



Sticker Hunt with Computer Vision

Semester Project Report

Tugdual Kerjan

Supervisors:
Krzysztof Lis
Mathieu Salzmann

Acknowledgments

I want to thank Krzysztof Lis for the guidance he gave me during my project

Abstract

We wanted to build an Augmented Reality game where one would search and find the various stickers placed around EPFL. The goal is to ultimately *collect them all*. This required the combination of a lot of different technologies, ranging from Computer Vision to Cloud Services.

Table of contents

[Tugdual Kerjan](#)

[Supervisors: Krzysztof Lis](#)

[Acknowledgments](#)

[Abstract](#)

[Table of contents](#)

[Introduction](#)

[A little history](#)

[Source: Original work](#)

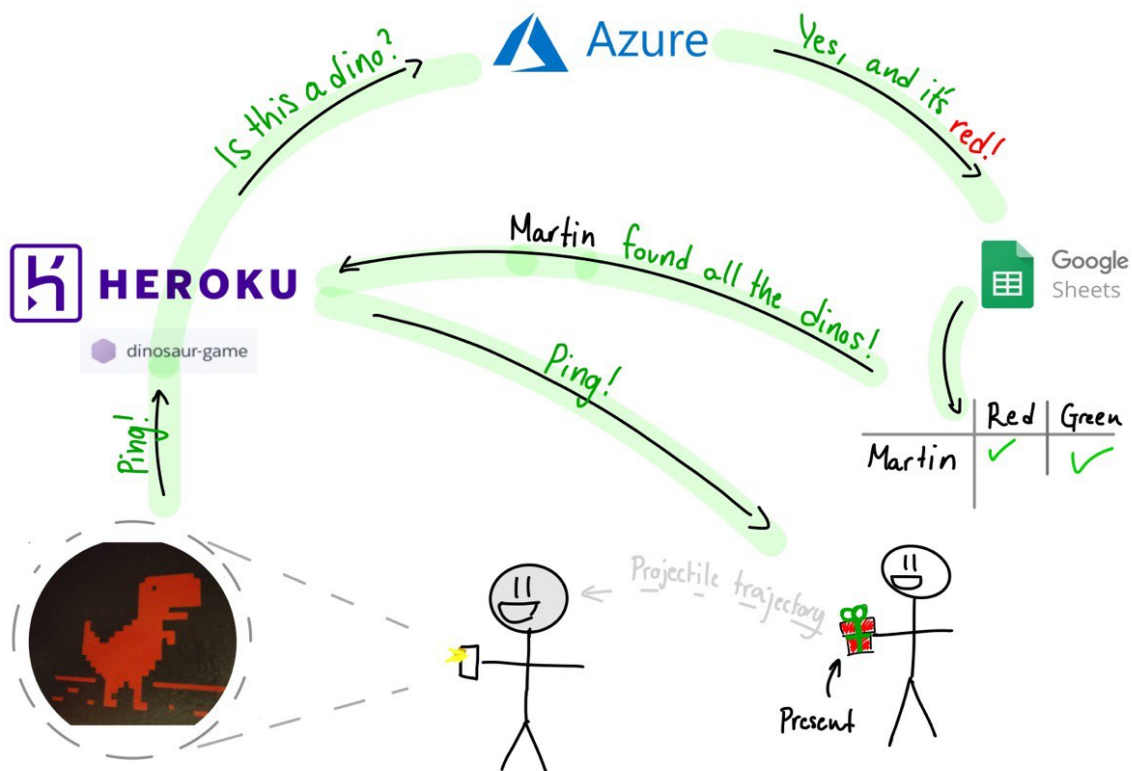
Introduction

We can all remember the hot summer of 2016 when all the kids including me were running around catching virtual pokemon with our phones, thanks to the craze surrounding Pokemon GO. It showed the untapped potential for Augmented Reality games. The ability to interact with the environment around the player really pushes the boundary for immersion, while usually only requiring a phone. This fact, along with random circumstances led me to want to expand my understanding in the field of computer vision.

A little history

This project actually arose accidentally while I was using a Silhouette Cameo 3, a machine that allows you to cut out vinyl and make custom stickers. The issue I was having was that I was wasting a lot of vinyl every time as I wasn't utilising the whole width of the space. My solution was to start filling the rest with, logically, dinosaurs.

Over time these dinosaurs started popping up inside the INM EPFL building, giving me the idea to gamify the process. The idea would be that one could attempt to find the dinosaurs, take a picture of them and I would manually check these pictures. Obviously, this quickly became tedious and I sought to automate the process by using Microsoft's Azure cloud services to detect the color of the dinosaur, which would then send the entry into a google sheet, and would notify me once someone had found all the stickers. I then simply had to verify the pictures, and offer a reward to the person, which happened to be a 3D laser cut dinosaur I had created at SKIL as well.



