



School of Computing and Computer Engineering

CSC 691: Topics in CS – Fall 2021

Project: Image Forgery Detection using Convolutional Neural Network

UNDER THE GUIDANCE OF

Dr. Kala R. Marapareddy By

Krishnakanth Yachareni (W10098930)

APPENDIX

Abstract.....	Page 3
Python.....	Page 4
TensorFlow	Page 5
Keras.....	Page 7
CNN (Convolutional Neural Network)	Page 8
Deep Learning.....	Page 9
Project Implementation.....	Page 11
Environment.....	Page 14
App Benefits.....	Page 22
References.....	Page 23

ABSTRACT

For the past several years, social media like Facebook, Instagram, and SNS (Social Network Service) have been used by many people and still, the emerging of their use is increasing accordingly. They have become part of our lives. In particular, the development of smart devices such as smartphones has a remarkable role in uploading and downloading images to those social networks.

In the meantime, there has been a technique for manipulating an image using various methods with a specific purpose. Image tampering can be done by counterfeit criminals for the purpose of counterfeiting. Digital forensic techniques are needed to detect the tampering and manipulation of images for these illegal purposes and much research has been studied on these forensic techniques. However, they use features designed by human intervention and their performance is totally dependent on the differentiation of these features among original, tampered, and modified images.

This project proposes an image manipulation detection algorithm using deep learning technology. The model based on a convolutional neural network (CNN) is designed. Especially, a high pass filter is used to acquire hidden features in the image rather than semantic information in the image. The convolutional layer is composed of 2 layers having maximum pooling, ReLU (Rectified Linear Unit) activation, and local response normalization. The fully connected layer is composed of 2 layers.

For the experiments, modified images are generated using median filtering, Gaussian blurring, additive white Gaussian noise addition, and image resizing for 256x256 images that were divided into 4 equal parts of Boss Base 1.01 images.

Quantitative performance analysis is performed to test the performance of the proposed algorithm.

Python

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most-used programming languages today. The Python programming language best fits machine learning due to its independent platform and its popularity in the programming community. A survey conducted by industry analyst firm **RedMonk** found that it was the most popular programming language among developers in 2020

The merits of using Python are:

- **Free and open source:** We can download Python for free, which means that Python developers are able to download its source code, make modifications to it and even distribute it. Python comes with an extensive collection of libraries that support us to carry out our tasks.
- **Extensive support libraries:** It provides large standard libraries that include the areas like string operations, Numpy, SciPy, Pybrain, web service tools, operating system interfaces and protocols.
- **Integration Feature:** Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython.
- **Easy to create prototypes:** Python is simple to learn and could develop websites quickly. Python requires less coding, which means that you can create prototypes and test your concepts quickly and easily in Python as compared to several other programming languages.

However, Python as a full-fledged programming language. It lacks the following important features:

- Python is an interpreted language and dynamically typed language. The line-by-line execution of code often leads to slow execution.
- To provide simplicity to the developer, Python must do a little tradeoff. The Python programming language uses a large amount of memory.
- Python does not support threading because of Global Interpreter Lock, i.e., GIL which is a mutex; this permits only a single thread to execute at a time.

