Capstone Project Report

06.08.2019

Kolja Maier

Definition

Project Overview

Neural Networks are inspired by biology - actual neurons did not only influence the name of 'neural networks' but also the theoretical foundations of deep learning in terms of neurons activation principles. Also advanced neural-network architectures, like CNNs (Convolutional Neural Networks), have resemblance to the human brain. Experiments even showed that activation patterns in the human cortex correlate with areas of a CNN when solving image classification tasks [https://arxiv.org/ftp/arxiv/papers/1608/1608.03425.pdf https://mindmodeling.org/cogsci2016/papers/0435/paper0435.pdf].

The roots of CNNs, as we know them today, are dated back to around 1990 when the famous scientist Yann LeCun did his pioneer work in that field. The foundations of Deep Learning are even older, reaching back to the 1950s. Hence the theory behind neural networks, deep learning and CNN is well understood.

But can we really be so certain about how CNNs learn?

In recent years criticism raised for using Neural Networks, because they rather behave as a black box in giving predictions for new data. More recently, several methods (for instance LIME https://homes.cs.washington.edu/~marcotcr/blog/lime/) have been developed to bring more transparency into the world of deep learning algorithms. For these reasons it is very important for every Deep Learning application to get a better understanding of their functionality and their usage.

In this project we will focus on some aspects in that domain:

a. how shape and texture of objects (like dogs) influence a CNNs training/learning and inference process
b. how & if it is possible to perform better from these insights (a.)

This project is based around the paper "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." [1], which claims that there is a general bias in CNNs towards learning the texture of objects, instead of the shape.

Particularly I will focus on the domain of dog-pictures, since the fur of dogs seems to hold lots of texture information (long, short, straight, curly,...), which seems to be promising for this use case. This projects domain can also be seen in Data augmentation, since the manipulation of source images and style-transferring their textures leaves us with more input data that is augmented.

Problem statement

[1] showed, that humans naturally recognize objects primarily by their shape and not their texture. [1] also showed, that CNNs actually have a bias towards learning the texture representation instead of the shape representation of an object. This can be problematic since lots of information sits inside the actual shape of an object. Particularly for object recognition (with the special case of image classification) shape is an very important feature. In this project we focus on the problem of image classification with CNNs (of dog pictures and their races) with the focus on data augmentation regarding shape and texture information. The assumption is, that I can increase the accuracy of the CNN model by providing more information on the shape of an object instead of its texture.

Especially in the domain of dog pictures this looks promising, since dogs from different breeds can have a similar fur-texture, but a very distinguished shape (see Figure 1).

(a) (b)

Figure 1. Images of (a) a Chihuahua and (b) a Doberman. Although their fur has very similar texture (short, straight and slightly shiny hair), their body shape is very different

Hence the concrete problem to solve is to write a CNN image classification application (for dog breeds) that reduces the inherent texture bias of CNNs and instead focuses on learning shape representation with shape information.

A big part of this project is the implementation of this application with a modern technology stack, so that the project is scalable and maintainable in the future. This aspect is important, as it one of the core aspects of Machine Learning Engineering. Concretely this will mainly achieved by using AWS SageMaker and PyTorch.

Metrics

Since we are facing a classification problem a valid approach for insights of the quality of a model is the accuracy metric which is defined as:

- $$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

In fact this is also exactly the metric we want to pay attention to in terms of our problem & solution statement, which should answer the questions:

- Is there an inherent texture bias in our benchmark model solution?
- Will the augmentation and new feature-attention (on shape) give better results?

We want to pay particular attention for a proper engineering process that separates the data into

- train set
- test set
- validation set

With this kind of data separation it is guaranteed that the model will only learn the representation from training data and is validated (accuracy) on the exclusively set of validation data.

As we learned in the project about fraud detection, it is very important to check the balance of class proportions in our training data. If it is not balanced enough, accuracy is not a great measure and should be replaced with recall for example. As we will see later on in the Analysis section our dataset is quite balanced so that the accuracy metric seems to be a good fit.

Analysis

Data Exploration & Visualization

Note: The images in this report can be replicated by executing the corresponding notebook in my repository.