Source: Original work

This is thus the story of how the idea to hunt spawned out of the blue. After looking around for a Bachelors project to work on, getting little inspiration, I decided I could try my luck and propose a project, with the goal of cutting out Azure from this diagram and building my own, more efficient and accurate dino recognizer. A big shout out to Krzysztof who is the only person to have given me a chance for this project!

Definition of the project

After some back and forth, it would be more interesting to work on detecting stickers of all styles, especially association related stickers at EPFL, in the goal of creating a similar experience, but on a more general scale. This would significantly increase the difficulty of this project and push me outside of my comfort zone.

Requirements

The main objective of this project is to be able to build an application that would be able to identify stickers from a camera, verify if they are already present in the database, and, if not, add them as a new class. Thanks to Julius Caesar we can divide and conquer this problem into 3 parts:

1. Sticker detection

- 1.1. Manage to train a model to detect stickers in a provided image, so as to then pass it to another model which would be dedicated to trying to figure out what is in the region of interest.

2. App development

- 2.1. Figure out on what medium the app would be available. Ideally on a phone as an android app it could work, but one can also think of using a webapp, or even building a dedicated device - you could also carry around a friend, they are great sticker recognizers.

3. Sticker Identification

- 3.1. Identify the association, club, faction, country, company a suggested part of an image that probably contains a sticker belongs to, and if it is unable to classify it, create a new category.

Sticker Detection

The first, and biggest problem encountered in this project is the ability for a neural network to detect stickers in an image. It must be able to distinguish it from other similar categories of objects like text and patterns. After looking around using paperswithcode.com it seemed that **Detectron2** (He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. 2017.), a library built by Facebook is leading the way in this domain.

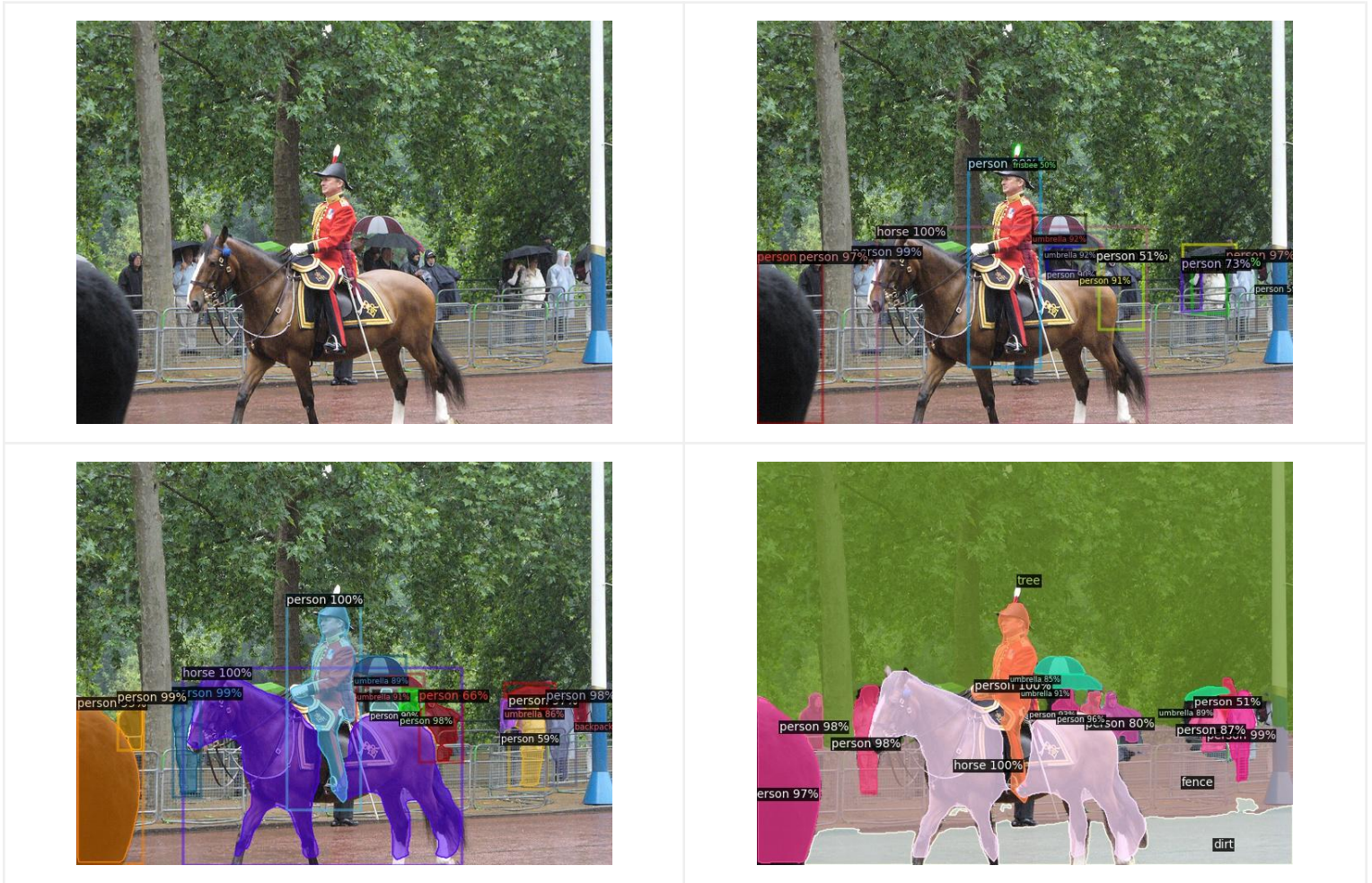
An important difference stated in object detection papers was whether the architecture of the NN was **proposal based**, or **proposal free**. The biggest tradeoff was a gain in performance over more blackboxing with proposal free. For our project we decided it would be more interesting to go with a **proposal based** NN as it would allow us to have more control over the fine tuning of the model, and see what was going on inside.

