CutOutRL - Visualizing Neural Networks with Scribbles

Yonatan Zarecki & Ziv Izhar, supervised by Elad Richardson

July 5, 2017

המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Contents

List of Figures

1 Abstract

Deep neural networks (DNNs) have been very successful in recent years, achieving state-of-the-art results in a wide range of domains, such as voice recognition, image segmentation, face recognition and more. In addition, reinforcement-learning (RL) training methods combined with DNN models (deep RL) have been able to solve a wide variety of games, from PONG to Mario, purely by looking at the pixel values of the screen.

Various “games” have been proposed for challenging neural networks, testing their capacity to learn complex tasks. Some tasks are designed to give us human insight about the way the model operates.

In this project, we challenged a deep RL model with the task of segmenting an image using scribbles. We force it to achieve good segmentations by using scribble-based segmentation in a way similar to humans. We hope to gain insight on the way the network does segmentation by looking at the scribbles it generates.

2 Introduction

Neural network have been successful at a wide range of task, however, little is known about their inner working and what makes them tick.

Some work have been done on the visualization of NNs, for example by calculating the de-convolution of an highly activated neuron and watching the resulting image. Attention has also been used as a way to see what the model “focused on”.

Not much work have been done on forcing NNs to work with “human tools”, tools that by using them we can gain insight on what the model “think”. Our project was born from an idea Elad had of forcing a NN to use his tool “CutOut”, meant to help humans segment images and training the network to use it for segmentation.

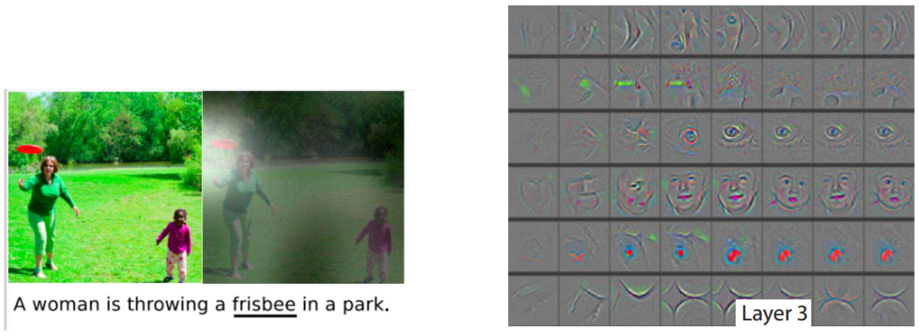

Figure 1: Other visualization techniques for CNNs

2.1 General Project Progress

The project's first stage consisted on us acquiring the theoretical and practical knowledge needed for implementing it. We've learned about CNNs in a previous project, and expanded our knowledge for CNNs in reinforcement-learning tasks.

After acquiring the knowledge we needed, we looked online for a basic model work with, a model that works for a wide variety of RL tasks, such as PONG and Breakout.

After getting to know the code and the framework we implemented the segmentation game environment (will be discussed further) and trained the model on it. After each training round ended, we would look at the results and hypothesize solutions for the problems we were seeing, implementing them for the next training iteration.

3 Technical Background

3.1 NN & CNN Background

Our model is based on convolutional neural networks (CNNs), trained with reinforcement learning (RL) algorithms for the segmentation task.

We'll elaborate more on NNs, CNNs and RL in the following subsections.

3.1.1 Neural Networks

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

> "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."
> In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

NNs are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the brain but on much smaller scales. A large NN might have hundreds or thousands of processor units, whereas a brain has billions of neurons. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.

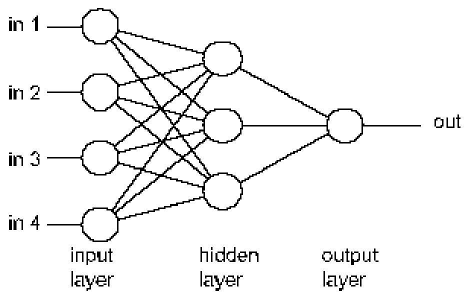

Figure 2: Example NN structure

Training NNs usually involves using the back-propagation algorithm [3]. The algorithm calculates the gradient for our network in an efficient way, using the structure of NN as layers. This algorithms was a crucial step toward the using NNs effectively.

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous subsection: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, runs its "activation function" and sends out the output to other nodes. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.

