**Practicing with various CNN structures**

In the following section we discuss some different approaches that we took to address the product authentication task. Although our data set included images from ten different features, we only focused on one feature, main\_logo. The reason was that according to our authenticator the main factor to decide their fake/real status is the quality or shape of the fonts and since more or less the same applies to two other features made\_in\_logo and brand\_code if a design works for one of them it’ll work for two others as well. Here, I report the results according to two different CNN structures: 1) Siamese Network 2) A network inspired by InceptionResNet

**Some experiment on data preprocessing**

Figure 1 shows six sample images from the main\_logo feature category of our data set. According to one of our authenticators the most important factor in deciding about their genuinity is the font used to print the logo. However, each image contains lots of fine details other than mere printing. Including color, material texture, lighting, etc. If the main factor to decide about the genuinity of a product based on its main\_logo is the font scripted then there is no reason for CNN to deal with these fine details. Also, the situation might get worse when these irrelevant factors are implemented very well in super fake products. In this case these factors weigh in and the result would be unpleasant. So, if the goal is to judge only based on the quality of printing and the font used then one way could be to somehow extract only the logo via preprocessing.

****

**Figure 1**

The image processing that we employed is a simple edge detection algorithm implemented in OpenCV. We used the Scharr edge extractor that is similar to Sobel but is believed to perform slightly better. As a result, images shown in Figure 2 are the kind of images that we are sending to CNN. Another motivation for this kind of preprocessing was to be consistent with some *signature authentication* models that we found in the literature so hopefully we can reproduce their result. As can be seen in Figure 2 these resulting images are more similar to a hand drawn signature.



Figure 2

**Other Dataset Manipulation**

As is shown in Figure 2, there are different styles of bags in our dataset and each comes with its specific logo printing. For this experiment we only chose one style. The one on top right of Figure one. The reason for picking this style for now was:

1. They all have the same font and the logo is inscripted by pressing. Also they all have close background color so we can be sure that the CNN will focus on the same features.
2. They are printed on a least textured material compared to other styles and thus CNN can mostly focus on printing.
3. Also they are the most frequent styles in our dataset, so provide us with more data points.

As a result our new dataset has the following statistics:

|  | Train | Validation | Test |
| --- | --- | --- | --- |
| Real | 773 | 121 | 133 |
| Fake | 258 | 51 | 60 |
| Total | 1031 | 172 | 193 |

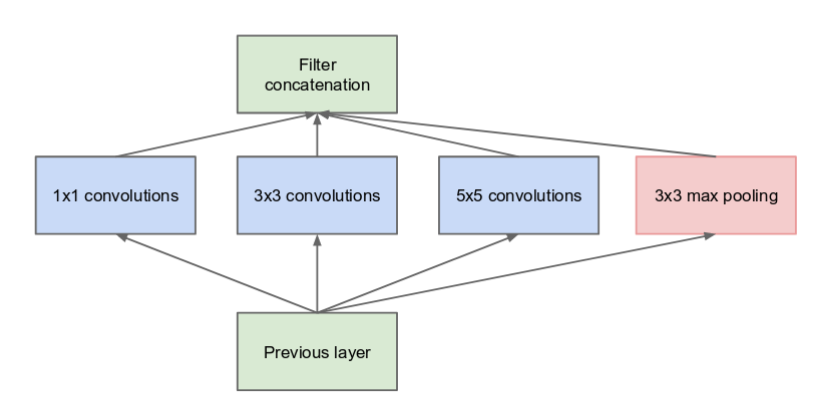
**Table 1**

Finally, we used data augmentation to address the imbalance that is present in our data set. As is clear from the above table, the majority of our data are from genuine bags. In order to address this problem we took two approaches. One was to use a loss function that is weighted toward one class and the second one was to use classic data augmentation but only on fake products.

**Network Design**

1. **InceptionResnet Inspired Network**

One network design that we used was a small network based on inception neural network. One block of this network is shown in Figure 3. The idea is to extend the network horizontally and make wider networks instead of putting layers on top of each other and going deeper and deeper.

