Using Deep Learning Methods to identify tampering of images

Presented By:

Sunil Burman
2K18/EC/177

Tushar Choudhary
2K18/EE/225

Project Overview

Image forgery detection is an active direction of research in the image forensics discipline. Moving a region from one image into another image, combining two images to form one image, or retouching an image are just a few examples of how an image can be doctored. Graphic designing applications like Adobe Photoshop can easily be abused to manipulate images. Furthermore, the advancements in AI like generative adversarial networks (GANs) have made it easier for someone to generate tampered images. These generated images are so flawless that it becomes challenging for even humans to detect a tampered image.

The project revolves around implementing a solution for detecting image tampering by leveraging deep learning methodologies.

Research Trends

Image tampering detection is a subset of digital image forensics divided into active forensics and passive forensics. In most passive forensics algorithms designed to detect image tampering, image features are extracted after image preprocessing, and then support vector machines are used to classify.

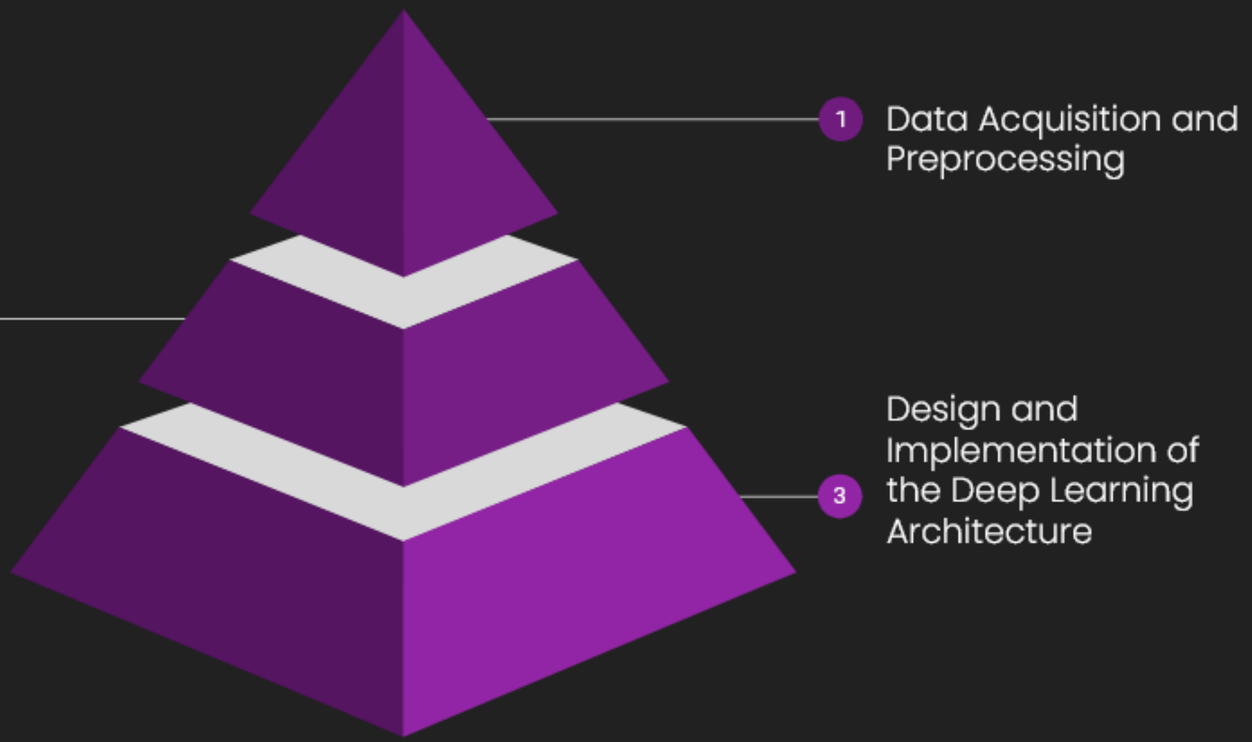
The field of image tampering has advanced a lot when it comes to image forgery detection. Several research papers propose that the YCrCb color space is easier to detect image forgery in, compared to the RGB color space. The developments in deep learning have also fueled the shift from hand-crafed features like DCT correlation to implicit DNN features. Another research trend to note in image forgery detection is that most methods focus on one specific type of forgery.

Research Trends

After the emergence of CNNs, they are the go-to deep learning architectures for computer vision applications. Hence, many CNN-based models have been proposed over the years for image forgery detection. One particular research paper also proposed AlexNet, the advanced CNN architecture capable of achieving high accuracies on very challenging datasets.

But, many other deep lerarning tehniques apart from CNN have been used to detect forgery in images. One of the studies presented a novel approach by proposing a hybrid LSTM and Encoder-Decoder architecture for detecting image forgery. We also came across a research paper that proposed a system that used GAN and one-class SVM to detect image forgeries as anomalies.