So what does change? CNNs architectures make the explicit assumption that the inputs are **images**, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. By knowing the input is an image, we can use convolutions (hence the name) on the image, which are known from CV algorithms to extract good properties from the image such as edges. Example activations for each layer of the CNN can be seen in Figure 3.

CNNs became very popular in recent years, starting with AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The success of AlexNet sparked an interest in CNNs which dominated the following years ImageNet challenge (almost all of the competitors used CNNs), and inspired researchers and practitioners to apply Deep NN to various other tasks such as speech-recognition [5], face-recognition[4], NLP tasks [6] and more.

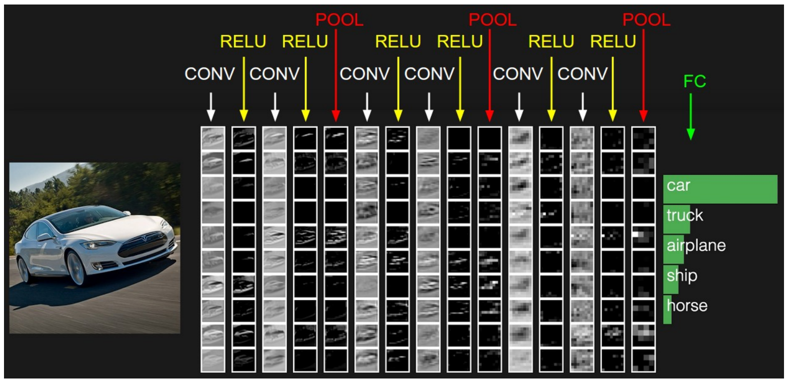

Figure 3: Example CNN activations for each layer

3.2 Reinforcement Learning Background

Reinforcement learning (RL) is usually about sequential decision making, solving problems in a wide range of fields in science, engineering and arts (Sutton and Barto, 2017).

The Reinforcement Learning Setup Problems solved using RL usually follow a simple setup and include the following components [7]:

1. **An environment** *Env*- the problem we're trying to solve, it is capable of receiving **actions** from an **action-space** $\mathcal{A}$ and for each action it returns an **observation** (or state) s_t and a **reward** r_t, which indicates whether the action was "good" if the score was high.

2. **An agent -** Capable of looking at the environment's observations and decide on actions to perform according to a policy $\pi(a_t|s_t)$. The agent's purpose is to maximize the environment's reward function.

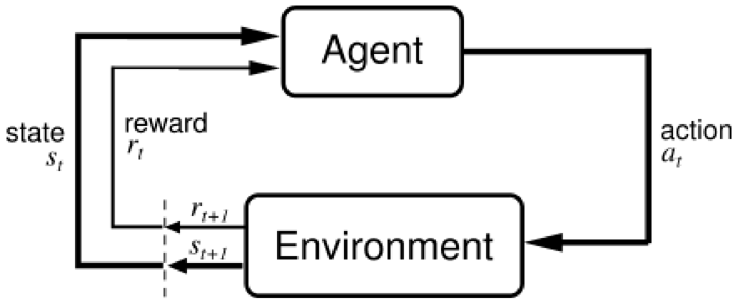

Figure 4: The typical RL setup

Several methods have been proposed in order to solve this problem.
Q-learning proposes learning an action-value function $Q(s, a)$ which predicts the expected accumulative future reward if an action a is applies in state s. In contrast, policy-based methods learn $\pi(a_t|s_t; \theta)$ directly, and update θ in order to maximize the reward.

In the last few years, we have been witnessing the renaissance of reinforcement learning, especially, the combination of reinforcement learning and deep neural networks, i.e., deep reinforcement learning (deep RL) has proved to be very powerful.
Deep RL algorithms have been successfully used to solve a large set of problems, such as ATARI games (from raw pixels), beating world champions at Go, guiding drones in a virtual world, and more.
Deep RL proposed to use deep neural networks in order to learn the value function $Q(s, a)$ or policy $\pi(a_t|s_t)$.

Deep NN have been used in many tasks that offer observations in the form of pixel data, such as PONG or Mario.
These tasks used CNNs, discussed earlier, as the core of their NN model for understanding the image, as we will do in this project.

