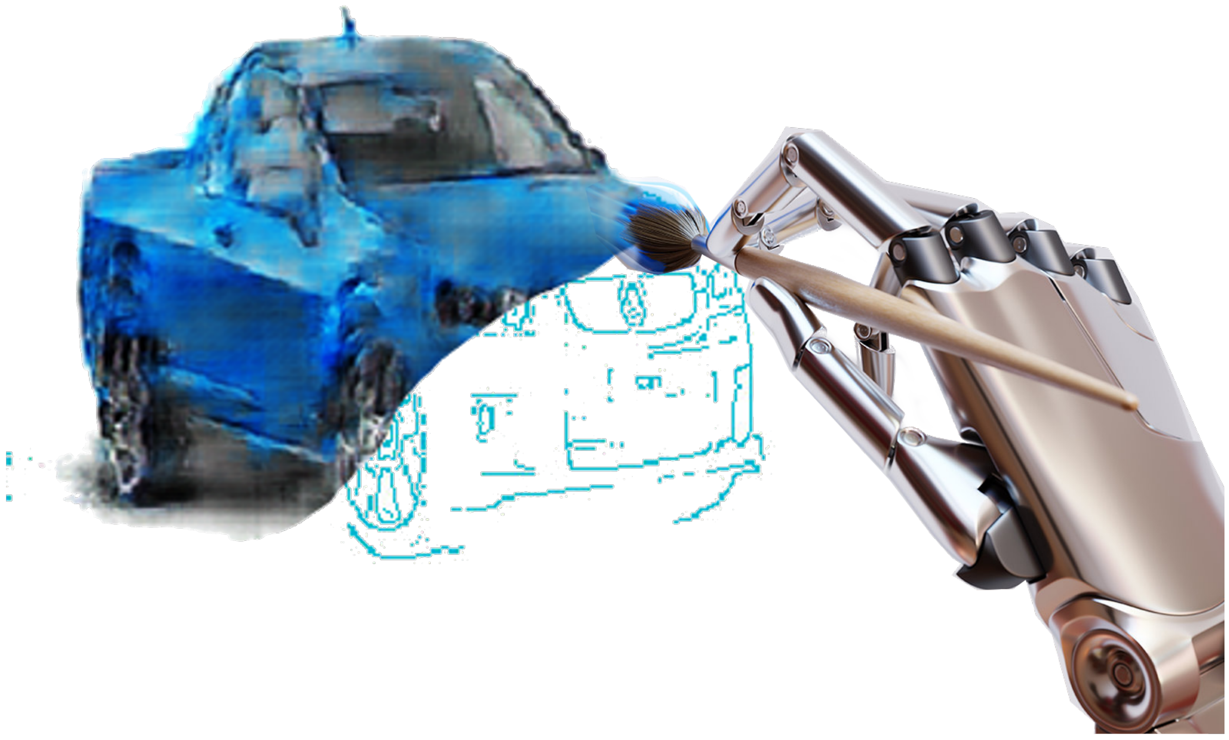
University of Dundee

Internship report
Generative Adversarial Networks



Student : David Albert

Supervisor : M. Jianguo Zhang

Tutor : M. Arnaud Knippel

Contents

Acknowledgments

First of all, I would like to thank my internship supervisor Mr Jianguo ZHANG for accepting me as a trainee and giving me the opportunity to work in the field of machine learning area. I would like to thank him for showing me and letting me discover machine learning applications for computer vision. Now, I have a global vision of this topic.

Otherwise, I would like to thank Mr Mahamadou Niakaté, computing specialist, for the time he spent in order tp allow me to work in the best conditions. Thanks for altering setups of my university account when it was necessary.

I especially thank my co-worker Mr George Chen for advices and references he gave me to learn by myself.

Eventually, I would like to thank all my colleagues for their kindness and the administration members for their hospitality.

Summary

Technician internship is a 12 weeks long training period that every third year student in mathematics department has to carry out. This internship allows students to put forward their skills in mathematics and computer sciences. Moreover, it allows to get new knowledges. I carried out my internship in an academic laboratory in computer vision and image processing. In this lab, searchers study machine learning techniques for computer vision. Most of them work on recognition of images for medicine anomaly.

Concerning my topic, I had to work on generative adversariel networks which is a new topic that my supervisor wanted to study. He intended me to study a particular application, called image to image translation.

I spent the first part of my internship listening machine learning courses, reading papers, and make some small applications in order to get necessary skills for futur projects. During this period, I learned machine learning's theoritical concepts from simple artificial neuron to deep artificial neural network. In the same way, I studied programming concepts, mainly tensorflow library, or technical concepts as parameters choice and tricks for train a neural network. As it was an important topic for computer vision and thus for my internship, I tried to make a small project about it. During my internship I was mainly focused on convolutional neural network and generative adversarial network. The first part of my internship report explains both.

The second part of my internship consisted of making two projects. Firstly, I worked on implementing a method to estimate human flow density on a picture. Then, in accordance to *"Image-to-Image Translation with Conditional Adversarial Networks"* paper [2], I applied a *pix2pix* architecture to transform car drawings into realistic car image. They are both explained in the second and third part of this report.

University Presentation

I carried out my technician internship at the Univeristy of Dundee, in Scotland. I intended discover AI research area. Therefore, find this intership about machine learning at the University of Dundee was a wonderful opportunity for me to acquire new skills in recent search area. Moreover, it was the opportunity to improve my english.

Figure 1 – Queen Mother Building

The university of Dundee was built in 1881 and is now hosting more than 17 000 students, including 25% of foreign students. The university is ranked as one of the world's top 200 universities according to Times Higher Education's World University Rankings and one of the world's top 100 universities for life sciences departement. In fact, life science and medicine research is very advanced. Otherwise, the university has an excellent sport facility, a huge library and some other student life facilities.

The place where I worked was the Queen Mother Building (fig. 1).The building is dedicated to computing science. The building architecture reflects both creativity and technicalities that require computer programming. I worked in an open-space area with PhD students and searchers. All of them were working on machine learning for computer vision. Besides, some of them were also beginners in this field.

Part I

Acquisition of knowlege and putting into practice

1 Machine Learning Overview

Machine learning is a huge topic. Way to use machine learning is multiple. That is why I will not explain machine learning from scratch but only what we need to know about this topic.

In machine learning problems, we consider n data $\{X_i\}_{i=1..n}$ and their associated results (or labels) $\{Y_i\}_{i=1..n}$. Having a new data X′, we want to predict the adequate and unknown result Y′. Let's call $(X, Y) = \{(X_i, Y_i)\}_{i=1,n}$, the learning set. The purpose is to estimate a non-trivial function f which provides the relationship between data and predictions.

Let's call f the function to be estimate, $f : E \to F$.

Given n examples $(X, Y) = \{(X_i, Y_i)\}_{i=1..n} \in E \times F$ and a model f_θ, we want to find θ, vector of parameters, which allows to minimise a cost function $\mathcal{L}$. In other words, we try to find θ such as $\theta = min_\theta\,(\mathcal{L}(\theta))$, where :

$$\begin{aligned}\mathcal{L}(\theta) &= \mathcal{L}(\theta_0, \theta_1, .., \theta_p)\\ &= ||f_\theta(X) - Y||_2^2 = \textstyle\sum_{i=0}^n |f_\theta(X_i) - Y_i|^2 \Leftrightarrow \mathcal{L}_2 \text{ loss}\\ &= ||f_\theta(X) - Y||_1 = \textstyle\sum_{i=0}^n |f_\theta(X_i) - Y_i| \Leftrightarrow \mathcal{L}_1 \text{ loss}\\ &= -\left[\tfrac{1}{m}\textstyle\sum_{i=1}^p y^{(i)} log(f_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - f_\theta(x^{(i)}))\right] \Leftrightarrow \text{log loss}\\ &= \dots\end{aligned}$$

To solve this problem, we can compute an iterative algorithm like gradient descent (1).

Algorithm 1 Gradient Descent algorithm

```
1: procedure GRADIENT_ DESCENT($x, y, \epsilon, \alpha$)          ▷ $\alpha$ the learning rate
2:     while $\mathcal{L}(\theta) > \epsilon$ do          ▷ We have the answer if $\mathcal{L}$ is lower that $\epsilon$
3:         for $j \leftarrow 0$ to $p$ do
4:             $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{L}(\theta_0, ..., \theta_p)$
5:     return $\theta$
```

NB : For each input, machine learning problem often consists in predicting a discret value (classification problem) or a continuous value (regression problem).

Examples of learning functions

- *Image Recognition* : X pixels of images and Y the object's name (1 : dog, 2 : table, etc)
- *Gender of anyone* : X features (height, weight, etc), Y = { Man, Woman }
- *House cost* : X (existence, surface area, location, etc), Y = cost

2 Image classification

The aim of image classification problems is to classify images into several groups corresponding to what they represent. For this task, we will see that the most powerful type of neural network are Convolutional neural networks.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNN) are a family of neural networks built for image classification. This type of network is famous since AlexNet network (a CNN) won the ILSVR competition vision with a very high score. Afterwards, CNNs had a huge success. The idea of CNN is that it compares parts of input images one by one and extracts important features. Indeed it is easier to detect similarities in smaller image areas than in the full images. A CNN consists of multiple layers which can be divided into 2 units. The first unit works as a "feature extractor". For that, the network uses convolution filtering operations with diferant kernels. Concerning the second unit, it is a basic connected network for classification task.

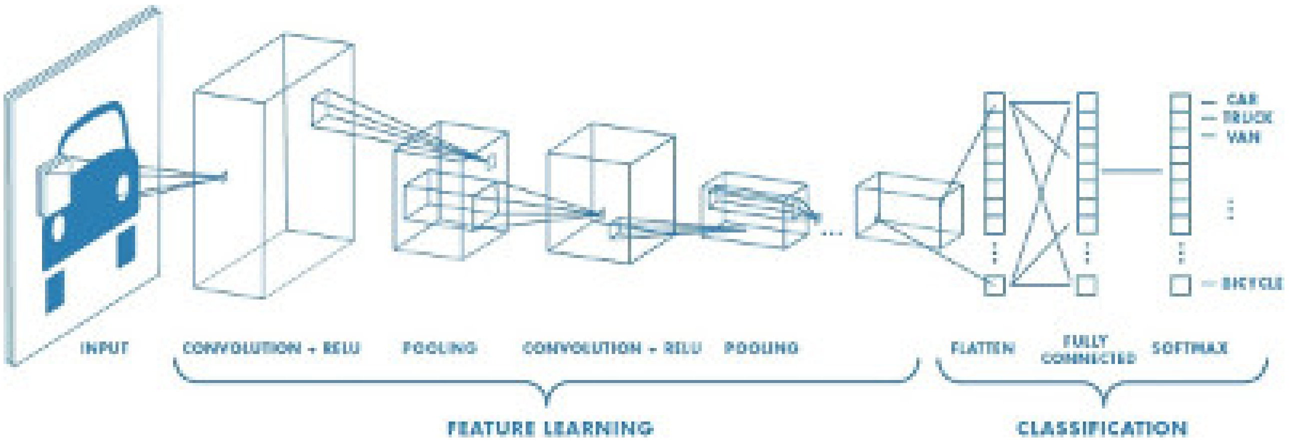


Figure I.1 – CNN architecture

Layers Description

It is usual to speak about Deep Learning when using CNNs because they have a lot of layers. Let me briefly describe each of them.

Convolutional layers

Convolutional layers have an image as input and calculate its convolution for each filter. These layers allow to find image features thank to the convolutional method shown in fig.

I.2. Features are learned during the learning step by gradient backpropagation.

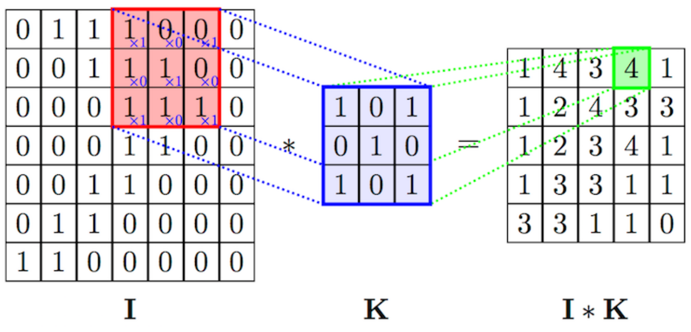


Figure I.2 – Convolutional Layer

ReLU layers

Rectified Linear Units (ReLU) layers apply $f(x) = max(0, x)$ for every value of the input matrix. They keep CNN in accordance with the maths model by replacing negative values by zero. There are other activation functions which can be used instead of ReLU. However, for CNNs, it is a reasonable choice to use ReLU.