****

**Figure 3**

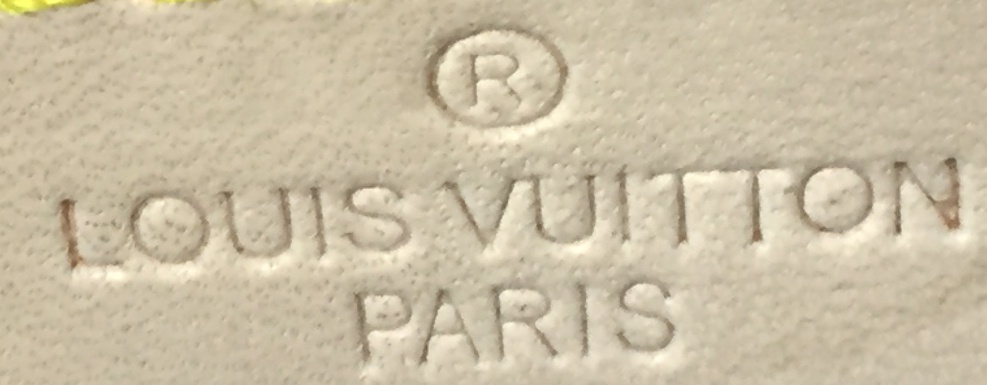
Our designed network consists of two such blocks. The only difference is that we use 2x2 filters instead of 1x1 convolution filters. For the loss function, we used NLLLoss in pytorch so we can easily set the weight for each of two classes which is reasonable for our dataset since we are dealing with high imbalance in data.

1. **Siamese Network**

Another network design that we used was Siamese network with contrastive loss. Siamese networks are very popular in face detection tasks where there are a few sample images of each person available and the model is expected to detect a desired person by confronting a few shots and sometimes only one shot. The standard approach in these networks is that the network is trained on triplets that the first two elements are images of people and the third one is label, normally 0 or 1 where 0 means the first two pictures belong to different people and 1 means they are photos of the same person. Figure 4 shows three of such triplets.

 1

Real Real

 1

Fake Fake

 0

Real Fake

The idea in leveraging this method was that if two images from some ROI are real or both are fake then they must carry enough information that a siamese network can differentiate between real or fake ROI provided that we facilitate it with enough images.

**Result**

Table 2 shows the results for both architecture:

|  | TP | FP | TN | FN |
| --- | --- | --- | --- | --- |
| Inception Net | 60 | 127 | 6 | 0 |
| Siamese | 1 | 132 | 1 | 59 |

**Table 2**

As is clear from the above table, while none of these results are reliable, the Siamese network is performing very badly. Apparently, these images are so similar that a Siamese network cannot mark them as different entities.

**Some Discussion and Suggestions**

The model that is developed by the group has acceptable accuracy but it also shows overfitting. Normally, when it happens the first suspicion is the complexity of the model. In order to test that I implemented various simpler architectures, including a simple ResNet network with only one skipping layer, as well as some other sequential network and the result was not good at least in accuracy compared to the original model implemented by Mihai, Claudiu and Jay. Also, it’s been said that overfitting might happen as a result of fine tuning a pre-trained weight of a large dataset over a much smaller one. In order to investigate this issue I tried ResNet18 both with pre-trained weight and from scratch and there was no improvement on overfitting. In fact the pretrained model performed slightly better.

**Fine-Grained Image Classification**

Fine-Grained image classification aims to recognize images belonging to multiple sub-categories of the same category. It is a difficult problem because the model should discriminate based on subtle and local variances in the images. As mentioned above, one challenge that we are facing with our data set is that the images of various features in our data set are too close and similar and thus it is difficult for neural networks to differentiate the fine details in fake and real images. Hence, employing a Fine-Grained neural network architecture might be helpful in this case. In recent years, many Fine-Grained models emerged in the computer vision community that try to address this problem. We investigated a few of them and decided to give a chance to one of them known as [API-Net](https://github.com/PeiqinZhuang/API-Net).

**Results for API-Net**

We trained API-Net on four features, *LV\_main\_logo, LV\_made\_in\_logo, LV\_main\_closure\_hardware, and LV\_zipperpull.* However, while we used the original dataset for the last three one*,* for *LV\_main\_logo* we used the same setting for the data set as we used in the Siamese network, i.e. separating the khaki images and only training on those images. From Table 3 you can see that it does not bring any advantage to API-Net.

The specificity and sensitivity scores are available in Table 3.

|  | main\_logo | made\_in\_logo | closure\_hw | zipper\_pull |
| --- | --- | --- | --- | --- |
| Specificity | .45 | .46 | .49 | .54 |
| Sensitivity | .47 | .50 | .49 | .52 |