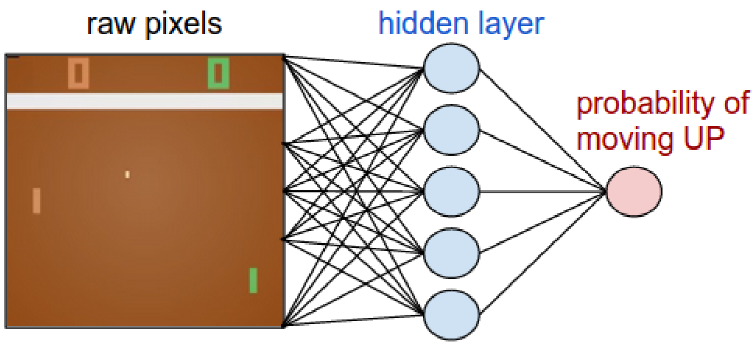

Figure 5: Deep RL with a policy-based method

4 CutOut Game Implementation & Experiments

In the previous section, we've presented some motivations for using deep RL, in this section we will present our implementation for the CutOut game, and go through the experiments and realization we made when training the model.

4.1 Looking for a basic RL implementation

Our first step was to find a working implementation for an RL tasks that learns from pixels, such as PONG or Breakout.
We looked at the top performing models on OpenAI's gym [1] on PONG, tried training a few and they did not work. Afterwards, we moved to models for Breakout and finally found a working model, we also trained it on PONG and it worked as well.

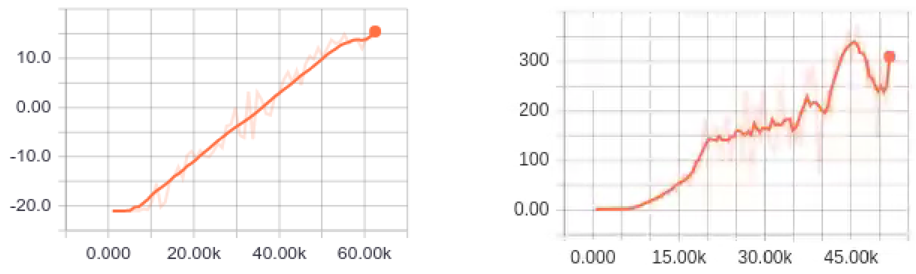

Figure 6: Training graphs for PONG and Breakout
x axis - the number of epochs
y axis - the accumulated score, returned by the environment

4.2 CutOut Game Definition & Implementation

As discussed earlier, in order to train a RL model on the task of segmenting an image with CutOut scribbles, we need to define the environment Env_{CutOut}, it's reward function r_{CutOut}, action space $\mathcal{A}_{CutOut}$ and observations s_t.

Each game start with 2 points, a point on the object p_{inside}, and a point outside the object $p_{outside}$.
At the start of the game the environment randomly selects a facial photo from the artificial faces dataset [2], which is then used in the course of the game.

[1]OpenAI Gym - https://gym.openai.com/

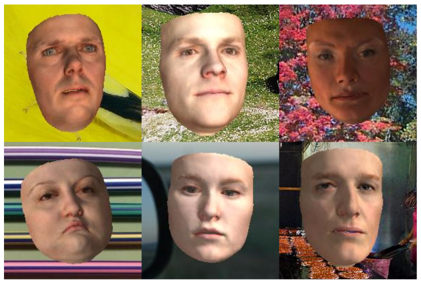

Figure 7: Atificial faces dataset [2]

In each state, the action space $\mathcal{A}_t$ is defined to be the 8 direction each point $\{p_{inside}, p_{outside}\}$ can move in the image, this is illustrated in Figure 8. Choosing a direction causes the "head" of the line to move a predefined number of pixels $step_{len}$.

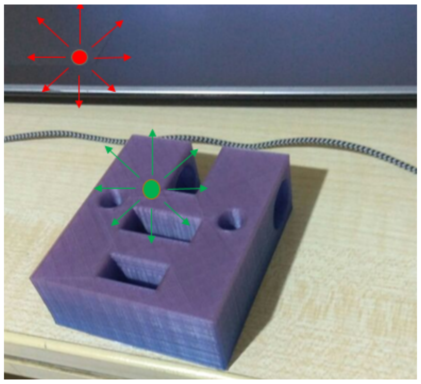

Figure 8: The action space $\mathcal{A}$ - each point can move 8 direction

The observation s_t for a particular step t is defined to be the original face image together with the current scribbles on it.