TensorFlow

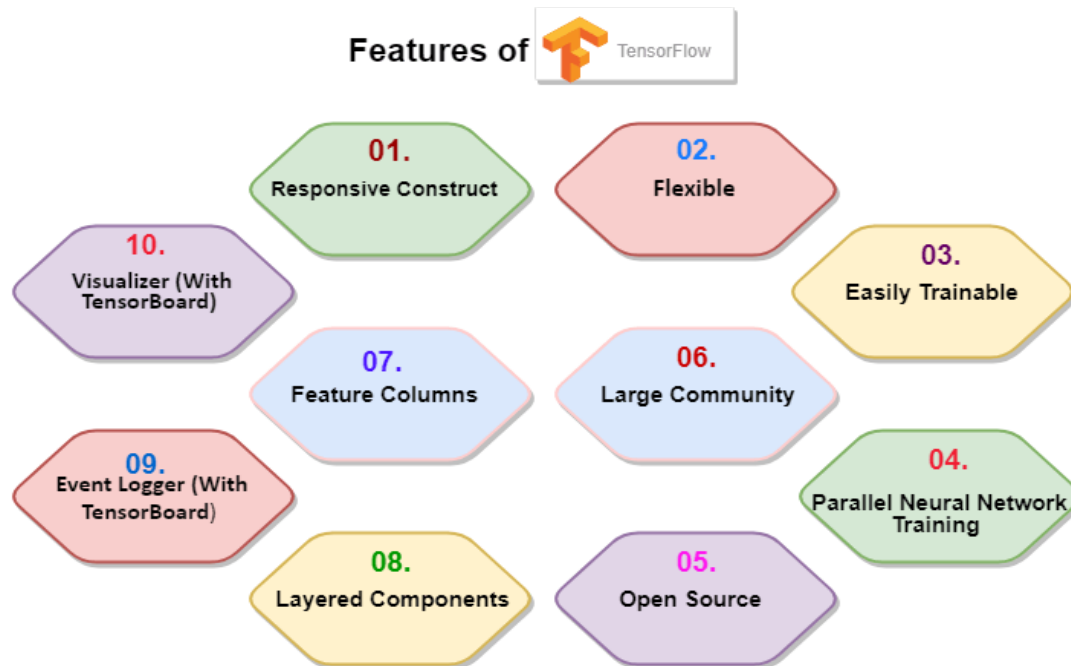
Tensorflow is an open-source library for numerical computation and large-scale machine learning that ease Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.



TensorFlow – Merits

- Tensorflow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++.
- It allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.
- TensorFlow has better computational graph visualizations. Which are inherent when compared to other libraries like Torch and Theano.
- It helps us execute subpart of a graph which gives it an upper hand as we can introduce and retrieve discrete data
- The libraries are deployed on a hardware machine, which is a cellular device to the computer with a complex setup.
- It has a unique approach that allows monitoring the training progress of our models and tracking several metrics.

TensorFlow – Features



TensorFlow – What are Tensors?

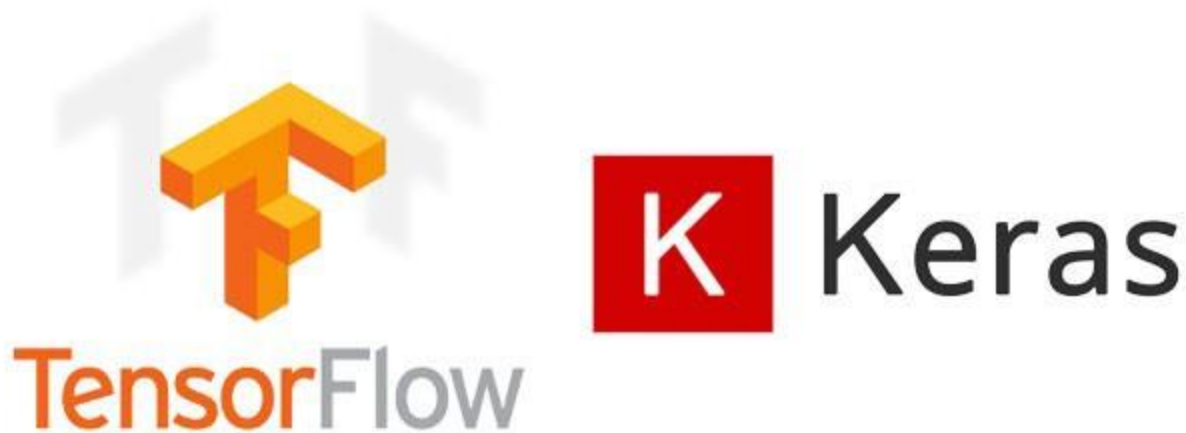
All the computations associated with TensorFlow involves the use of tensors. A tensor is a vector/matrix of n-dimensions representing types of data. Values in a tensor hold identical data types with a known shape. This shape is the dimensionality of the matrix. A vector is a one-dimensional tensor, matrix a two-dimensional tensor. Obviously, a scalar is a zero-dimensional tensor.

In the graph, computations are made possible through interconnections of tensors. The mathematical operations are carried by the node of the tensor whereas the input-output relationships between nodes are explained by a tensor's edge.

Thus, TensorFlow takes an input in the form of an n-dimensional array/matrix (known as tensors) which flows through a system of several operations and comes out as output. Hence the name TensorFlow. A graph can be constructed to perform necessary operations at the output.