Other activation functions : $sigmoid \Leftrightarrow \frac{1}{1+\exp^{-x}}$ | $tanh \Leftrightarrow \tanh(x)$ | $Leaky\_ReLU \Leftrightarrow \max(0.01x, x)$

Pooling layers

Pooling layers are very important. They allow to reduce the number of pixels. Thus, they improve the effectiveness of the algorithm which has less parameters to learn. Moreover, they drop overfitting probability and make position and orientation of the features less important for prediction.

Max pooling (fig. I.3): Usually, we use maxpooling to reduce image size. It means that we keep the pixel with the biggest value.

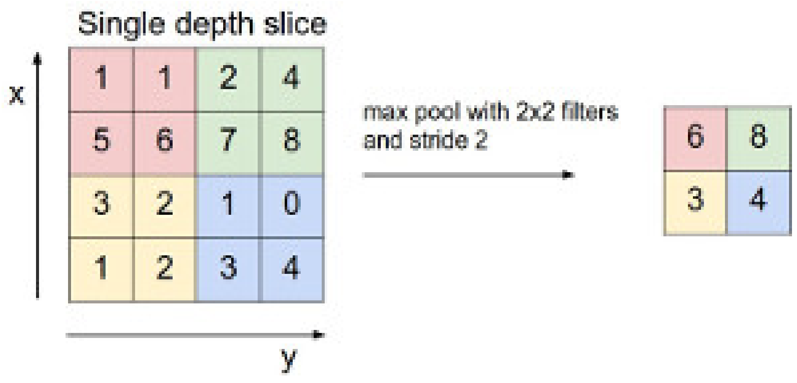


Figure I.3 – Max Pooling layer

Fully-connected layer

These layers are last in CNN. Fully-connected layers, in opposition to previous layers, are not specific for CNNs. They get a vector as input and provide another vector of likelihoods. Each unit of the output vector corresponds to the likelihood that the input data x is from the i^{th} class.

Build a CNN

There are several question to wonder to build a CNN. How many features (and how many pixels each) for convolutional layers ? What is the length of pooling window ? How many layers for each kind ?

Hyper-parameters of Convolutional layers

We consider some parameters K, F, S and Z :

K - Number of filters (= features)

F - Size of filters

S - Gap of filters' window

P - Zero-padding = number of pixels added all around of the image

NB : Zero-padding avoid to quick lose picture information

For each image of dimension $W * H * D$, outputs of convolutional layer are of dimension $W_c * H_c * D_c$ where :

$$W_c = \frac{W - F + 2P}{S} + 1$$

$$H_c = \frac{H - F + 2P}{S} + 1$$

$$D_c = K$$

Thus, if $P = \frac{F-1}{2}$ and $S = 1$, the output have the same width and height than the input. In practice, we often build such architectures. Moreover, it is usual to take $F = 3$, $P = 1$ and $S = 1$ or $F = 5$, $P = 2$ and $S = 1$.

Hyper-parameters of Pooling layers

F - Size of pooling filter

S - Gap of filter window

For each image of dimension $W * H * D$, outputs of pooling layer are $W_p * H_p * D_p$ dimension, where :

$$W_p = \frac{W - F}{S} + 1$$

$$H_p = \frac{H - F}{S} + 1$$

$$D_P = D$$

. Usually, we choose $F \in \{\mathbf{2}, 3\}$ et $S = 2$.

2.2 Small project : Yu-Gi-Oh card recognition

In order to apply what I learned about CNN and TensorFlow, I started by implementing a small Deep Learning API, Code 2 & 3 p.35. The goal of this API is to provide a way to solve usual machine learning problems easier in reusing some already implemented classes. This project was a good start for me to improve my python and TensorFlow knowledge. Afterwards, I used the Classifier1D class with a VGG network to train a Yu-Gi-Oh card classifier. That is to say, a network which is able to determine if an image is a such card or not. Why did I train a Yu-Gi-Oh card classifier ? Because I had downloaded some of them in another small project (see DCGAN project, p.16).

Results

As I thought, for a simple task like that, VGG network provide a very high accuracy. On the test dataset componed by 77 items, VGGnet makes a good classification for 76 of them. So that the accuracy is 98.7%.

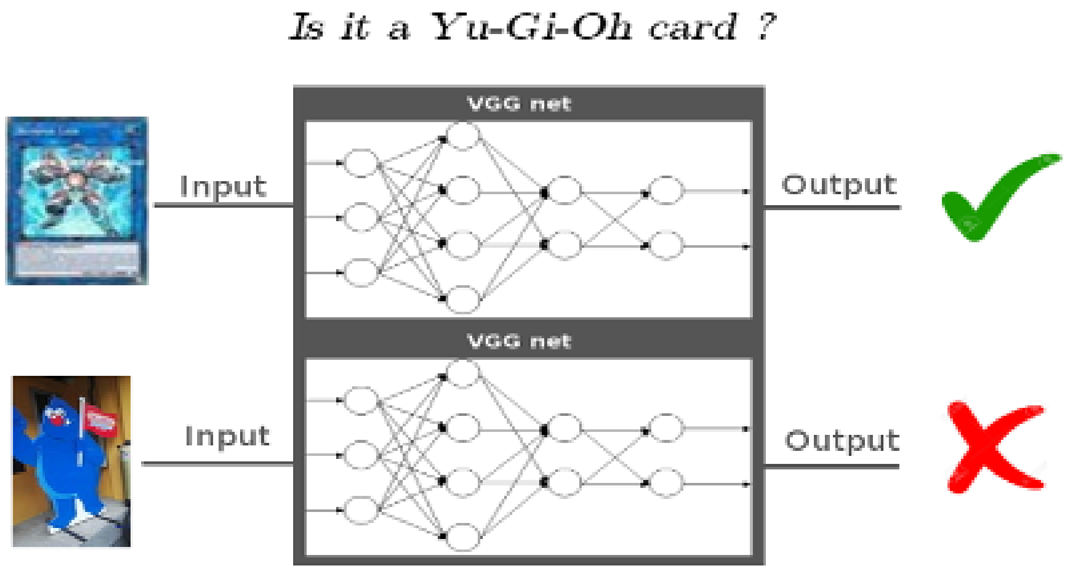


Figure I.4 – Classifier 1D

3 Generative Adversarial Network

The main topic of my internship was related to Generative Adversarial Network (GAN). GANs were introduced by Ian Goodfellow in 2014. This is a very interesting new concept of artificial intelligence algorithm which is used in different image processing problems. GANs are usually used new images generation. Let now see the theoritical aspect of generative models and more especially GAN.

3.1 Generative Models

In generative models, given learning data, we want to generate new samples from the same and unknown distribution.

We consider $x_1, x_2, ..., x_n$, n samples of the same random variable X, $X \sim P_{data}(x)$. $P_{data}(x)$ is an unknown distribution. x_i is the i^{th} learning data. And we consider G, another random variable, such as $G \sim P_{model}(x)$. Each generated sample is a sample of G. We want to have $P_{model}(x)$ as close as possible to $P_{data}(x)$.

Examples of data :

Data	Discrete representation	Tensor's Dimension
Numerical value	$\mathbf{R}$	0-Tensor
Sound / Function	$\mathbf{R}^{freq}$	1-Tensor
Music	$\mathbf{R}^{freq} \times \mathbf{R}^{time}$	2-Tensor
2D Image	$\mathbf{R}^{width} \times \mathbf{R}^{height} \times \mathbf{R}^{color}$	3-Tensor
3D Image	$\mathbf{R}^{width} \times \mathbf{R}^{height} \times \mathbf{R}^{depth} \times \mathbf{R}^{color}$	4-Tensor
2D Movie	$\mathbf{R}^{width} \times \mathbf{R}^{height} \times \mathbf{R}^{color} \times \mathbf{R}^{time}$	4-Tensor

Explanations : We consider samples from a random variable X. X follows $P_{data}(x)$ which is unknown. All of the samples seem to take randomly numerical values between 0 and 1. It makes sense to think that these values are distributed according to the Uniform distribution in [0,1]. The main purpose of generative models is to build a likelihood density $P_{model}(x)$ close to $P_{data}(x)$. In practice, $P_{data}(x)$ will be complex. Thanks to generative models, we are now able to generate new samples from X.

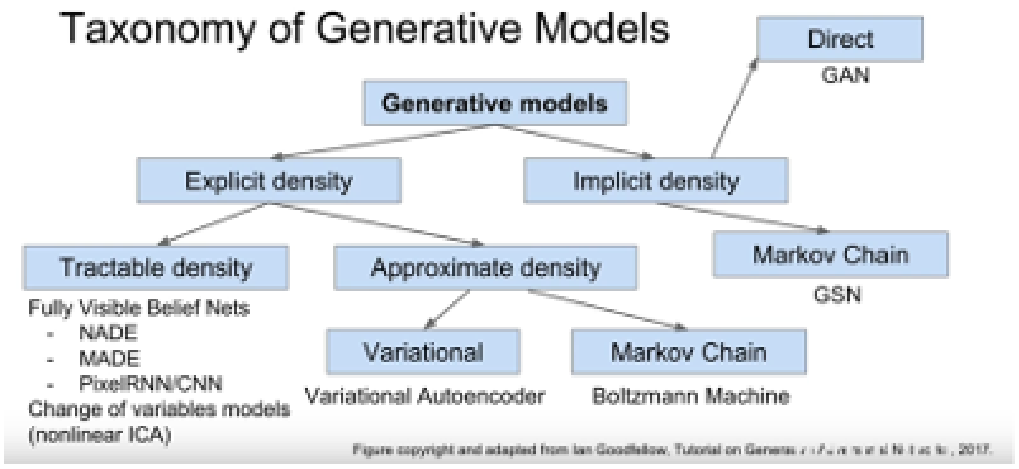


Figure I.5 – Generative models Overview

We can use generative model to :

1) define and solve explicitly $P_{data}(x)$
2) learn a model to produce samples from $P_{model}(x)$ without explicit density

Let's now speak about one type of generative model, its derivatives, and its applications.

3.2 GAN

Generative Adversarial Networks are a family of generative models (fig. I.5). They can be defined as implicit. In fact, GANs enable to generate random samples from an unknown distribution without define explictly an approximation of this distribution. Concretely, for this kind of model, the algorithm learns a transformation from a simple distribution sample (e.g random noise or uniform) to a more complex distribution sample. Neural networks are a powerful method to learn an unknown and non linear function. That's why we will use artificial neural networks to represent this complex transformation.

GANs training

We can say that GANs are amazing because of their particular neural networks training. This model is compouned of two neural networks which will be trained together. The first network, called generator, allows to generate samples from $P_{model}(x)$. Its aim is to generate samples the closest possible of $P_{data}(x)$. As for the second network, called discriminator, it allows to distinguish samples from $P_{data}(x)$ and samples from $P_{model}(x)$.

Training period will look like a 2-players zero-sum game where generator and discriminator networks are players and both try to maximize its reward by minimizing opposite reward.

Goal for each "Player":

- **Player 1 = Discriminator network** : Try to distinguish real and fake samples (images) from the generator.
- **Player 2 = Generator network** : Try to fool the first player by generating samples similar to data (real-looking images).

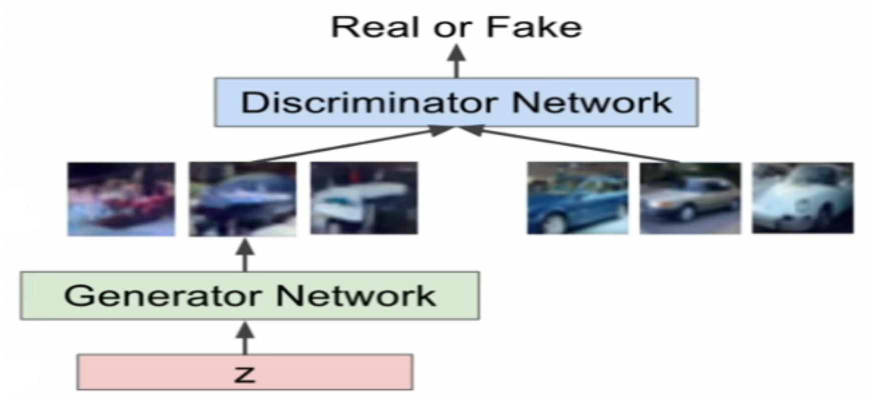


Figure I.6 – GAN training

The decision rule for this game is the minmax rule, which can be rewrite as follow :

$$\min_{\theta_g} \max_{\theta_d} \left[E_{x \sim P_{data}} log(D_{\theta_d}(x)) + E_{z \sim P_z} log(1 - D_{\theta_d}(G_{\theta_g}(z))\right]$$

- $D_{\theta_d}(y)$: discriminator output for y. It is the likelihood that y is real.
- x : real data (from the training data set)
- z : generator input (usually a random vector from Uniform or Gaussian distribution)
- $G_{\theta_g}(z)$: generated data (fake data)

In other words, D wants to maximise objective such as D(x) is close to 1 (real data prediction) and D(G(z)) is close to 0 (fake data prediction). In an other hand, G wants to maximise objective such as D(G(z)) is close to 1 (D is fooled into thinking generated G(z) is real).

