TDT4265: Computer Vision and Deep Learning

Final Project

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

March 24, 2019

- **Delivery deadline: Specified below. NOTE! there are three deliveries for this project.**
- **Final presentations will be held: 29. & 30. April, 2019**
- **This project count towards 16% of your final grade.**
  - 11% for the final presentation.
  - 3 % for the progress report.
  - 2 % for the project proposal.
- You can work on your own or in groups of up to 3 people.
- Upload your code as a single ZIP file.
- You will have access to use GPU resources on the Google cloud platform and the computers in Tulipan.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.
- The project proposals might change slightly. For a updated document, go to: https://www.overleaf.com/read/xgqfysbtbcpd.

Introduction

This will be your final project before the end of the semester and will in total count 16% of your final grade. For this project, the TAs have all made different project proposals, but you can also propose your own project. During the project, your main communication will be through the advisor for your selected project.

For the project, we will have three deliveries. Each delivery is specified in detail below and they have the following deadlines:

1. **24. March, 2019 by 23:59 PM:** Project Proposal (1-1.5 pages).
2. **7. April, 2019 by 23:59 PM:** Presentation slides draft.
3. **28. April 2019 by 23:59 PM:** Presentation slides and final code.

Delivery

Project Proposal (2 points)

Deliver this as a PDF.

The project proposal will be a short document, 1 - 1.5 pages, describing what you want to do and what the goal of your project is. We want you to briefly go over the following questions:

- What is the goal of your project?
- What is the relevant literature for your project? (Previous papers, projects, etc)
- What model architecture are you going to use?
- What dataset are you planning to use? (If applicable)
- Are you writing your code from scratch, or are you basing it on some existing code? If you are basing your project on existing code, please link the github/code.

This delivery is designed to make you start early on your project and ensure that your selected scope is neither too small or large for this course.

Presentation Slides Draft (3 points)

Deliver this as a PDF.

The presentation slides should be delivered as a set of slides and can be considered as a starting point for your final presentation. We want you to cover the following points:

- What is the goal of your project?
- What is the relevant literature for your project? (Previous papers, projects, etc)
- What dataset are you using? (if applicable)
- What method are you using? (e.g: the chosen model architecture)
- Report the current results (if you don't have any results, please state why)
- What work remains before the presentation deadline? What will be your main focus?

This delivery is designed to make you start early on your project and your presentation. **All of the slides made in this delivery can be reused for the final presentation!**.

Final Presentation (11 points)

Students working alone will have 9 minutes to present, groups of two will have 10 minutes to present, and groups of three or more will have 11 minutes to present. **We will stop any group using more then the maximum presentation time.** What we want you to deliver is the following:

- Powerpoint slides as a PDF file
- All code in a ZIP file.

Note: There is no final report for this project, due to limited staff for grading these fairly. Therefore, your final project will be graded based on the final presentation, the delivered code, and the powerpoint slides.

The final presentation will be a significant part of your grade. I recommend you to start early and prepare yourself well for the actual presentation. For general guidelines for presenting a project, Professor Charles Elkan has some good advice.

Topics you should cover are the following:

- **Main goal of the project.** Shortly describe what you have been working on. Maybe show a illustration of the result of your method?
- **Previous work on this problem.** This should be very brief. Shortly summarize previous solutions.
- **The dataset.** Shortly summarize the dataset your are using.
- **The model architecture.** Describe the model you have used. If you used a previous architecture, explain what you changed. You should include the final hyperparemeters of your model.
- **Results.** The results should be clear and concise. When you are presenting the results, you will show the most knowledge of the subject by discussing the result and not reading up the results.

- **Discussion.** The discussion of your result and method should be a major part of your presentation. What worked and what did not? Were there any results that were unexpected? What modifications did you make that gave you significant improvement?
- **Conclusion/Summary**

Remember, when you are presenting, show your knowledge about the problem you have chosen and don't waste time on nitty, gritty details. Often students struggle with managing the time during the presentation and spend all their time on previous work/dataset/model architecture. This leads to a very rushed discussion section, making it hard for the evaluators to understand the work gone into the project and the student's understanding of the underlying concepts.

Project Suggestions

1 Your Own Project

Advisor: TBD

You are free to propose your own project. As this is a computer vision(CV) course (focusing on modern DL methods) the proposed project must be related to CV one way or another. Your project must be approved before you can start to work on it. This is not very complicated, just drop me a some lines regarding the envisioned content of the project. Send this to Frank Lindseth and Håkon Hukkelås. hakon.hukkelas@ntnu.no and frankl@ntnu.no.

2 Image-to-Image Translation with GANs

Advisor: Håkon Hukkelås (hakon.hukkelas@ntnu.no)

NB: I will not be in Trondheim March 12-23 and April 8-14. I will be available on mail in this period.

This project proposal will focus on generative models and more specifically, generative adversarial networks. The proposed project is to translate satellite images into street view images, as shown in Figure 2. If you want to work on a different project related to generative models, please send me a message such that we can approve the project!

2.1 Main Idea

[Isola et al., 2017] proposed an impressive network(Pix2Pix) that was able to perform image-to-image translation for several domains, including satellite images to street map images. They proposed a model based on generative adversarial networks, where they used a conditional input to both the generator and discriminator. They used an encoder-decoder architecture based on the U-Net network, originally proposed as a segmentation network for medical segmentation. A general u-net architecture is shown in Figure 1.