The dataset I use will be the dogs images set, that we know from our Notebook exercise. Since it is quite large I will provide the link to it instead of a whole copy. This dataset holds

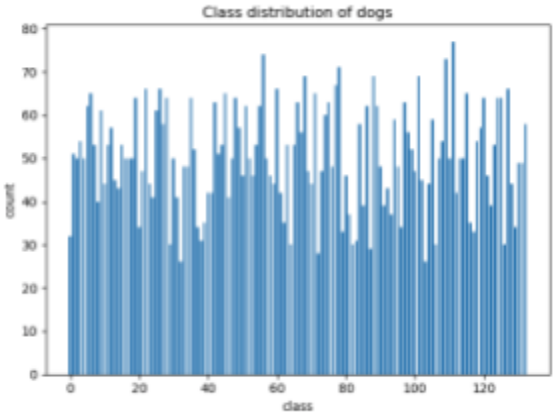
various pictures of dogs and their corresponding race-labels - in total there are 133 different breeds of dogs. It holds image data that suits our problem definition from above.

One very important aspect is the class balance in a dataset, which also influences the choice of our Metric (see below). Hence we take a quick look at the class distributions of our training data:

```
import os
import matplotlib.pyplot as plt
sizes = []
sum = 0
for root, dirs, files in os.walk(mypath):
    sizes.append(len(files))
    sum += len(files)

sizes = sizes[1:] # drop the root directory size as it holds no images
print(sum)

plt.bar(range(len(sizes)), sizes, 1/1.5)
plt.title('Class distribution of dogs')
plt.xlabel('class')
plt.ylabel('count')
plt.show()
```



We see that our class distribution is quite balanced. Some classes are more present than others, but in general the distribution is uniformly shaped.

In total there are 6680 pictures of dogs in the training set across the various breed classes. For the test and validation set there are 836 and 835 pictures respectively.

To get a better understanding of the actual image data, we sample a few random pictures of one dog breed class:






As you can see from the image, the dogs have no 'standardized pose' neither is the environment fixed. Also the pictures can have different sizes.

While sampling and displaying a random batch of training data can be helpful it is always recommended, to manually go through a part of the data to get a better understanding. By this I found some interesting pictures:




As we see from these pictures above, the dataset is very versatile - even for one class of dog breed! This can be manifested in different backgrounds, multiple dogs on one picture, different angles & sizes of dogs or other objects that are very present (like persons, mail containers, carpets,...). Under these circumstances it does not seem so easy for a CNN to learn a good representation of 'dogs'.

