Report Forensic - Hades Challenge

Bùi Quang Long

February 2020

Contents

1 Introduction

In this report, I will present my work on Image Forgery Detection.

1.1 The flow

To detect whether an image is tampered or not, I apply a sliding window with stride 16 the image into smaller patches (size of 32x32). Then, using a trained ANN, I can predict if a patch is tampered or not, combined with the predicted result of its neighbors I can calculate its reliability rate. If its reliability rate exceed a definite threshold, this patch is consider a real tampered patch. A tampered image is an image having at least **one** tampered patch.

In order to train an ANN, I need to generate tampered and non-tampered patches as input training data, which lead to the section below.

1.2 Some definitions

Let us define some terms for easier understanding:

- **A positive patch**: A tampered patch, whose predicted result after applying ANN is 1.
- **A negative patch**: A non-tampered patch, whose predicted result after applying ANN is 0.

1.3 Organization

I divided it into 4 parts: *Data pre-processing*, *Training phase*, *Testing phase* and *Demo part.*

- **Data pre-processing**: Generate data for training ANN. This part will generate a mix set of positive and negative patches of size 32x32, then extract theirs features and end up with a matrix of shape $(n\_sample, n\_feature)$ and a label vector of shape $(n\_samples,)$. In this assignment, this work is written in *dataHandler.py* and *featureExtractor.py.*
- **Training phase**: Train the ANN and save the model. The input is a list of patches and the output is corresponding labels 0 or 1, standing for negative or positive patch respectively. In this assignment, this work is written in *training_phase.py.*
- **Testing phase**: Predict a list of images haven't been seen by the ANN. Then some metrics are applied to evaluate the model. In this assignment, this work is written in *testing_phase.py.*
- **Demo part**: We can test some images via this part. Given a list of images, *demo.py* will point out its prediction for each image.

  INSERT FLOW

1.4 Some definitions

2 Data pre-processing

Our dataset (Casia2) has 7491 authentic images and 4481 out of 13443 tampered images that have been labeled and pointed out the edges of the tampered area. Thus, in this assignment, let us only focus on all authentic ones and 4481 preprocessed tampered ones. I divided:

- **Training phase**: 7000 authentic images and 4000 tampered images.
- **Testing phase**: 491 authentic images and 481 tampered images.

2.1 Data generation

This process can be called "Patch generation" as we are generating positive and negative patches. With each type of patch, we have different approach. In this assignment, I take 10000 patches each type.

2.1.1 Negative patches generation

Due to the fact that the large number of negative patches, we just need to randomly choose some patch in each authentic image.

2.1.2 Positive patches generation

Positive patches are carefully selected.

- First, we resize a tampered images into a specific size, which will help us to use a sliding window easier.
- Second, we apply a sliding window of size 32x32 to select a patch. We define a good positive patch is a tampered patch whose tampered area and non-tampered area can be clearly discriminated. Patches with prominent tampered or non-tampered area will affect our model. Thus, the rate of tampered area must be in some particular range. In this assignment, I set it to $[0.25, 0.75]$. In conclusion, a positive patch is chosen if:

  $$0.25 <= tampered\_rate <= 0.75$$

  To calculate $tampered\_rate$, we can use data from folder "edges", this should be equal to the portion of white area divided by the size of the patch.

2.2 Feature extraction

Having a list of selected patches, I apply some feature extracting steps on a patch:

- **Converting it from RBG channel to YCrCb channel**
- **Drop the Y channel**: Due to the fact that values in the Y channel of patches are approximate each other, so they are not discriminative enough to help us in classification.
- **Using Daubechies Wavelet transform**: By applying 5 level-3 Daubechies Wavelet transforms (db1 to db5) on each channel, we obtain 100 matrices (2 channels x 5 transform x 10 result matrices). The feature vector of a patch will have values of the sum, mean and variance of these matrices. This work ends up with a vector of size 300.

2.3 Saving pre-processed data

After collecting and extracting feature of 20000 patches, we obtain a training set of shape $(20000, 300)$. In this assignment, I saved it to a file called "*train.csv*" in folder "*preprocessed_data*". When beginning the training phase, we just need to import data from here, which saves us a bunch of time.

3 Training phase

Now let us import pre-processed data first. Before training, we need to normalize the data:

$$X_{train} = \frac{X_{train} - X_{mean}}{X_{var}}$$

Futhermore, mean and variance vectors will be save to normalize data in testing phase and demo part.
The next step is to build an ANN for classification, then train it and save it for testing phase. More detail will be presented in "Result" section.

4 Testing phase

In this phase, we will test nearly 1000 images (491 authentic ones and 481 tampered ones). With each image, following steps are applied:

- A sliding window with size 32x32 and stride 16 is applied to select patches. With a particular patch, some feature extracting process is applied:
  - Converting to YCrCb channel.
  - Dropping Y channel
  - Applying Daubechies Wavelet transform
  - Normalizing using saved mean and variance from training phase
  - Applying trained ANN to predict whether this patch is tampered or non-tempered.

  The whole process will result in a matrix with values are prediction for each patch. This matrix will be delivered to post-processing process below.

- Post-processing: In this process, we will evaluate if a predicted tampered patch is truly tampered or not via its neighbors. Each value in the predicted matrix above has value of 0 or 1, standing for non-tampered or tampered prediction. Let us consider a particular position (x, y) in that matrix. This position will have at most 8 neighbors. Assume that (x, y) has k neighbors. Let us define a_i as the predicted value for i^{th} neighbor, for $i = 1..k$, and a_0 as the predicted value for patch (x, y). The reliability rate can be calculate as:

$$R = \frac{\sum_{i=0}^{k=8} a_i}{k+1}$$

If this rate exceeds a threshold $\alpha (0 < \alpha < 1)$, we consider this patch is truly tampered. An image is considered tampered if there is at least one patch is truly tampered.

5 Results

5.1 Using Multi-layered perceptron with sklearn

I use sklearn to build a simple MLPClassifier with 5 hidden layers (450, 300, 200, 150, 100), 'adam' solver and 'relu' activation for hidden layers. The accuracy of MLP only reaches 89%. Applying this model to testing phase, the accuracy score is only 80%. When I print out the accuracy for each type of image, it turned our that the model mostly mispredicts the non-tampered images.

5.2 Using keras to build ANN

I used keras to build a more complex ANN. Please check it here
https://colab.research.google.com/drive/1w85yye8m-HkAJpmIaRH8fdrOKbzgHQHz