Installation

Installing the programs on my computer was a little bit tedious, I learned the how and why of python environments, and installed **PyTorch**, **Torchvision**, **Detectron2** and **OpenCV**. Although I initially used Conda, I decided to fall back on **VirtualEnv** as I would later be using the **SCITAS** servers which came with VirtualEnv.

Testing

After a while I was able to run the demo.py provided by the [Detectron2 git](#), which yielded these results:



Source: My computer

I also discovered the power of **Jupyter Notebooks** through the [getting started tutorial](#), which provided abundant information and explanations as to how to run the demo, but also a hint as to what was next, which was training a custom dataset. But before that, I needed to get something better than my PC to train my model.

More power

1 year prior to this project I had followed an introductory course on how to use the SCITAS clusters of EPFL, in the goal of using them for **photogrammetry**. (Yes, I turned myself in 3d). With this knowledge I

was able to quite quickly connect again to the cluster, and reinstall the programs on **SCITAS** so as to see if it would be possible to train a model, as the latest cluster, IZAR had CUDA available GPUs. I learned how to use **scp**, **rsync**, more about **virtualenv** and how to use **bash** scripts to do all the tedious bits faster.

How to train it to recognize stickers

The biggest question that remained to answer was how to train the model so as to get it to recognize stickers. Thankfully, the answer was not to spend weeks going around EPFL photographing, cropping, and annotating hundreds of images of stickers. Instead, we opted to contact the **University of Augsburg** which provided us with links to the dataset specialized in the recognition of logos. (**FlickrLogos32** and **FlickrLogos47**). After making a blood sacrifice and spending around 8 hours to understand how Detectron2 loads data, I was able to load and sample one of the images in the dataset:

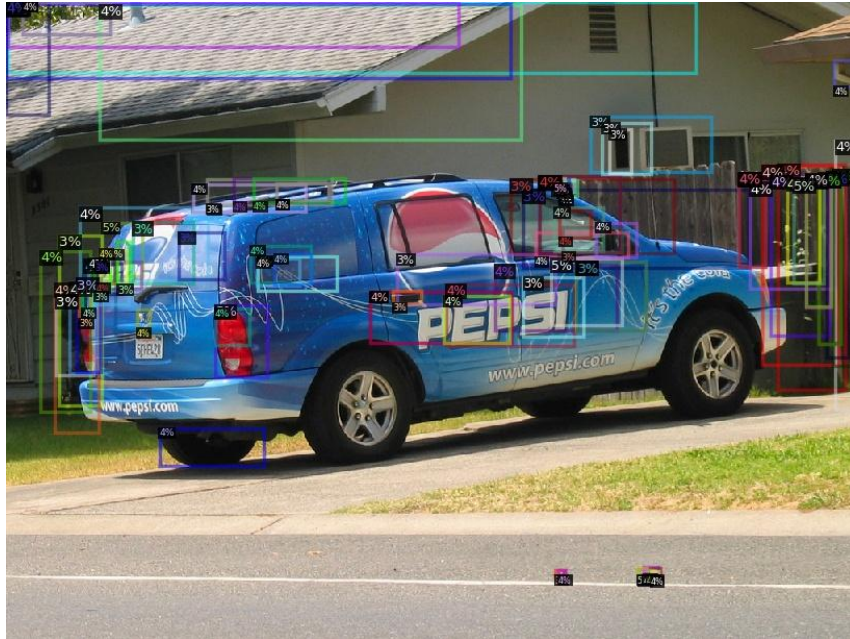


Source: FlickrLogos32

From that point on I was able to copy the dataset onto my partition on SCITAS, and attempt my first training of Detectron2 on a new dataset.

