Machine Learning II_DATS_6203_11
Dr. Amir Jafari

ML2 - Individual Project Report

Photoshop Detector

Poornima Joshi

Abstract

The aim of this project was to use Neural Networks to determine whether a photo has been altered by photoshop. During the course of this project, we have attempted to explore several different types of neural networks (like Dilated Residual Networks, ResNets, vgg's) and frameworks (pytorch, keras, tensorflow). In addition we also experimented with fastai library is order to speed up the training. The main focus throughout has been to achieve better accuracy in identifying photoshopped images. We attempted to isolate the faces in the images in order to reduce the noise in the background. This research is part of a broader effort to improve digital image forensics; a field dedicated to detecting the authenticity of images.

Introduction

Earlier this year Adobe and UC Berkley published a paper in which they proposed a Neural Network architecture to detect photoshopped images. Previous research on image manipulation detection involved detecting changes in original images that were obtained from techniques like splicing, cloning, and removal, whereas Adobe's effort focuses on the Face Aware Liquify feature in Photoshop because it's popular for adjusting facial features, including making adjustments to facial expressions with user-friendly operations like "'increase nose width and decrease eye distance". They also mention that while GAN based methods to generate false images like DeepFakes have garnered a lot of attention, most fraudulent images that practitioners in digital image forensics deal with are relatively small manipulations created from tools like Adobe Photoshop. They used a special type of CNN (see Network) to solve the problem, and were able to achieve 90% accuracy on their test set, 50 original images and edits to them made by a professional artist. Although that is commendable, the small size of the test set and the fact that their accuracy was driven by their ability to predict original images (96% compared to 84% for photoshopped images) leaves room for improvement and further validation. Therefore, in our project we would like to validate the model that they published to github and use other networks / frameworks to attempt to perform better than their model on a dataset we selected.

Data

We aim to use a dataset from Kaggle that is created by the Computational Intelligence and Photography Lab, Yonsei University. Here is a link to the dataset. https://www.kaggle.com/ciplab/real-and-fake-face-detection. The dataset contains expert-generated high-quality photoshopped face images. The photoshopped images are composites of different faces, separated by eyes, nose, mouth, or whole face.

Dataset Size

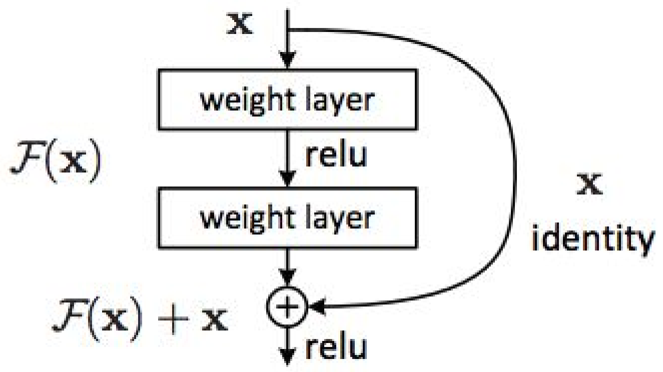
Inside the parent directory, training_real/training_fake contains real/fake face photos, respectively. Is it a large enough dataset since there are 1081 real and 960 photoshopped images. The photographs have a 2000Mb high quality images sized at 600x600x3

Outline of the shared work

The first approach in our project was that Sean would try to replicate the Adobe's paper, I would try to explore ResNet and Chirag would explore VGG16. We decide to work on these networks individually and finally compare the results. Once we had a few models that performed reasonably well, we wanted to ensemble them. In the end, Chirag implements the data preprocessing techniques, Sean implemented the Adobe's paper, which was our baseline. I implemented the ResNet and MobileNetV2. I came up with the idea of using FastAI for all other models and implemented it using ResNet. Chirag expanded on the different FastAI networks. And finally Sean performed the ensembling.

