ML2 - Final Project

Photoshop Detector

Sean Pili
Chirag Jhamb
Poornima Joshi

Abstract

The aim of this project is 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 etc,) 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[1] *"Detecting Photoshopped Faces by Scripting Photoshop"* 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". One could consider most of these alterations made with Adobe's Face Aware Liquify tool 'beautifications,' are usually quite subtle. They also mention that while using 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 made to authentic images with tools like Adobe Photoshop. Adobe used a DRN-C-26 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 attempted to validate the model that they published to github[2] and used other networks / frameworks to attempt to perform better than their model on a dataset we selected. Ultimately, we were able to beat Adobe's baseline by a negligible .2% (67.77% to 67.97%) by ensembling the logits of a VGG16 and ResNet50 via a soft voting ensemble classifier that combined random forest, logistic regression, and adaBoost models.

Data

We used a dataset from Kaggle that is created by the Computational Intelligence and Photography Lab, Yonsei University[3].The dataset contains expert-generated high-quality photoshopped face images. The photoshopped images are composites

of different faces. The features of each original face that can be swapped out with features from another face (or faces) are the eyes, nose and mouth, either individually, or in any combination of those items.

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

Experimental Setup

Dataset Preprocessing

Cropping images: Getting the faces and cropping was done using MTCNN[3], a face detector package written in tensorflow 1.x. It gives a box of x, y, width and height values, which we can use to crop the face and ignore the rest of the image. In a lot of cases the chin area of the face was cropped out as well, so we did +15 to the height to adjust for that.

Original Image vs Cropped Image

Subsetting dataset: For fake image, we had easy, medium and hard images based on how easy it is for humans to detect the photoshop editing performed. In order to avoid getting just one category in test and getting a very good or bad accuracy, we took 20% of each difficulty level of fake images and 20% of real images for the test set.

Hard *Medium* *Easy*

Photoshop levels

Batching/Resizing

Batch sizes and image sizes across experiments differed, as different members of the group were using different cloud providers, and networks with varying levels of computational complexity. Image sizing was used as a hyperparameter to tune for the experiments, as we were not sure if there was an optimal size that would make computation faster but would not reduce accuracy.

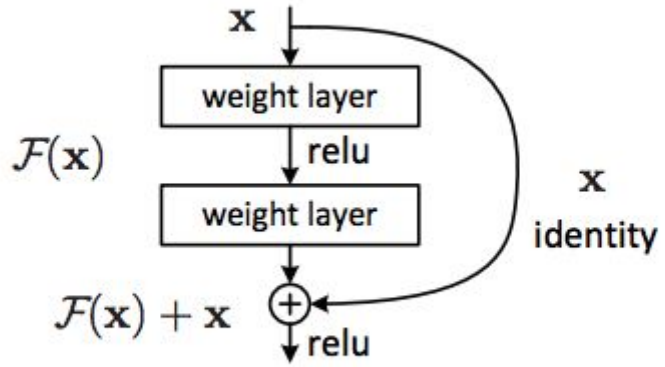
Training Parameters

The learning rate was changed in 2 ways. 1st, for those team member(s) that used fastai, it automatically optimized over the learning rate. The team member(s) that did not use fastai tried the following approaches to determine the learning rate: trial and error, as well as manual learning rate decay. I.e. they would multiply the learning rate by a fraction after every epoch. One group member also used trial and error to find the best optimizer to train the baseline model, and even tried switching from faster converging optimizers like adam and adadelta to SGD after 5-10 epochs.

Network Architectures

ResNets18 and ResNets50

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.

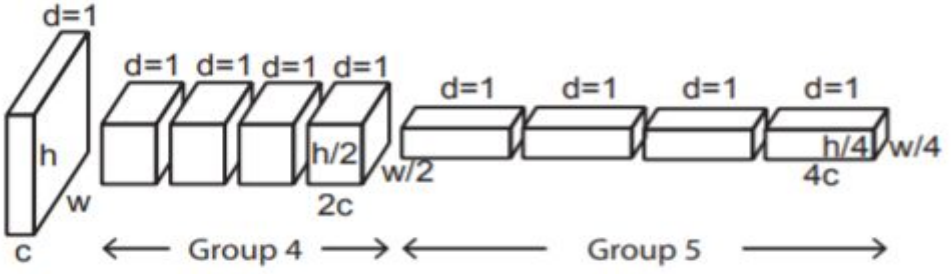


Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover.