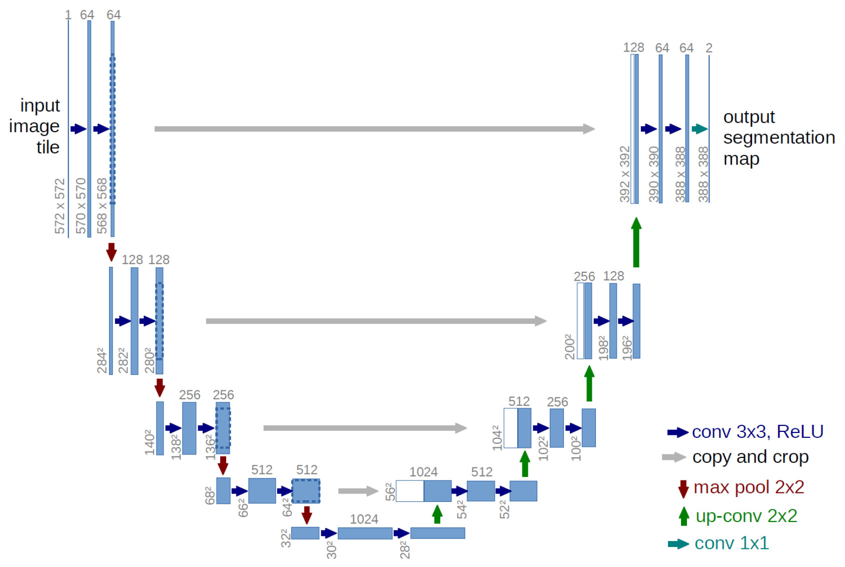


Figure 1: An illustration of the U-net architecture. Note, this is not the specific generator architecture used by [Isola et al., 2017], but a simple illustration explaining the main features of the U-net architecture.[Figure source].

In this project, I propose that you should base your model on the proposed solution of Pix2Pix and from this baseline, improve the network and training procedure. The main results of Pix2Pix were introduced in 2016 and since then, we have seen major advances in generative adversarial networks. The general training procedure of Pix2Pix is very unstable and somewhat outdated. Their loss function is based on the non-saturating loss originally proposed by [Goodfellow et al., 2014], but they introduce an additional reconstruction loss; the mean L1 loss over the whole reconstructed image. This additional reconstruction loss has shown to cause reconstructed images to be blurry, which might be diminished if we are able to train the network without the L1 loss. *The improved wasserstein loss*([Gulrajani et al., 2017]) is an alternative to the non-saturating loss and their proposed objective function has shown to be more robust to mode collapse. From empirical experiments, they show that the improved wasserstein loss is substantially more stable than the non-saturating loss.

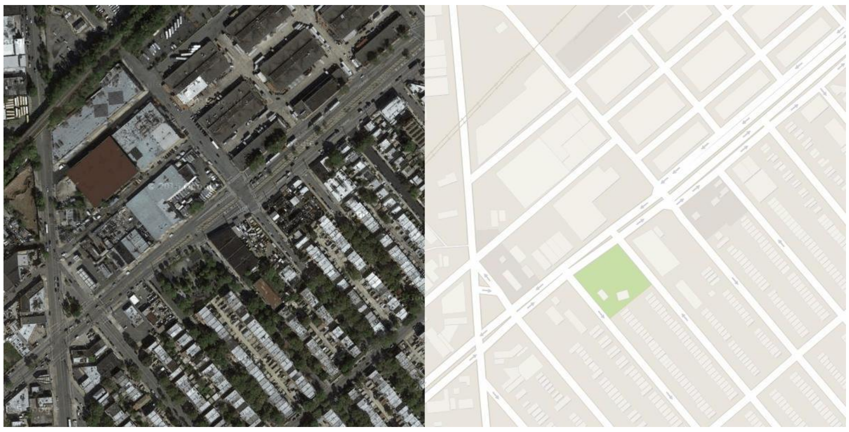


Figure 2: Example image from the Pix2Pix satellite to street maps images. The left input is the input image and the right image is the target street maps image.

One major contribution to training GANs is the method of *progressive growing of GANs* ([Karras et al., 2017]). Training the network progressively improve the overall physical training time and improve the stability of the generative adversarial network. Their basic idea is that they want to train the network with curriculum learning. Intuitively explained, this is that when a human learns about math and we want to learn how to multiply two numbers, we first have to learn how to perform addition of two numbers. They transfer this simple idea to GANs; initially, we want to learn about the basic shape and color of the image we want to generate. Then, we want to learn more high fidelity details, such as the color of eyes, or placement of teeth. They perform this by initially training the network to generate 4×4 images, then progressively grow this by increasing it to 8×8, 16×16, and so on until they reach 1024×1024.

2.2 Datasets

The two datasets I propose you to work with is the original dataset used in the Pix2Pix paper([Isola et al., 2017]) and the second dataset is a dataset scraped by me from the game Grand Theft Auto 5(GTA). You can see examples of the datasets in Figure 2 and Figure 3.

2.2.1 Pix2Pix dataset

The dataset from Pix2Pix can be downloaded from the author's website linked below. Images in this dataset are 600×600 pixels large and each sample includes an input satellite image and a target image that use solid colors and outlines to identify roads, buildings, urban areas and bodies of water. In total the dataset contains 2194 images, 1096 designated for training and 1098 designated for validation.

Due to the large image size and a rather small dataset, I recommend you to test and train your model on the GTA dataset initially.

2.2.2 GTA dataset

The GTA dataset is scraped from a high-resolution satellite, road and aerial maps from the game. These are pre-processed into smaller images. The images are generated from a game world, which makes the images and targets highly accurate. The dataset has a resolution of 128×128 and contains about 1300 images without any data augmentation. By introducing rotation, reflection, it is possible to increase the data amount significantly. To download the dataset, contact me and I'll send you the script.

