Copy Move Forgery Detection

Siddhant Pande

Pratik Parale

June 2022 - August 2022

Mini Project Report

By

Siddhant J Pande

Pratik R Parale

Roll No - 2021510043, 2021510045

Guide:

Prof. Sakina Salmani

Department of Masters of Computer Applications

Sardar Patel Institute of Technology

Mumbai 400 058, INDIA.

2022-2023

ABSTRACT

The increasing popularity of the internet suggests that digital multimedia has become easier to transmit and acquire more rapidly. This also means that this multimedia has become more susceptible to tampering through forgery. One type of forgery, known as copy-move duplication, is a specified type that usually involves image tampering. With the exponential growth of high-quality fake images in social networks and media, it is necessary to develop recognition algorithms for this type of content. One of the most common types of image and video editing consists of duplicating areas of the image, known as the copy-move technique. Traditional image processing approaches manually look for patterns related to the duplicated content, limiting their use in mass data classification. In contrast, approaches based on deep learning have shown better performance and promising results, but they present generalization problems with a high dependence on training data and the need for appropriate selection of hyper parameters. To overcome this, we propose a approach that use deep learning.. Additionally, the problem of generalization is addressed with images from 1000s of different open access datasets. Finally, the models are compared in terms of evaluation metrics, and training and inference times.

Keywords: copy-move forgery detection; computer vision; deep learning; fake image; transfer learning;

1 Acknowledgements

ould firstly like to extend our heartiest thankfulness to our esteemed guide Prof. Sakina Salmani ma'am for inspiring us throughout the project and for her sustained interest, guidance and support. She has been thoroughly instrumental in showing us the right direction at all points in time. We would like to take this opportunity to thank our respected Head of Department, Dr. Pooja Raundale Ma'am ..

TABLE OF CONTENTS

LIST OF FIGURES

2

Digital editing is becoming less and less complicated with time, as a result of the increased availability of a wide array of digital image editing tools. Image forgery, which is defined as “the process of cropping and pasting regions on the same or separate sources [1], is one of the most popular forms of digital editing. Copy-move forgery detection technology can be applied as a means to measure an image’s authenticity. This is done through the detection of “clues” that are typically found in copy-move forged images.In the field of digital image forensics, copy-move forgery detection generally falls into two categories: key point-based and block-based [2].

This paper will focus on the latter category. Block-based copy-move forgery detection approaches employ image patches that overlap. From these, “raw”pixels are removed for forgery testing against similar patches [3]. Of the many strategies currently being employed in image forgery detection, several use statistical characteristics across a variety of domains [4]. Regardless of the forgery category, the forgery detection application will deal with active image copy-move forgery and/or passive copy-move forgery. In the former type, the original image includes embedded valuable data which makes the detection process easier, whereas in the latter, the original is imaging that makes the detection more challenging and difficult. Image forgery localization is even more difficult to carry out [5]. While forgery detection only seeks to know if an image is in whole or in part fake or original, image forgery localization tries to find the exact forged portions [5].Furthermore, in image forgery localization, the focus is on building a model rather than looking at only certain features or domains.

The model will be used to automatically detect specific elements based on a form of advanced deep neural network. Examples of these types of networks include deep belief network [6], deep auto encoder [7], and convolutional neural network (CNN) [8]. Of these three neural networks, CNNs are most commonly used in vision applications.

These approaches employ local neighborhood pooling operations and trainable filters when testing raw input images,thereby creating hierarchies (from concrete to abstract) of the features under examination. Because the image analysis and computer vision in the CNN strategy are so highly advanced, CNN generally provides excellent performance [9,10] in image forgery detection, through the composition of simplistic non-linear and linear filtering operations (e.g., rectification and convolution) .

This present paper proposes a novel approach for image forgery detection and localization which is based on scale variant convolutional neural networks (SVCNNs). An outline of the proposed method is presented in Figure 3. For this approach, sliding windows that incorporate a variety of scales are included in customized CNNs with the aim of creating possibility maps that indicate image tampering.Our main focus is both copy-move forgery detection and localization through the application of elements removed via the use of CNNs.The rest of the paper is organized as follows. In Section 2, we introduce an overview of the literature that has contributed to the advancement of CNNs in copy-move forgery detection and feature extraction procedures. In Section 3, we introduce the proposed model and the training processes.In Section 4, the experiment’s environment and results are discussed. Finally, in Section 5, we present the study’s conclusions.

3 Related Work

This section provides an overview of related works in copy-move forgery detection using neural network CNN's and related concepts.CNN for forgery detection based on discrete cosine transformation (DCT): Numerous researchers have approached the problem using CNN's for forgery detection. As discussed , CNNs can be used in analysis for gray-scale images, where the CNNs first layer features a single high pass filter to filter out the image content. An image model is developed for detecting image-splicing detection. In this approach, the researchers used discrete cosine transformation (DCT), to remove relevant features out of the DCT domain .The DCT domain feeds the input of the CNN by transferring the row of quantized DCT coefficients from the JPEG file to data classification. The processing of the data in the classification stage will generate a histogram for each patch and concatenate all of the histograms to feed the CNN .Deep learning methods applied to computer vision problems resulted in a local convolution feature data-driven CNN, while in other research, copy-move forgery detection algorithms were mostly based on computer vision tasks such as image retrieval , classification , and object detection .Along with CNN, graphics processing unit (GPU) technologies have helped to fuel the latest improvements in computer vision tasks . Unlike traditional strategies for image classification,which mostly use local descriptors , the latest CNN-based image classification techniques use end-to-end structure. Because deep networks typically incorporate classifiers and features that are high, mid, or low level using end-on-end multi layers, the various feature levels are enriched according to the number of hidden layers. The most recent convolutional neural networks (e.g.,VGG (Convolutional network for classification and detection) , significantly enhance performance in object detection and image classification tasks. [**?**]