Keras (The Python Deep learning API)

Keras is a deep-learning framework and open-source library for Python. It provides a Python interface for artificial neural networks and a suitable way to define and train nearly any kind of deep-learning model. It was originally developed for scholars and researchers. Its main aim was to enable rapid experimentation.



Keras key features are

- Keras permits the same code to run flawlessly on CPU or GPU.
- Its user-friendly API makes it easy to fast prototype deep-learning models.
- This was built-in support for computer vision& sequence processing or any combination of both.
- It may be freely used in profitable projects. It's well-matched with any version of Python from 2.7 to 3.6.
- Keras emphasizes being user-friendly, extensible, and modular due to its design to allow fast research with deep neural networks.
- It allows operators to productize deep models on smartphones, the web, and on the Java Virtual Machine.
- It also lets users of distributed training of deep-learning models on clusters of GPU and TPU.

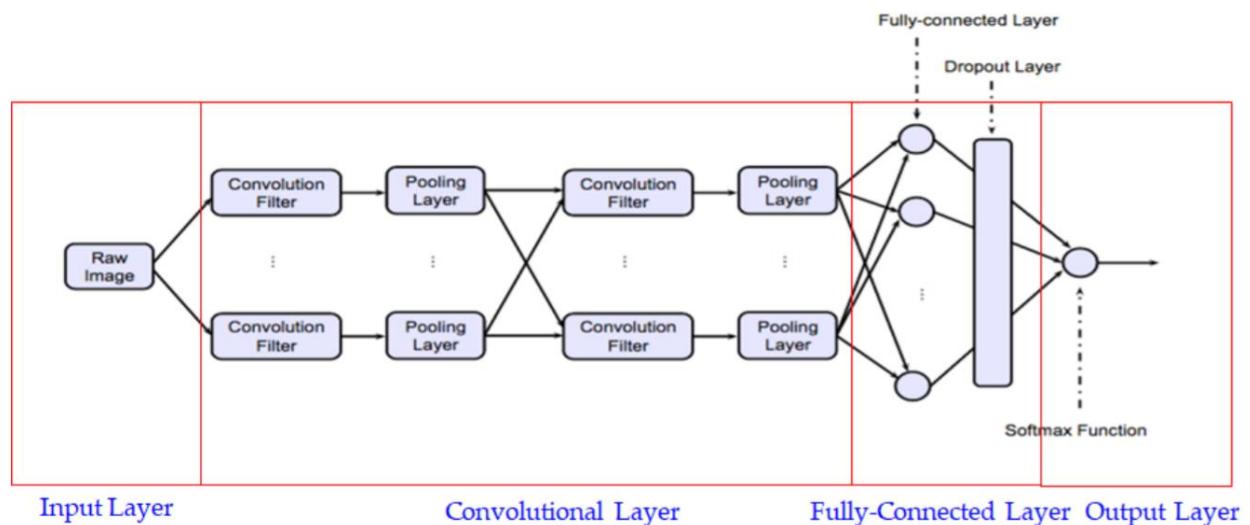
CNN (Convolutional Neural Networks)

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

Deep Learning

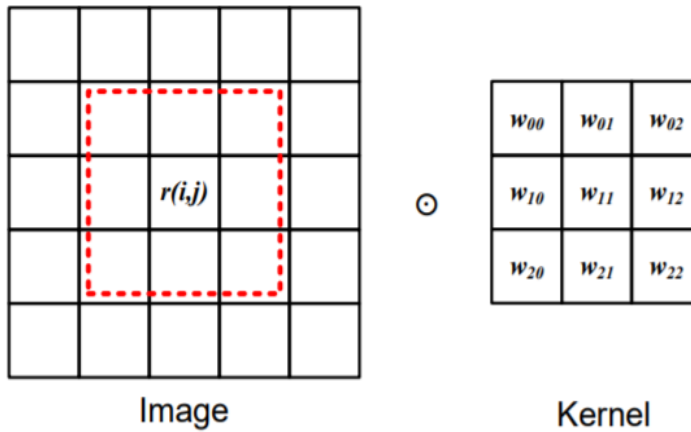
The computation of high complexity has been solved through the development of hardware performance. Furthermore, a variety of deep learning models have been proposed. One of them is a convolutional neural network model which is specialized in image processing. The convolutional neural network includes an input layer, a convolution operation layer, a fully connected layer, and an output layer as shown in below Fig.