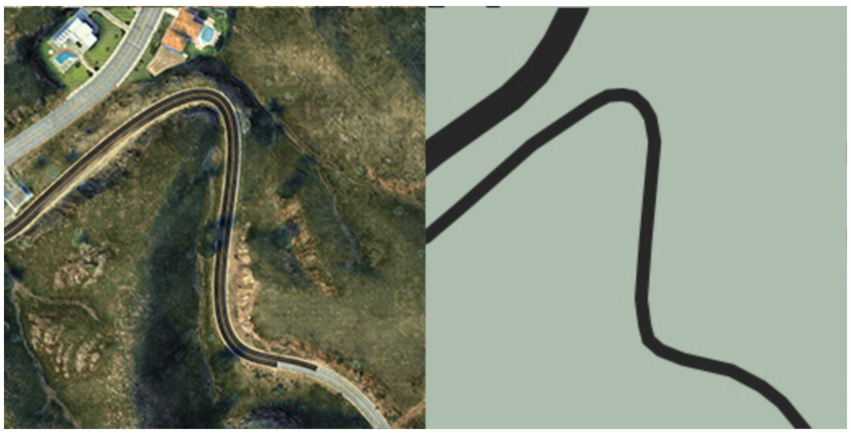


Figure 3: Example image from the GTA5 dataset. The left image is the input image (satellite) and the right image is the target image (street maps).

2.3 Recommended Readings

If you contact me, I can help you get started on a starter code. This github page contains the code for Pix2PixHD, which can be a very good starting point.

1. Image-to-Image Translation with Conditional Adversarial Nets, Philip Isola et. al. [CVPR 2017]
2. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, Jun-Yan Zhu et. al [ICCV 2017]
3. Progressive Growing of GANs for Improved Quality, Stability, and Variation, Tero Karras et. al. [Arxiv Preprint]
4. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs, Ting-Chun Wang et. al. [CVPR 2018]
5. Improved Training of Wasserstein GANs, Ishaan Gulrajani et. al. [NIPS 2017]

3 Autonomous Driving Using End-to-end Learning

Advisors: Hege Haavaldsen (hegehaav@stud.ntnu.no), Max Aasbø (mmaasbo@stud.ntnu.no)

In recent years, substantial progress has been made towards a vehicle's ability to operate autonomously. Primarily, two different approaches have emerged. The prevailing state of the art approach is to divide the problem into a number of sub-problems and solve them by combining techniques from computer vision, sensor fusion, localization, control theory, and path planning. This approach requires expert knowledge in several domains and often results in reasonably complex solutions, consisting of several cooperating modules.

Another approach is to develop an end-to-end solution, solving the problem using a single, comprehensive component, e.g., a deep neural network. While many believe that the black-box behavior of such systems makes them untrustworthy and unreliable, others point to the recent year's advances in deep-learning and argue that end-to-end systems are the future.

In this project you will explore the latter approach, building a deep neural network to achieve some level of autonomous driving.

3.1 Main Idea

The main idea of this project is to design a deep neural network that accepts an image (from a driver's perspective) as input and produces a proposed steering angle for the vehicle. By training the network on actual driving data, the model should hopefully learn to imitate a human driver.

Training and testing models for autonomous driving in the physical world can be expensive and impractical. Thus, for this project, you shall use a simulator to both train and test your model. A simulator can effectively provide the variety of corner cases needed for training, validation, and testing; while removing any safety risks and material costs.

3.2 Data Collection

In this project, you will need to collect your own training data. This is done by manually driving a vehicle in the simulator while capturing images from the driver's perspective along with the vehicle's associated steering angle. This data can then be used to train and validate your model using images' pixel values as input, and the associated steering angles as targets.

3.3 Model Architecture and Framework

A crucial part of this project is to choose a suitable network architecture for your model. It is strongly recommend that you research existing CNN-architectures used for autonomous vehicle control, and use these as a guide when building your own (e.g., the DAVE-II architecture proposed by NVIDIA).

You are allowed to use any machine learning framework to design and train the model. However, **it is recommended that you use Keras for this project** as the helper code provided only supports this framework.