Algorithms & Techniques

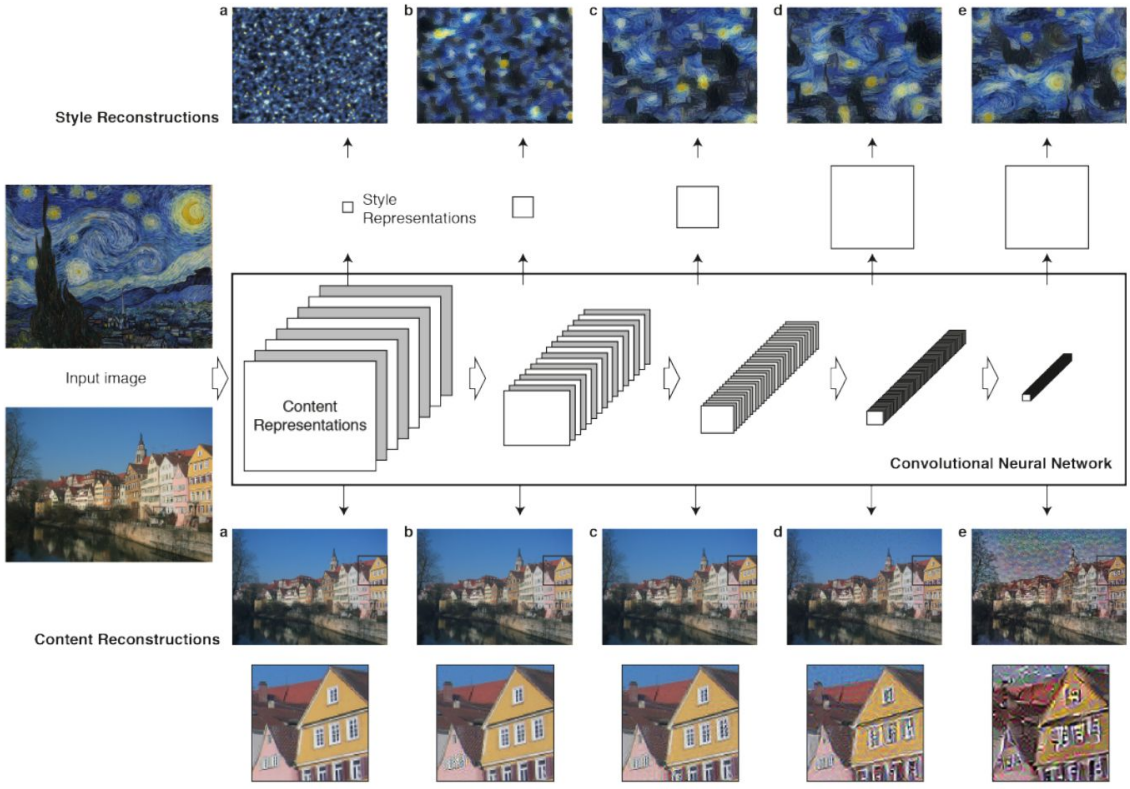
Style transfer & Image Augmentation

Image augmentation is the process of manipulating the raw input data and enrich it with new information. A classical example is the rotation of images - with this augmentation technique the model (i.e. a CNN) is forced to learn a more general representation of an object. Whether a new test images comes in 10 degree or -10 degree rotated does not matter anymore. The CNN is rotation-invariant. This project is built around a very special augmentation technique: Style transfer. The aim is to enrich raw images with different styles (texture images). This is, of course, best explained by an example (taken from [1]):



The input image is style transformed by a texture image - elephant skin in this case. The result is the same picture with different texture styles applied.

The above process uses a technique called AdaIN style transfer. This algorithm uses a complex architecture:



Key point is, that the lower level reconstruction representations/visualizations (see a,b,c,d,e in the above image). The trick then is to minimise the loss function between content and style representation during the image synthesis process. The result is a new synthesized image that has content and style features.

Further information can be found at [https://arxiv.org/abs/1508.06576]. There is also a great open source implementation of AdaIn https://github.com/naoto0804/pytorch-AdaIN for PyTorch, that was referenced in this project.

Transfer Learning

The core idea of transfer learning is to use so called pretrained models. These models can be complex CNN architectures, that consist of millions of parameters. Usually they were trained on a large corpus of data - so large that it took weeks to train them.

After training, the weights of the CNN are stored and can be re-used in the same CNN architecture. The initialization of such a CNN-architecture just loads the saved model weights and hence utilizes the trained model, without the actual training taking place.

While this seems to be a miracle cure, we need to remind ourselves, that the parameters were trained on a specific data set. In case of the VGG16 architecture it was trained on the popular dataset ImageNet. This dataset includes a lot of categories like birds, humans and food. In total it holds more than 20.000 categories!

While this dataset also includes pictures of the category dog, it was not trained on our very custom data set - that is the different dog breeds. That is why we need to retrain the last layer of VGG16 to adapt the parameters to our needs.

Benchmark

The benchmark model will be the CNN we were instructed to implement during the dog-classification notebook in the courses material.

This model leverages some advanced techniques like transfer learning and hence is a quite ambitious baseline. It is based on the VGG16 architecture - the last layer will be used for re-training on our dogImages dataset.

Methodology

Data Preprocessing

Note: We are dealing with lots of image data here. Even with GPU instance of type **ml.p2.xlarge** the style transform of the dogImages corpus took round about 9h!

The data processing part for this project is focused on how to 'force' the CNN to focus on learning object shapes.

This will be done in augmenting and manipulating the input images. To visualize this, have a look at Figure 2. A lot of different textures are applied to the same source image. This results in multiple input pictures for the same object. Since there are various textures for the same object, the CNN is forced to learn the shape of this object, since this is now the most distinct description feature. You can also see how the curly hair texture of the original source image is washed away by applying different textures.