NB : The drawback of this model is that it trains two networks in the same time and thus it can be unstable.

To train a GAN model we alternate between :

1) Gradient ascent on discriminator to maximise :

$$C_1(\theta_d) = E_{x \sim P_{data}} log(D_{\theta_d}(x)) + E_{z \sim P_z} log(1 - D_{\theta_d}(G_{\theta_g}(z))$$

2) Gradient ascent on generator to maximise :

$$C_2(\theta_g) = E_{z \sim P_z} log(D_{\theta_d}(G_{\theta_g}(z))$$

Algorithm 2 Training GAN

```
1: procedure TRAIN-GAN($number\_epochs, steps, \epsilon, \alpha$)
2:     for number of epochs do
3:         for k steps do
4:             •Sample batch of m noise sample $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ from $p_z$, the latent space.
5:             •Sample batch of m real data $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ from $p_{data}$.
6:             •Update the discriminator by ascending its stochastic gradient:

                   $$\nabla_{\theta_d} \frac{1}{m} \sum^{m} [logD(x^i) + log(1 - D(G(z^i))]$$

           end for
7:         •Sample batch of m noise sample $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ from $p_z$, the latent space.
8:         •Update the generator by ascending its stochastic gradient:

                   $$\nabla_{\theta_g} \frac{1}{m} \sum^{m} log(D(G(z^i))]$$

       end for
```

3.3 Derivatives and applications

Since Ian Goodfellow introduced GAN, mutiple models based on GAN have been developped and the applications of GANs have been multiplied. I will introduce some of them in this section.

Deep Convolutional GAN

Deep Convolutional GANs are an improvement and a specific case of GANs where generator is a network producing images thanks to convolutional layers which increase the number of pixels. In the other side discriminator is a convolutional neural network.

Conditional GAN

Conditional GANs (cGAN) try to learn data associations. That is to say that given a condition c, G learn to generate data y in accordance with this condition and D have to distinguish real and fake paires "condition/data". We rephrase the problem by saying that c is a parameter for both generator and discriminator :

$$\min_{\theta_g} \max_{\theta_d} \left[E_{x,c \sim P_{data}(x,c)} log(D_{\theta_d}(x,c)) + E_{z \sim P_z, c \sim P_{data}(c)} log(1 - D_{\theta_d}(G_{\theta_g}(z,c),c) \right]$$

$\rightarrow$ *Couple examples : text/image ; black and white image/color image; summer photo/winter photo*

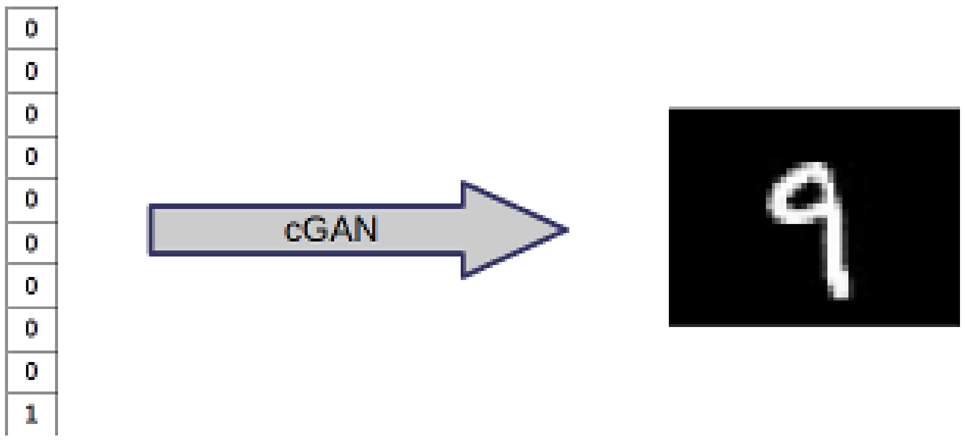


Figure I.7 – Number to hand-written digit

Pix2pix

Pix2pix is the application of cGAN for image-to-image translation. See Part III, p.26.

Stack GAN

Stack GAN is a kind of conditional GAN which generate photo-realistic images conditioned on text descriptions.

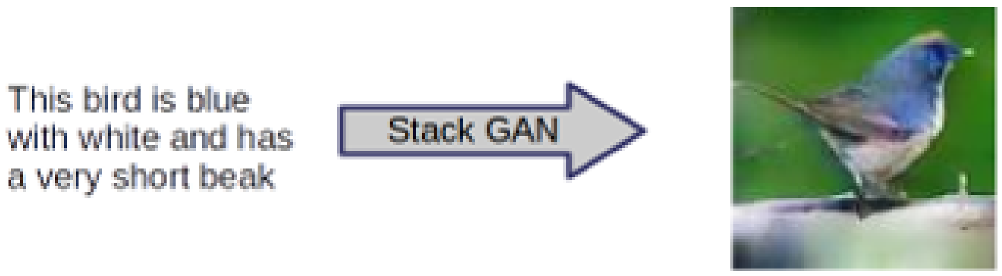


Figure I.8 – Text to image

Cycle GAN

Cycle GANs is an approach learning to translate an image from a source domain X to a target domain Y in the absence of paired examples.

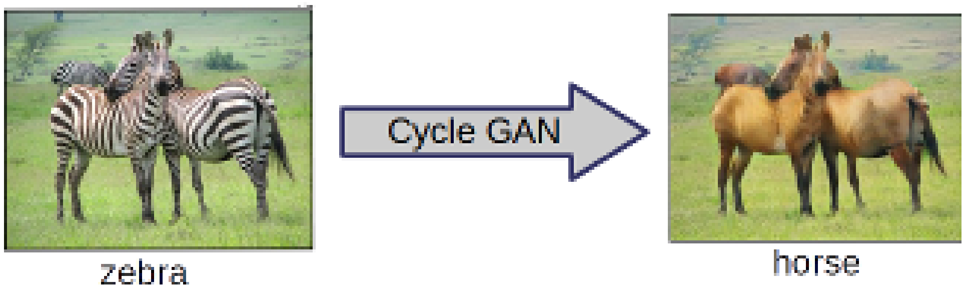


Figure I.9 – Zebra to Horse translation

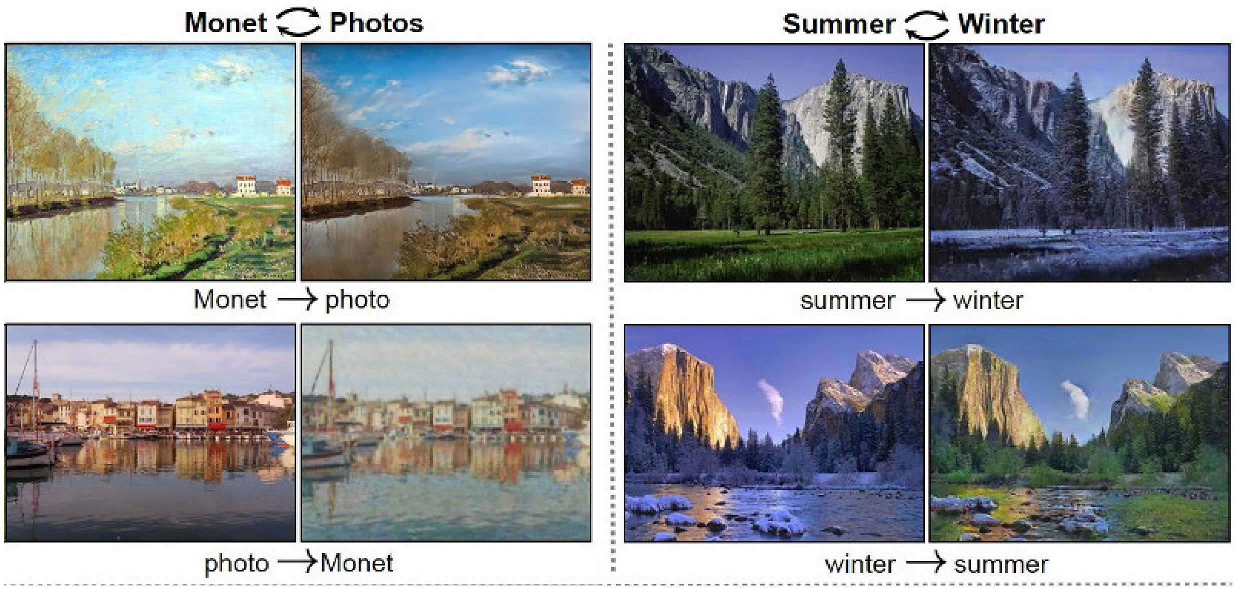


Figure I.10 – Other translations

3.4 Small project : Understand and apply DCGAN network

I first studied a *DCGAN code* [7] from carpedm, understood how it works and applied it on my own images. I chose to generate Yu-Gi-Oh cards because Yu-Gi-Oh cards have similarities and so I thought that it could give good results for GAN. Moreover, Yu-Gi-Oh cards can be easily found on the internet [10]. Thus, I wrote a bash code to download more than 3000 Yu-Gi-Oh cards and I used them as input of DCGAN algorithm.

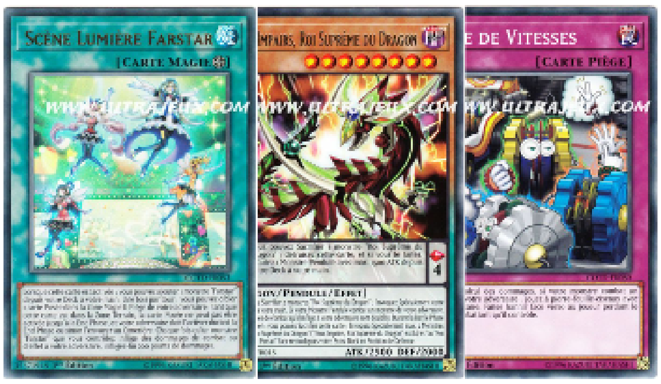


Figure I.11 – Real Yu-Gi-Oh cards

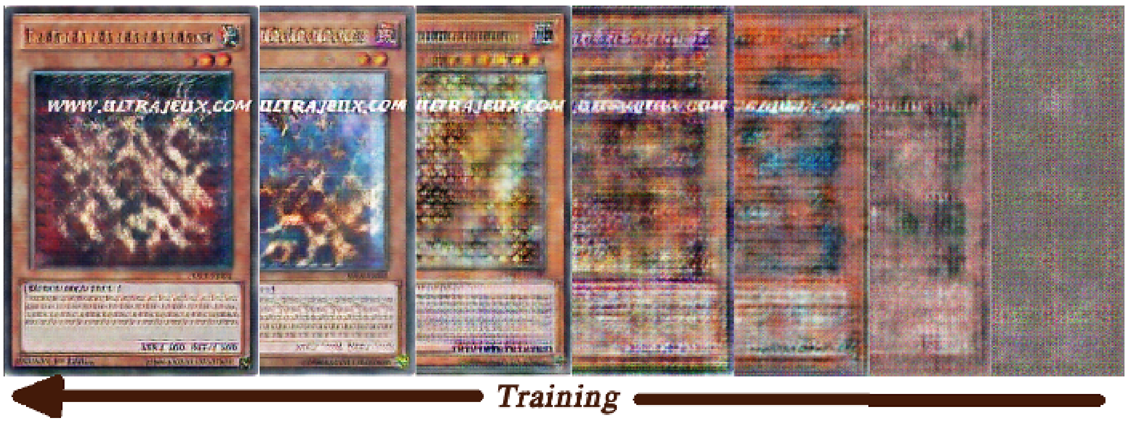


Figure I.12 – Training progress

Other random generated examples are available p.36.