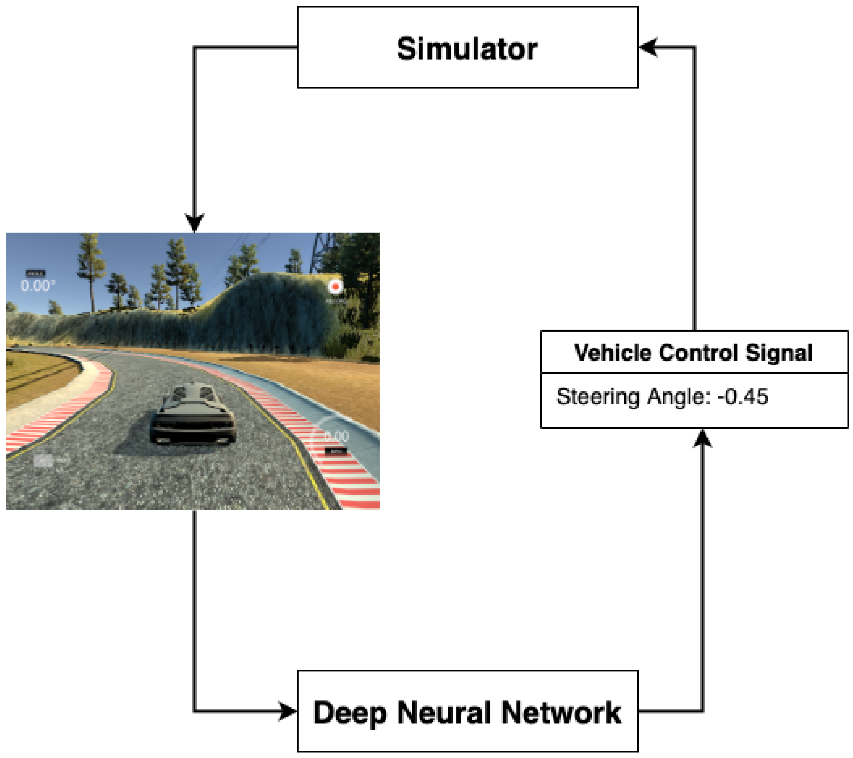


Figure 4: Information flow when real-time testing a model.

3.4 Testing your model

Since the calculated accuracy and loss after training provide little indication on the model's actual driving capabilities, the model should be real-time tested using the simulator. To achieve this, you will need to create an interface that feed the model images from the simulator, and send the produced steering angles back to the simulator (Figure 4). If you have chosen to use Keras and the Udacity simulator, this interface is provided (see section 3.6).

3.5 Simulator

For this project, it is strongly recommended that you use the Udacity Simulator. You will be able to use the provided interface to communicate with the simulator, and the project advisors will not be able to help you with any other simulator.

The Udacity simulator has two different modes: **training mode** and **autonomous mode**. In the training mode, you drive the car manually to record the driving behavior. You can use the recorded images to train your machine learning model. In the autonomous mode, you are testing your deep neural network to see how well your model can drive the car without dropping off the road / falling into the lake.

Specifically, in autonomous mode, the simulator is acting as a server that continuously streams road images, while listening for vehicle driving instructions. The custom interface (*drive.py*) can thus receive the road images from the simulator, predict driving instructions using your trained model, and send them back to the server.

To get started with the Udacity Simulator read the provided guide *README.pdf*.

3.6 Provided Code

If you have chosen the Keras framework and the Udacity Simulator following provided code can be used:

- *drive.py*: a Python script that you can use to control the simulator autonomously, once your deep neural network model is trained
- *video.py*: a Python script that can be used to make a video of the vehicle when it is driving autonomously

You can download the provided code, along with a guide to get you started, here.

3.7 Final Presentation

Your final presentation should include the following:

- The goal of your project
- A *brief* presentation of previous work on this problem
- Your model architecture
- Your model's hyperparameters
- The data augmentation preformed on your dataset
- A video showing your model driving a vehicle in a simulator
- A discussion of your result and method
- A conclusion/summary

3.8 Tips and Tricks

- Data augmentation is vital to achieve a stable driving behaviour.
- Will a model be able to take right turns if the dataset solely consists of left turns? A balanced dataset is crucial for good generalization.
- Does the entire road image contain useful information, or should you crop away irrelevant areas?
- A high loss on both the training and validation set could indicate underfitting. Possible solutions could be to increase the number of epochs or add more convolutions to the network.
- A low loss on the training set, but a high loss on the validation set, could indicate overfitting. A a few ideas could be to: use dropout or pooling layers, use fewer convolution or fewer fully connected layers, collect more data or further augment the data set.
- Does your model consistently drive off the track? It's important to feed the network examples of good driving behavior so that the vehicle stays in the center and recovers when getting too close to the sides of the road. How is the distribution of left and right turns in you data set? Does the training set contain any recovery data, so the model knows how to recover from a mistake? Have you performed enough data augmentation on the dataset?

3.9 Recommended Readings

1. End to End Learning for Self-Driving Cars
2. ImageNet Classification with Deep Convolutional Neural Networks
3. Solving the MNIST classification problem using Keras, achieving a test accuracy of 99.25 %
4. Introduction to Udacity Self-Driving Car Simulator

4 Object detection

Advisor: Åsmund Brekke (aasmunhb@stud.ntnu.no)

4.1 Main Idea

This project focuses on training an object detection architecture on a dataset of your choice. Since designing a object detection module from scratch is cumbersome, this project will focus more on how you configure the model to accomodate your new dataset.

In the last years, object detection using deep neural networks have achieved great results, outclassing previous feature-based methods such as HOG (Histogram of Oriented Gradients). Two popular architectures for object detection are Faster RCNN [Girshick, 2015] and the YOLO architectures by [Redmon et al., 2016]. The former uses a region proposal network (RPN) to generate regions of interest (RoI), while the latter divides the input image into a grid, and then predicts a number of bounding boxes for each cell.

Your task is to train an object detection architecture. Important things to consider might be how you decide your hyperparameters based on your specific dataset, your choice of data augmentation, the class distribution of your dataset (if doing multi-class object detection). In addition, it may be interesting to investigate the strenghts and weaknesses of your model. For example, is your model capable of detecting small objects, or does it struggle with a specific set of classes?

4.2 Datasets

4.2.1 Pneumonia detection

The RSNA pneumonia dataset is a dataset from the "RSNA Pneumonia Detection Challenge", which was a kaggle competition where 346 teams competed in detecting pneumonia from chest radiographs. The training set contains 30000 labeled images, but you do not have to use all images, since it might be too computationally expensive. To detect lung opacities is a very challenging task, as they may be very hard to see even for a trained professional (see Figure 5 for a sample image). If you are using YOLO, the code for converting the dataset to the expected format of YOLO can be found here.