The input layer is the set of input units. It is a passage through which pixels of an image for learning are entered. Its size is related to the number of image pixels [7]. The convolutional layer consists of various convolution filters. Across the convolutional layer, the result values are passed to the next layer in a nonlinearity. In the pooling layer, the dimensionality of the data is reduced. Next, in the fully connected layer, the classification is performed according to the learned results. Multiple fully connected layers can be stacked, Drop-out can be applied between each layer to prevent over-fitting or underfitting. Finally, the output layer is learned to score each class and in general SoftMax function is used.

In particular, the convolutional layer consists of various combination of convolution, pooling, and activation operations. The computation of convolution in a neural network is a product of a two-dimensional matrix called an image and a kernel or mask. This is depicted in Fig below. Through this convolution, local features considering neighboring pixels can be extracted.

Figure 1: Overall structure of convolutional neural network



$$r_{(i,j)} = r_{(i-1,j-1)} \cdot w_{00} + r_{(i,j-1)} \cdot w_{01} + r_{(i+1,j-1)} \cdot w_{02} + \dots \\ + r_{(i-1,j+1)} \cdot w_{20} + r_{(i,j+1)} \cdot w_{21} + r_{(i+1,j+1)} \cdot w_{22}$$

Figure 2: Convolution operation

The pooling layer appearing after convolution is to select a pixel value having a certain characteristic among pixels in a specific region, such as maximum pooling and average pooling. Figure 3 shows an example of maximum pooling to select maximum values. Through this pooling, the size of input data can be minimized to improve the time performance. However, in aspect of detecting image manipulation, it is possible to lose important traces to determine the modifications. The activation function is used to change the result of the hierarchy nonlinearly. Generally, ReLU, Sigmoid, tanh etc. are used.

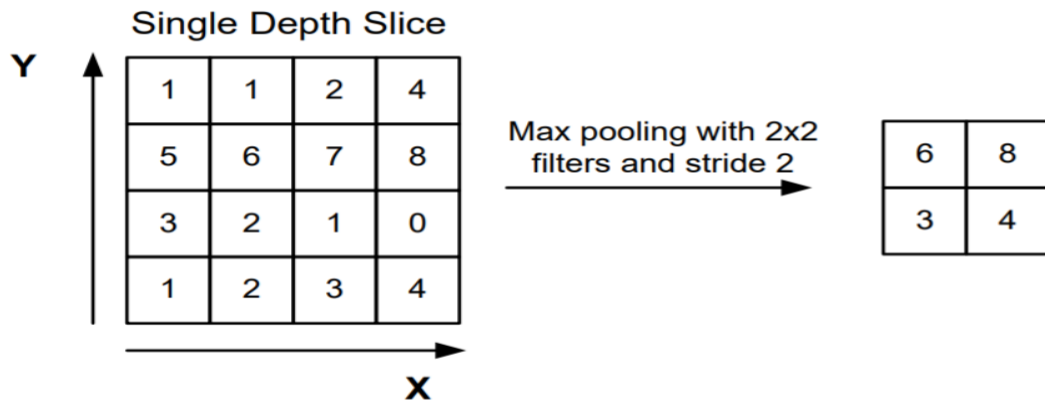


Figure 3: Maximum pooling operation

PROJECT IMPLEMENTATION

Detection Algorithm

The proposed algorithm is composed of two steps: learning and testing. Below Fig 4 shows a series of learning and testing processes. First, we manipulate the original image to produce a manipulated image. The generated image is classified into a learning set and a test set. Then, the learning set is fed into the proposed CNN model. Weights are then updated via back propagation. After learning, the test set is fed into the model that has been learned, and the accuracy is calculated by analyzing the result