4 Tools for machine learning

Machine learning techniques need a lot of computational resources. Thus, to train a machine learning model, it is essential to use some tricks which can make the running time shorter. In fact, to train a deep neural networks from scratch, it can last days or weeks. To face this problem, several methods must be used.

(a) Colaboratory (b) Tensorflow

Figure I.13 – Tools

4.1 Graphics processing unit

The first thing to do in order to improve running time is to run the code on a Graphic processing unit (GPU). A GPU is a processor specialized for display functions. It performs parallel operations and is useful for deep learning algorithms which perform a huge number of operations. I worked with two GPU.

I first had access to Queen Mother building GPU. Thanks to my server account, I could log on my account and use this GPU by SSH, tranfer files and run my codes. Moreover, I used Colaboratory platform, figure I.13a, and the associated GPU for research projects. Colaboratory is a Google project created to help disseminate machine learning education and research.

4.2 TensorFlow library

I worked with an open-source machine learning framework, TensorFlow I.13b. Firstly, TensorFlow was developed by the Google Brain team for internal Google use. Later, it became free to public. It can be run on multiple CPUs and GPUs and provides Python and C APIs. This is one of the most popular machine learning framework because it provides many powerful functionalities. TensorFlow allow users to build their own graphs (neural networks), train their model from scratch or with an already trained model, visualize their model, follow in real time the progress and, the most important thing, save some "checkpoint files" which contains every weights learned while training. Then checkpoint files could be used to retrain the model with other parameters and avoid to restart training from scratch.

To visualize our model and its progress, we can use the TensorFlow visualization tools called TensorBoard. When I trained a classifier to determine if an image was a Yu-Gi-Oh card or not (see Yu-Gi-Oh card recognition, p.10), it looks like fig. I.14 and fig. I.15.

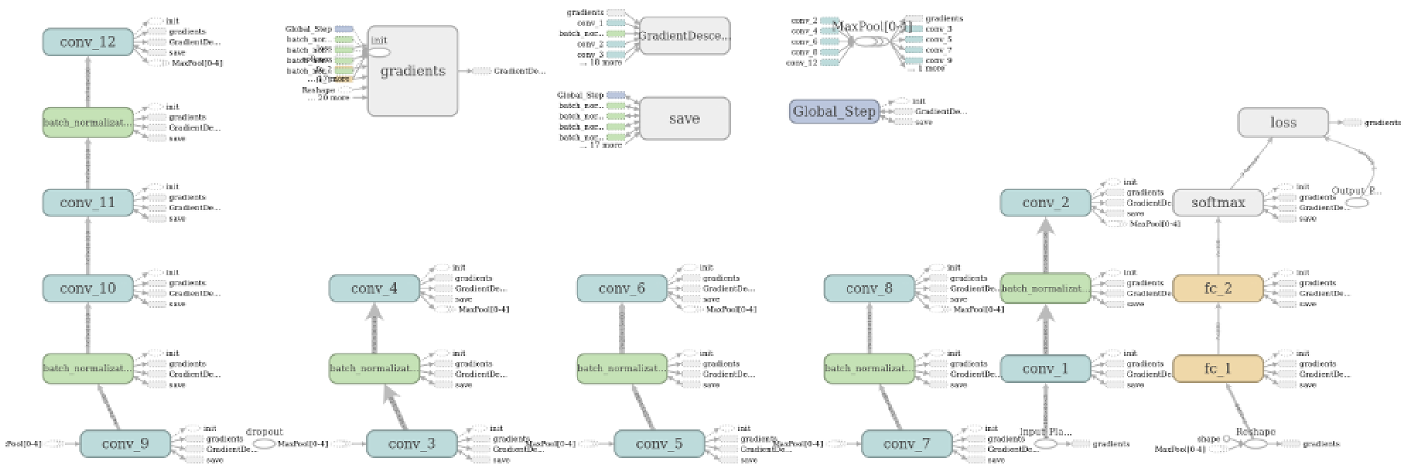
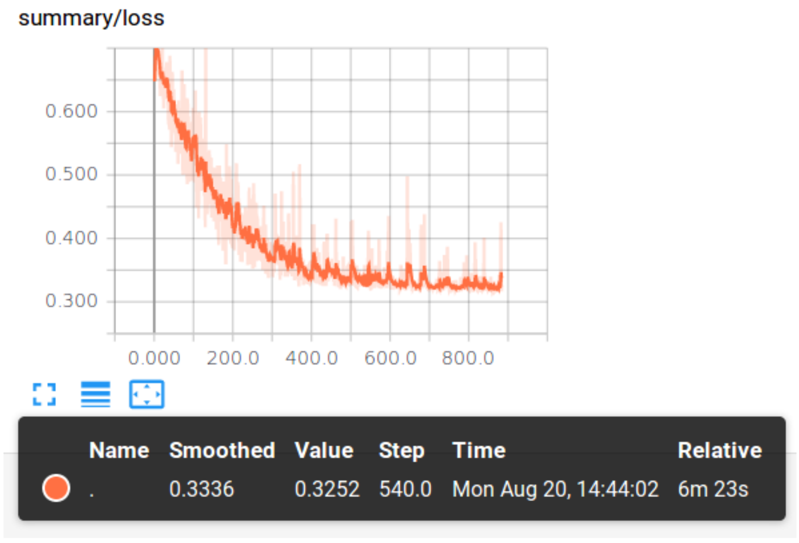


Figure I.14 – TensorBoard Graph



	Name	Smoothed	Value	Step	Time	Relative
●	.	0.3336	0.3252	540.0	Mon Aug 20, 14:44:02	6m 23s

Figure I.15 – TensorBoard Loss

4.3 Additional machine learning methods

Using good tools, as mentionned above, is important to get the best result in reasonable time. However, other methods exist to accelerate training process and prevent some problems. Transfer learning and dropout can be point out.

Transfer learning

Given a model developped and trained for a task, transfer learning consists in reusing this model as a start for a model on a second task. This method allows rapid progress and can improve performance for a second more specific task. For instance, we can use transfer learning to classify very similar images like breeds of dog. We can use a ResNet model trained for ImageNET classification, change the last fully-connected layer and trained it on our dogs images. In this case, transfer learning will allow to classify dogs without require a large number of images for each breed.

Dropout

Dropout is an important regularization technique for deep learning. Dropout consists in deactivate some neurons in both hidden and visible layers of a neural network. Thus, it avoids neurons to develop co-dependency during training. Applying dropout does not face directly the waste of time but prevents against overfitting, one of the most regular deep learning problem.

Part II

Detecting and counting people on an picture

1 2018 Baidu Star Developer Competition

The idea of my first project, Code 4 p.35, came from one of my colleague, George Chen. He wanted to take part in a chinese computing contest. This competition was organized by Baidu Inc, a Chinese multinational company specialized in Internet-related services and artificial intelligence. The main purpose for participants was to develop a universal human flow density estimation algorithm. That is to say participants were asked to provide an algorithm or a model which predicts the total number of people in a given image. First, we needed to train our model for a given dataset and then apply this model on testing data in order to get the most accurate number of people for each testing data.

While George was attempting to build a model for this competition I decided to implement a model using data to predict position of people on a picture. I thought that if we were able to detect people in a picture, we should easily count how many they are.

1.1 Data Description

For this contest, the main problem was the data. Data consist of two parts, training data and testing data. Both of them are divided into two more parts: one folder containing the whole set of picture and one JSON file containing annotations for data. I will describe in details training data which was the most important part for building an adequate model.

First, we had 3619 images of public places like roads, cafeterias, elevators or airports as showed in the following figure.

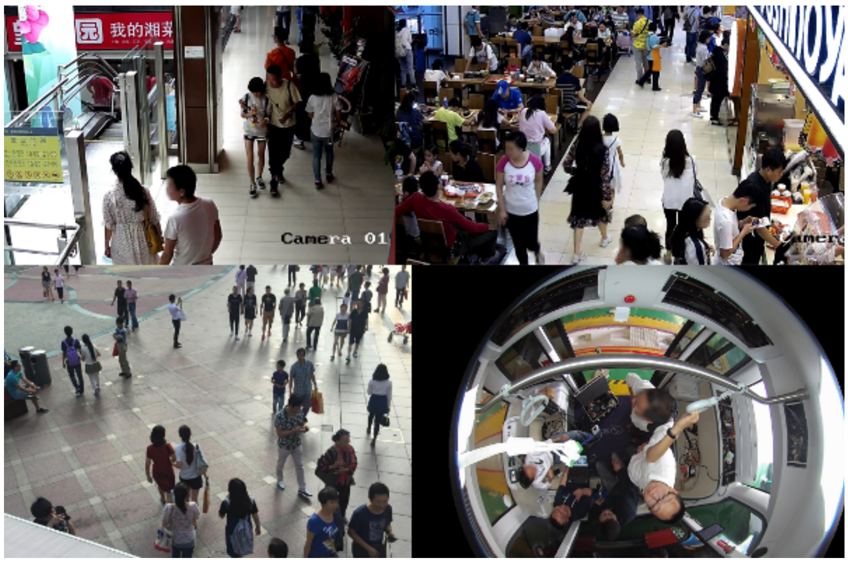


Figure II.1 – Example of training image

Finally, we had a JSON file which provide several annotations for training process. This file consists of a 3619 units' vector (one for each image).

Each unit is compouned as follow :

$$unit = \begin{bmatrix} name \\ id \\ num \\ ignore\_region \\ type \\ annotation \end{bmatrix} = \begin{bmatrix} "stage1/train/e6bc4d3e343ce1d6b01b947f7630c506.png" \\ 1851 \\ 5 \\ [] \\ "dot" \\ \begin{bmatrix} \{"y" : 298, "x" : 138\} \\ \{"y" : 345, "x" : 505\} \\ \{"y" : 451, "x" : 145\} \\ \{"y" : 398, "x" : 602\} \\ \{"y" : 203, "x" : 412\} \\ \{"y" : 240, "x" : 19\} \end{bmatrix} \end{bmatrix}$$

where :

- *name* is the file name
- *num* is the number of people on this image
- *ignore_ region* is a coordinate list for a possible region to ignore
- *type* represent the type of annotation used to localize people ("dot" for a dot on his head and "box" for a box over him)
- *annotation*, the dot or box list

It can be useful to precise that most of localization annotation are dots and not boxes, that is why I decided to build a model using only dot localization.

2 General model

2.1 Choice of the model and explanation

At this point I started many researches about how I could solve the problem of counting people. The first idea I thought about was to build a person detection algorithm. It would be easy, after detecting people, to count them. At this stage of my internship, I had had time to learn that it exists algorithms which allow to make object detection in real-time, a very interesting topic for me. That is why I chose to use object detection model to count people. I first thought about using Faster-RCNN model which is one of the most powerful model for object detection in real-time with a large accuracy (70-75 % accuracy and $\sim$ 20 FPS). This algorithm is an improvement of Fast-RCNN which is itself an improvement of R-CNN (Region proposal Convolutional Neural Network). R-CNN family are classification problems and include a region proposal network. It means that the model proposes some regions (usually 2000 regions) where it is possible to find an object and then compute the classification for each of the 2000 proposals. However, computing so much classification can take a very long time. Moreover, the model is easy to understand but the regions' proposal in Faster-RCNN is more difficult to implement. For these reasons, I studied another approach, called YOLO (You Only Look Once). YOLO is different because it treats the object recognition task as a regression problem. The main advantage of YOLO is that it can achieves more than 50 frames per second so it turns out to be useful for real time object detection. However, YOLO is less accurate.

2.2 Architecture

The YOLO network is quite similar to CNN architecture. Indeed, it is compounded of a convolutional feature extractor and two fully connected layers. Only the output is quite different, because this is a 3D-tensor instead of 2D for CNN. In YOLO, the convolutional part is adapted from Googlenet.