Abysmal results

As the image below testifies, the model seemed to have taken one step forward and 4 steps back:



Source: My Computer

A simple fix

After playing around with the model, I simply increased the Learning rate from a tiny 0.000025 to 0.0002, which managed to significantly improve the performance of the model. From then on it was just a matter of trying out different models and empirically checking the images for the accuracy of the trained models.

A working model

Finally I was able to obtain a decent model that is saved as model.pth, copy it to my computer and test it on some custom images



Launching an application

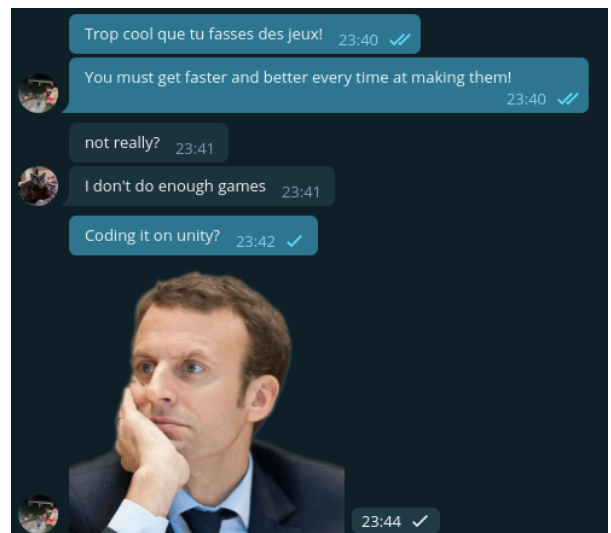
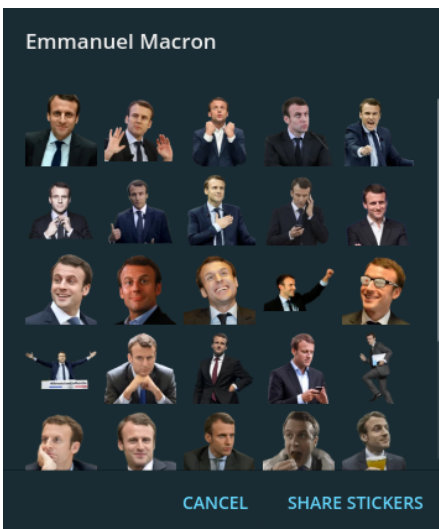
Selecting a platform

As stated, we could either choose to run the application on android, the web, or dedicated hardware. I believe that dedicated hardware defeats the purpose of being able to easily and quickly play with friends. I was also uneasy with running the program on android as iPhone players wouldn't have access to this clearly amazing game. Although I was originally planning on going with developing a web application as it is available on most devices, an idea I had pushed me to use telegram.

Although using Telegram as a platform on which to easily interface with, receive and send images was extremely useful for some quick prototyping, it remains very restrictive at the UI level. It's a trade off I was willing to take given the amount of time left I had to finish the project.

Automatic sticker pack creation (A side project)

After witnessing the power of Detectron2 and its ability to segment objects in images, I had an idea which would eat up the next two days of my time on this planet. On telegram, one can make **custom sticker packs**, which can reference inside jokes, people, etc.



Source: <https://t.me/addstickers/Macron>

Usually these sticker packs are tedious to do as one has to cut out the people from images. I thus decided it was time to save the world from the disease of masking people for Telegram sticker packs. I thus wrote a Telegram bot which accepts an image, detects the big objects in the image, cuts them out and formats them to 512x512 png and automatically adds them to a sticker pack. This cuts down the time for sticker creation **from 10 minutes** (Open computer, load photoshop, cut out, save to png, send to telegram bot for sticker packs...) **to 10 seconds**. That's a **x60 increase in productivity!**



Detecting and masking people

Source: My telegram bot, Prof. Dillenbourg

The biggest and hardest part in the deployment of this application was getting it to run online, learning to **containerize** my application, learning how to link up **Heroku** and running a segmentation model that would take less than **500MB** in memory (Hard limit on Heroku). I thus attempted to load **vovnet** models, which are purpose built to be as efficient as possible in regards to memory and accuracy. Unfortunately, although it seemed to work on my PC I was unable to run the vovnet python library on Heroku.

Portable logo detector

Naturally one bot isn't enough, and once I got the gist of how to host Telegram bots running Detectron2 online, I realized that I could continue to use Telegram as the platform on which to run the sticker detection, just like the dinosaur bot at the start of this project. I created a simple bot that simply would reply to the sent image with the output of my trained model for logos. This allowed me to easily do some field testing instead of tediously having to load the collected images from my phone to the computer and run them there.