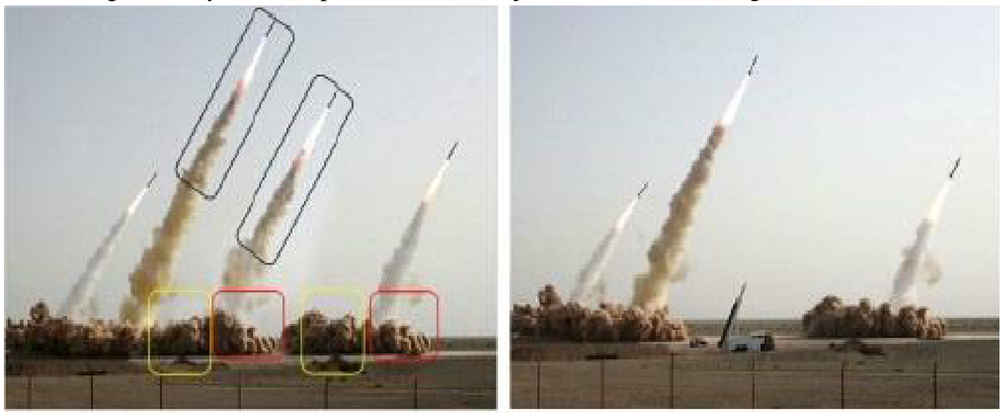


Figure 1: Example of Copy Move Forgery

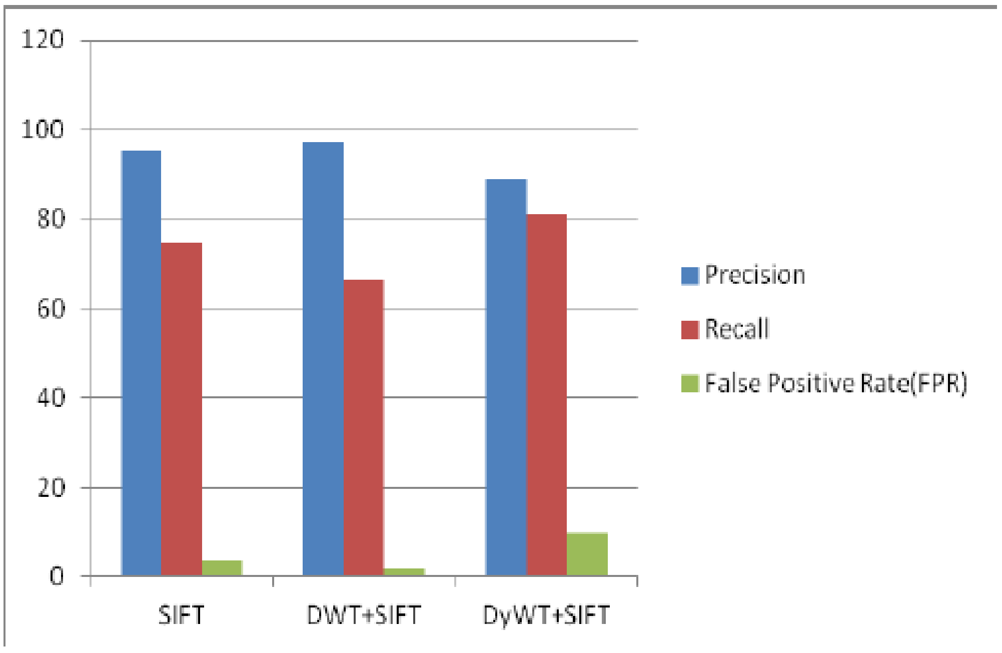


Figure 2: Comparision between different methodologies

4 Dataset

We trained and tested our model on Cifar 10 and Cifar 100 dataset for detecting Forgered regions using cnn .The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. . The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.d [**?**]

