch size give an influence towards the model performance and the answer is yes.

Table 3: Image Size Influence Analysis Table

Image Size	***200x200***	***100x100***	***50x50***
Training Accuracy	0.7796	0.6934	0.6501
Test Accuracy	0.6184	0.6178	0.5967

What about image size of the input ? Will it affect the performance of the model ? And the answer is yes , as seen in the results above. By changing the image size , the difference in training accuracy is really huge. Not only that , the test accuracy is also affected. The difference in test accuracy between 200x200 and 100x100 as input is actually very small , however the results of 50x50 image is far lower than the other two , which mean as long as difference in image size is not too drastic , then image size shouldn't affect the final result too much. But , the results might be better with higher image size as input.

Now that we have seen the data that show the difference in output , we can conclude that the parameters does give such a huge impact in the model performance. A more fitting or suitable parameter can increase the performance and help us getting closer to the target accuracy even when all of the result so far have not reach the target.

And for the last question that needs to be answered in this section , will there be a difference in model performance between the usage of greyscale and image chroma (red) in the model performance? The table shows the difference between test accuracy of three run each for greyscale and red chroma based input. Another thing to add is that data collection for table below was done after fixing a fatal error where input image was not properly processed.

Table 4: Input Image Influence Analysis Table

Learning Rate	**Training Accuracy**		**Test Accuracy**	
	Greyscale	***Red Chroma***	***Greyscale***	***Red Chroma***
$1x10^{-6}$	0.9131	0.7796	0.5936	0.6184
$1x10^{-7}$	0.7186	0.6224	0.5930	0.6042
$1x10^{-8}$	0.5989	0.5935	0.5788	0.5788

It can be clearly seen that greyscale and red chroma give a completely different results. But there is something interesting , as shown on the table despite a higher training accuracy greyscale image as input have a lower training accuracy compared to red chroma as input. Because of those results , the only way we can clearly see which type of input image is better by comparing the difference between their training and test accuracy , and by that logic red chroma as input which have a lower difference between training and test accuracy is definitely the one works better as input for the model. However , can they be improved ? The method of pre-processing can be changed without deviating from the general architecture of the model which have given the results below.

Table 5: Advance Input Image Influence Analysis Table

Input	***1 channel : red chroma***	***3 channels : RGB***	***3 channels : CbCr channels , high contrast greyscale***
Training Accuracy	0.6934	0.7061	0.6945
Test Accuracy	0.6178	0.5807	0.6103

As a note , the content of table above are the results of implementation with lower image size compared to previous table which explain why it has a lower results. New pre-processing methodes as explained in the previous section were tested. As seen above , the method with the highest test accuracy is red chroma , however the difference between the test accuracy of red chroma with CbCr method is very close to each other with 0.0075 in differnce , which means the new method involving CbCr can compete and might even surpass red chroma results with further increase in epoch. However , the method with RGB channels as input does not perform too well where despite higher result in training accuracy , the test accuracy is far lower than the other two. These might have been caaused by similar problem with the combined feature processed greyscale in figure 3 , where there are to many object of interest in image input for neural network to learn about and the neural network might studying and make decision based on the wrong object of interest. The results of this implementation of changes in input have proven the reason why Asghar et al. [1] and Wang et al. [11] use image chroma as the input for their model in favor of a normal RGB or greyscale image , as it also proven by our implementation result that image chroma help better at detecting image forgery.

Unfortunately , despite numbers of change and testing in the impementation have been done , the model have not reach the 70% target accuracy , which mean the current states of the model would not be able to reliably detect image forgery. From numbers of testing , the most likely reason of such failure to reach the target is the very same reason why in an attempt to use RGB image with individually processed channel also end in worse result for the test accuracy in contrast to it’s training accuracy which tower over the different input , and that reason is the

neural network model confused the object of interest in the model which cause them to flop once the model must predict the result of the testing accuracy that the model have not seen before. In other word , the problem reside in the pre-processing , not the combined feature pre-processing as a whole but on the individual pre-processing of Gabor filter and Chan-Vese segmentation. The problem in pre-processing might have been caused by the fact that in the implementation of the model , the pre-processing have been barely touched to the point that the parameters in them have not been changed through out the data collection process. Unsuitable paramter for pre-processing might cause some unnecessary object of interest existence in the output of the process , that as said earlier leads to the neural network confuse the actual characteristic of an image with just random object that appear thanks to unfit pre-processing.