Project Workflow



About the Dataset

CASIA V2 is a popular dataset used specifically for image forgery classification. It contains 7491 original images and 5123 tampered images. The dataset contains images in two formats- JPG and TIFF.

We will be using 2000 original images and 2000 tampered images for training our model.

Feature Extraction

We have extracted the GLCM matrix for the edge matrices of the image in Cb and Cr color space separately.

```python
import cv2

def Scharr_Operator(imgYCbCr, threshold=192):
    x = cv2.Scharr(imgYCbCr, cv2.CV_16S, 1, 0)
    y = cv2.Scharr(imgYCbCr, cv2.CV_16S, 0, 1)
    dst = cv2.addWeighted(abs(x), 0.5, abs(y), 0.5, 0)
    dst = np.clip(dst,0,threshold-1)
    return dst
```

```python
from skimage import feature
def GLCM(imgRGB, threshold=192):
  imgYCbCr = imgRGB.convert('YCbCr')
  imgYCbCr = np.array(imgYCbCr)
  Cb = Scharr_Operator(imgYCbCr[:,:,1], threshold)
  Cr = Scharr_Operator(imgYCbCr[:,:,2], threshold)
  Cb_GLCM = feature.texture.greycomatrix(Cb, [1],
                                         [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4],
                                         levels=threshold)[:, :, 0, :]
  Cr_GLCM = feature.texture.greycomatrix(Cr, [1],
                                         [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4],
                                         levels=threshold)[:, :, 0, :]
  GLCM_feature = np.concatenate((Cb_GLCM,Cr_GLCM),axis=2)
  return GLCM_feature
```

Data Processing

Iterating over the whole dataset, we randomly select the extracted features of 2000 original images. Same process is repeated for tampered images.

```
from PIL import Image
import numpy as np
import os
import random

X = []
Y = []

original_images_path = '/content/CASIA2/Au'

for dirname, _, filenames in os.walk(original_images_path):
  for filename in filenames:
    try:
      image_path = os.path.join(dirname, filename)
      imgRGB = Image.open(image_path)
      glcm = GLCM(imgRGB, 192)
      X.append(glcm)
      Y.append(1)
    except:
      continue
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,
                                                    Y,
                                                    test_size = 0.2,
                                                    random_state=5)
print(len(X_train), len(Y_train))
print(len(X_test), len(Y_test))
```

```
3200 3200
800 800
```

Model Architecture

The model contains 5 Conv2d layers and 3 fully connected layers. The kernel size used is (5, 5) in the first Conv2D layer and (3, 3) in other layers.

conv2d_5 (Conv2D)	(None, 192, 192, 64)	12864
conv2d_6 (Conv2D)	(None, 192, 192, 128)	73856
conv2d_7 (Conv2D)	(None, 192, 192, 256)	295168
max_pooling2d_3 (MaxPooling2	(None, 96, 96, 256)	0
dropout_4 (Dropout)	(None, 96, 96, 256)	0
conv2d_8 (Conv2D)	(None, 96, 96, 256)	590080
max_pooling2d_4 (MaxPooling2	(None, 48, 48, 256)	0
dropout_5 (Dropout)	(None, 48, 48, 256)	0
conv2d_9 (Conv2D)	(None, 48, 48, 512)	1180160
max_pooling2d_5 (MaxPooling2	(None, 24, 24, 512)	0
dropout_6 (Dropout)	(None, 24, 24, 512)	0
flatten_1 (Flatten)	(None, 294912)	0
dense_2 (Dense)	(None, 256)	75497728
dropout_7 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 64)	16448
dropout_8 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 2)	130