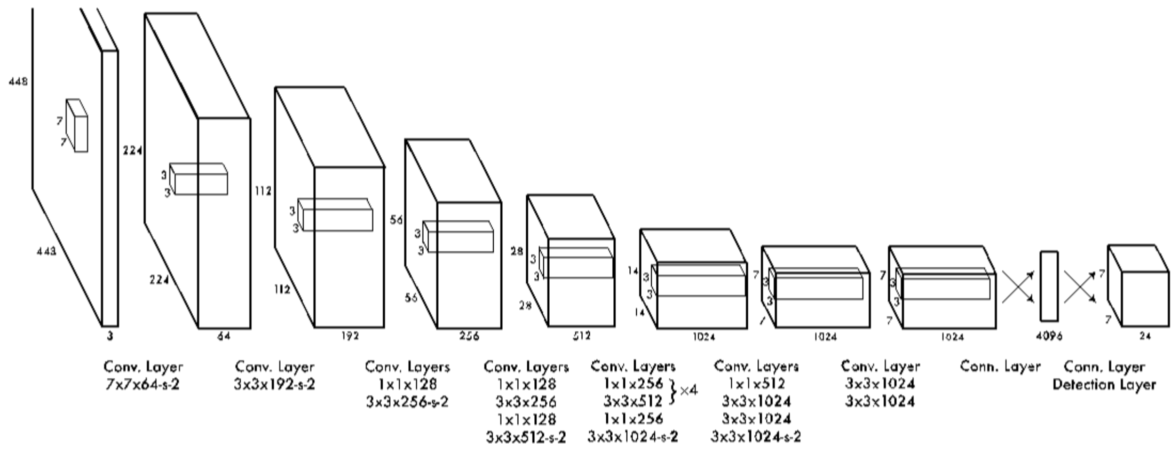

448
448
7
7
224
224
3
3
112
112
3
3
56
56
3
3
28
28
3
3
14
14
3
3
7
7
3
3
7
7
3
3
7
7
3
64
192
256
512
1024
1024
1024
4096
24
Conv. Layer 7x7x64-s-2
Conv. Layer 3x3x192-s-2
Conv. Layers 1x1x128 3x3x256-s-2
Conv. Layers 1x1x128 3x3x256 1x1x128 3x3x512-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x256 3x3x1024-s-2
Conv. Layers 1x1x512 3x3x1024 3x3x1024 3x3x1024-s-2
Conv. Layer 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer Detection Layer

Figure II.2 – YOLO v2 architecture

Input of YOLO neural network is an image and the output consists in a 3D-tensor with dimensions $7 \times 7 \times 24$. Let's get into more details. YOLO predicts $7 \times 7 = 49$ vectors corresponding to 49 cells of input image, as showed fig. II.3 with $5 \times 5 = 25$ cells.

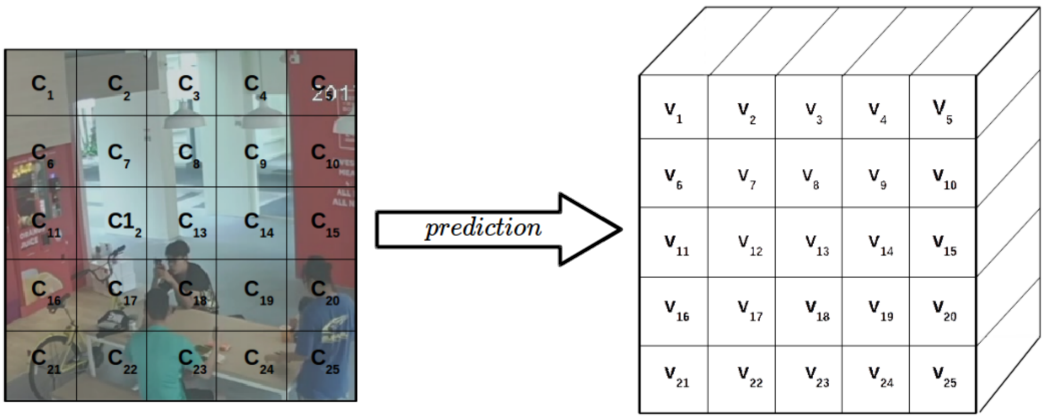


Figure II.3 – YOLO Prediction

sont les probabilités que le rectangle prédit de la cellule i soit de la classe j

Moreover, each predicted vector is 24th length $\Rightarrow v_i = (x_i, y_i, w_i, h_i, l_{i,1}, l_{i,2}, ..., l_{i,20})$, where :

- (x_i, y_i, w_i, h_i) are predicted position, weight and height of object bounding box in cell i
- $\{l_{i,j}\}_{j=1..19}$ are likelihoods that the predicted box, in cell i, is an object of class j (similar to image classification problem)
- $l_{i,20}$ is the likelihood that the predicted box, in cell i, is a part of the background

I used YOLO v2 (fig. II.2) and modified a bit intial model in order to be in accordance with my data and what I needed to do with it. To this end, I changed the last fully connected layer to predict a 15th depth tensor instead of a 24th depth tensor. In fact, for each cell, I decided to predict $B = 5$ relative coordinates (x, y) and give the livelihood that a person is present at this point $\Rightarrow v_i = (p_{i,1}, x_{i,1}, y_{i,1}, p_{i,2}, x_{i,2}, y_{i,2}, ..., p_{i,5}, x_{i,5}, y_{i,5})$

Example : In fig. II.3, we intend our model predict : $v_{20} = (1.0, 0.45, 0.2, 0, 0, 0, ..., 0)$

2.3 Loss function

I have first trained the model with (II.1) as loss function and then with (II.2) which is more similar than the original YOLO loss function.

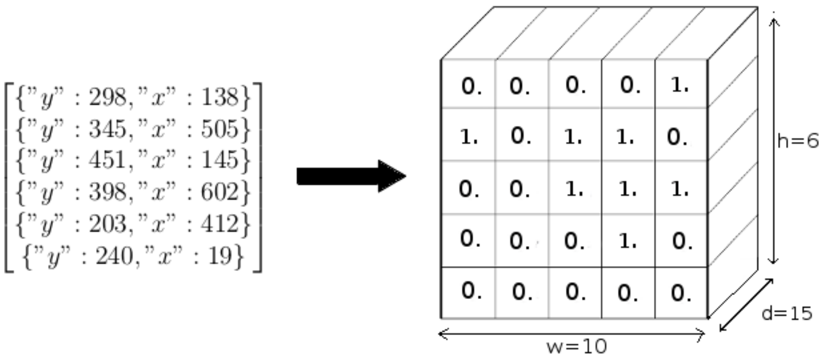
$$\mathcal{L}_1 = \lambda_{coord} \sum_{i=1}^{h\times w} \sum_{j=1}^{B} \mathbb{1}_{i,j}^{obj} \left[(x_{i,j} - \widehat{x_{i,j}})^2 + (y_{i,j} - \widehat{y_{i,j}})^2 \right] + \lambda_{obj} \sum_{i=1}^{h\times w} \sum_{j=1}^{B} (p_{i,j} - \widehat{p_{i,j}})^2 \tag{II.1}$$

$$\mathcal{L}_1 = \lambda_{coord} \sum_{i=1}^{h\times w} \sum_{j=1}^{B} \mathbb{1}_{i,j}^{obj} \left[(x_{i,j} - \widehat{x_{i,j}})^2 + (y_{i,j} - \widehat{y_{i,j}})^2 \right] + \lambda_{obj} \sum_{i=1}^{h\times w} \sum_{j=1}^{B} (p_{i,j} - \widehat{p_{i,j}})^2 \tag{II.2}$$

where $\mathbb{1}_{i,j}^{obj}$ is 1 when there is at least j person in the cell i, else 0. λ_{coord} and λ_{obj} are ratio relating to importance of coordinates predictions and object livelihood prediction.

3 Preprocessing data and visualisation

For this project, I used Google Colaboratory, which provides a powerful GPU. The first part was to preprocess data. George's code has helped me to this end. The goal was to alter data from JSON file to a 4D-tensor of shape $(n, w, h, p) = (3619, 10, 6, 15)$. In fact, for each of the 3619 image annotations, I had the following alteration to compute :



In order to verify that the data have been well preprocessed and in order to follow training progress, I have implemented a displaying function (fig. II.4).

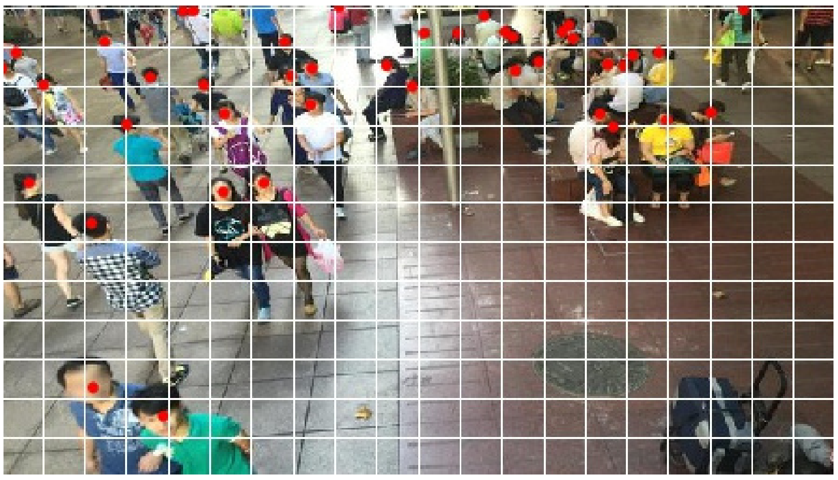


Figure II.4 – **Procedure** DISPLAY_ IMG(img, label, grid=True)

Then, I built my model and train it on the data. See Code 4. p.35.

4 Learning and results

4.1 Training

After some attempts, I chose to train the model with $\lambda_{coord} = 0.5$ and $\lambda_{obj} = 1.0$ because for baidu competition it was more important to have good person prediction than their position. The training period didn't take so much time ($\approx 5h$ to process 15 epochs). However, I saw that a problem was coming. In fact, the loss quickly fall down to 0. As a consequence, application of the model on training data provided good results, whereas on testing data it was bad. The problem was due to overfitting. It means that the network have learned a too specific function which works well with the training data but can't supply a general function to predict new samples. It is likely that overfitting is important here due to the fact that training data are compouned by similar images. Data show only around fifteen places where people often stand at the same position. Moreover, another reason of overfitting is that the network is very deep whereas number of training images is not large. To face this problem, several solutions can be applied.

4.2 Improvements

The first solution I tried was to create new data thanks to the original data. We call this technique data augmentation. Therefore, I extended the number of data from 3 619 to 72 360 by flipping, reflecting and cropping each of them. Moreover, I added dropout to avoid overfitting. Then, I retrained the model on 72 360 pictures. I trained the first epoch with $lr = 10^{-3}$ to go down quicker and then with $lr = 10^{-5}$ to adjust. It took more time ($\approx$ 1 epoch /hour). Unfortunetly results weren't good yet for the problem of localization. However, compared to expected results, density of population was not bad. Convinced that there are other ways to improve training, I was looking for a new method. I thought that the best method was to train or use a pretrained model on image classification (like ImageNET dataset) to learn useful features. Then, I should move last fully-connected layer to do object-detection prediction instead of classification. In other words, I decided to use transfer learning as explained p.18.

To this end, I used a pretrained VGG model on ImageNET dataset and changed the last fully-connected layer to detect and count people.

This project was interesting for me because I discovered object detection algorithms and learnt how YOLO, a very encouraging model, works.

Part III

Realistic image generation of cars from drawings

1 Project Description

For this last project, Code 5 p.35, I tried to apply a kind of conditional GAN, sometimes called *pix2pix*. A very interesting website gives an interactive interface of pix2pix examples [11]. I found this idea when my supervisor M.Jianguo Zhang asked me to study the *Image-to-Image Translation with Conditional Adversarial Networks* paper [2] and to find a project about it. In this paper they introduce some ways to use cGAN for altering an image to another one. Applications are multiples, this process can be used to image colorization or realistic image reconstruction from shapes for instance. One particular application caught my interest. It consists of a network able to generate realistic images from drawings. It is used to design new clothes or to create new images thanks to an original drawing.

For this project, I chose to train a model to generate realistic colored cars (output) thanks to car edges and a color (input = label = condition).