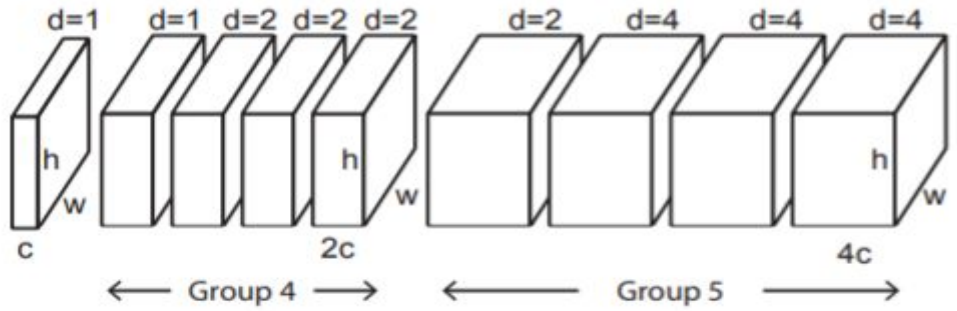
Dilated Residual Networks

The key idea in a DRN (Dilated Residual Network) is to preserve spatial resolution in convolutional networks for image classification, and they achieve this by a technique named dilluted convolutions, which sets DRN's apart from ResNets. The formula for dilated convolution applied to groups of layers in a residual network is shown below where each group of layers is written as G^l and the ith layer in that group is G_i^l which filter f_i^l is convolved over: $(G_i^l * f_i^l)(p) = \Sigma_{a+(d*b)=p} G_i^l(a) f_i^l(b)$. Note, the domain of p is the feature map in G_i^l. One can think of it as adding an extra kind of striding, between each point in a kernel. For example, a 3x3 kernel with a dilation of 2 touch the 4 corners of a standard 5x5 kernel, and touch the first, third and fifth pixel in the third row. In this way, although one is still technically using a 3x3 kernel, they are capturing a wider scope of information. Although progressive downsampling has been observed to be very successful in classifying digits or iconic views of objects, the loss of spatial information may be harmful for classifying natural images and can significantly hamper transfer to other tasks that involve spatially detailed image understanding. This is the case in our context since we deal with human face images. Natural images like human images often feature many objects whose identities and relative configurations are important for understanding the scene. The classification task becomes difficult when a key object is not spatially dominant. What's worse, if the object's signal is lost due to downsampling, there is little hope to recover it during training. However, if we retain high spatial resolution throughout the model and provide output signals that densely cover the input field, backpropagation can learn

to preserve important information about smaller and less salient objects. Therefore, it made sense to us that it DRNS may be better than ResNets at detecting copy-paste photoshops, because it is easier to determine whether someone's nose has been pasted on someone else's face if more information about the face surrounding the nose is available. Using this idea, we implemented the adobe's approach to identify fake / photoshopped images. The results of this attempt are discussed in the below section.



(a) ResNet



(b) DRN

DRN-C-26

A DRN-C-26 is a modified version of ResNet 18. Resnet18 has 5 groups of layers, which is extended to 8 in the DRN-C-26. Both Netowrks output with 16 7x7 featuremaps in their first layers. However, ResNet18 uses Maxpooling before moving to its second group of layers and the DRN-C-26 does not use maxpooling. It instead continues its first layer by adding 2 layers that both output 16 3x3 featuremaps with dilations of 1 (normal convolution.) It then adds the output of the initial layer to the net input of the last aforementioned layer, and feeds that into the first layer in its second group of layers, each of which output 32 3x3 featuremaps, which also includes a shortcut. Then, the architectures of both networks are the same for the next two groups. They each feed to a group of 4 layers which output 64 3x3 featuremaps and contains a shourtcut with a dilation of 1 , then feed into another group of four layers