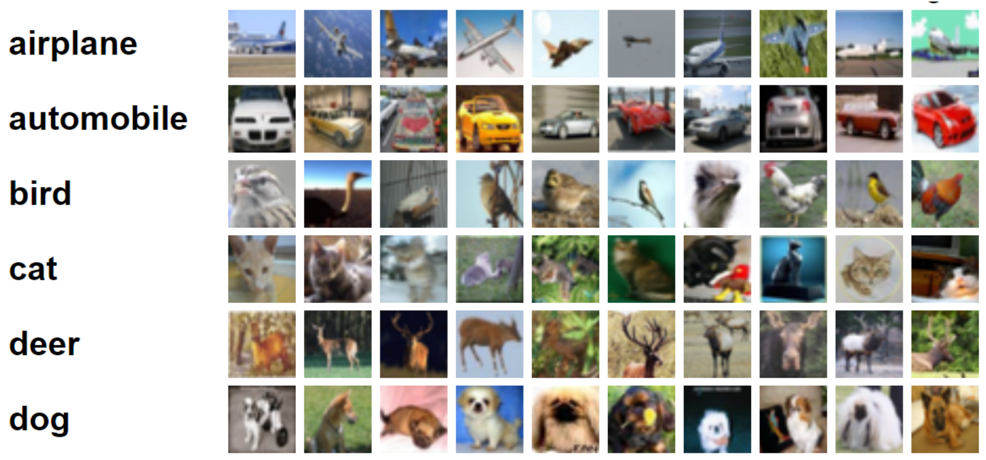


Figure 3: CIFAR 10 dataset

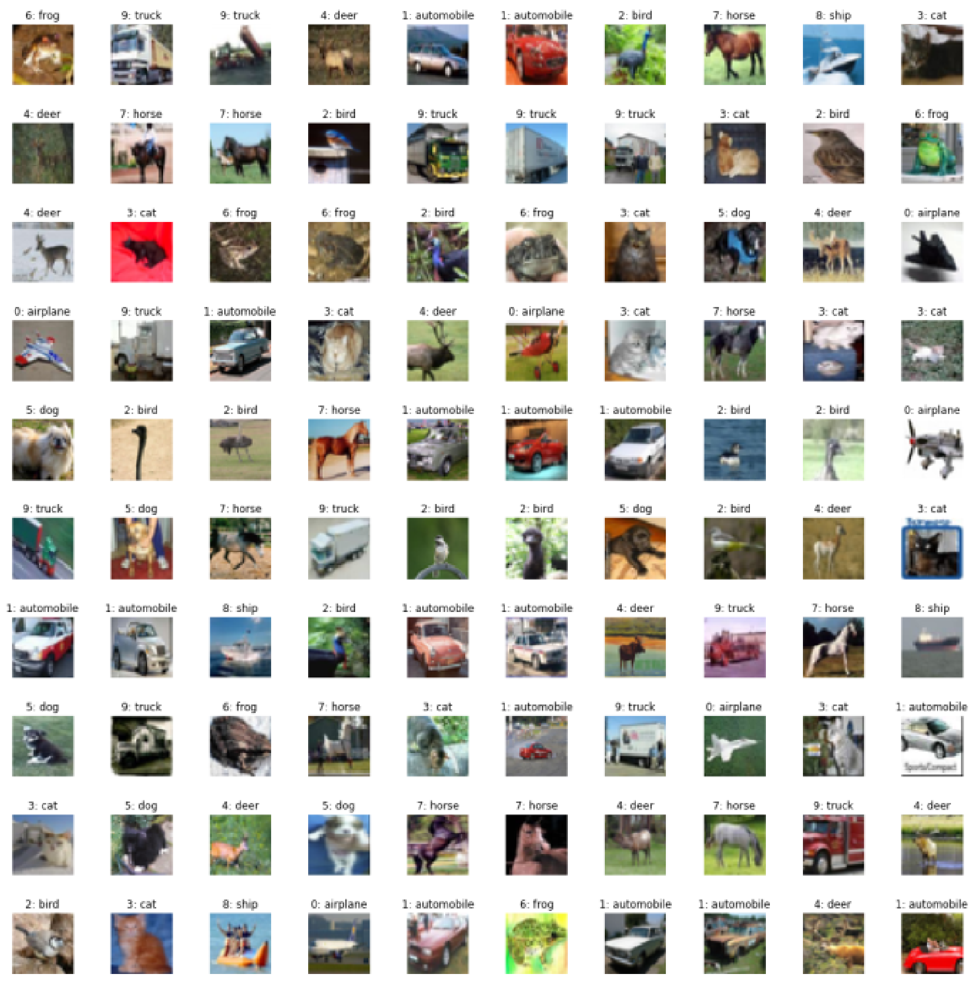


Figure 4: CIFAR 100 dataset

5 Proposed Method

CNN's are nonlinear interconnecting neurons based on the construct of the human visual system.Applying CNN's for forensic purposes is a somewhat new approach, but their ability to segment images and identify objects is thus far unsurpassed [36]. In one study, where CNNs were used to extract input images' features in order to classify them, the method outperformed all previous state-of-the-art approaches. Therefore, and based on the method from [36], our proposed CNN will be used as a feature extractor for image input patches in the training stage and, later on, for the testing stage as well (see Figures 2and 3). CNN's can be deconstructed into building blocks known as layers. Layer L i will accept relevant input Hi×Wi×Pi for feature maps or vectors sized asPi. This layer then gives the output Hi+1×Wi+1×Pi+1for feature maps or vectors sized asPi+1. In the present study,we use six different kinds of layers: convolutional, pooling, ReLU, softmax, fully-connected, and batch normalization. A brief description of each type of these layers is given below.

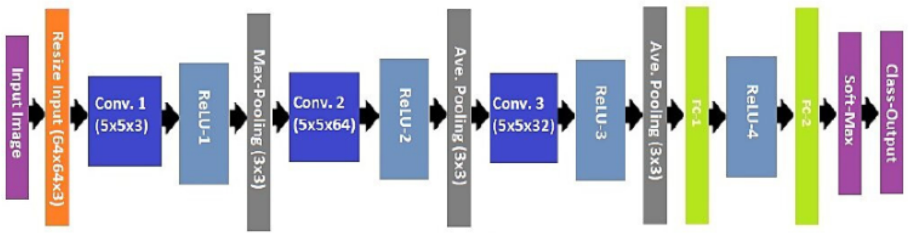
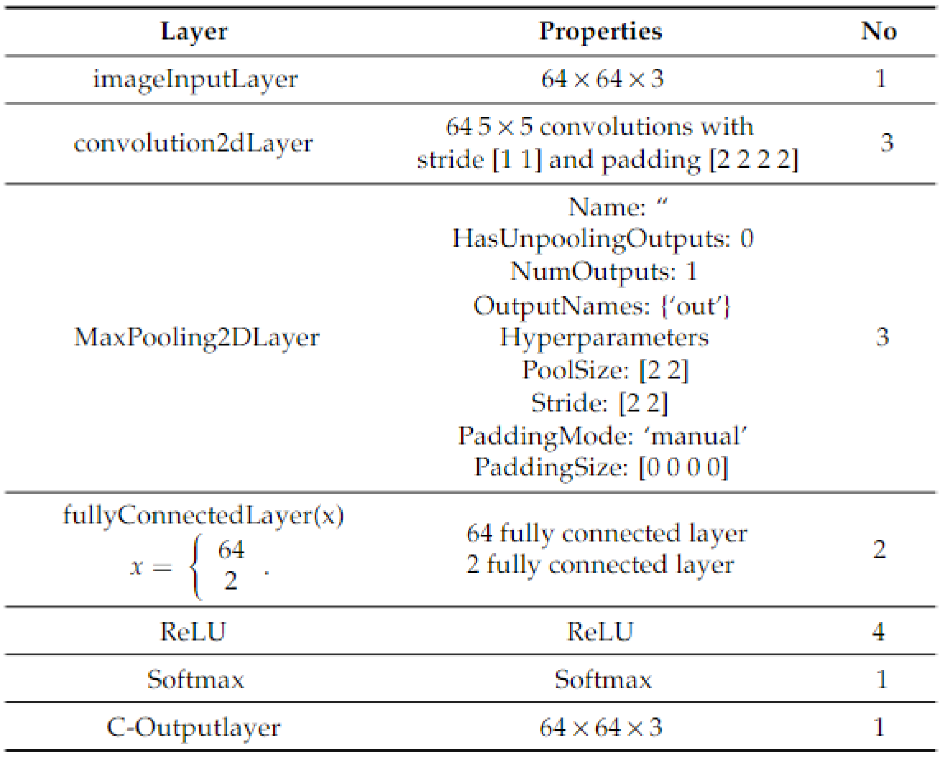


Figure 5: TensorFlow Importation

1)In a convolutional layer, the convolutions are performed using stride Stand S w for the first two axes of the input feature maps, along with Pi+1filters Kh×Kw×Pi[37]:Hi+1="HiKh+1Sh(5)Wi+1=WiKw+1Sw(6)Pi+1(7)(2)In a pooling layer, which occurs following convolutions, the layer chooses pixel valuations of specific characteristics (e.g., average pooling or maximum pooling) within a given region. If a max-pooling layer is chosen, it then carries out maximum element extraction, i.e., stride ShandSw for the initial two axes in a neighborhood Kh×Kw for every two-dimensional piece of input Symmetry 2019,11, 1280 6 of 17the feature map. The input block's maximum value is, therefore, returned [37]. This approach is commonly applied in deep learning networks. In our proposed strategy, the max-pooling layer will decrease the input image patch resolution, as well as enhance network robustness,in the face of possible valuation changes in the motion residuals of the frame's absolute difference image [38].Hi+1="HiKh+1Sh(8)Wi+1=WiKw+1Sw(9)Pi+1=Pi(10)Input image patches for CNN models use two-dimensional array image blocks measuring 3×(64×64), with 3 indicating the channel number in the RGB-scale. Thus, if we use 3×3 as the window size and 3 as the stride size, then the image patch resolution decreases by half to 32×32from its original 64 ×64, following the initial max-pooling layer [37].(3) ReLU layer performs element-wise nonlinear activation. Given a single neuronx, it is transformed into a single neuron y with:y=max(0, x)(11)(4)Softmax layer turns an input feature vector into a vector with the same number of elements summing to 1. Given an input vector x with Pineuronsxji[1, Pi], each input neuron produces a corresponding output neuron:yj=exjPk=Pik=1exk(12)(5)In a fully-connected (FC) layer, dot multiplication is carried out between flattened feature maps(i.e., the input feature vector) and the weight matrix using Pi+1rows, along with columns of Pior (Hi.Wi.Pi) [37]. Meanwhile, the output feature vector presents Pi+1elements [37]. Trained CNNs can also remove meaningful information in images that have not been used to train the network. This particular characteristic enables forgery exposure of previously unidentified images as well [37].(6)In a batch normalization layer, every input channel is normalized in ultra-small (or mini) batches.The batch normalization layer initially normalizes every individual channel's activations by subtracting the mini-batch mean and then dividing the result by the standard deviation of the mini-batch [37]. Next, the input is shifted by the layer using the learnable offset, after which it scales the input using the learnable scale factor[37]. Batch normalization layers can also be used between convolutional and non linearities (e.g., ReLU layers) to increase CNN training and lessen any sensitivities that might arise during the initialization of the networks. Batch normalization can normalize inputs xi through formulating the meanμBand variance 2B for a mini-batch and input channel, after which it formulates the normalized activation's [37]:^xi=xiμBq2B+(13)As can be seen in the expression above, or Epsilon is used to enhance numerical stability if the variance of the mini-batch variance presents as being too small. Furthermore, in cases where the zero mean input and unit variance are not suited to the subsequent batch normalization layer, it is then scaled and shifts its activations as follows:yi=^xi+(14)