4.2.2 Multi-class object detection with PASCAL VOC

The PASCAL VOC challenge is an object detection benchmark where the goal is to detect 20 classes such as person, cat, cow and aeroplane. In total, the dataset contains over 11000 images, with over 27000 labeled objects. A sample image from the dataset is shown in Figure 6. If you are intending to use YOLO or Faster RCNN you can easily get up and running using the setups found here and here. The dataset can be downloaded here.

4.3 Recommended Readings

If you intend to use YOLO, the official github page and this page are both good starting points. For faster-rcnn or mask-rcnn, the model zoo here can help you get started. Note that there exists multiple unoficcial implementations of these architectures, so it should not be an issue to find an implementation in your favorite framework. Different object detection architectures may require different format on label

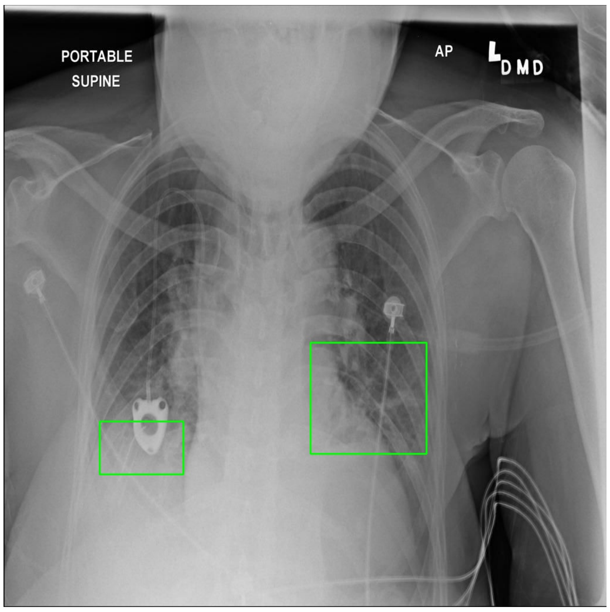


Figure 5: Sample image from the pneumonia dataset. The ground truth bounding boxes are marked in green.

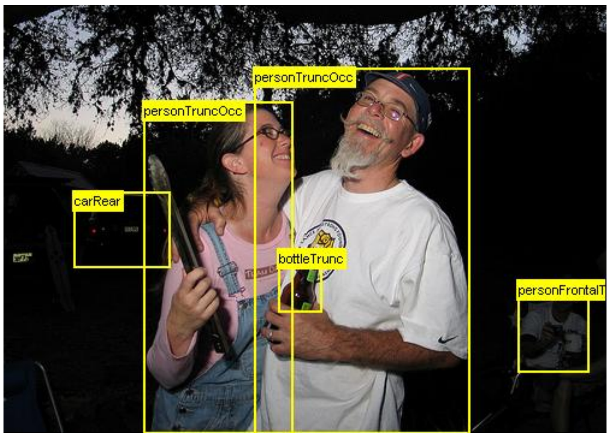


Figure 6: Sample image from voc dataset.

files for training, but it should not be to much of a hassle to convert the training labels for your dataset into your desired format.

- You only look once: Unified, real-time object detection, Redmon, Joseph, et al.
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Ren, Shaoqing, et al.
- PASCAL VOC 2012 Challenge description
- Tensorflow object detection API

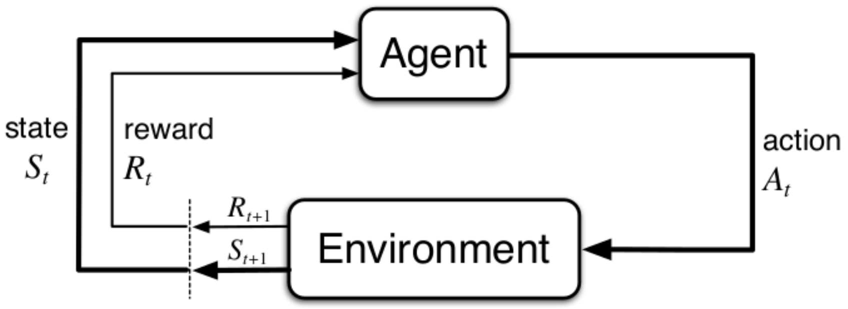


Figure 7: Reinforcement learning loop. Starting at time-step t, the agent observes the state s_t and reward r_t. Whenever an agent takes an action, a_t, the environment returns a new state, s_{t+1} and a scalar reward value, r_{t+1}, representing the state and reward in time-step $t+1$.

5 Deep Reinforcement Learning

Advisor: Marcus Loo Vergara (mmvergar@stud.ntnu.no)

Reinforcement learning is a sub-field of artificial intelligence that focuses on creating agents[1] that maximize an objective in some environment. Examples of problems where reinforcement learning is applicable, is, for example, the task of learning how to play a video game, learning how to use robotics locomotion to walk, or learning how to drive a vehicle. In this project, we will look into how modern reinforcement learning uses deep learning to train agents to e.g. play Atari games with only an input image and a reward signal. However, you are free to choose whichever reinforcement learning problem/environment you want as long as the underlying method uses deep learning in some capacity.

5.1 Main idea

5.2 Reinforcement Learning – Motivation

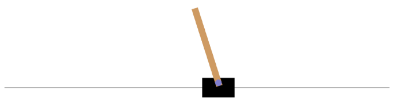
Reinforcement learning is a field that has been extensively studied over the last seventy years. It is, however, only recently that we have seen major leaps within the field as a direct result of the success of deep learning methods. Examples of agents that use deep reinforcement learning include DeepMind's AlphaGo, which in 2017 beat the number #1 ranked Go player, OpenAI's Dota 2 agent that has repeatedly beat multiple high-level players and teams in showcase matches, to simpler but equally impressive things, such as training humanoid agents and real-life robots how to walk and maneuver. In reinforcement learning, the agent learns to how behave through its experiences, and it is a framework that intuitively emulates how we imagine we learn ourselves.