Detecting and masking stickers

Source: My telegram bot

Once the Telegram bot was up and running the *only* thing left to do was the classification - probably the hardest part of the project.

Sticker Classification

Few Shot Image Classification (FSIC)

The goal of Few Shot Image Classification is to be able to classify an image given very few examples of the category. (After seeing a crab 4 times, the model has to be able to identify the 5th image of a crab). This is ideal for the project as there are a theoretical infinite amount of stickers to detect, where we need to be able to automatically classify a newly discovered logo after it has been found and manually classified 4-5 times. This actually creates a natural threshold criteria for the addition of stickers: Users need to have come across it enough for it to become an “official” sticker.

In a more explicit manner: Few Shot models can train on loads of images of a certain category (Alphabets (Omniglot), Animals (minilImageNet), Logos (FlickrLogos42)...). This allows them to learn what abstractions are important to pick up on, be it curvature of a letter, fluffiness of an ear, or contrast of a logo. Then, when given n shots of k classes, and q query images the model needs to be able to identify which queries belong to which classes.

The 3 pillars of FSIC and the 2 Datasets

There are three usual ways of being able to do classification efficiently without much data:

- Learning about similarity of the data (How to differentiate well)
- Learning about learning (How to adapt to the new classes well)
- Learning about the data (How to augment it to learn better)

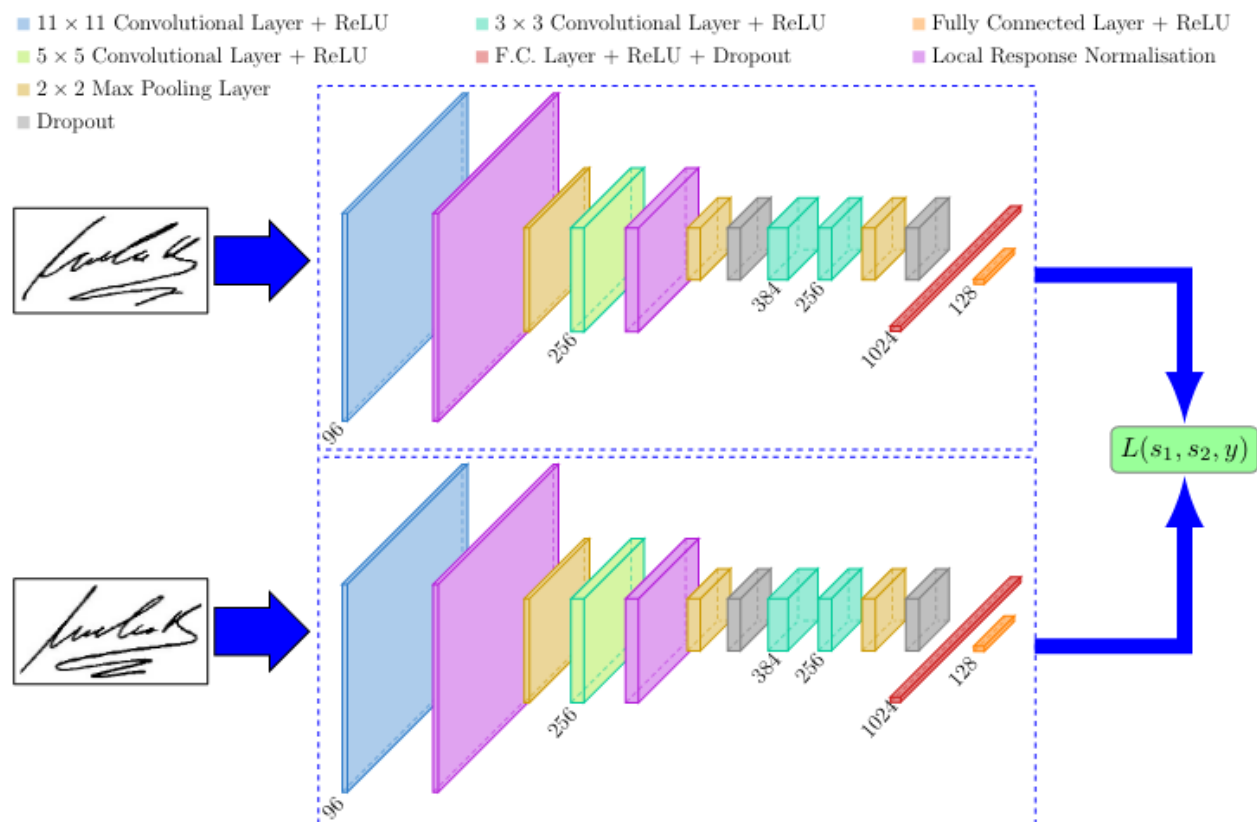
I decided to go with Prototypical networks which learn to differentiate well because of their efficiency, understandable model and the fact that code was already available online.