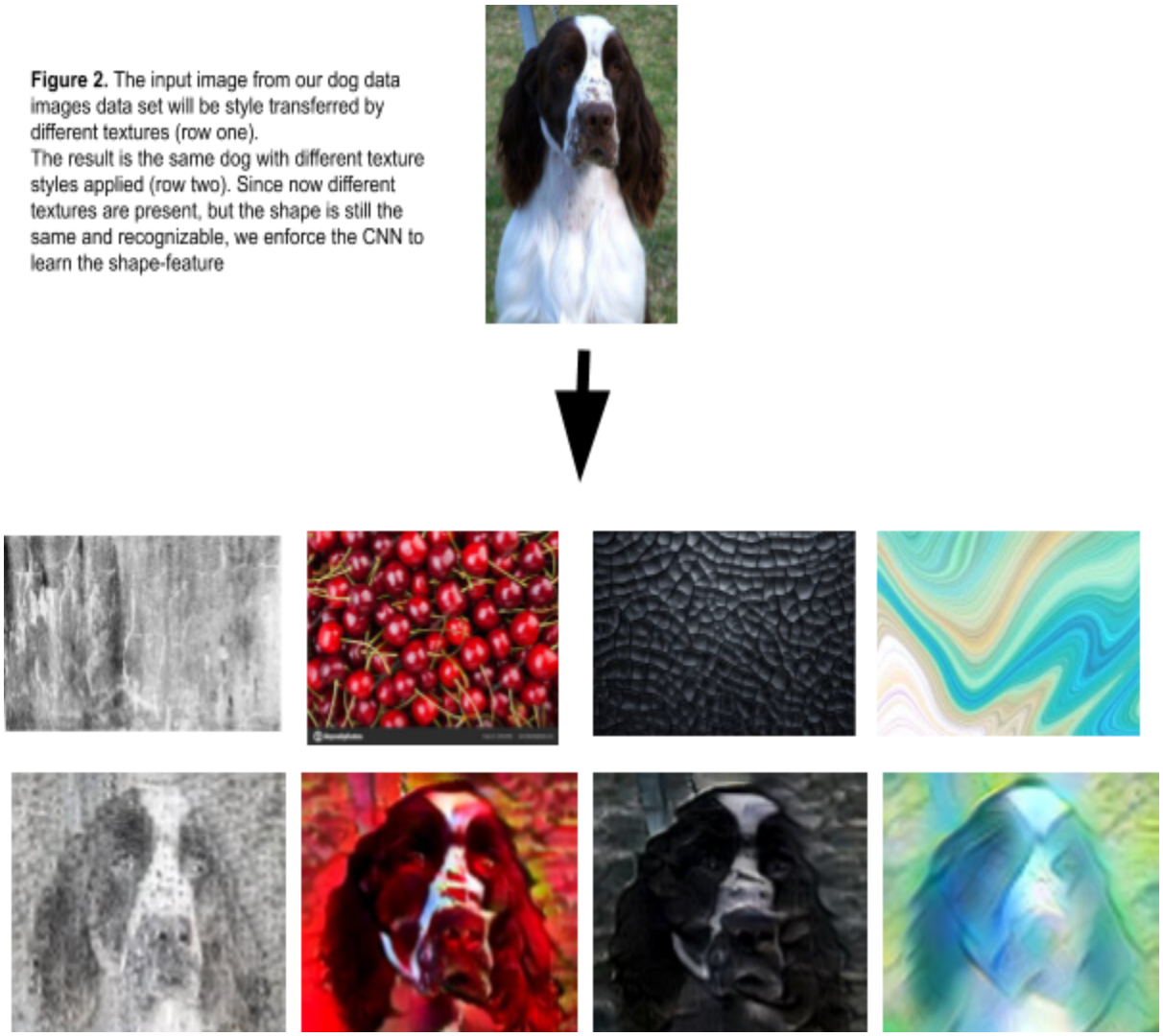


Figure 2. The input image from our dog data images data set will be style transferred by different textures (row one).
The result is the same dog with different texture styles applied (row two). Since now different textures are present, but the shape is still the same and recognizable, we enforce the CNN to learn the shape-feature

Since there is not only one texture for the same object, the CNN is forced to focus on the actual shape of the object. The hypothesis is to reduce the texture bias that usually occurs in training CNNs and afterwards get a more robust and performant (in terms of accuracy) CNN classifier.