Layer	Properties	No
imageInputLayer	$64 \times 64 \times 3$	1
convolution2dLayer	64 5×5 convolutions with stride [1 1] and padding [2 2 2 2]	3
MaxPooling2DLayer	Name: " HasUnpoolingOutputs: 0 NumOutputs: 1 OutputNames: {'out'} Hyperparameters PoolSize: [2 2] Stride: [2 2] PaddingMode: 'manual' PaddingSize: [0 0 0 0]	3
fullyConnectedLayer(x) $x = \left\{ \begin{array}{l} 64 \\ 2 \end{array} \right.$.	64 fully connected layer 2 fully connected layer	2
ReLU	ReLU	4
Softmax	Softmax	1
C-Outputlayer	$64 \times 64 \times 3$	1

Figure 6: TensorFlow Importation

Symmetry 2019,11, 1280 7 of 17Interestingly, the offsetand scale factorproperties appear as learnable properties that can be updated throughout the network training process. At the end of the network training, the batch normalization layer then formulates both the mean and the variance across the entire training set, after which it retains them as properties named Trained Mean or Trained Variance [37]. Then, if the trained network is applied for new image prediction, the layer will utilize the trained mean/variance rather than the mini-batch mean/variance for activation normalization [37].The three main characteristics are representative of CNN models and indicate their potential for image forgery detection. These characteristics are presented below:Convolution operation: This is defined as adding image pixels within local regions, thereby accumulating into large values the duplicate patches in the area. The large-value accumulation could result in easier detection of forged images among pristine ones [39].CNN model convolutional: This is a form of exploitation of any strong spatially local correlations which could occur in input images. Embedded copy-move distorts image pixel local correlation, which then differentiates it from correlations of pristine images via the process of correlation-based alignment.In this way, any distinctions between distorted and natural images are easily perceived through the CNN models [39].Nonlinear mappings: In CNN models, this type of mapping enables them to derive deep and rich features that therefore means they can be used to classify all types of images. Such features are automatically learned via network updates and would be difficult to apply using the traditional non-CNN method [39].The literature, as mentioned above, has introduced different types of algorithms used for image forgery in general and in copy-move forgery detection in particular. However, CNNs are emerging now as a powerful method to do the job. The CNN pipeline begins the extracting process of the features from the image using the different layers and then feeds them into the specific classifier to detect the copy-move forgery if it exists. However, before we go over the different parameters used in this CNN as shown in Table 2, we should first clarify why CNNs are generally a more viable option for this task. The fact that CNNs are a learnable method makes them a better choice overall,as compared to other methods, for achieving the same goal. In the evaluation section, we show the output performance of this algorithm versus the state-of-the-art. Second, the classifier here can work at the feature level as well as the pixel level, which eliminates the challenge of losing pixel interaction if we use a pixel vector. The CNN uses the first convolution layer to down sample the image by adjacent information of the pixels. The convolution is thus a summation of the weight of pixel values in the input image. This is achieved, in the proposed network, by convoluting the input image 64×64 with a5×5 Kernel filter. The operation (using a weight matrix) will produce a new image with a smaller size. Each convolutional layer in the CNN will produce multi convolutions, thus generating a weight tensor according to then number of the convolutions, and in this case, the tensor will be 5×5×n.The first convolution layer in the CNN will give a weight matrix of 64×5×5, which will produce 1600parameters. At the end of the network, we use a prediction layer to support the final classification task.For the last two convolutional layers we padded them with 2, however, the max-pooling layer has a pool size of 3 ×3 and a stride of 2 ×2. Symmetry 2019,11, 1280 8 of 17Table 2. Summary of CNN layers with their chosen parameters.Layer Properties No image Input Layer 64 ×64 ×3 1convolution2d Layer 64 5 ×5 convolutions with stride [1 1] and padding [2 2 2 2] 3MaxPooling2D Laye r Name: “Has Unpooling Outputs: 0 Num Outputs: 1 Output Names: ‘out’Hyper parameters PoolSize: [2 2]Stride: [2 2]Padding Mode: ‘manual’Padding Size: [0 0 0 0]3fully Connected Layer(x)x=(642.64 fully connected layer2 fully connected layer 2ReLU ReLU 4Softmax Softmax 1C-Output layer 64 ×64 ×3 13.1. The Proposed CNN Architecture Recent studies show that CNNs are performing remarkably well in image forgery [29–34].Therefore, in this paper, we propose an end-to-end deep learning CNN to handle and detect copy-move forgery. The proposed CNN includes the following main operational layers: an input layer, convolutional layers, fully connected layers, classification layer, and output layer, with each convolutional layer including different convolutional filters. The main benefit of using CNNs in a copy-move forgery detection model is the strategy's success in feature extraction, which improves the model overall performance. Moreover, improvements in the output results are based on CNN learning skills which can be boosted by increasing the input samples and training cycle. CNNs also lower the cost of detecting copy-move forgery, as compared to the classic method. Finally, a wide range of input images can be used by CNN which, indeed, increases the output accuracy of the model.In this paper, the CNN structure is intended for copy-move forgery detection. To that end,we layered the CNN in a specific sequence such that it could function as a type of feature extraction system that uses filter sets of a certain size. The filters are arranged in parallel to the input image regions, incorporating an area of overlap known as the stride. Every convolutional filter output per convolutional layer stands for a feature map or learned data representation. The subsequent convolutional layers likewise extract features from maps, which were learned from earlier convolutional layers. The proposed CNN will learn how to detect similarities and differences in image features through a number of hidden layers. Each individual hidden layer will enhance the CNN's learning feature ability in order to increase its detection accuracy. Note that, hierarchical feature extractor output is added to an FC to carry out a classification task learning weight, which is first randomly initiated and then learned via a back propagation method [40]. However, the hierarchical convolutional layers create an enormous amount of feature maps, rendering the CNN's impractical from both cost and computational perspectives [41]. The network, shown in Figure 3, applied to the present study, features15 layers in total: one each of input and output classification layers, one SoftMax layer, one max-pooling layer, two average-pooling layers, two FC layers, three convolution layers, and four ReLU layers.