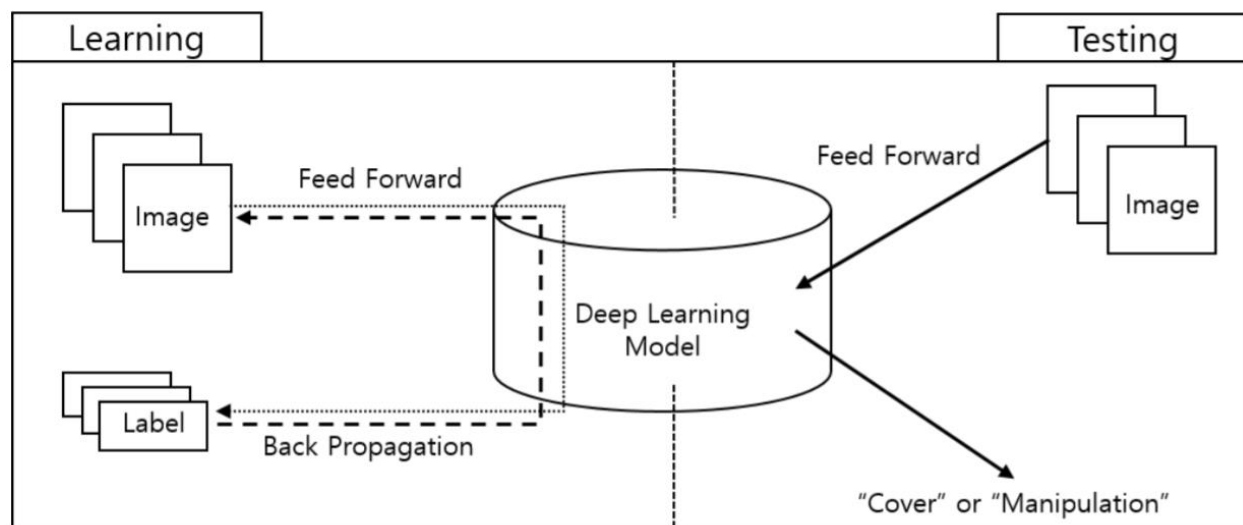


Figure 4: Image manipulation detection algorithm

CNN-based deep learning model for detecting image manipulation is shown in Fig. 5 below. The model is composed of 1 high pass filter, 2 convolutional layers, 2 fully connected layers, and 1 output layer.