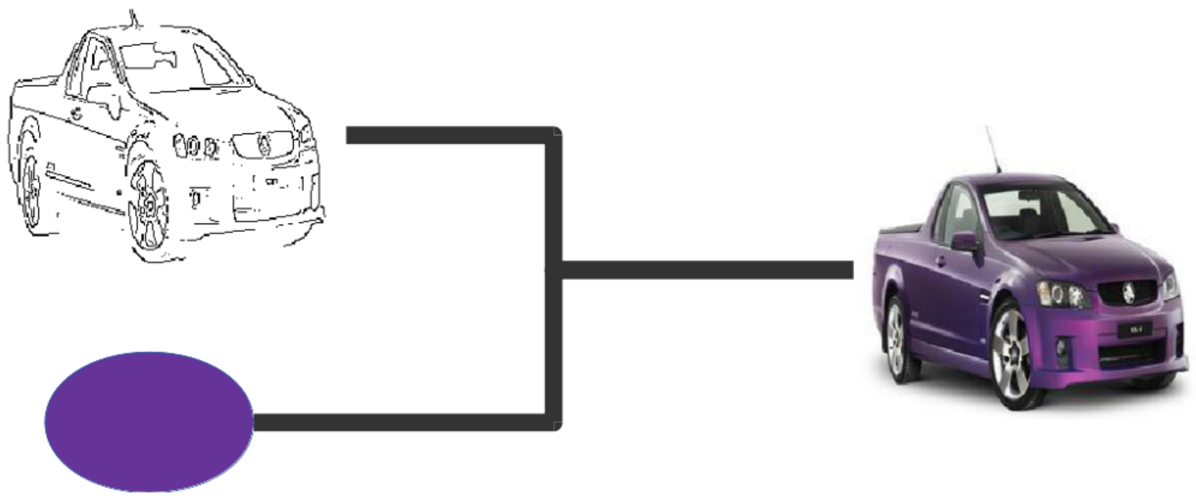


Figure III.1 – Project purpose

1.1 Data Description and preprocessing

As every machine learning project, the first part was to find an adequate dataset and preprocess data. In this case, I had to find a large range of car images. So I runned code of Hardik Vasa [13] in order to download more than 3 500 car images from Google image. Then I had an important work to preprocess them. Indeed, I knew since DCGAN project p.16 that it is more accurate to use similar images than very different ones. In fact, convolutional layers detect features in images. Then, when every images have different features the network cannot extract similarities. For this reason, I decided to keep only images without background because there were too much backgrounds and the goal was to generate car image and not background. So, I first deleted all images with a background. Besides, I removed every images that were not "jpg" and "png" files. As a result, there were only 1 050 images. With these 1 050 images, it was then possible to work.

After that, I had to find a way to generate labels, that is to say the edge images and the car color. To generate labels, I tried 3 edge detection methods.

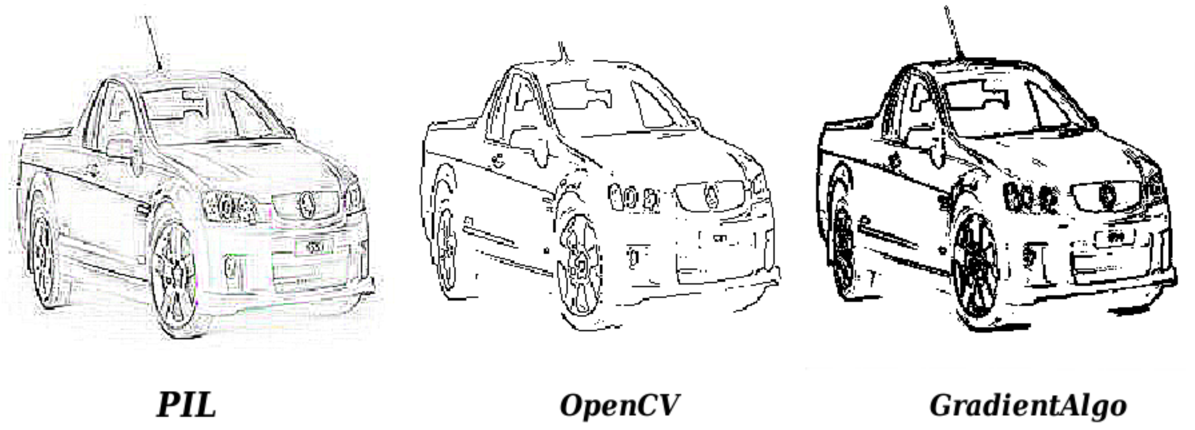
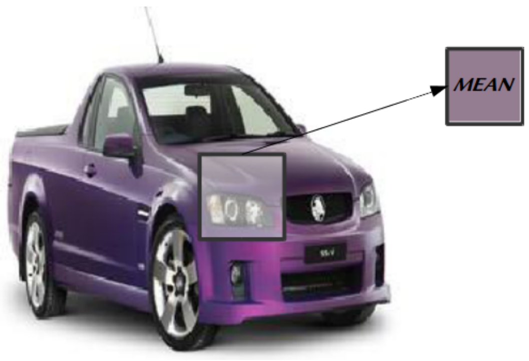


Figure III.2 – Edge Detection algorithms

I finally decided to use OpenCV method because this method provides images with less edges and so that, quite similar to basics drawings.

To finish the preprocessing part, I had to detect car color. It was not an easy task because cars have different orientations, have headlights, wheels and a windscreen of another color. Moreover, there are lights and shadows on the car, so it is not obvious to find a similar area for every 1050 images where the car color is mainly present. The empirical method that I chose is to use the average color of a small square (5×5 or 8×8 pixels) in the middle of the image.



2 Model

According to Manish Chablani website [12], pix2pix is a conditional GAN which learns a mapping from an input image to an output image. I used this model for my project. As every GAN model, pix2pix is compouned of two neural networks, a generator and a discriminator. Let's talk about both architectures.

2.1 Generator

The structure of the generator is an *encoder-decoder*. It looks like a variational autoencoder (VAE) structure. The generator is divided into two units. The encoder unit takes an input image and try to reduce it into a smaller vector with some convolutional layers. This vector can be seen as a compressed data of the input. In the decoder unit, the input is the output vector of the encoder and the output is a new image. Decoder uses *"deconvolutional"* layers to get higher dimensions.

In the paper *Image-to-Image Translation with Conditional Adversarial Networks* [2], authors used another architecture, called *U-Net*, in order to improve the performance of the image-to-image transformation. *U-Net* has skip connections, which connects the correspond layers from the encoder to decoder (see fig. III.3).

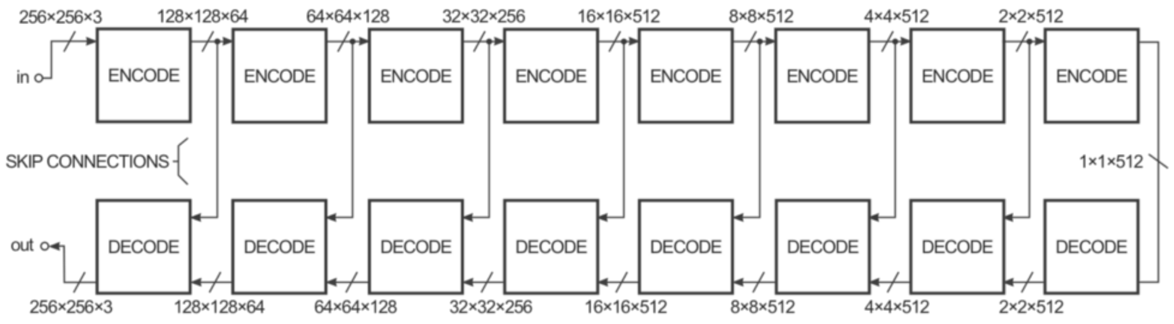


Figure III.3 – Generator Structure

2.2 Discriminator

Concerning discriminator, its structure looks a lot like the encoder unit of the geneator. Its goal is to predict if a image is due to the generator or is a real image from the dataset. The main difference with encoder unit is that the output is a 30×30 matrix of likelihoods in $[0, 1]$. Each value represents how believable is the corresponding section of the input image.

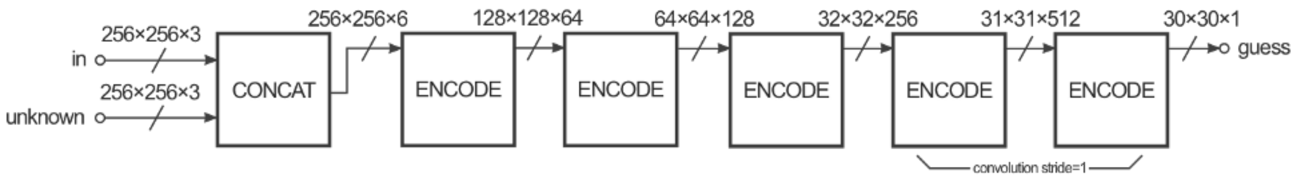


Figure III.4 – Generator Structure

2.3 Loss functions

As every GAN model, to train this model, we have to train alternatively discriminator and generator. I used *AdamOptimizer* to minimise each generator and discriminator losses. Let x be the colored edge data and y the real data.

Discriminator loss:

$$\mathcal{L}_D = -\sum_{i=1}^{k\times k} log\left[D_{\theta_d}^{(i)}(y)\right] + log\left[1 - D_{\theta_d}^{(i)}(G_{\theta_g}(x))\right]$$

Generator loss:

$$\mathcal{L}_G = -\frac{\lambda_{FD}}{k\times k}\sum_{i=1}^{k\times k} logD_{\theta_d}^{(i)}(G_{\theta_g}(x)) + \frac{\lambda_{L1}}{n\times n\times 3}\sum_{i=1}^{n\times n\times 3} |y_i - G_{\theta_g}^{(i)}(x))|$$

where $k = 30$ is the dimension of discriminator output and $n = 256$ is the dimension of generator output.

λ_{FD} and λ_{L1} are ratio relating to the importance of fooling the discriminator and looking like real data.

3 Learning and results

3.1 Training

I trained the model with $\lambda_{FD} = 1$ and $\lambda_{L1} = 100$ as advised in the original paper [2]. Making λ_{L1} high allows a quick convergence to the realistic images. So it saves time (only 10 epochs needed to be train $\approx 2h$). As shown in fig. III.5, before the first epoch, generated cars already have the shape of real cars. However, it needs more epochs in order to have a good accuracy and to color cars well.

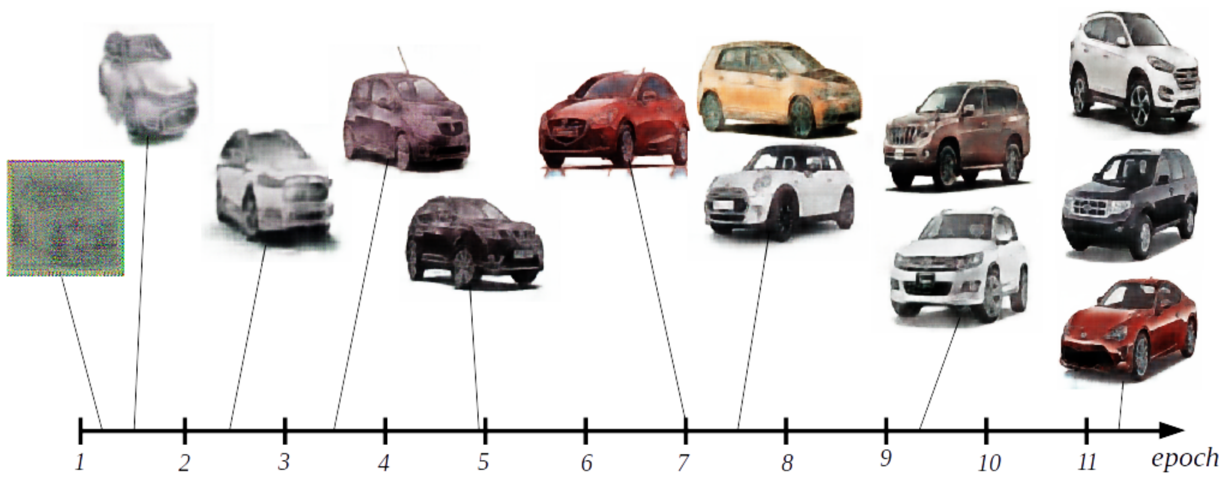


Figure III.5 – Training progress

3.2 Testing

I tested the network on different data. Results are quite good and interesting to analyse. We can notice that the network have learned several ideas. I would like to put forward ideas of coloration, shadow and light.

I used the network to change original color of the purple 4×4 displayed above.