In this study, we propose a simple Convolutional Neural Network (CNN) to classify CIFAR 10 and CIFAR 100 images. Because this tutorial uses the Keras Sequential API, creating and training your model will take just a few lines of code. The main procedures includes importing TensorFlow in GOOGLE COLAB as described in Figure 5. Next step includes Downloading and training the CIFAR10 and Cifar100 dataset The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them. See Fig 7. Next we verify the data To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image: Next step includes creation of convolutional Create the convolutional base The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

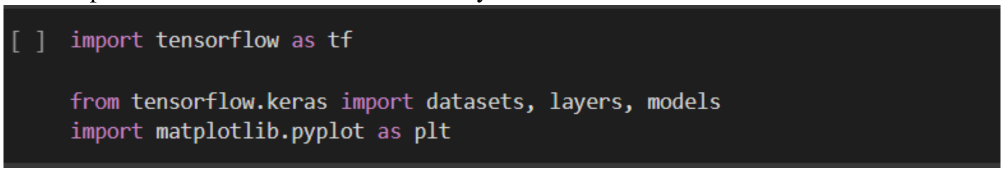
As input, a CNN takes tensors of shape (image$_height, image_width, color_channels), ignoringthebatchsize.Ifyouarenewtothe

Above, you can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 or 64). Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer. Then we will Add Dense layers on top To complete the model, you will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so you use a final Dense layer with 10 outputs.

The network summary shows that (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

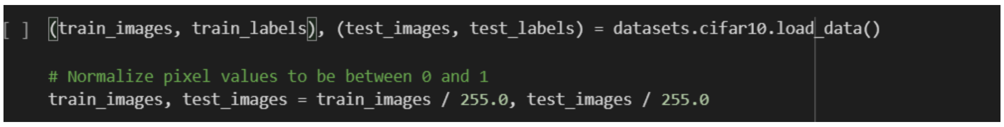
After that we have complied our model using the optimizer "Adam" and Evaluated it Using Epoch to get the accuracy.

Our simple CNN has achieved a test accuracy of over 70



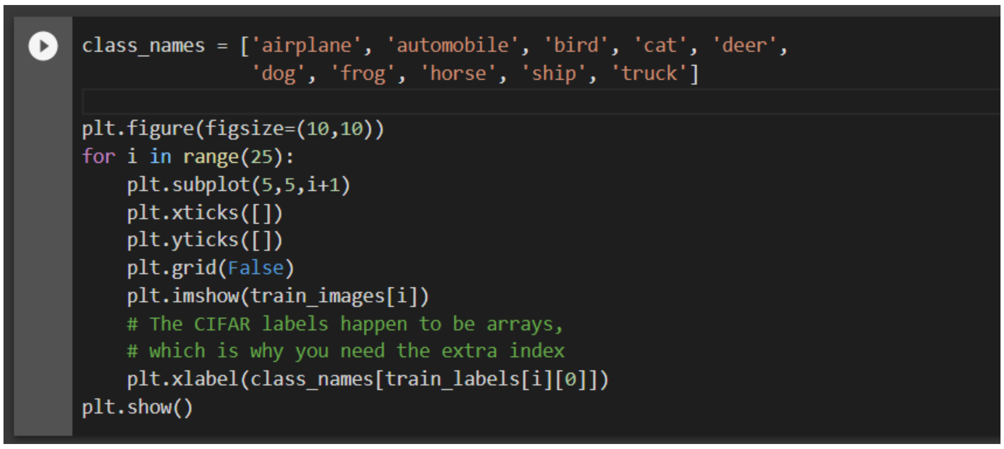
```
[ ]  import tensorflow as tf

     from tensorflow.keras import datasets, layers, models
     import matplotlib.pyplot as plt
```

Figure 7: TensorFlow Importation



```
[ ]  (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

     # Normalize pixel values to be between 0 and 1
     train_images, test_images = train_images / 255.0, test_images / 255.0
```

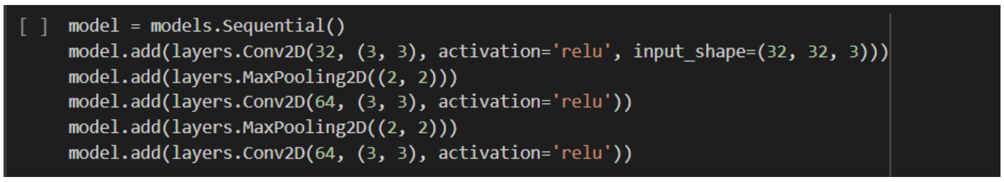
Figure 8: Training the Model



```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

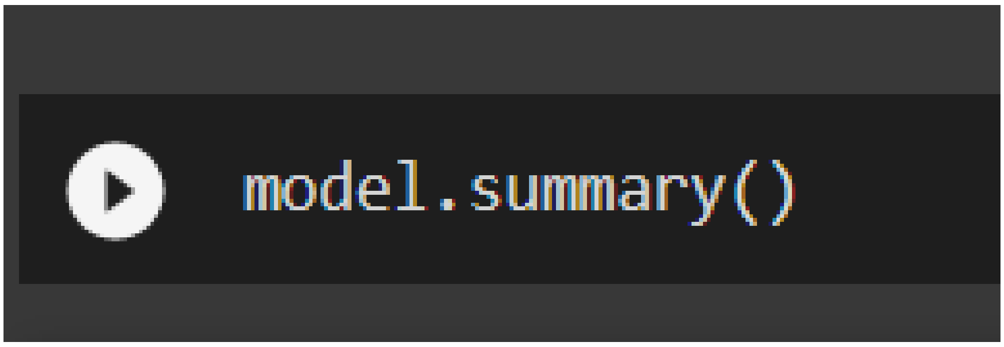
Figure 9: Verifying the classes

]



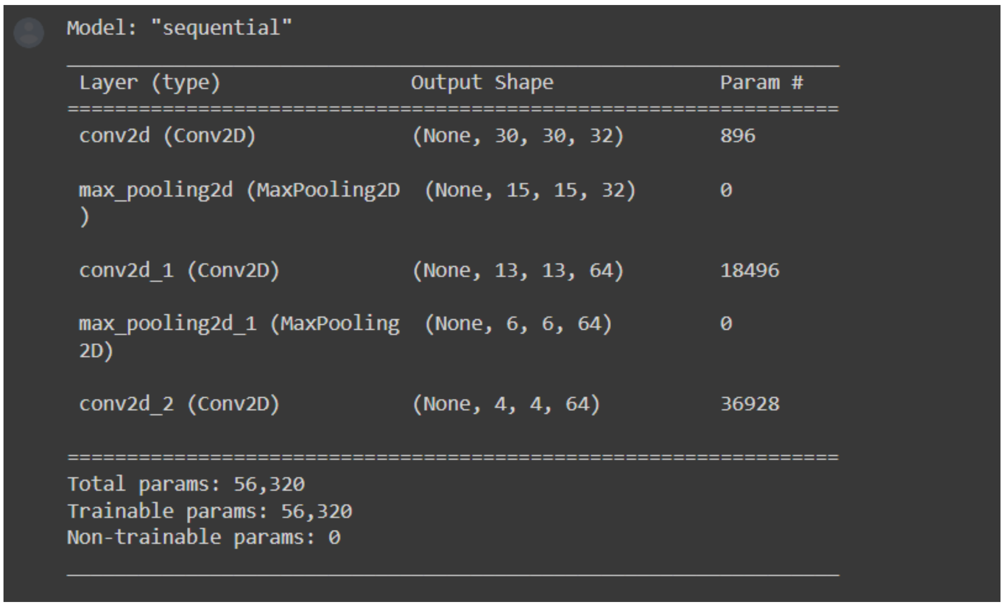
```
[ ]  model = models.Sequential()
     model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Figure 10: TensorFlow Importation