that output 128 3x3 featuremaps with dilations of 1 and contain feedbacks. After this, both networks feed into a group of four layers that output 256 3x3 featuremaps that contains a shortcut, but the DRN-C-26 uses a dilation of 2. Then, the DRN-C-26 doubles its dilation to 4, whereas Resnet18 still uses a dilation of 1 for its next and final group of four layers which output 512 3x3 featuremaps and contains a shortcut (which would then connect to a linear layer for classification purposes.) The DRN-C-26 then feeds into its 7th group of 2 layers that output 52 3x3 featuremaps and contains a shortcut and uses a dilation of 2, and then feeds into its 8th group of layers which also output 512 3x3 featuremaps, but only uses a dilation of 1 (and then would feed into a linear layer for classification.) To implement it for the project, we wrote the aforementioned architecture in pytorch with subclassing and used the pretrained weights provided on Adobe's github. We then added a series of linear layers as follows: we attached the output of the last convolutional layer to a linear layer that outputs 100 neurons, applied the relu transfer function, then applied dropout with a probability of .5 before adding another linear layer that outputted 2 neurons and applied the LogSoftmax transformation as we used NLLLoss. To train the network, we experimented with using the following optimizers, SGD, Adamax, Adadelta, Adam and AdamW. The only setting we modified besides the learning rate was momentum and weight decay when applicable, though weight decay always seemed to yield worse results. Although we did not write an early stopping algorithm, we never tested models that reduced test loss for more than 3 epochs in a row to combat overfitting. We achieved the best results by running adadelta for 5 epochs with Pytorch's default parameters. We tried using SGD after the first 5 epochs, but we could not lower the accuracy on our validation set. The results on the test set are shown in the results section.

fastai

Fast.ai is synonymous to transfer learning and achieving great results in a short amount of time. The fastai library simplifies training fast and accurate neural nets using modern best practices. It's based on research in to deep learning best practices undertaken at fast.ai, including "out of the box" support for vision, text, tabular, and collab (collaborative filtering) models.

Vgg16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

MobileNetV2

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. MobileNetV2 builds upon the ideas from MobileNetV1, using depthwise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture:
1) linear bottlenecks between the layers, and
2) shortcut connections between the bottlenecks
The intuition is that the bottlenecks encode the model's intermediate inputs and outputs while the inner layer encapsulates the model's ability to transform from lower-level concepts such as pixels to higher level descriptors such as image categories. Finally, as with traditional residual connections, shortcuts enable faster training and better accuracy.

Results

Below is a table of results that were obtained from various networks we attempted to use during this project. It is evident that baseline model from Adobe's paper using DRN performed the best. All other models are also in the range of 60%-65%.

Network	Epochs	Batch Size	Accuracy	Fake Accuracy	Real Accuracy

ResNet18	10	64	62.53%	72%	52%
FastAI-ResNet152	n/a	n/a	Out of Memory	N/A	N/A
FastAI-resnet50	14	64	63%	58%	65%
Baseline	5	17	67%	66%	69%
MobileNetV2	50	8	60%	58%	62%
FastAI-VGG16	15	64	64%	61%	65%

Ensembling of Models

Hard-Voting ensembling and a variety of ensembling methods were used on the top 4 performing non-baseline models in an attempt to beat its accuracy. Using hard-voting on VGG16 and both ResNets was able to beat the baseline model with a hard voting accuracy of 68%, excluding VGG16 from the top four models yielded a hard voting accuracy of 67%, excluding resnet 50 yielded a hard voting accuracy of 66% and excluding resnet18 from the top four models yielded a hard voting accuracy of 65%. Hard voting was not applied to all four models simultaneously because a tie-breaking method would need to be applied, which would reduce the reliability of results.

Soft-Voting ensembling was also employed by averaging the top 4 final model's probabilities for each sample in the test set. Since it is almost impossible for the average of 4 model probabilities to be equal to each other for each class, a simple average was taken of the model probabilities for each of the top four models for each sample in the test set, and it was assigned to the label with the highest averaged probability. As suspected, none of the averaged probabilities generated from the top four models on the test set were exactly .5 for either class, so a random-number generator or a set decision (anything with .5 is assigned to be a photoshop) would not be necessary to break ties as it would be in hard-voting.

Conclusion

In conclusion, we can say that individually the models listed in the results table did not perform better than our baseline (Adobe's DRN). However, they performed much better while ensembling using randomforest. Therefore, ensembling was the key to improved performance. The overall accuracy of our model is **67.97**%The key takeaways are, cropping photos and performing data augmentation helped boost the models the most. Hyperparameter tuning significantly changed the performance of

the models both individually and while ensembling. FastAI enabled to quickly prototype while having reasonable results.

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