Machine Learning II_DATS_6203_11
Dr. Amir Jafari

ML2 – Final Project

Photoshop Detection

Sean Pili

Individual Report

Machine Learning II_DATS_6203_11
Dr. Amir Jafari

Introduction

For my project, I tried to extend the work of Adobe's research team;**might wanna add those sparkly citations….** evaluating a model that they trained to detect photoshops generated by a tool that alters the features of an authentic image of a face on a dataset containing copy-paste photoshops of different facial features on authentic faces, and trying to build a network (or ensemble of networks) that improve upon the original model's accuracy. More specifically, Adobe and UC Berkeley used a DRN-C-26 to achieve 90% accuracy on a held out set of 50 photoshopped images created by using Adobe's own Face Aware Liquify tool, which is often used for 'beautification' transformations Since the network's accuracy on fake faces was 84%, I believed that there was definitely room to improve upon Adobe's work. Ultimately **we were able to beat/not beat the baseline model by doing blah blah blah…**

I had come up with the idea for the project before I found a team, so I found our team's dataset and decided that I would be one to train the baseline model since I had spent the most time familiarizing myself with Adobe's Github repository. Poornima, Chirag and I each researched and implemented data augmentation methods, but Chirag was the one to implement the data augmentation that was most beneficial to all of us. In terms of the rest of the responsibilities, Chirag wrote code that ensured we loaded the test set in the same order and that our model predictions were in the correct order before we tried ensembling them. All three of us trained 5+ model architectures, and I think it would be fair to say we put the same amount of work into modeling. However, Poornima and Chirag experimented with fastai whereas I only used pytorch. Poornima wrote code that allowed for useful metric printing while training models in Pytorch. The fact I did not use fastai could be the reason none of the models I create could compete with the baseline, but Poornima and Chirag's models could. I.e. the only model I trained that you will see in the results slide is the baseline. Poornima essentially created all of the slides for the presentation herself and all three of use put in an equal amount of work to create the final report. I wrote all of the model ensembling code myself.

Individual work

As mentioned previously, I created our research question, found our dataset and trained our baseline model. To train their model, I used a pytorch subclass that adobe created for the DRN-C-26, loaded the pretrained weights they provided me and added 2 linear layers, the first of which outputted 100 neurons and the second outputted 2, the number of class labels. When training, I experimented with ReLU, PReLU, CELU and SELU as transfer functions between the linear layers of the baseline model, and tried applying different dropout rates between them. I used NLLLoss as my performance index, so I applied LogSoftmax to the output of my model. The last parameters I changed in the model were the type of optimizer and the learning rate. For each experiment, I loaded the images as 500x500x3 and used a batch size of 17. I used the former size because it yielded higher accuracy, and I used the latter size because that was the largest size I could use when training it on AWS with that image size. Before using the data augmentation from mtcnn to crop faces, I could not best 55% accuracy on the held out set. I achieved that result by using the Adam optimizer and a learning rate of .000002. Typically, the validation loss would fluctuate wildly, but oddly, most times the test accuracy would improve after 5 epochs. After using the mtcnn library to crop faces, I achieved my best test set accuracy, 67.77%, by using adadelta with the default parameters, and relu as the transfer function.

After training the baseline, I tried training other models to compete with it, but did not have much luck. The models I tried were, GoogleNet (max test accuracy 52%), AlexNet (max test accuracy 51%), shufflenet (accuracy 55%) and mnasnet (max test accuracy 53%) Each group members decided to try different architectures to make the work faster. In addition to running networks individually, I trained mnasnet and shufflenet in parallel, concatenated their predictions and used a linear layer to predict the ensembled result. Below is an example of the code.

```
sFN = models.shufflenet_v2_x1_5(pretrained=False)
sFN.fc =
nn.Sequential(nn.Linear(1024,1000,bias=True),nn.ReLU(),nn.Dropout(.4),nn.Linear(1000,
2))
mna = models.mnasnet1_0(pretrained=True)
mna.classifier =  nn.Sequential(nn.Dropout(p=0.2,
inplace=True),nn.Linear(in_features=1280, out_features=1000,
bias=True),nn.ReLU(),nn.Dropout(.35),nn.Linear(1000,2))
class MyEnsemble(nn.Module):
    def __init__(self,mna,sFN):
        super(MyEnsemble,self).__init__()
        self.mna = mna
        self.sFN = sFN
        #self.vg16 = vg16
        self.classifier = nn.Linear(4,2)
        self.ls = nn.LogSoftmax(dim=1)
    def forward(self,x1,x2):
        x1 = self.mna(x1)
        x2 = self.sFN(x2)
        #x3 = self.vg16(x3)
        x = torch.cat((x1,x2),dim =1)
        x = self.classifier(torch.relu(x))
        x = self.ls(x)
        return x
```

That, or training VGG16 and shufflenet in parallel did not improve any of the model performances. That was not the only kind of model ensembling I tried, as I wrote all of the model ensembling code used by my team, which included a hard-voting classifier and a simple soft-voting classifier that voted using the average of the softmax outputs. The hard-voting classifier beat the baseline by 1% I also used an ensemble classifier using scikit learn, and was able to use it to beat the baseline by two percentage points.

I also studied data augmentation techniques, and although chirag was the one to actually implement the mtcnn library to crop faces, I was the one who told him to use that method after I had read about it. During my search I tried using the haarcascade method from opencv and a python package named autocrop, but they did not work as well as mtcnn did in that it recognized more faces in our training data and got higher quality crops.

304-44 / 34+26+68+40+300 =55.5% of my code was found or copied from the internet. This is mostly because the architecture of the DRN-C-26 is quite verbose.

References

1. https://arxiv.org/pdf/1705.09914.pdf
2. https://peterwang512.github.io/FALdetector/
3. https://www.kaggle.com/ciplab/real-and-fake-face-detection
4. https://arxiv.org/pdf/1512.03385.pdf
5. https://en.wikipedia.org/wiki/Residual_neural_network
6. https://docs.fast.ai/
7. https://arxiv.org/abs/1801.04381
8. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html
9. https://neurohive.io/en/popular-networks/vgg16/
10. https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803
11. https://github.com/ipazc/mtcnn