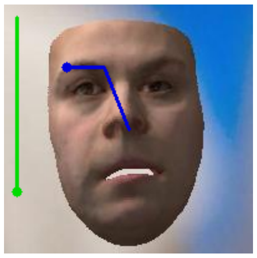

Figure 9: CutOut environment observation example

The reward function is defined to be intersection-over-union of the ground truth segmentation seg_{GT} against the proposed segmentation received by running CutOut on the current lines $seg_{proposed}$, $IoU = \frac{seg_{GT} \cap seg_{proposed}}{seg_{GT} \cup seg_{proposed}}$.

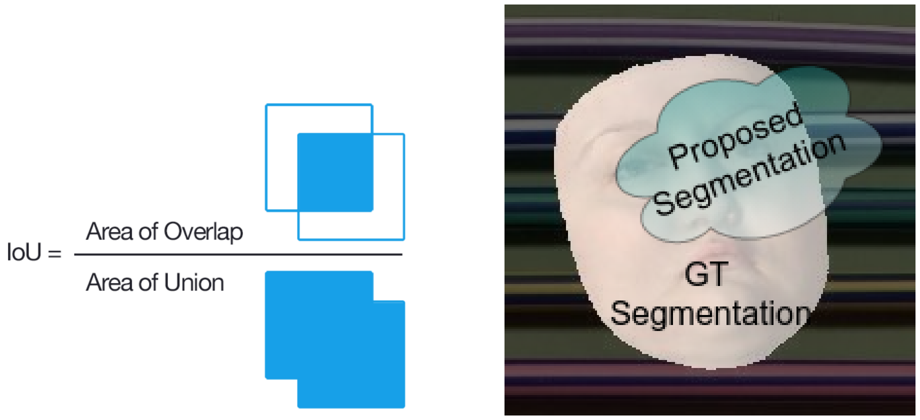

Figure 10: CutOut reward function r_t - Intersection-over-union

The game continues until a predefined number of steps $step_{count}$ (defaulted to 16) is passed, create a scribble from all the steps made in the course of the game.

After defining the game mechanics and formulas, we implemented the environment in Python, using Elad's CutOut C++ and Python's PIL environment. For compatibility with other RL algorithms we implemented the abstract methods of OpenAI's Gym Env class.

4.2.1 Tensorflow - An open-source software library for Machine Intelligence

Tensorflow [1] is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use.
Tensorflow has a great support for GPUs which are used to accelerate scientific computations , and is very important when training neural nets.
Tensorflow have become very popular since its recent release at the end of 2015, and a strong community have gathered around it, one reason for it is the platform's support for python in contrast to Torch's Lua.

We chose Tensorflow as it currently has a very active community and because the implementation we found for Breakout uses it for the definition and training of it's neural net.

4.3 Experiments

After finding a solid basic code and implementing the CutOut game environment, we were ready for training.

4.3.1 First Results - Off-the-shelf Model

We started by running the off-the-shelf implementation we had (switching the Breakout environment with CutOut) for about 24 hours.

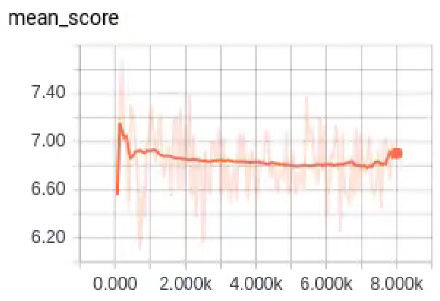

Figure 11: Results using off-the-shelf implementation

The results we found were disappointing, no real progress was made in the scores received by the model.

4.3.2 Improvment Iterations

After thinking over the results of the off-the-shelf model, we decided to change the environment's observations from an image with scribbles, to the original image plus 2 layers one for the scribble on the object and the scribble outside the object. We hoped separating these elements to different layers will help the network to better recognize the lines and their locations.

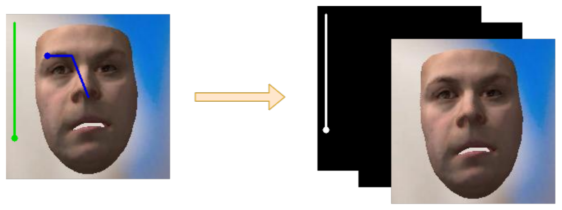

Figure 12: Observation change for the CutOut environment

Increasing $step_{count}$ We visualized the model's lines for a few times we have seen that the number of steps $step_{count}$ is too small, and significant lines didn't form. We've tried other values for $step_{count}$, 16, 40, 60 and 100, and only 60, 100 steps were enough for 2 significant lines to form during the game.