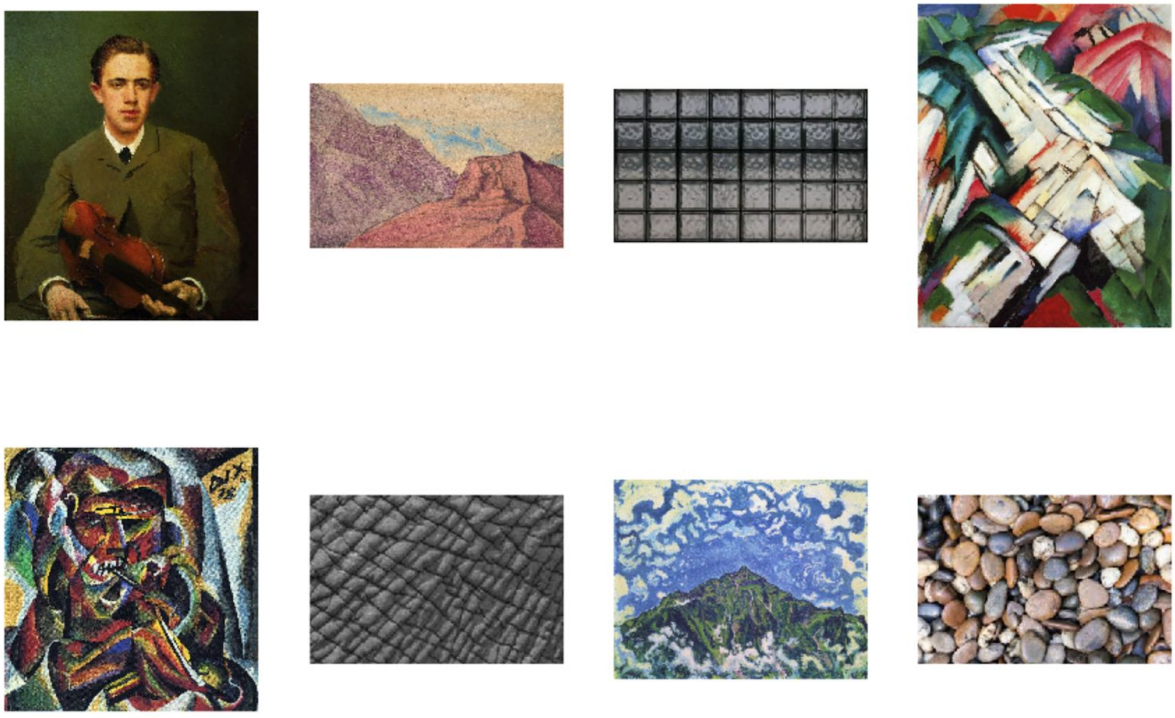
This was achieved by including the AdaIn https://github.com/naoto0804/pytorch-AdaIN algorithm into our project, that is able to style transfer the pictures.

Originally, I planned to work with different textures images (that were included in my project proposal assignment). Sadly it turned out, that many of these textures did not perform well, to leverage the shape of dogs. The following pictures were created with the old texture images:

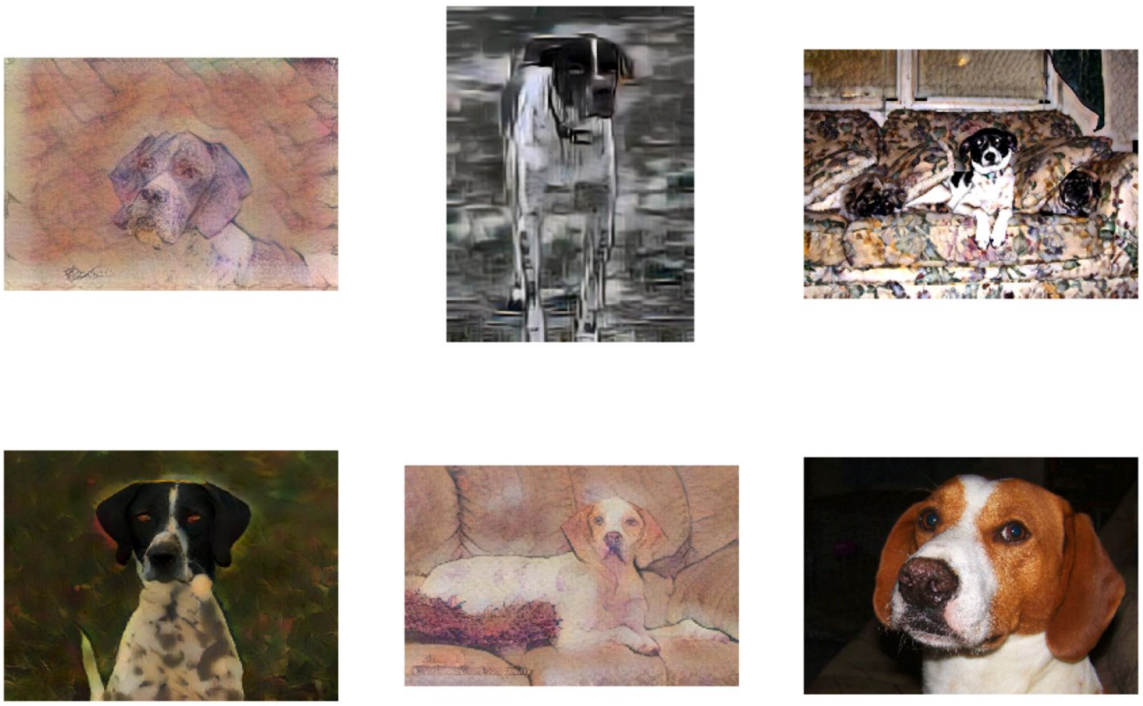
You see how often times the dogs shape is barely recognizable (even for us humans). This makes it very hard for CNNs to learn a proper representation on shape.

Because of this I chose new texture images, from the same dataset where also the original paper sampled from. You can find that dataset at https://www.kaggle.com/c/painter-by-numbers/data .

The new corpus of 8 style images is the following:



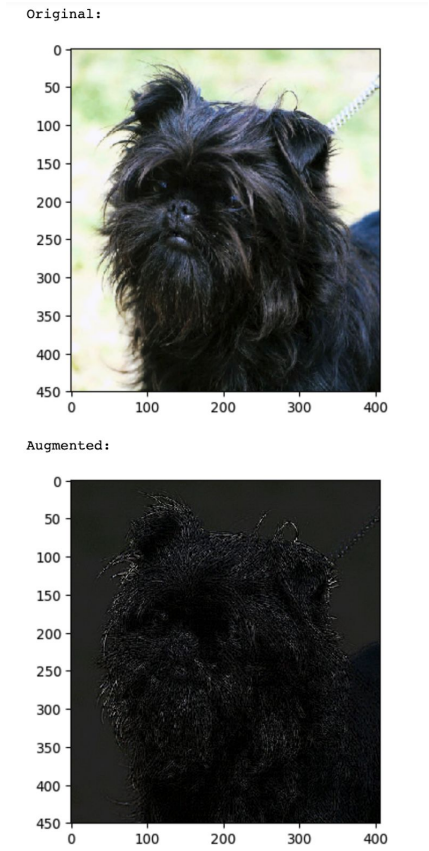
After style transferring the dogImages corpus with the above texture images and specifying the data transforms (see notebook), we display a batch of training data (that includes style transferred images & original ones) to see how these new textures amplify the dogs shape:



Advanced Preprocessing

While the torchvision.transforms package has lots of useful and important pre-processing capabilities (like Resize, CenterCrop, Greyscale, ...) it lacks capabilities of advanced image augmentation. After some research I stumbled upon a great library called imgaug that leverages some advanced image processing techniques.

For instance, imgaug has capabilities to for edge detection. This feature seems very promising for our task at hand, because edges define the shape of an object. And the shape of an object is the characteristic we want our CNN to focus on. With imgaug we can easily lay an edge detector on our images:



Implementation

The main part of my implementation consists of the described preprocessing pipeline and the specification of a transfer learning model in PyTorch and the integration of both in AWS SageMaker. I would like to invite the reader to go through the corresponding notebooks, that are all executable and self-documented.

Note: Be sure to use the referenced GPU instances as described in the notebooks, otherwise the training might take very long.