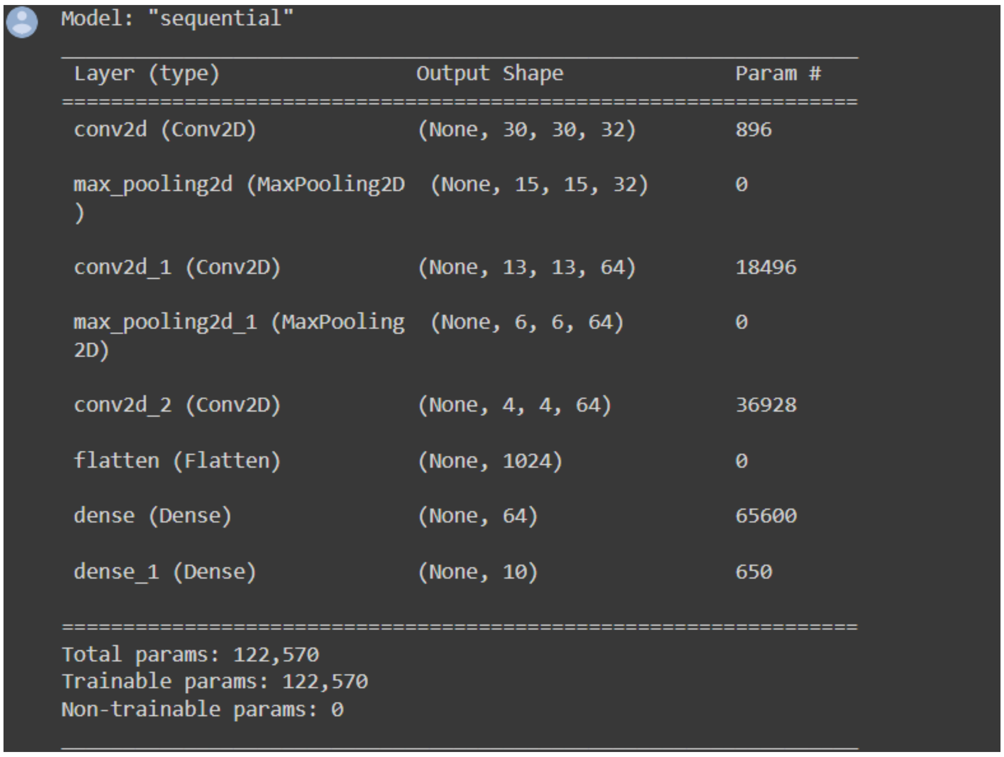
```
model.summary()
```

Figure 11: Summary of the Model



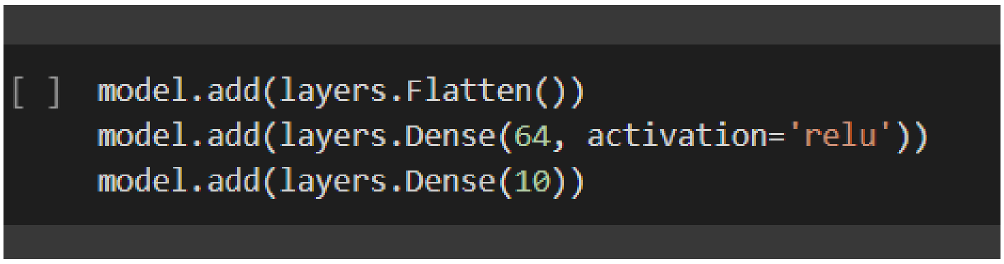
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: Resultant Summary



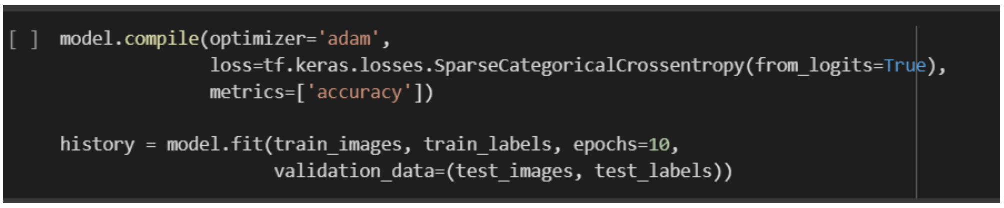
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Resultant Summary 2



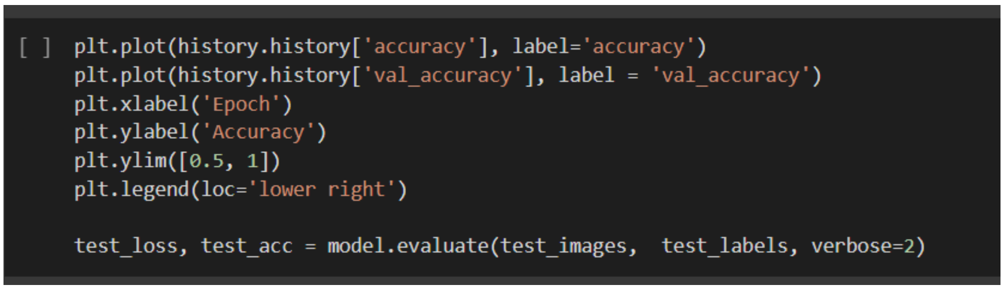
```
[ ]  model.add(layers.Flatten())
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(10))
```

Figure 14: Addition of Dense Layer



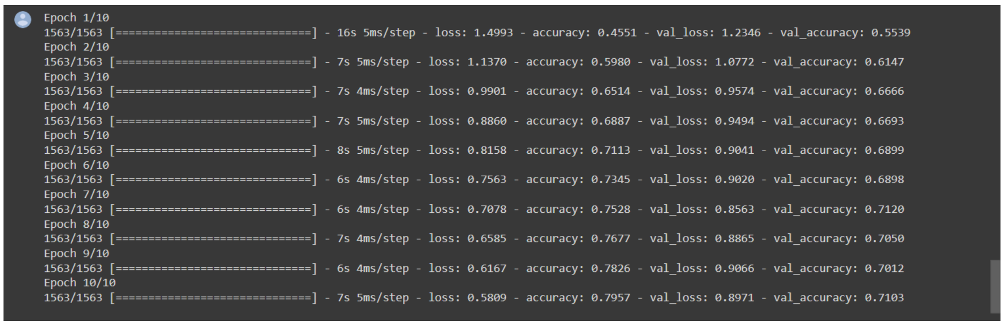
```
[ ]  model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])

     history = model.fit(train_images, train_labels, epochs=10,
                         validation_data=(test_images, test_labels))
```