Hyperparameters

```
epochs = 50
batch_size = 25

init_lr = 1e-4
optimizer = Adam(learning_rate = init_lr, decay = init_lr/epochs)
```

```
model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

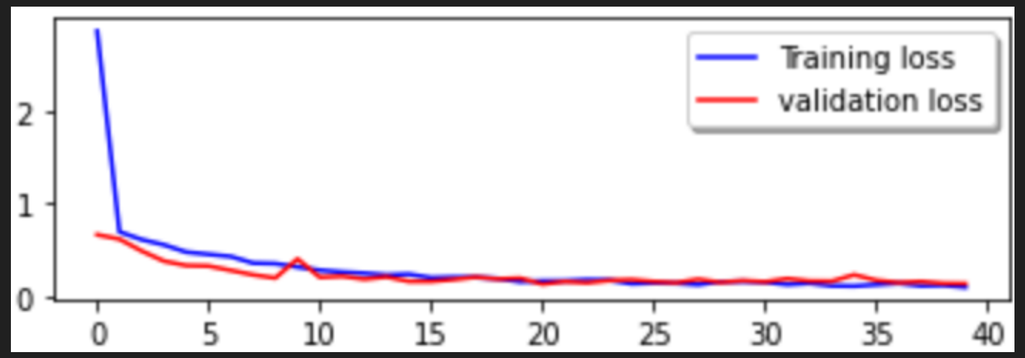
```
early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0,
                               patience = 20,
                               verbose = 0,
                               mode = 'auto')
```

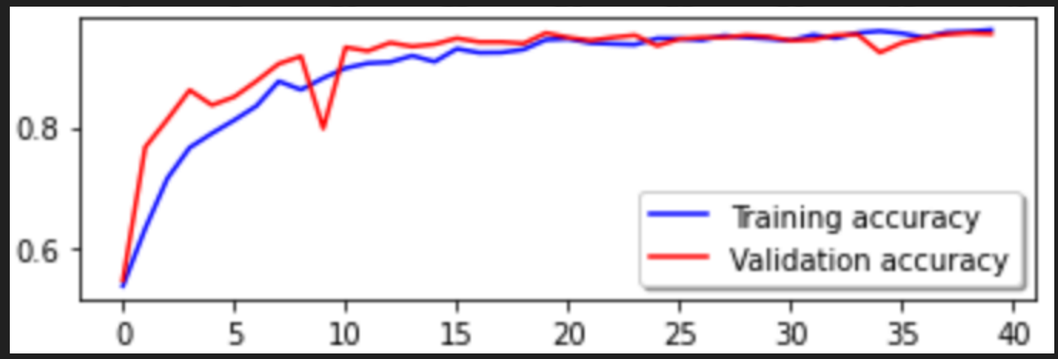
```
hist = model.fit(X_train,
                 Y_train,
                 batch_size = batch_size,
                 epochs = epochs,
                validation_data = (X_test, Y_test),
                callbacks = [early_stopping])
```

Results

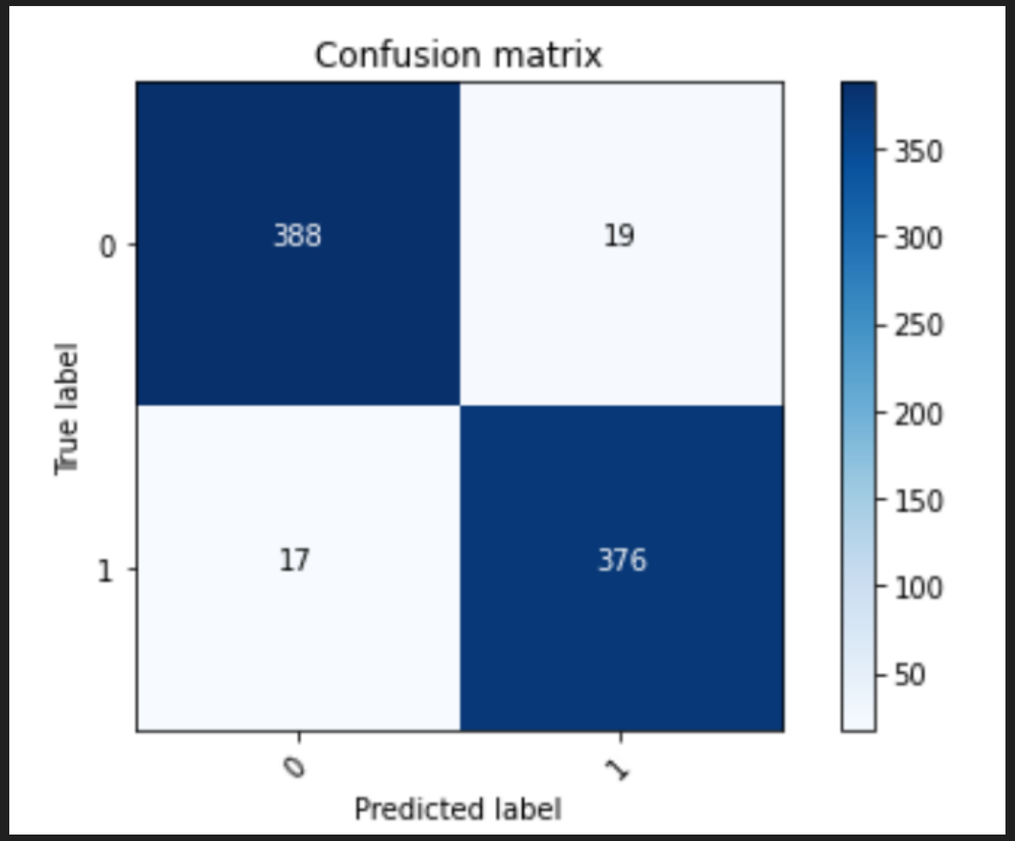
```
from sklearn import metrics

print("The accuracy of the model on the test set is:", metrics.accuracy_score(y_test, predictions))
```

```
The accuracy of the model on the test set is: 0.955
```





Results: Confusion Matrix



Future Direction

Leveraging GLCM for image forgery direction has proved to be phenomenally effective. It is a much more enhanced approach as compared to other approaches in terms of both accuracy and computational expense.

In the future, we would like to explore the effectiveness of the implemented solution for GAN generated images.