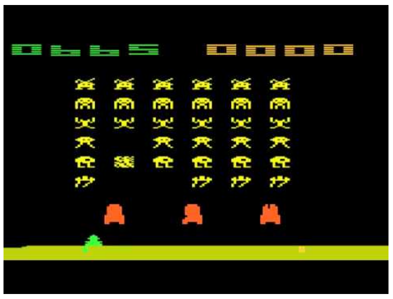
5.3 Deep Q-learning

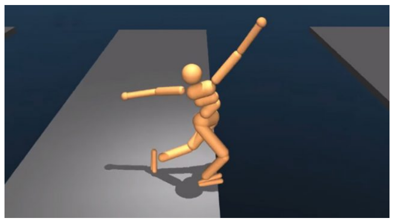
Deep Q-learning is a method that builds on the classical reinforcement learning algorithm, known as Q-learning, or *quality* learning. In Q-learning we iteratively update a quality function, denoted $Q(s_t, a_t)$. This is a quantity that represent *how good* taking action a_t in state s_t is, and is dynamically updated with the help of the Bellman equation[2]:

[1]Please refer to Section 5.4 if for reinforcement learning terminology.

[2]For further explanation Q-learning and the Bellman update equation, check this video, or Chapter 6.5 from *Reinforcement Learning: An Introduction* by Sutton and Barto if you prefer.


(a) CartPole-v0: The task here is to balance a pole on a cart that can move either left or right


(b) SpaceInvaders-v0: Atari's classical Space Invaders game


(c) MuJoCo, or Multi-Joint dynamics with Contact; an environment for RL-based locomotion problems

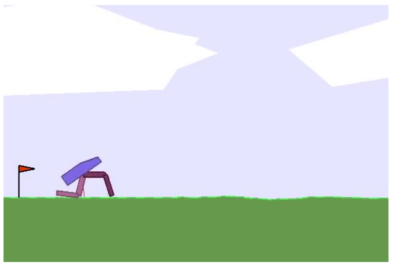

(d) BipedalWalker-v2: A simple locomotion problem in OpenAI gym

Figure 8: Some example environments

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{1}$$

Normally, $Q(s_t, a_t)$ stored in a $|A| \times |S|$ table, where each column represents an action, and each row represents a state. Over time, and through sufficient exploration of the environment, we should arrive at a table that accurately approximates the quality of every possible state-action pair of the environment. However, maintaining such a table for environments with a high number of states (possibly an infinite number,) or a high number or continuous action-spaces is often impossible.

Deep Q-learning seeks to fix the state-space limitation by approximating a Q-table with deep learning. Given a state, we try to fit a function, $f(s) \approx Q(s, a)$, that accurately estimates the quality of each action given a state s, and if this function is accurate enough, we can simply pick the action with the highest Q-value as the optimal action. Note that the state in deep Q-learning is often represented as an image – taken as the agent is acting in the environment. The algorithm uses only these images coupled with a reward signal to learn how to play. Please review the resources listed in Section 5.5 for more in-depth explanations of Deep Q-learning and other RL topics.

Implementation tips:

- You can use a *deque* to stack frames and to manage a replay buffer.
- Use a ϵ-greedy exploration strategy.
- Do preprocessing to reduce the dimensionality of the input images (common preprocessing step is turning the images into greyscale and resizing to 84x84,) and make sure that the images capture all the information in the image that is relevant for the AI to learn properly.
- Prototype in a simple environment! CartPole (part of the OpenAI gym Python package) is the

simplest one, but it does not output images by default. I recommend trying a fully-connected architecture in this environment to verify that your method is working correctly.

- Plot or print the total episode reward over the duration of the training. Your agent is working correctly if this value is increasing over time. Note that a greedy strategy ($\epsilon = 0$) should be used when *evaluating* the agent's performance.
- Once it is working correctly, try to run the method on a more complicated environment e.g. Atari's SpaceInvaders or in VizDoom's basic scenario.
- A simple, 3 layer convolutional or fully-connected should suffice.
- Leave a good amount of time for training; training DQN to convergence may take a full 24 hours!

Be aware that DQN only works with discrete action spaces – meaning each action the agent takes is represented by either True or False (e.g. button presses – can be several buttons.) If you wish to train an agent that works with continous action spaces (e.g. if you want to train a MuJoCo agent,) I would recommend looking into either Deep Deterministic Policy Gradient (DDPG) – an algorithm that enables a DQN-like approach for continuous action spaces – or potentially looking into Proximal Policy Optimization or Soft Actor-Critic, both of which are current state-of-the-art deep RL methods.

For the final presentation, I recommend including:

- A video showing how the the final agent behaves. Additional clips showing how the agent learned over time would also be good to include.
- A graph showing the evaluation reward over episodes (use a greedy policy for evaluating.)
- Analysis of training time and agent behaviour.
- Explanation of your method (w/ modifications and experiments) and some words on limitations.

5.4 Terminology

Here is a short overview of over some useful terminology used in reinforcement learning.