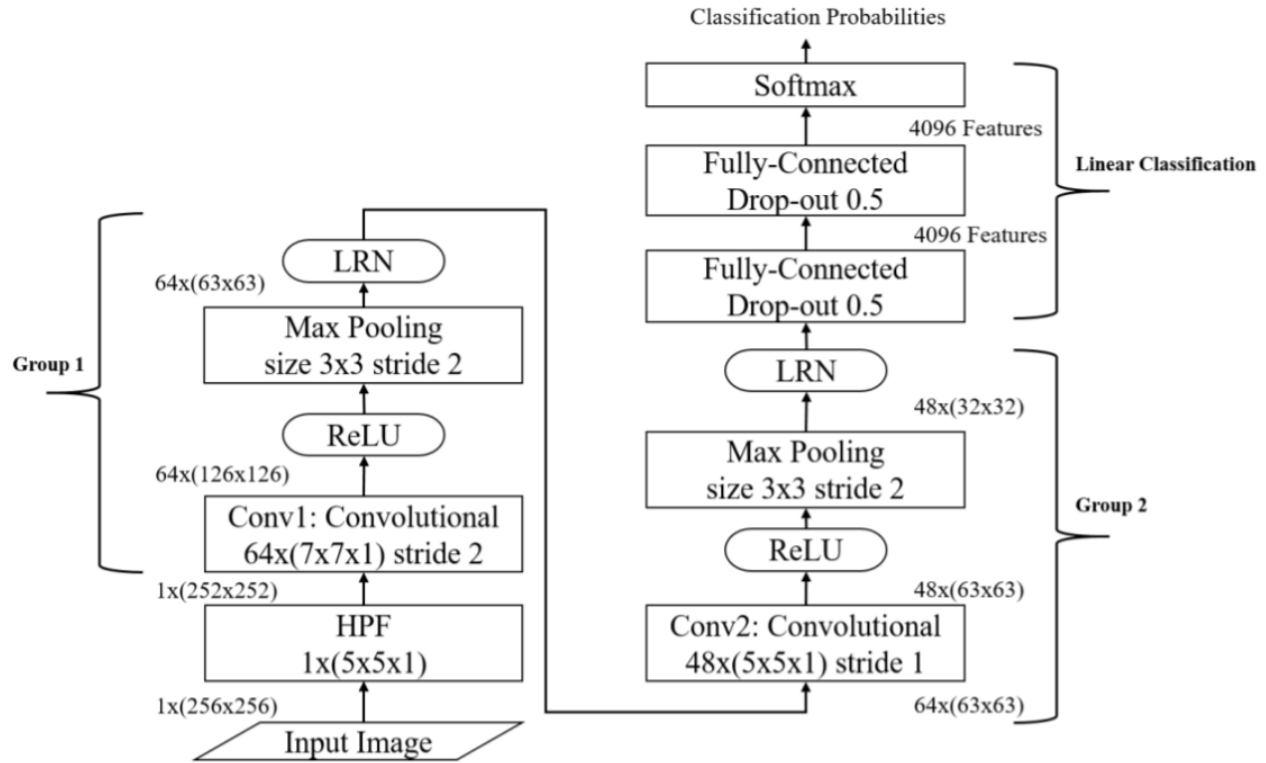


Figure 5: CNN model for image manipulation detection

High Pass Filter

In general, for image classification research, there is use of image pixel data. It behaves as if it was personally classified. However, our research targets those that cannot be distinguished by human eyes. So, we apply HPF called by High Pass Filter to extract hidden features within the image as follows: Before we delve deep into the solution, let us discuss about the different types of entities involved in our application. The important entities in our application are,

$$\text{HPF} = \frac{1}{12} \begin{pmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & 1 \end{pmatrix}$$

Convolution Layer

In the process of passing through the above HPF high frequency filter, one channel image of 256x256 size is converted into 252x252. The neuron values of 252x252 pass through a 7x7 kernel of Conv1 layer with stride 2. Initialization of weight values in deep learning is important enough to influence overall performance. In this paper, we use Xavier initialization. Xavier initialization is the square of the input value and the input value of the random number value of the output value. Conv1 uses the Xavier initialization mentioned. Then, it goes through the pooling layer, which have been described above, and pass to the Conv2 layer. The Conv2 layer allows 126x126 neuron values to pass through a 5x5 kernel. The stride value of Conv2 is 2, and initialization uses Xavier initialization like in Conv1. At every convolution layer, we use an activation function called ReLU (Rectifier Linear Units). ReLU changes the output to the value of $f(x) = \max(0, x)$.

Pooling Layer

Convolutional neural networks generally have a very large number of neurons. It has been shown that this increases the complexity of learning problem. To solve this problem, we apply the pooling layer, such as maximum pooling or average pooling. Maximum pooling is used in this study. In the layer behind Conv1 and Conv2, the kernel size is 3x3 and the stride is 2x2. Also, after the Maximum Pooling layer, the brightness is standardized through a layer called LRN (Local Response Normalization). In this model, the depth radius is 4, the bias is 1.0, the alpha value is $0.001 / 9$, and the beta value is 0.75.

Pooling Layer

In this study, two fully connected layers are used. Each has 4096 neurons, with a standard deviation of 0.4. This layer has the most basic DNN structure. We also have applied a drop-out of 0.5 to each layer to solve the over-fitting problem. Only 2048 of each of the 4096 neurons will be used for learning. We also use the SoftMax layer as the last layer for discrimination.

ENVIRONMENT SETUP

I use Google TensorFlow as a deep learning framework. The GPU of NVidia GeForce 1080Ti is used and the image batch size is set to 100. The images are 10,000 images of 512x512 size of Boss Base 1.01. First, 40,000 images of 256x256 size are created by dividing the images into two horizontally and vertically divided images. The total of 200,000 images including the original images and images from 4 algorithms: median filtering, Gaussian blurring, AWGN (additive white Gaussian noise addition), and Re-Sampling are used. Of these, 80% (160,000 images) are used as learning data and 20% (40,000 images) are used as test images. Samples of test images are shown in Fig. 6 below.

As shown in Table 1, for each algorithm, a 5x5 kernel is used for median filtering, Gaussian blurring with a standard deviation of 1.1, and AWGN with a standard deviation of 2. In the case of Re-Sampling, bilinear interpolation is performed at a magnification of 1.5 times to enlarge and reduce to its original size.