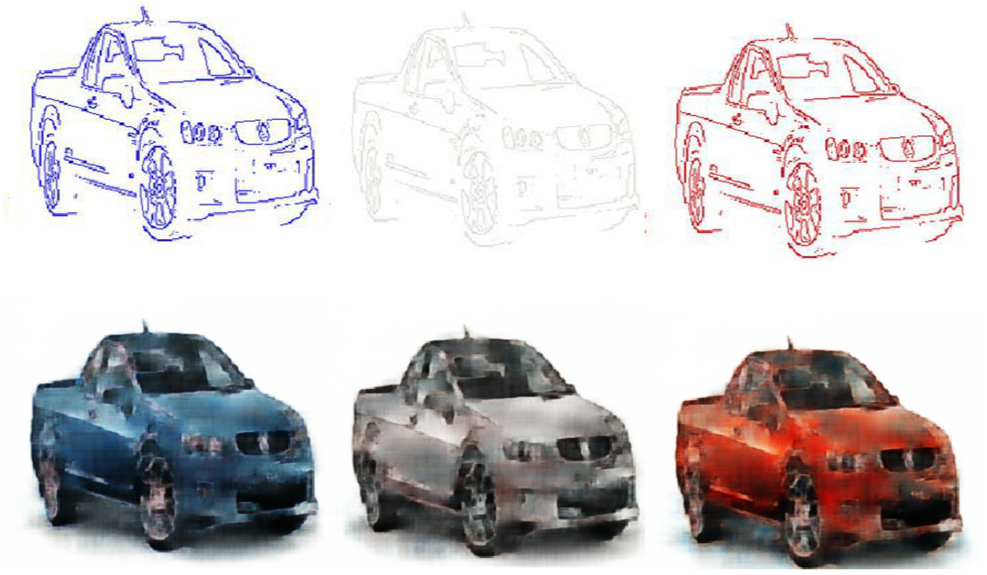


Figure III.6 – 4x4 in different colors

I also used it to color some drawing from *Google Image*, some drawings of my friends and mines, p.37 .

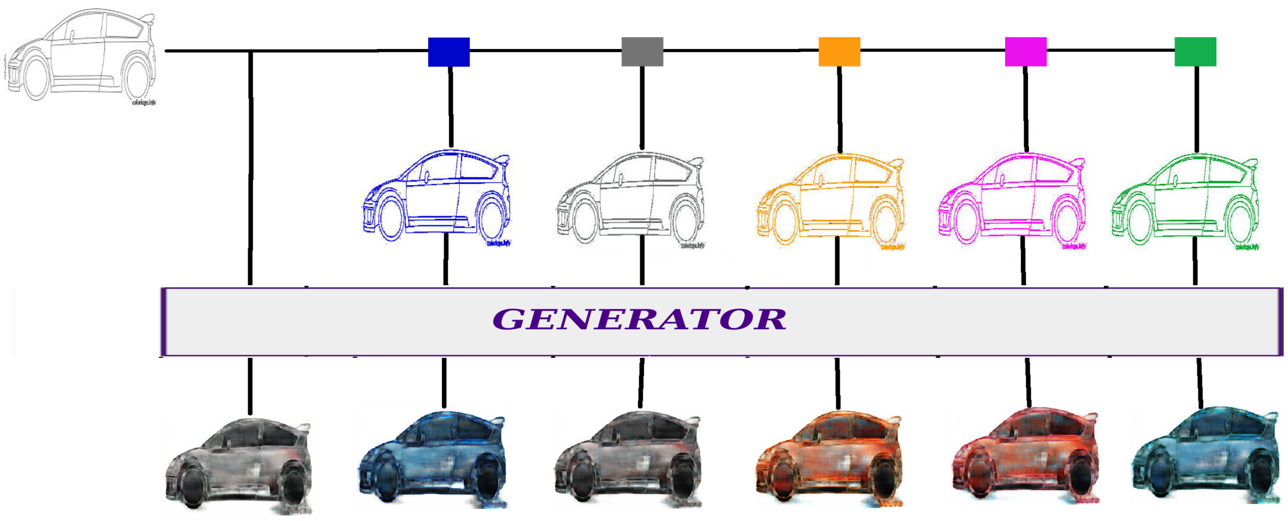


Figure III.7 – Image to image translation procedure

3.3 Improvements

I quickly understood that there was a limit to the trained model. As shown on fig. III.8, the network seems to be more able to generate black, gray, red and blue cars than other colors. Moreover when I tryied to generate a car with two or more colors, it provides bad results.

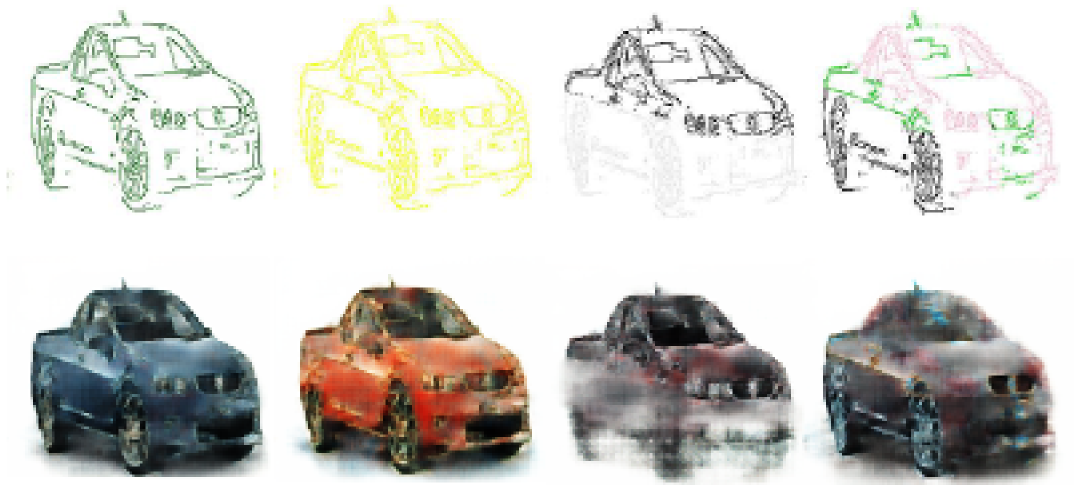


Figure III.8 – Wrong coloration

One way to solve coloration problem is to train the model with more colored images like yellow, pink or green cars for instance. Although these kind of images are more difficult to find, I downloaded new car images of seven different colors. I got only sixty for each color. I retrained the model on all images, including new ones. Results were quite better for the coloration task because it was able to provide a larger range of car’s color. For example, I obtained different shades of red or shades of yellow.

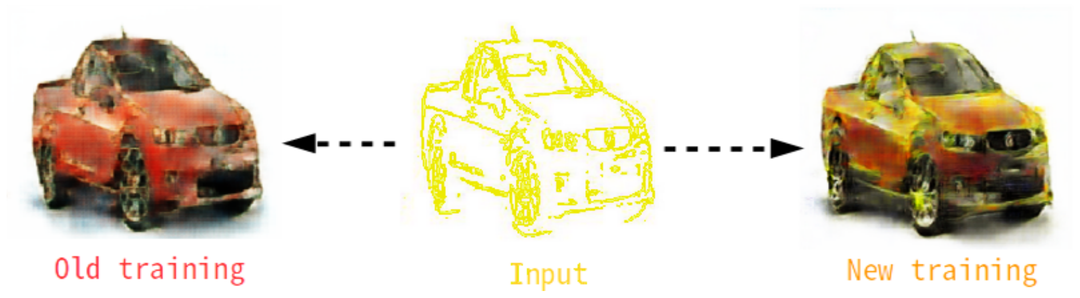


Figure III.9 – Output difference

For this project, results are interesting and quite good. In the future, we could add cars data and train the model longer to get best results.

Conclusion

This internship allowed me to discover different things.

First of all, living in a flatmate with foreigners in an English-speaking country allows me to discover different cultures. All of these cultures are different from the French one. As an example I can mention the Chinese, Hungarian and Thai cultures. I also discovered different type of culinary specialties. I will not hesitate to cook some of them again.

As I already mentioned, the discovery of an attractive domain was for me a significant asset. In fact, deep learning is at the heart of current researches in artificial intelligence. On the other hand, this internship allowed me to acquire the necessary knowledge in order to have an efficient practice of machine learning methods. This topic has a particular interest to me. I already have some ideas of projects that I would like to carry out in the future using the skills I earned.

Finally, and probably the most important, I discovered the work of a searcher at once with the work of my PhD student colleagues, my supervisor and mine. In addition, I discovered the world of research in an international context owing to the multiple nationalities of people working in premises of Queen Mother Building. This first experience allows me to confirm my interest for this job.

Internship Progress

Week 1	- Meet up with my supervisor M.Jianguo ZHANG and his colleagues - Lecture about cybersecurity and how to choose a good password - Learning of techniques for classification problems - Learning of Python language
Week 2	- Learning of ML techniques for image classification (Stanford Univ.) - Learning of TensorFlow library - Small project about comparison between a CNN and a Fully Connected network
Week 3	- Learning of ML (Stanford Univ.) - Learning of TensorFlow library - Small project DCGAN
Week 4	- Specific study of generative models (VAE and GAN) - Reading of papers about GAN derivatives and applications. - Looking for a project
Week 5	- Study of pix2pix applications. *It is fun !!* [11] - First idea of a project and research about it (gave up) - Introduction to Baidu contest by George
Week 6	- Implementation of a deep learning API - Study of object detection algorithm (Faster R-CNN and YOLO)
Week 7	- Implementation of a deep learning API - Small project for Yu-Gi-Oh card recognition - Building a model for Baidu Contest using YOLO
Week 8	- Study of George's code and implementation of my model for counting people problem
Week 9	- Implementation of the second project
Week 10	- Work on Baidu project to avoid overfitting
Week 11	- Work on edge2car project
Week 12	- Writing of the report and improvement of the project models

Bibliography

[1] *NIPS 2016 Tutorial: Generative Adversarial Networks*, Ian Goodfellow, https://arxiv.org/abs/1701.00160, 2016.

[2] *Image-to-Image Translation with Conditional Adversarial Networks*, Phillip Isola, Jun-Yan Zhu, Tinghui Zhou and Alexei A. Efros, https://arxiv.org/abs/1611.07004, 2016.

[3] *You Only Look Once*, Joseph Redmon , Santosh Divvala , Ross Girshick and Ali Farhadi, https://arxiv.org/abs/1506.02640, 2016.

[4] *Stanford University CS231n, Spring 2017*, Fei-Fei Li, Justin Johnson and Serena Yeung, http://cs231n.stanford.edu/, 2017

[5] *Object Recognition for Dummies*, Lilian Weng, https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html, 2017.

[6] *TensorFlow tutorial*, TensorFlow team, https://www.tensorflow.org/get_started/ 2015.

[7] *DCGAN-tensorflow*, Taehoon Kim (*carpedm20*), https://github.com/carpedm20/DCGAN-tensorflow, 2016.

[8] *pix2pix-tensorflow*, Yen (*yenchenlin and MIT*), https://github.com/yenchenlin/pix2pix-tensorflow, 2017.