There are two datasets commonly used for this task:

- Omniglot, over 20 images of 1600 characters hand drawn by [Amazon's Mechanical Turk](#), spanning 50 different alphabets
- minilImageNet, a subset of ImageNet containing 600 classes of animals with 100 images each.

Prototypical networks

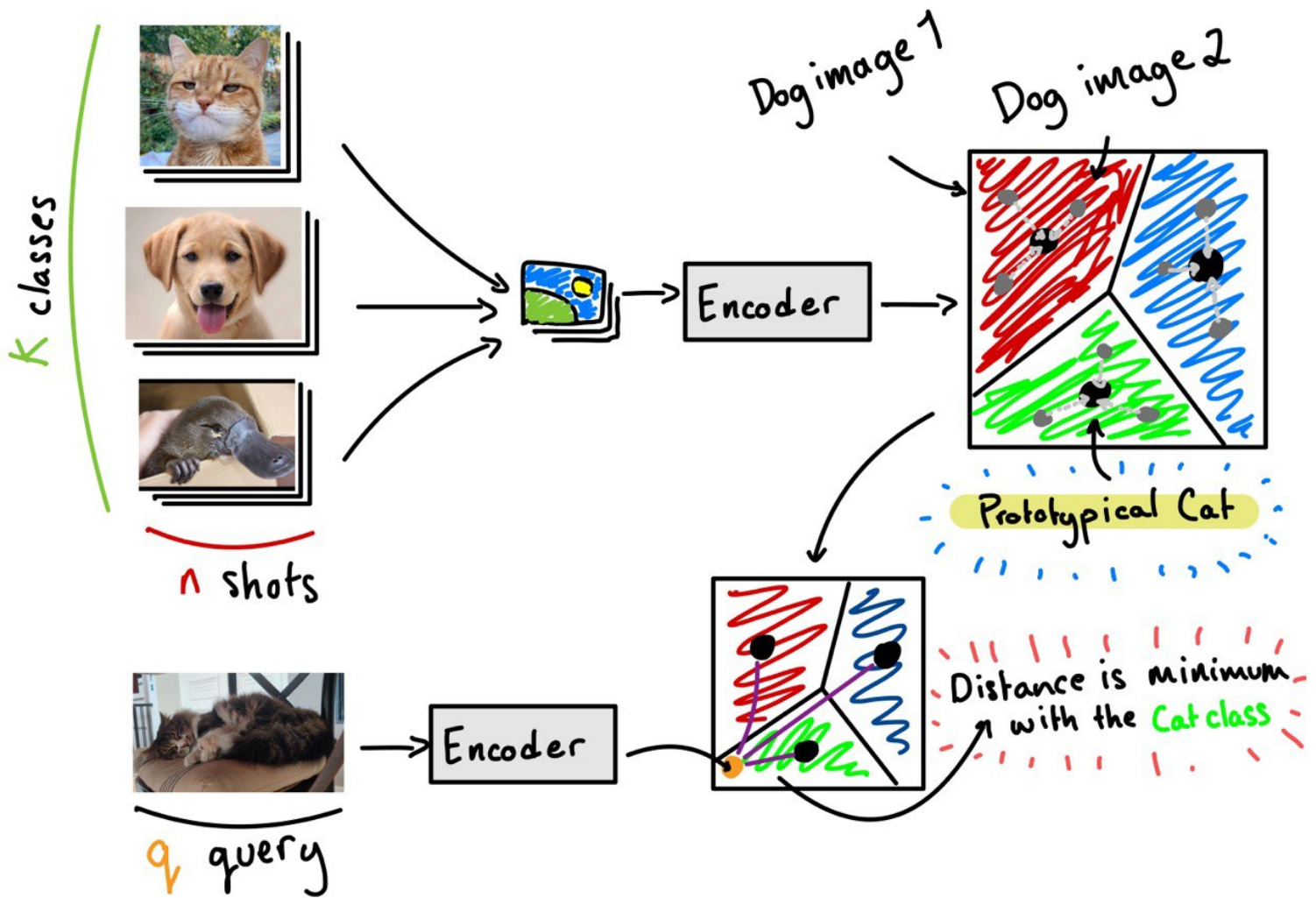
Neural networks are great at abstraction, and finding curves, then ellipses, then eyes, then faces in images. As we can see with auto encoders, they are thus able to find the most relevant pieces of information to then encode them in a smaller dimension. What siamese networks do with this is compare two images encoded with an energy function to see if they are similar:



Siamese networks on signatures

Source: Dey, Sounak, et al. "Signet: Convolutional siamese network for writer independent offline signature verification." *arXiv preprint arXiv:1707.02131* (2017).