Manipulation operation	Parameter
Median filtering	5x5 kernel
Gaussian blurring	5x5 kernel, Standard deviation=1.1
AWGN (Additive White Gaussian Noise)	Standard deviation = 2
Re-Sampling	Scaling = 1.5

Table 1



Figure 6: Original images and manipulated images

Implementation code:

```
import cv2
import os
from PIL import Image
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import imageio
from imageio import imread

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
from keras.layers import BatchNormalization
from sklearn.model_selection import train_test_split
```

Cropping (512 x 512) 10000 images into (256 x 256) images

```
def crop_images(folder, dest_folder):
    count=1
    for filename in os.listdir(folder):
        print(filename)
        image = cv2.imread(folder+filename,cv2.IMREAD_UNCHANGED)
        height, width = image.shape[:2]

        #TOP LEFT
        start_row, start_col = int(0), int(0)
        end_row, end_col = int(height * .5), int(width * .5)
        cropped_top = image[start_row:end_row , start_col:end_col]
        #print(start_row, end_row)
        #print(start_col, end_col)
        cv2.imwrite(dest_folder+str(count)+'.jpg',cropped_top)
        count+=1
```



```

#TOP RIGHT
start_row, start_col = int(0), int(width * .5)
end_row, end_col = int(height * .5), int(width)
cropped_top = image[start_row:end_row , start_col:end_col]
#print(start_row, end_row)
#print(start_col, end_col)
cv2.imwrite(dest_folder+str(count)+'.jpg',cropped_top)
count+=1

#BOTTOM LEFT
start_row, start_col = int(height * .5), int(0)
end_row, end_col = int(height), int(width *.5)
cropped_top = image[start_row:end_row , start_col:end_col]
#print(start_row, end_row)
#print(start_col, end_col)
cv2.imwrite(dest_folder+str(count)+'.jpg',cropped_top)
count+=1

#BOTTOM RIGHT
start_row, start_col = int(height * .5), int(width * .5)
end_row, end_col = int(height), int(width)
cropped_top = image[start_row:end_row , start_col:end_col]
#print(start_row, end_row)
#print(start_col, end_col)
cv2.imwrite(dest_folder+str(count)+'.jpg',cropped_top)
count+=1

```

Below snippet produces 40000 Median Blur images (256 x 256) images

```

def median_blur(folder, dest_folder):
    count=20001
    for filename in os.listdir(folder):
        print(filename)
        img=cv2.imread(folder+filename,cv2.IMREAD_UNCHANGED)
        img_median=cv2.medianBlur(img,5)
        cv2.imwrite(dest_folder+str(count)+'.jpg',img_median)
        count+=1

```

Below snippet produces 40000 Gaussian blur images (256 x 256) images

```
def gaussian_blur(folder, dest_folder):
    count=40001
    for filename in os.listdir(folder):
        print(filename)
        img=cv2.imread(folder+filename,cv2.IMREAD_UNCHANGED)
        img_gaussian = cv2.GaussianBlur(img, (5,5),1.1)
        cv2.imwrite(dest_folder+str(count)+'.jpg',img_gaussian)
        count+=1
```

Below snippet produces 40000 Resampling images (256 x 256) images

```
#Resampling
def resampling(folder, dest_folder):
    count=60001
    for filename in os.listdir(folder):
        print(filename)
        img = cv2.imread(folder+filename, cv2.IMREAD_UNCHANGED)
        scale_percent = 150
        width = int(img.shape[1] * scale_percent / 100)
        height = int(img.shape[0] * scale_percent / 100)
        dsize = (width, height)
        output = cv2.resize(img, dsize)
        cv2.imwrite(dest_folder+str(count)+'.jpg',output)
        count+=1
```

Below snippet produces 40000 AWGN images (256 x 256) images

```
def awgn(folder, dest_folder):
    count=80001
    for filename in os.listdir(folder):
        print(filename)
        img = cv2.imread(folder+filename, cv2.IMREAD_UNCHANGED)
        # print(img.size)
        gauss = np.random.normal(0,2,img.size)
        gauss = gauss.reshape(img.shape[0],img.shape[1]).astype('uint8')
        # Add the Gaussian noise to the image
        img_gauss = cv2.add(img,gauss)
        cv2.imwrite(dest_folder+str(count)+'.jpg',img_gauss)
        count+=1
```

High Pass Filter

```
def my_filter(shape, dtype=None):
    f=np.array([[-1, 2, -2, 2, -1],
                [2, -6, 8, -6, 2],
                [-2, 8, -12, 8, -2],
                [2, -6, 8, -6, 2],
                [-1, 2, -2, 2, 1],
                ])
    k=1/12
    f=f*k
    f=f.reshape((5,5,1,1))
    print(f.shape,shape)
    assert f.shape == shape
    return tf.keras.backend.variable(f, dtype='float32')
```