- **Agent**: Some entity that we are able to control and issue **actions** to. E.g. a Roomba vacuum cleaner is an agent living in a vacuum cleaning **environment**.
- **Environment**: The world in which the **agent** lives and can act in. E.g. Atari's *Breakout* is considered an environment.
- **State**: Some representation of the state environment. Denoted s_t for the state at timestep t. This could for example be an image from the agent's point of view, or the positions, velocities, etc. of all objects in the environment.
- **Action**: Some command signal that makes the agent act in the environment. Denoted a_t for action at timestep t. Examples: *move left*, *fire*, etc.
- **Reward**: A signal telling the agent how useful **action** a_t was in state s_t. Reward is denoted by r_t for the reward received at timestep t. In a game, this signal would correspond to the increase of score after taking the action.
- **Return**: Total (or discounted) sum of **rewards** starting at some time step. Denoted $R(t)$.
- **Episode**: A single run of the agent until it reaches a **terminal state**.
- **Value**: Represents the agent's **expected return** starting in a particular state s_t.

- **Quality**: Represents the agent's **expected return** when taking action a_t in state s_t.
- **Policy**: The parameters that determine how the agent acts when given a state s. Typically denoted π_θ where π means "the policy" and θ are the parameters/weights of the network.

5.5 Further Reading

1. OpenAI Spinning Up – Key Concepts in RL: Quick and simple introduction to RL concepts
2. YouTube – Arxiv Insights – An introduction to Reinforcement Learning: High-level overview of the landscape of RL
3. YouTube – RL Course by David Silver: Video lectures by David Silver (co-author DQN paper)
4. Getting Started with OpenAI Gym in Python
5. Medium – An introduction to Deep Q-Learning: let's play Doom
6. Playing Atari with Deep Reinforcement Learning (DQN paper) by Minh et.al.
7. Continous Control With Deep Reinforcement Learning (DDPG paper) by Lillicrap et.al.
8. OpenAI Spinning Up – Deep Deterministic Policy Gradient: Implementation details of DDPG and other algorithms (however, no DQN)

6 Medical Image Segmentation

Advisors: Jenny & Mathias P (jennysj@stud.ntnu.no & mathiaap@stud.ntnu.no)

6.1 Main Idea

Medical imaging is concerned with creating images of the inside of the human body. This includes both internal structures and its functioning. The goal is to map the structure of organs in the body or identify abnormalities in order to diagnose and treat disease. In medical image segmentation, the different pixels or voxels of an image is assigned to one of several predefined classes.

U-Net is a type of fully convolutional neural networks commonly used for medical image segmentation. It was originally developed by Ronneberger et al. in 2015. [Ronneberger et al., 2015]. In recent years U-Net has been developed further, e.g. by addition batch normalization, strided convolutions and how upsampling is achieved. In this project you should experiment with one (or more) architectures based on U-Net.

Find and modify a model suited to your dataset. Play with the hyperparameters and see how it affects the results. You should also experiment with augmentation methods as the number of training images is often insufficient for generalization.

State in your report all the existing code you are using in your project.

6.2 Datasets

For this project you must pick a dataset consisting of medical images.

Example datasets can be found at https://goo.gl/QzVZcm. We recommend using the Lung dataset or the Heart dataset, but you are free to select any medical dataset of similar complexity.

For this project it is sufficient to perform binary segmentation using 2D slices from the 3D images. However, you are free to experiment with using multiple slices as input or even use 3D volumes for segmentation.

In most medical image datasets the majority of pixels are labelled as background. A loss function that takes class imbalances into account is necessary, otherwise the model may simply ignore the less frequent classes. Loss functions like Jaccard Dice or weighted cross entropy work well for this type of problem. Please include the Dice score of the training set and the validation set in your delivery.

The tool ITKSnap can be used to visualize 3D images. See figure 9.

6.3 Recommended Readings

1. U-net: Convolutional networks for biomedical image segmentation, Ronneberger et. al.
2. An Introduction to Biomedical Image Analysis with TensorFlow and DLTK (Focus on basics, .nii files, normalization, and augmentation)
3. Review: U-Net (Biomedical Image Segmentation)

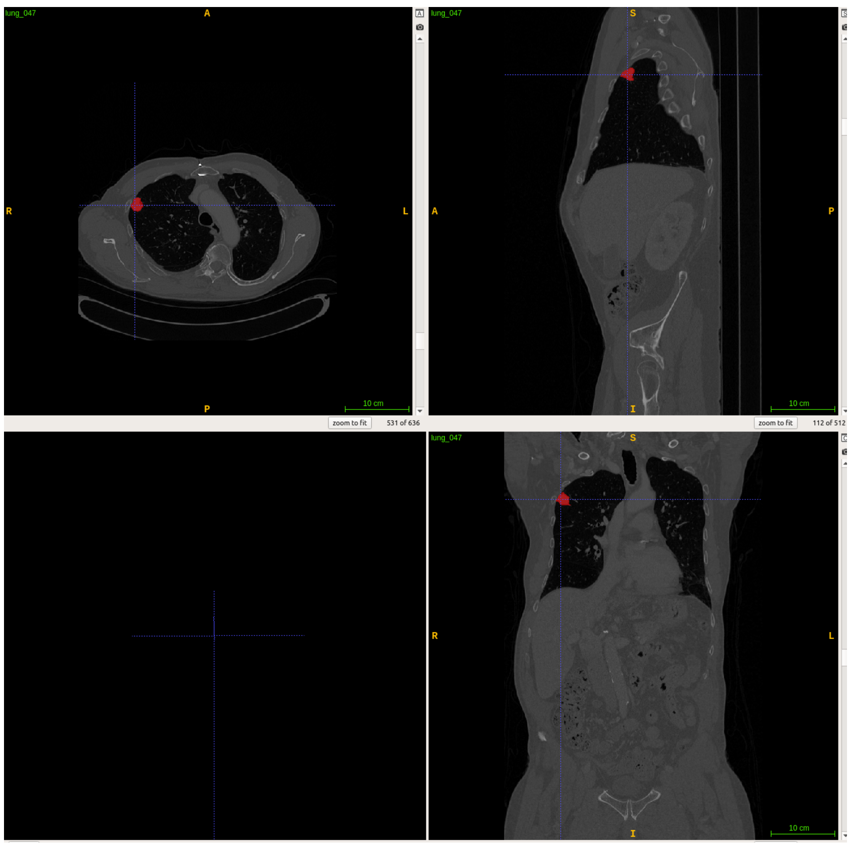


Figure 9: Ground truth label of lung cancer tumor shown in ITKSnap.