However, training with a larger step count did not result in better scores.

Changing Environment to Reward-at-end Hoping to see why the model took so long to train, while doing a relatively few epochs, we measured how much time the model spends on each part of the system. Our result was that 90% of the time was spent calculating the reward function using CutOut, 7% in Tensorflow and 3% in other calculations and GPU utilization was very low. This result is abnormal, as typical RL algorithms spend most of their time calculating the neural network and their next step. We knew CutOut was being calculated $step_{count}$ times in each game, as it is called each time for the calculation of the reward r_t. We tried to solve this problem by changing our reward to a "reward-at-end", where the environment returns $r_t = 0$ for all non-final steps, and return the real reward only for the final step. This means we call CutOut only once, at the end of our game. The model should be able to learn (similarly to learning to play Chess) to deal with the change and learn to maximize it as well.

Unfortunately, Our high hopes did not match reality. Even tho GPU utilization was very high, and our model made much more epochs, the resulting IoU was very low at 0.6. Visualization also showed our model did not make meaningful decisions when choosing steps. We returned to a normal reward function afterwards.

Changing the Learning Rate It is common knowledge in NN community that the learning-rate (LR) is the most important hyper-parameter. So we tried to change it and see if it makes a difference in our training graphs. We tried multiplying the LR by 2, and seeing if it converges faster. This time it did make a difference, and the model achieved an IoU of 7.5 after only 18 hours, much higher than before.
Our visualization were not as good as we hoped, still making unreasonable lines as can be seen in the figure.

Enlarging the Epoch Size Another problem the model wasn't performing was that the epoch size was too small, the common intuition is that larger epoch sizes means more stable training. We tried increasing the epoch's size by the multiple of 4, 8, 16 and even tho the training was much slower (as CutOut ran in each step's reward, and the epoch was larger) we did not see any change in the training graph.

Enlarging the Step Length As discussed earlier, running CutOut in the reward function takes most of the training process's runtime. In another attempt to reduce each game's total runtime we tried to enlarge the step's total length in pixels $step_{len}$. This way the model can still make significant lines in much less steps. We've set $step_{len}$ to 18 pixels (from 8) and $step_{count}$ to 30 (from 60), effectively cutting the runtime for a game in half.

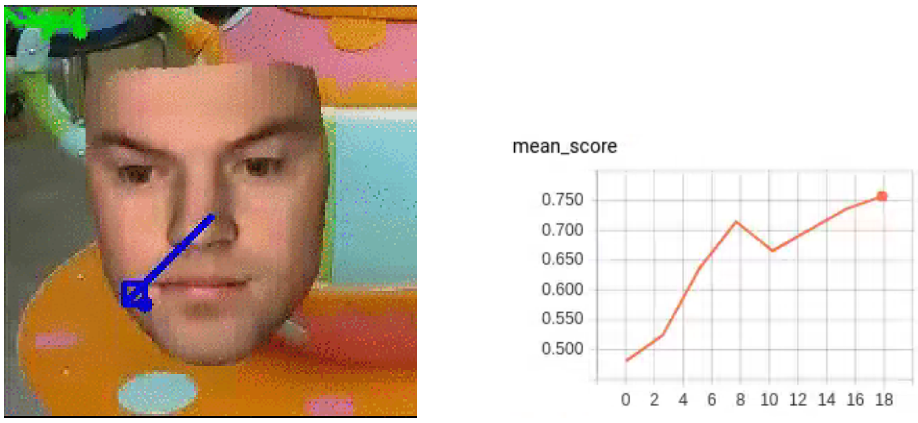

Figure 13: Changing the LR - A visualization of the output line, and the training graph

The ideas has indeed shorten our runtime, but even after running the model for a whole week we didn't go past the established 0.8 IoU score limit we received earlier.