Normalization Layer

```
from keras import backend as K
# from keras.engine.topology import Layer, InputSpec
from tensorflow.python.keras.layers import Layer, InputSpec

class LocalResponseNormalization(Layer):

    def __init__(self, n=5, alpha=0.0005, beta=0.75, k=2, **kwargs):

        self.n = n
        self.alpha = alpha
        self.beta = beta
        self.k = k
        super(LocalResponseNormalization, self).__init__(**kwargs)

    def build(self, input_shape):

        self.shape = input_shape
        super(LocalResponseNormalization, self).build(input_shape)

    def call(self, x, mask=None):
        if K.image_data_format == "th":
            _, f, r, c = self.shape
        else:
            _, r, c, f = self.shape
        squared = K.square(x)
```

```

pooled = K.pool2d(squared, (5, 5), strides=(1, 1),
padding="same", pool_mode="avg")
if K.image_data_format == "th":
    summed = K.sum(pooled, axis=1, keepdims=True)
    averaged = self.alpha * K.repeat_elements(summed, f, axis=1)
else:
    summed = K.sum(pooled, axis=3, keepdims=True)
    averaged = self.alpha * K.repeat_elements(summed, f, axis=3)
    denom = K.pow(self.k + averaged, self.beta)

return x / denom

def get_output_shape_for(self, input_shape):
    return input_shape

```

CNN Layer Model for Image Detection

```

#Get a model
def get_model(num_classes=5):
    model = Sequential()

    model.add(Conv2D(1, kernel_size=5, kernel_initializer=my_filter, input_shape=(256, 256, 1)))
    model.add(Conv2D(64, (7, 7), strides=2, padding='same'))
    model.add(Conv2D(64, (7, 7), strides=2, padding='same'))
    model.add(Activation('relu'))

    model.add(MaxPooling2D(pool_size=(3, 3), strides=2, padding="same"))
    model.add(LocalResponseNormalization())
    # model.add(BatchNormalization())
    #model.add(tf.nn.lrn(input=, depth_radius=4, bias=1, alpha=0.001/9, beta=0.75, name=None))
    #model.add(tf.keras.layers.Lambda(tf.nn.local_response_normalization))
    model.add(Conv2D(48, (5, 5), strides=1, padding="same"))
    model.add(Activation('relu'))

    model.add(MaxPooling2D(pool_size=(3, 3), strides=2, padding="same"))
    #model.add(tf.keras.layers.Lambda(tf.nn.local_response_normalization))
    #model.add(tf.nn.lrn(depth_radius=4, bias=1, alpha=0.001/9, beta=0.75, name=None))
    # model.add(BatchNormalization())

```


```

model.add(LocalResponseNormalization())
model.add(Flatten())
model.add(Dense(units=4096))
model.add(Dropout(0.5))
model.add(Dense(4096))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

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

return model

```

 model.summary()

 Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 252, 252, 1)	26
conv2d_1 (Conv2D)	(None, 126, 126, 64)	3200
conv2d_2 (Conv2D)	(None, 63, 63, 64)	200768
activation (Activation)	(None, 63, 63, 64)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 64)	0
module_wrapper (ModuleWrapper)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 48)	76848
activation_1 (Activation)	(None, 32, 32, 48)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 48)	0
module_wrapper_1 (ModuleWrapper)	(None, 16, 16, 48)	0
flatten (Flatten)	(None, 12288)	0
dense (Dense)	(None, 4096)	50335744
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 5)	20485
activation_2 (Activation)	(None, 5)	0
=====		
Total params: 67,418,383		
Trainable params: 67,418,383		
Non-trainable params: 0		

BENEFITS OF THIS PROJECT

Forgeries are not new to mankind but are a very old problem. In the past it was limited to art and literature but did not affect the public. Nowadays, due to the advancement of digital image processing software and editing tools, an image can be easily manipulated and modified. It is very difficult for humans to identify visually whether the image is original or manipulated. There is rapid increase in digitally manipulated forgeries in mainstream media and on the Internet. This trend indicates serious vulnerabilities and decreases the credibility of digital images.

This image manipulation detection algorithm verifies the integrity and authenticity of the digital images. Because it is very important, especially considering that the images are presented as evidence in a court of law, as news items, as a part of medical records, or as financial documents, in this sense, image forgery detection is one of the primary goals of image forensics.

REFERENCES

1. <https://www.tensorflow.org/>
2. <https://keras.io/>
3. http://www.ripublication.com/ijaer17/ijaerv12n21_156.pdf