Impediments

I underestimated the volume of my training corpus after style transferring it. This resulted in very long run times (see section Result) of training. Furthermore the migration to AWS SageMaker required some refactorings and custom package versions. For instance, I needed to put pillow==5.4.1 in my requirements.txt for SageMaker to downgrade the library due to bug report https://github.com/python-pillow/Pillow/issues/3769 . On the other hand the PyTorch-SageMaker-Estimator has this library in version 6.0.0 (check the dependency table at https://github.com/aws/sagemaker-python-sdk/tree/master/src/sagemaker/pytorch).

Results

Model Evaluation and Validation

The model key characteristics are summarized in the following table (the best score is highlightet):

model name	**used model**	**dataset**	**training time**	**accuracy**
vgg16_imgaug	vgg16	style transferred pictures & original pictures + imgaug pipeline	5 hours	81.58%
vgg16_style	vgg16	style transferred pictures & original pictures	2 hours	81.58%

baseline (vgg16)	vgg16	original pictures	**37 min**	**83.61**%

original screenshots of AWS SageMaker interface:

sagemaker-pytorch-2019-08-04-19-50-35-540	Aug 04, 2019 19:50 UTC	5 hours	Completed
sagemaker-pytorch-2019-08-05-11-30-00-513	Aug 05, 2019 11:30 UTC	2 hours	Completed
sagemaker-pytorch-2019-08-06-06-15-30-723	Aug 06, 2019 06:15 UTC	37 minutes	Completed

As we can see, the baseline performs better in accuracy. While I was hoping to improve the accuracy with my advanced image augmentation techniques, it could not be achieved in practice.

Searching for different reasons, I came back to the paper [1], that inspired this work. Investigating their results, I found the following performance table:

name	training	fine-tuning	top-1 IN accuracy (%)
vanilla ResNet	IN	-	76.13
	SIN	-	60.18
	SIN+IN	-	74.59

In the column 'training' you can find the dataset, that was used to train the corresponding model. **IN** is the original dataset (comparable in our case to the original dog images), **SIN** is the style transferred version of IN. Looking at the accuracies, they actually reflect my findings pretty well: By training the same model (in their case it is a ResNet) with only the original training corpus and then with the original corpus plus the additional style transferred corpus the models accuracy decreases!

While the accuracy decreases, the model (see paper) "is more robust towards distortions than the same network trained [on IN]". For our use case this needs more exploration to be done - and maybe also another metric would be an idea (more thoughts on that in the Outlook).

Also my result answers the question mentioned above "Is there an inherent texture bias in our benchmark model solution?". Indeed there seems to be a bias in CNNs towards learning texture, because our trained model on style transferred pictures had a harder time (lower accuracy) on classifying the test dataset right. This also coincides the finding of the quoted paper.

Interestingly, the model that was trained on only the style transferred & original pictures (vgg16_style) yielded the same accuracy as the model that was trained on style transferred & original pictures with the additional imaug augmentation pipeline (vgg16_imgaug).

In terms of runtime the vgg16_imgaug model really stands out with 5 hours! Of course this stems from the style transferred corpus, that is 8x the original volume. But it is also because of the usage of imgaug library. Sadly, this library does not have GPU support (yet). Also, to make this library work in combination with AWS SageMaker, I needed to set the PyTorch DataLoader to num_workers=0 - no parallelization! This explains the high running time.

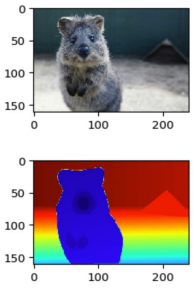
Outlook & Summary

While all of my models scored above 80% accuracy, the vgg16_imgaug model could not reach better accuracy values than the baseline model.

The model was forced to learn a different representation of objects (the dog images), that is because we fed in style transferred images, that amplified the shape of the dog pictures.

Throughout this project I had lots of ideas for further work regarding this domain. I want to share some of these ideas here:

- Try out a different metric, that better catches the robustness of the model in comparsion to the baseline model. While the accuracy was not improved [1] also describes, that the robustness of a CNN is improved, when trained on shape representations
- Integrate some heatmap/segmentation features from imgaug library. These heatmaps capture the shape of an object even more distinctly:



- Invest more research in what texture images should be used best
- Visualize the last layers of our CNN to demonstrate, that the model actually learned the shape representation and not the texture representation