Figure 15: Compiling the images



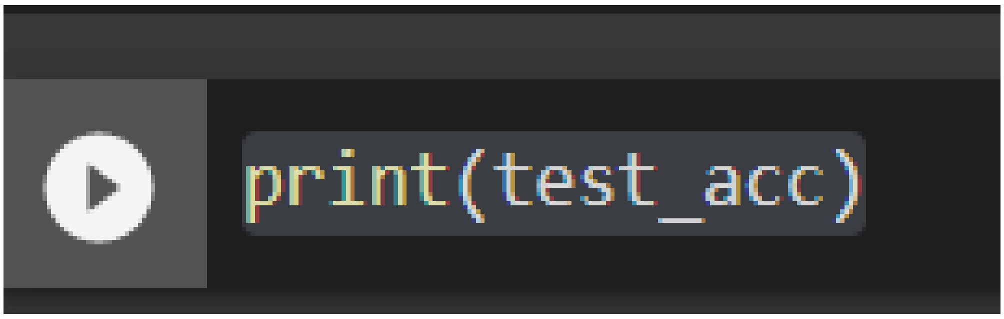
```
[ ]  plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0.5, 1])
     plt.legend(loc='lower right')

     test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

Figure 16: Evaluating our Model



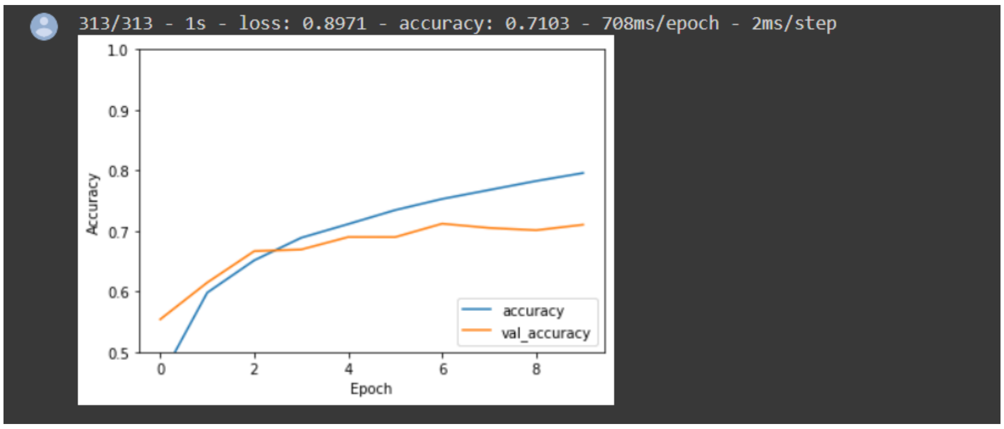
```
Epoch 1/10
1563/1563 [==============================] - 16s 5ms/step - loss: 1.4993 - accuracy: 0.4551 - val_loss: 1.2346 - val_accuracy: 0.5539
Epoch 2/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1370 - accuracy: 0.5980 - val_loss: 1.0772 - val_accuracy: 0.6147
Epoch 3/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9901 - accuracy: 0.6514 - val_loss: 0.9574 - val_accuracy: 0.6666
Epoch 4/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8860 - accuracy: 0.6887 - val_loss: 0.9494 - val_accuracy: 0.6693
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8158 - accuracy: 0.7113 - val_loss: 0.9041 - val_accuracy: 0.6899
Epoch 6/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.7563 - accuracy: 0.7345 - val_loss: 0.9020 - val_accuracy: 0.6898
Epoch 7/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.7078 - accuracy: 0.7528 - val_loss: 0.8563 - val_accuracy: 0.7120
Epoch 8/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.6585 - accuracy: 0.7677 - val_loss: 0.8865 - val_accuracy: 0.7050
Epoch 9/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.6167 - accuracy: 0.7826 - val_loss: 0.9066 - val_accuracy: 0.7012
Epoch 10/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.5809 - accuracy: 0.7957 - val_loss: 0.8971 - val_accuracy: 0.7103
```

Figure 17: Evaluating our Model



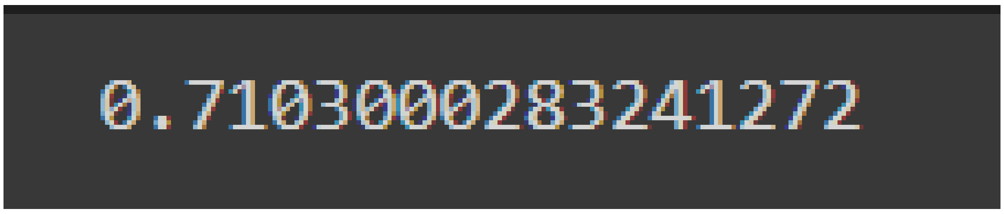
```
print(test_acc)
```

Figure 18: Printing Accuracy



```
313/313 - 1s - loss: 0.8971 - accuracy: 0.7103 - 708ms/epoch - 2ms/step
```

Figure 19: Printing Accuracy

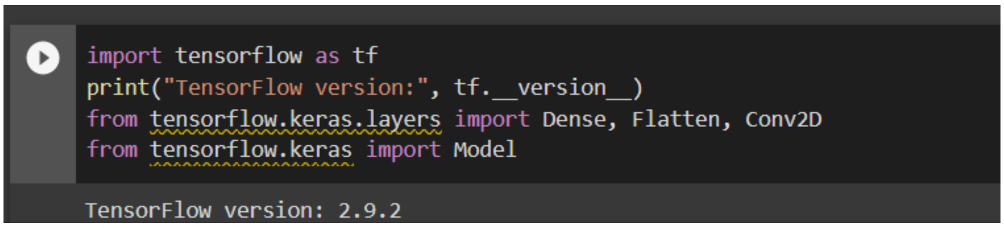


```
0.7103000283241272
```

Figure 20: Printing Accuracy

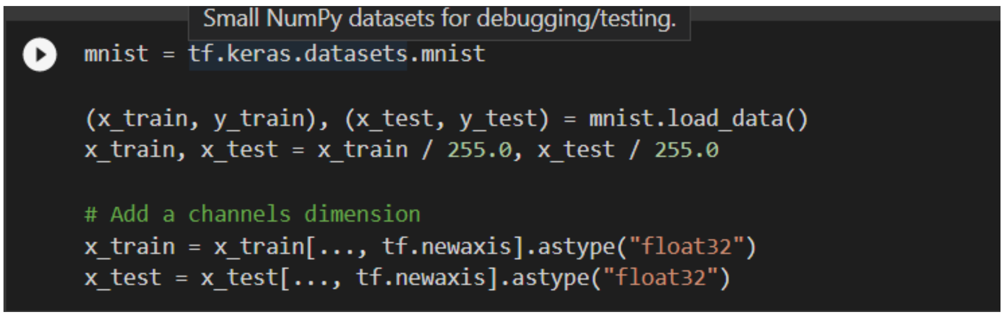
6 MNIST DATASET

To test our Model, we tried to train it with one another Dataset. For this we have used MNIST Dataset. First of all we imported TensorFlow and Keras in our program used Conv2D layer.See fig 21. Then after loading the MNIST Dataset , we trained it using x and y dimensions from tensor slices. see fig 22 Then we built tf.keras using Keras and optimized it using optimizer Adam. see fig 23. After that we tested the model using Epoch to get the accuracy and we managed to get the accuracy of the model of 99