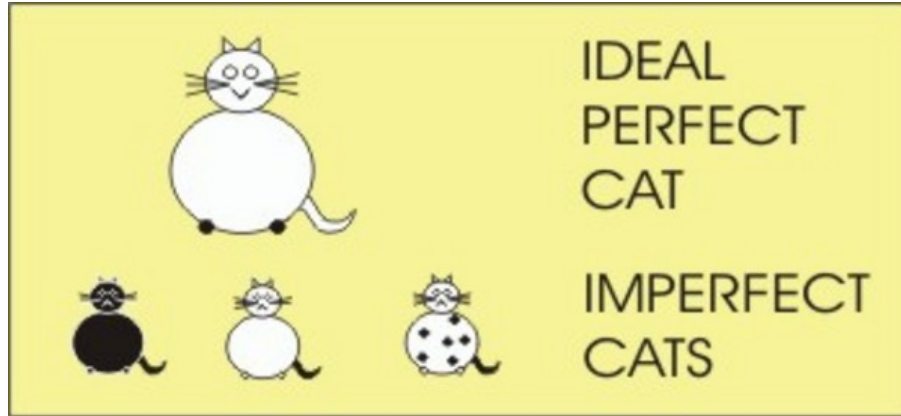
The advantage that Prototypical networks give is that instead of constantly having to do one on one checks with other items in the dataset, one is able to send multiple images of each class through the encoder, and, in the embedding space take a mean of the embeddings to create a “prototype”, ideal representation of the class in question.



Overview of how Prototypical networks work

Source: Tugdual Kerjan (Me again)

One could say this is the 2021 version of Plato's Theory of the Forms, where for all classes there exists an ideal (prototypical) object that represents the essence of what that class is.



An explanation for Plato's theory of Ideal forms

Source: Midnight Media Musings... (midnightmediamusings.wordpress.com)

Thanks to the amazing work done by Oscar Knagg which implements three Few Shot papers including Prototypical Networks for Few-shot Learning, I was able to quickly download his [git](#). I was then able to get some tests running.

Running some tests

I downloaded minilimageNet and Omniglot datasets, the two being very popular in this Classification problem. Running them through Oscar's program on SCITAS gave me the same results as obtained in the papers. It was time to do some experiments!

Customizing the program

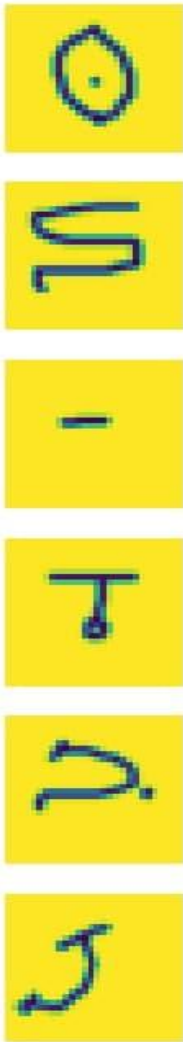
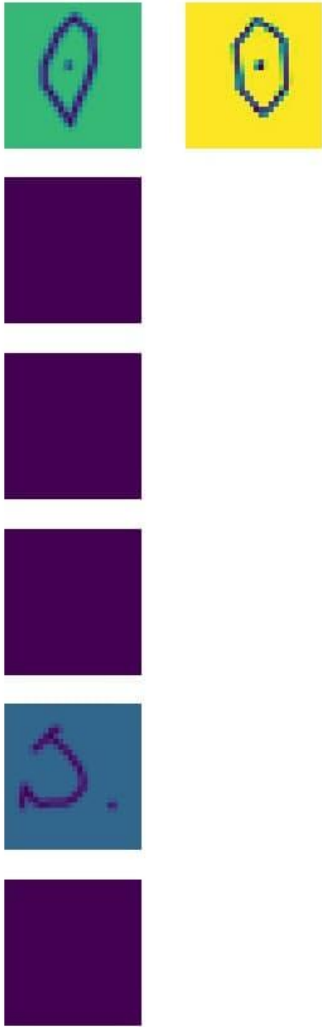
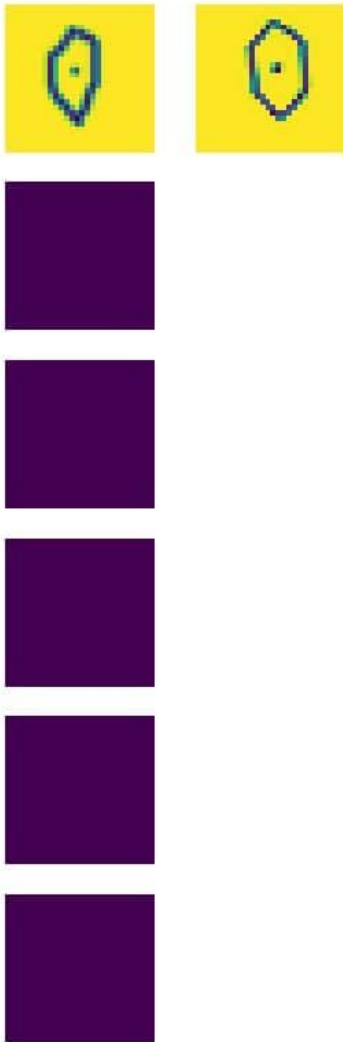
After spending a good day understanding how the machinery of the program worked, I was ready to start modifying it. First the program is only able to run on CUDA hardware, I had to change the device used by torch and the functions that took advantage of it. I then modified the fact that it was giving a multiple k of query images when it was going through a batch instead of 1. Changing this allowed me to simply provide one image for the model to try to classify. Next, I wanted to be able to run the program by providing it with my own hand drawn image. This is easier with the Omniglot dataset due to it's more basic nature. I chose to redraw the Arcadian "a".