[9] *Wikipedia*, Wikipedia Team, https://fr.wikipedia.org/

[10] *Yu-Gi-Oh card dataset*, ultrajeux.com, http://www.finalyugi.com/yugioh-cartes.html

[11] *Image-to-Image Demo - Affine Layer*, Christopher Hesse, https://affinelayer.com/pixsrv/, 2017

[12] *CycleGANS and Pix2Pix*, Manish Chablani, https://towardsdatascience.com/cyclegans-and-pix2pix-5e6a5f0159c4, 2017

[13] *Google Images Download*, Hardik Vasa (*MIT*), https://github.com/hardikvasa/google-images-download, 2015

Codes

- **Code 1** - Dense NN vs CNN - *Colab*
  → https://arxiv.org/abs/1701.00160
- **Code 2** - Deep Learning API - *Github*
  → https://github.com/davHub/DeepLearningAPI
- **Code 3** - Yu-Gi-Oh card recognition - *Github*
  → https://github.com/davHub/DeepLearningAPI
- **Code 4** - Counting People - *Colab*
  → https://drive.google.com/open?id=1hlmlDtVtL89Xp0fLIVd3SNkytIF9Am1P
- **Code 5** - Car Generator - *Colab*
  → https://drive.google.com/open?id=1vOLj9Mx7SRtyeZAbfayE8jiw6mE4S-D-

Extensions

Yu-Gi-Oh Generation

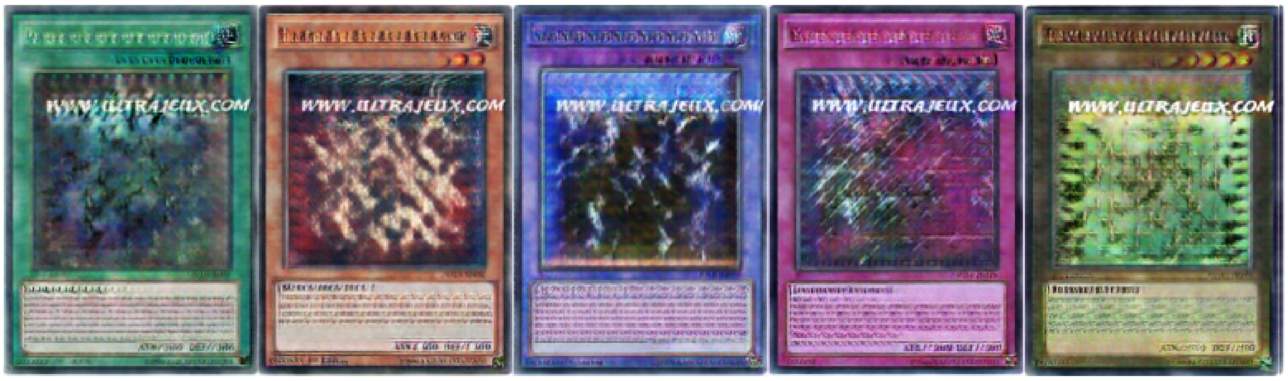


Figure III.10 – Extension Yu-Gi-Oh 1

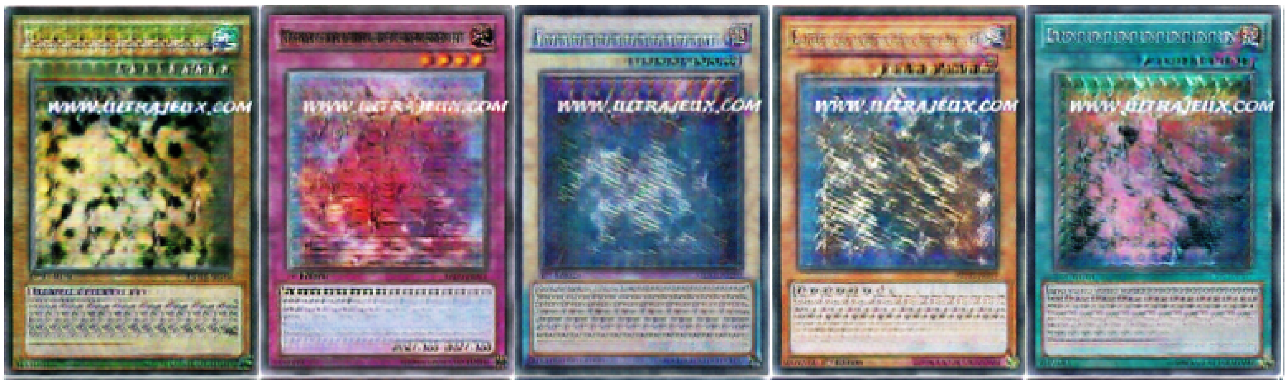


Figure III.11 – Extension Yu-Gi-Oh 2

Car Generation

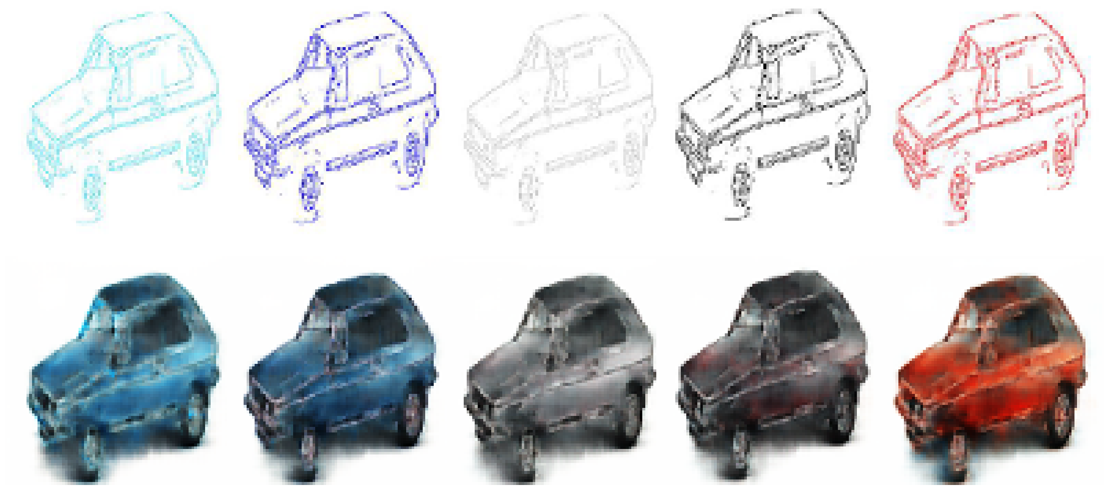


Figure III.12 – Extension Car 1

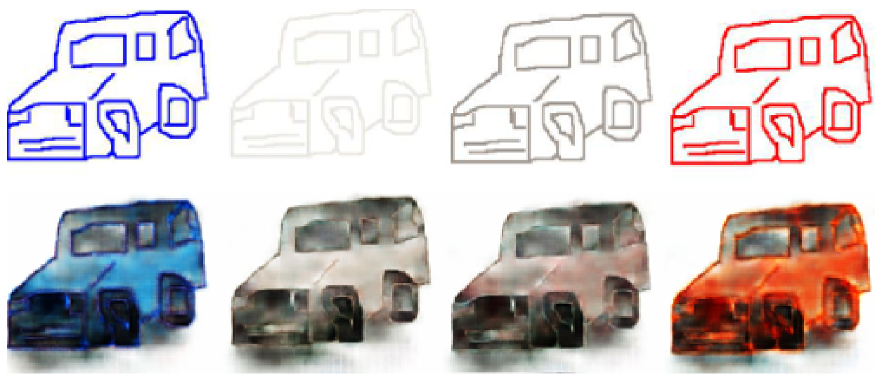


Figure III.13 – Extension Car 2

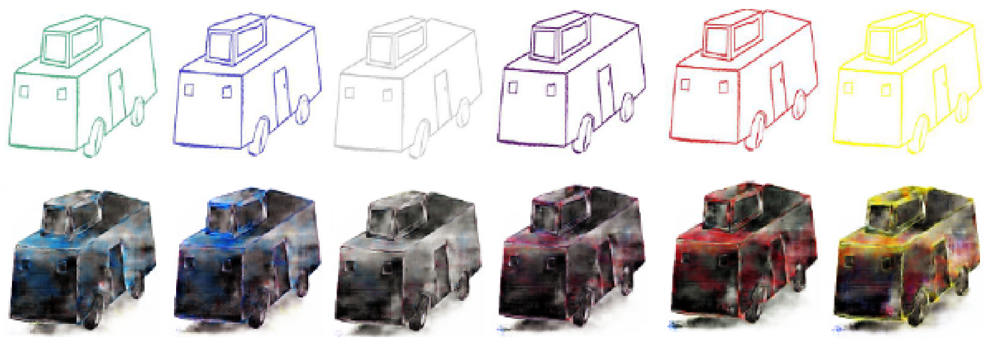
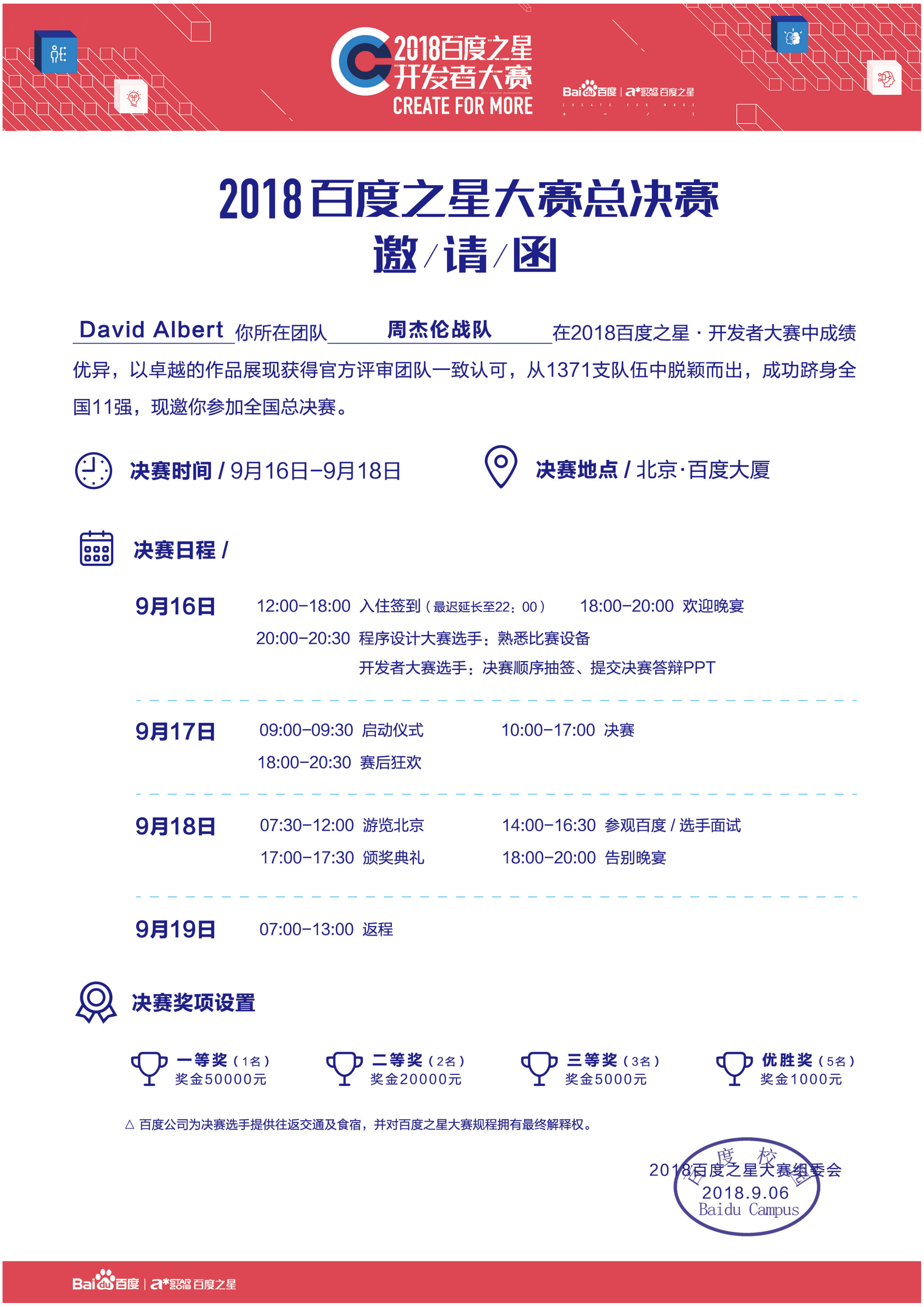


Figure III.14 – Extension Car 3


2018百度之星
开发者大赛
CREATE FOR MORE
Bai du 百度 | a* 百度之星

2018百度之星大赛总决赛

邀/请/函

David Albert 你所在团队 **周杰伦战队** 在2018百度之星 · 开发者大赛中成绩优异，以卓越的作品展现获得官方评审团队一致认可，从1371支队伍中脱颖而出，成功跻身全国11强，现邀你参加全国总决赛。

决赛时间 / 9月16日–9月18日　　**决赛地点 /** 北京·百度大厦

决赛日程 /

9月16日
12:00–18:00 入住签到（最迟延长至22：00）　18:00–20:00 欢迎晚宴
20:00–20:30 程序设计大赛选手：熟悉比赛设备
开发者大赛选手：决赛顺序抽签、提交决赛答辩PPT

9月17日
09:00–09:30 启动仪式　10:00–17:00 决赛
18:00–20:30 赛后狂欢

9月18日
07:30–12:00 游览北京　14:00–16:30 参观百度 / 选手面试
17:00–17:30 颁奖典礼　18:00–20:00 告别晚宴

9月19日
07:00–13:00 返程

决赛奖项设置

一等奖（1名）奖金50000元　**二等奖**（2名）奖金20000元　**三等奖**（3名）奖金5000元　**优胜奖**（5名）奖金1000元

△ 百度公司为决赛选手提供往返交通及食宿，并对百度之星大赛规程拥有最终解释权。

2018百度之星大赛组委会
2018.9.06

百度校园
Baidu Campus

Bai du 百度 | a* 百度之星

Figure III.15 – Baidu Invitation for the final competition