Work that I did on the project

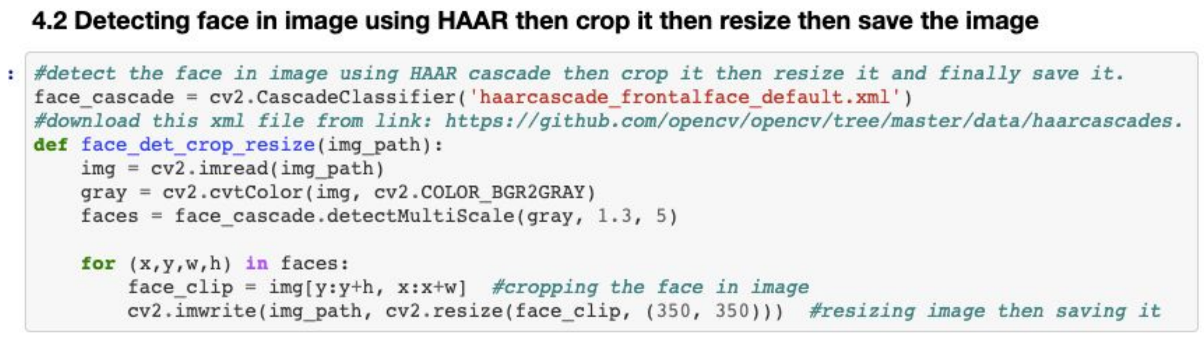
In the very beginning of the project, I organised the github and co-wrote the proposal. Then, I figured a way to load the dataset from Kaggle. I used the blog "How to Train an Image Classifier in PyTorch and used it to Perform Basic Inference on Single Images" as a reference to load my dataset [1]. As we progressed through the project, I had a ResNet model, which was working but did not have a great accuracy in the first week. Since this was the first attempt, I tried to implement the ResNet18 using pytorch with randomly initializing the weights. The network did not perform very well. I went ahead to use the pretrained weights. I still ended only with 56% accuracy. This was used from the bentrevett github page [2]. I learnt much more about the working of ResNets from the paper and the blog "Introduction to ResNets" [5]. A residual neural network (ResNet) is a type of neural network that skips connections, or short-cuts to jump over some layers. Typical ResNet models are implemented with double or triple layer skips that contain ReLU and batch normalization in between them. Models with several parallel skips are referred to as DenseNets. Diagram of the residual block is shown below. Finally, after playing around with image sizes, batch size and epochs, I could achieve 59% accuracy.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

At this stage, I was wondering what kind of data augmentation and preprocessing we could do. Our 2 classes were quite balanced too. So I went ahead to explore fastai tutorials. I developed an understanding of how I can use fastai for classification problems and how I can choose the split of the learning rate in order to make fastai work for us. Since we did not include this code in our project github, I have attached it in my personal github [3]. We realised that, fastai was helping us get good results, Chirag started using it together in order to improve our accuracy.

After further brainstorming, we decided to try out other models. InceptionNet and Alexnet which were my next obvious choices. Since both of these yield 52% and 51% each, we decided to ignore them. DenseNet did not perform very well. However, mobilenetv2 was really fast. I wanted to know why mobilenets were so fast, I started reading the paper on “MobileNetV2: Inverted Residuals and Linear Bottlenecks” [4]. I understood that the MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. This architecture helped us reach upto 60% accuracy. Feeling quite pleased, we went ahead to think about data preprocessing.

During a discussion, we realised that the key features in identifying the images is by zooming into the faces. This would help us to eliminate all the noise. Here is the code we used initially to identify faces from the ”Real Time Facial Expression Recognition” post [5] intially. Further Chirag found the MTCNN which gave us the better results, since it could even identify the different features in a face.


4.2 Detecting face in image using HAAR then crop it then resize then save the image

```
#detect the face in image using HAAR cascade then crop it then resize it and finally save it.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#download this xml file from link: https://github.com/opencv/opencv/tree/master/data/haarcascades.
def face_det_crop_resize(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        face_clip = img[y:y+h, x:x+w]  #cropping the face in image
        cv2.imwrite(img_path, cv2.resize(face_clip, (350, 350)))  #resizing image then saving it
```

Finally, we ensembled all the models we build by exporting the pickle files of our predictions.

Results

Below is the table of results of different models all of us tried together. It can be observed that the baseline model performs the best individually. However, on ensembling the models, we could achieve an accuracy of 70%.

Network	Epochs	Batch Size	Accuracy
ResNet18	10	64	62.53%
FastAI-ResNet152	n/a	n/a	Out of Memory

FastAI-resnet50	14	64	63%
Baseline	5	17	67%
MobileNetV2	50	8	60%
FastAI-VGG16	15	64	64%

Summary and conclusions

The biggest takeaway from this project is that, just changing the model does not help significantly. Therefore it is important to understand the data and utilize other techniques in order to perform better. In our case, cropping out background and focussing on the face helped. Also, when we looked carefully into the dataset, it was difficult to see if an image is photoshopped even for human eye. Therefore it may have been hard to identify those.

Finally, I learnt to use different frameworks and a way to compare the results of different models and frameworks. Ensembling was a good learning exercise too specially to maintain consistency.

Percentage of the code

- Percentage of the code that you found or copied from the internet : 370 + 304
- Code from the internet I modified : 35 + 23
- Code from the internet I added : 11 + 62

Total = 674 - 58 / 674 + 73 = 616/747 *100 = 82.2%

References

1. https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5
2. https://github.com/bentrevett/pytorch-image-classification
3. https://github.com/poornimajoshi/jupyter/blob/master/fake_and_real.ipynb
4. https://arxiv.org/abs/1801.04381
5. https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4
6. https://medium.com/datadriveninvestor/real-time-facial-expression-recognition-f860dacfeb6a