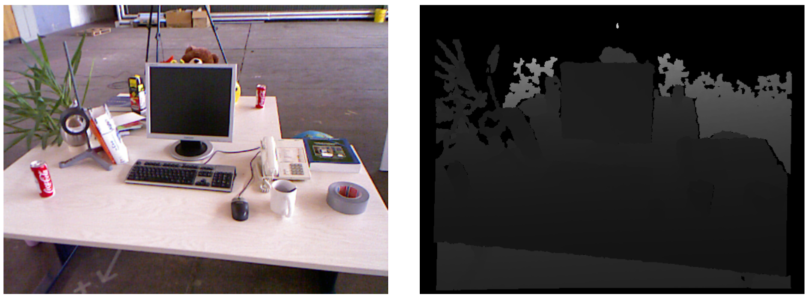

(a) RGB Image (b) Depth Image

Figure 10: RGB-D Dataset Example

7 Visual Odometry

Advisor: Matias Christensen (matiasvc@stud.ntnu.no)

Odometry is the use of data from sensors to estimate the ego-motion of an of an object. Visual Odometry is motion estimation with cameras as the primary sensor. It is widely used in autonomous vehicles, robotics and AR/VR and is a basis for almost all modern SLAM(Simultaneous localization and mapping) methods. Both are active areas of research both in academia and industry. An example of a state-of-the-art Visual Odometry algorithm can be seen in this video.

7.1 Main Idea

For this project you will implement a simple visual odometry system using a dataset of RGB-D sequences, recorded with a Kinect sensor. This means that the image contains a depth channel in addition to RGB, as shown in figure 10. Your system will estimate the translation and rotation of every frame of the video, in relation to the first frame. The datasets contains a groundtruth trajectory that can be used to visualize the accuracy of the estimated trajectory.

There are a wide variety of solving this problem. You will be provided with Python starter code for a KLT-tracker based implementation, but you are free replace some, or all, parts of the algorithm steps with alternative methods. You are also free to implement this project in C++. I can provide guidance on how to implement the algorithm in C++, but you must write all of the code yourself as no C++ starter code will be provided.

The steps of the KLT-tracker based Visual Odometry system is as follows:

- **1: Harris Corner Detection:** This is used to find corners in the image, which are easily distinguishable points that can be tracked over multiple frames. This step is not run on every frame, but is instead run every k frames to replace tracked points that may have been lost in step 2. You will receive some helper code, but you must implement the majority of this step yourself.
- **2: KLT(Kanade–Lucas–Tomasi) Feature Tracker:** Given two frames, frame N and frame $N+1$, and a set of points in frame N, this method estimated the movement of each point in the image between frame N and frame $N+1$. If no match is found, then the point is discarded and replaced in step 1 to keep a sufficient number of tracking points at all times. You will receive some helper code, but you must implement some of this step yourself.
- **3: 2D to 3D projection:** For each frame you must project the tracked 2D points into 3D space. This is done using the intrinsic parameters of the camera in the dataset, and the depth from the

depth channel of the RGB-D image. You will receive some helper code, but you must implement the majority of this step yourself.

- **4: Transformation Optimization:** Given the projected 3D points found in both frame N and $N+1$, an optimization algorithm is used to find the translation and rotation that best describes the transformation between the two frames. We can than calculate the transformation of frame N as the product of the transformation of all preceding pairs of frames: ${}^{N}_{0}T = {}^{N}_{N-1}T \cdot {}^{N-1}_{N-2}T \cdots {}^{1}_{0}T$. You will receive the majority of the code for this step, but may need to implement some minor parts of it yourself.

In addition to this, you will receive helper code to load the dataset and the visualize the results. As well as a document describing the methods you will need to implement in more detail that is done here.

7.2 Dataset

For this project we will be using the RGB-D SLAM Dataset from The Technical University of Munich, which can be found here. The starter code will contain a script to download the sequences to the correct location.

The dataset contains multiple RGB-D sequences with a variety of environments and movements, all containing a ground-truth.

7.3 Recommended Readings

- Wikipedia: Harris Corner Detector
- YouTube: HARRIS Corner detector explained
- Original Harris Corner Detector Paper
- UiO Video Lecture (Norwegian): Corner Features
- UiO Lecture Notes: Corner Features
- YouTube: KLT-Tracker Lecture
- CMU Lecture Notes: KLT Tracker
- UiO Video Lecture (Norwegian): Pose in 2D and 3D
- UiO Lecture Notes: Pose in 2D and 3D
- UiO Video Lecture (Norwegian): Basic projective geometry
- UiO Lecture Notes: Basic projective geometry
- UiO Video Lecture (Norwegian): The perspective camera model
- UiO Lecture Notes: The perspective camera model
- Interactive Numerical Optimization Tutorial

References

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

[Isola et al., 2017] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

[Karras et al., 2017] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.