The letter “a” of the Arcadian alphabet

Source: My hand, my apple pen and my iPad












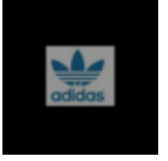

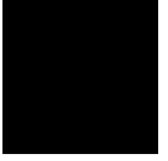


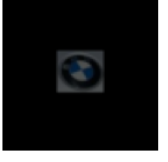
Armed with this letter it’s simply a matter of formatting it correctly (resize to 28x28 pixels using **ImageMagick's** convert tool), write a program that loads the prototypes of a few selected classes (including the Arcadian “a”) and then sends my drawing, comparing its embedding to those of the other classes. Here are some simple results:

6 Random classes including the Arcadian “a”	2 Examples of my drawing and its closeness to the others in the embedded space represented by the brightness of the 6 classes	
		

Source: My computer

Custom dataset

After I confirmed that I was able to load custom content, I decided to train the program on a custom dataset, by using the **FlickrLogos47** dataset, and preparing it before by simply getting cropped image with just the logo instead of the whole image as our goal is to classify not detect. After setting up the new Dataset (extending **torch.utils.data**), I trained the model on **SCITAS**. Downloading the .pth model and running it with a selected image on internet yields this result:

5 Random classes including the Adidas	Cherry picked examples that show it working sometimes (Again, brightness indicates similarity)			
    	     		     	

It was now time to tie it all together in one application: From detecting the logos in the image to feeding it to the few shot classifier so as to be able to return an answer to the user. This is a simple copy paste operation and modification of the code to accommodate for the png's instead of jpegs first used by the few-shot. After a little more coding we end up with a program that will first detect the logos in an image, and then attempt to classify them:



A POC for a bot capable of identifying stickers

Source: Original work

Conclusion

What I learned

This project has allowed me to learn a *lot* about Computer Vision, and computers in general. The absolute monstrous amount of tools I learned can only begin to scratch the surface with this probably incomplete list: git, python, numpy arrays, pytorch tensors, Pillow images, pandas dataframes, git, ssh, scp, rsync, ImageMagick, aptitude, Heroku, Docker, Markdown, bash, Slurm, OpenCV, VirtualEnv, Conda, Jupyter Notebooks, VSCode.

I also solidified the concepts of CNN's, encoders, and generally how to find the right information on the internet, as well as a new confidence in downloading gits and learning how to run the code and tinker with it.

Going further

Supposing I had the time to finish this project, there are a few aspects I would like to refine. First, it would be interesting to merge the models into a single, continuous one so as to streamline the process and avoid passing images / processing them between models. I would also like to move from Telegram to an independent platform, most likely on the web with Three.js for example. Finally, it would be nice to have a fully working program, the main part missing here is the feedback so that users can correct the bot, teach it to add new classes to it's inventory.

References

- Dey, Sounak, et al. "Signet: Convolutional siamese network for writer independent offline signature verification." *arXiv preprint arXiv:1707.02131* (2017).
- He, Kaiming, et al. "Mask r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.
- Wang, Xin, et al. "Frustratingly simple few-shot object detection." *arXiv preprint arXiv:2003.06957* (2020).
- Vinyals, Oriol, et al. "Matching networks for one shot learning." *arXiv preprint arXiv:1606.04080* (2016).
- Romberg, Stefan, et al. "Scalable logo recognition in real-world images." *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*. 2011.
- Detectron2 git: <https://github.com/facebookresearch/detectron2>
- Advances in few-shot learning: a guided tour Oscar Knagg, November 30 2018
(<https://towardsdatascience.com/advances-in-few-shot-learning-a-guided-tour-36bc10a68b77>)