```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model

TensorFlow version: 2.9.2
```

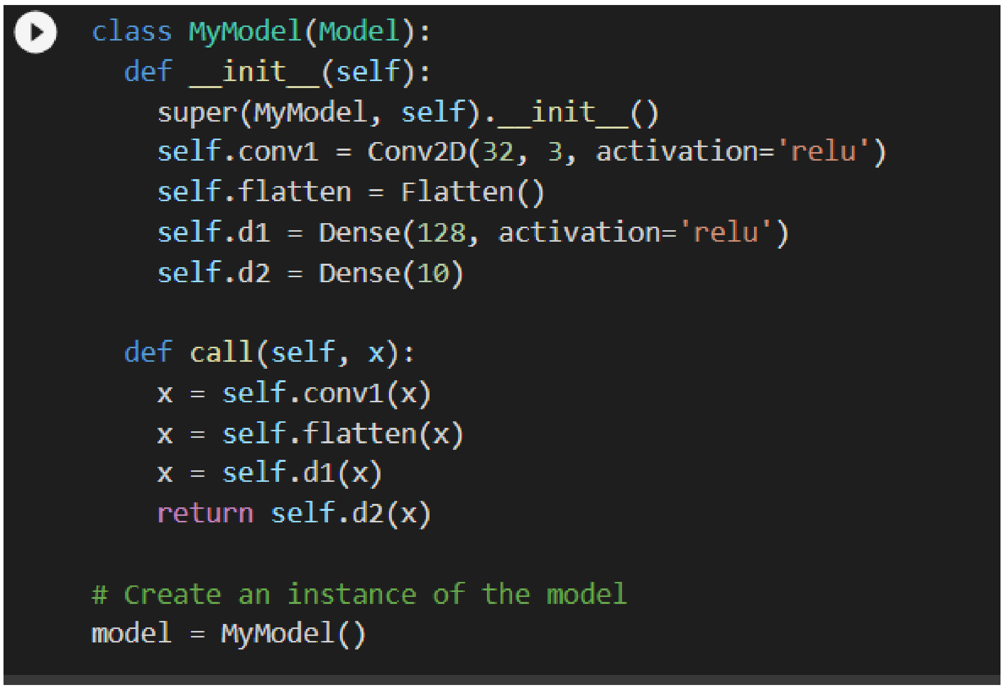
Figure 21: Tensorflow



Small NumPy datasets for debugging/testing.

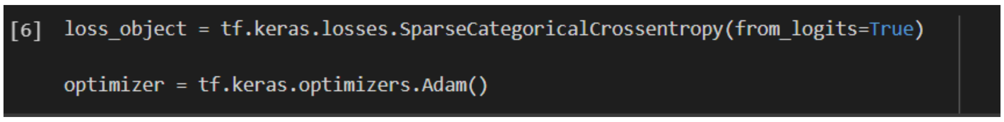
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add a channels dimension
x_train = x_train[..., tf.newaxis].astype("float32")
x_test = x_test[..., tf.newaxis].astype("float32")
```

Figure 22: MNIST



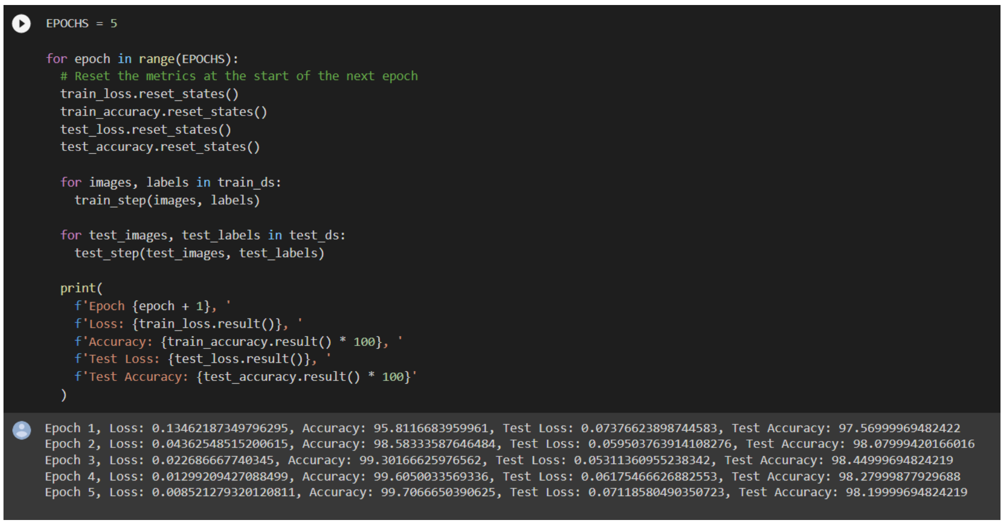
```
class MyModel(Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10)

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)

# Create an instance of the model
model = MyModel()
```

Figure 23: Keras



```
[6] loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

    optimizer = tf.keras.optimizers.Adam()
```

Figure 24: Optimizer



```
EPOCHS = 5

for epoch in range(EPOCHS):
  # Reset the metrics at the start of the next epoch
  train_loss.reset_states()
  train_accuracy.reset_states()
  test_loss.reset_states()
  test_accuracy.reset_states()

  for images, labels in train_ds:
    train_step(images, labels)

  for test_images, test_labels in test_ds:
    test_step(test_images, test_labels)

  print(
    f'Epoch {epoch + 1}, '
    f'Loss: {train_loss.result()}, '
    f'Accuracy: {train_accuracy.result() * 100}, '
    f'Test Loss: {test_loss.result()}, '
    f'Test Accuracy: {test_accuracy.result() * 100}'
  )
```

```
Epoch 1, Loss: 0.13462187349796295, Accuracy: 95.8116683959961, Test Loss: 0.07376623898744583, Test Accuracy: 97.56999969482422
Epoch 2, Loss: 0.04362548515200615, Accuracy: 98.58333587646484, Test Loss: 0.059503763914108276, Test Accuracy: 98.07999420166016
Epoch 3, Loss: 0.022686667740345, Accuracy: 99.30166625976562, Test Loss: 0.05311360955238342, Test Accuracy: 98.44999694824219
Epoch 4, Loss: 0.01299209427088499, Accuracy: 99.6050033569336, Test Loss: 0.06175466626882553, Test Accuracy: 98.27999877929688
Epoch 5, Loss: 0.008521279320120811, Accuracy: 99.7066650390625, Test Loss: 0.07118580490350723, Test Accuracy: 98.19999694824219
```

Figure 25: Final Output

7 Conclusion

In this study, the major strategy of our proposed algorithm focuses on a single tampered region detection. And we have proposed keypoint-based image forensics for copy-move forgery images based on CNN. As previously presented in the experiments, it is clear that the proposed method is highly robust against many kinds of forged images, such as CIFAR 10, CIFAR 100 AND MNIST Datasets. However, the current method is not robust against symmetric, recurring, and smooth patterns for tampering region. Progress in detecting symmetric, recurring, smooth forgery images, and tampering region copied multiple times will be a major focus in the future.