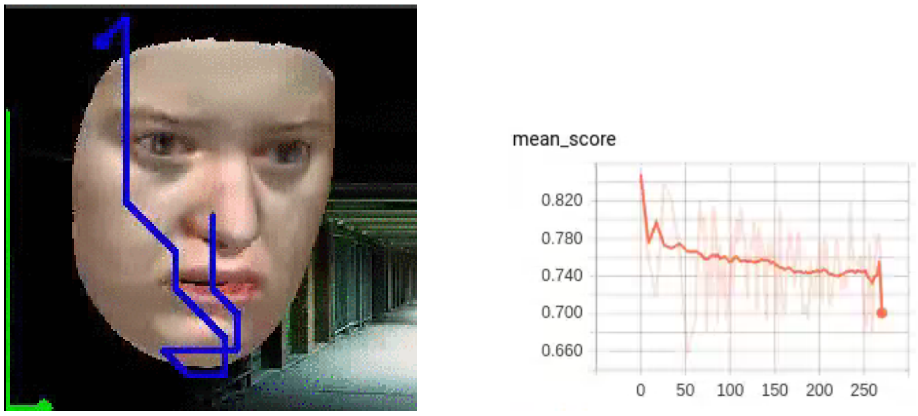

Figure 14: Enlarging $step_{len}$ - IoU score over hours spent training

Enlarging the Input Image's Size As a last attempt, we thought maybe our input image lacked details when reduced to 128x128 pixels. We tried enlarging the input image to 256x256 and train, the results were similar to earlier, settling around 0.8.

Closing up After making all improvements and attempts presented above, our model still wasn't able to fully "solve" the CutOut game. Qualitative analysis

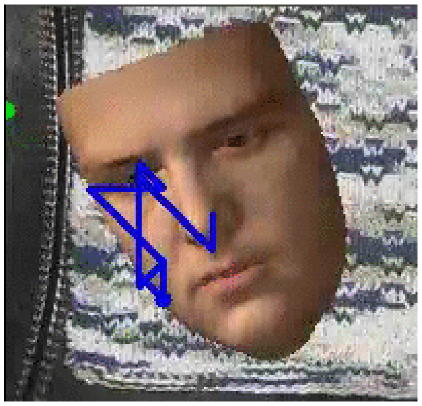
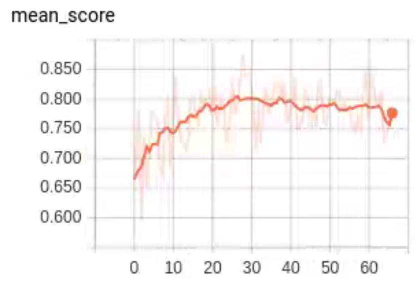

Figure 15: Enlarging the input image's size - IoU score over hours spent training

of the lines generated exemplified that the lines generated by the model were not very similar to lines a human would make. Also the IoU scores can be much higher if the task is done correctly. We suspect that if further research is done on this task, much better results can be accomplished.

5 Summary

Our project introduces a new task for NNs, making them work with humans tools for image segmentation. We implemented this task and experimented with a model training to excel at it, making improvements iterations along the way.

For us, this project was a realization of the skills we acquired after working on our previous project DeepFlowers.
We approached a much more difficult problem and tackled it head-on, working out issues as we go. This time our path was not as clear like in the previous project and we had to try different approaches to make our model succeed better or run faster. Even though our model did not fully solve the task at the end of the project we enjoyed doing it and learned a lot along the way. These experiences, we are sure, will help us in the future when tackling difficult tasks in NNs or in other domains.

We would like to thank our supervisor Elad for the continuous support in this project. You continued to give us support even as things didn't exactly go the way we hoped, and helped us figure things out for our own. We want to thank you for the freedom you gave us during this project, letting us take our time and make mistakes for ourselves, even tho it made the project stretch out a bit :)
We would also like to thank all of GIP's staff, for supporting us when Elad was away and helping us get through technical issues.

6 Bibliography

[1] **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**
Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, et al.
http://tensorflow.org/

[2] 3D face reconstruction by learning from synthetic data
E Richardson, M Sela, R Kimmel
http://ieeexplore.ieee.org/abstract/document/7785121/

[3] **Learning Representations by Back-propagating Errors:**
David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams
https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf

[4] **DeepFace: Closing the Gap to Human-Level Performance in Face Verification:**
Yaniv Taigman Ming Yang Marc'Aurelio Ranzato, Lior Wolf
https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf

[5] **Deep Neural Networks for Acoustic Modeling in Speech Recognition:**
Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, Brian Kingsbury
https://research.google.com/pubs/pub38131.html

[6] **Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank:**
Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng and Christopher Potts
nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

[7] DEEP REINFORCEMENT LEARNING: AN OVERVIEW:
Yuxi Li
https://arxiv.org/pdf/1701.07274.pdf