The Next Generation Image Translation App with GPU Acceleration

Building from heavy-training to ready-to-use Deep Learning Model to deployment

Kar Chun, CHAN
Department of Computer Science and Engineering &
Department of Mathematics
The Hong Kong University of Science and Technology
paddy.chan@connect.ust.hk

Qiwei, LI
Department of Computer Science and Engineering &
Department of Mathematics
The Hong Kong University of Science and Technology
qlicd@connect.ust.com

ABSTRACT

In the recent century, the development of deep learning brings us to handle computer vision tasks from impossible-to-be-done to possible-to-be-done. This revolution to computer vision is attributed from the advancement of deep learning by which the learning model is getting complex enough to resolve many problems from image classification, Captioning, Object Detection, Object Tracking, Style Transfer, Image Translation, Deep Dream and Semantic Segmentation. Several deep learning architectures have been explored for solving image classification, object detection, object tracking and activity recognition challenges which are widely adopted in the real world.

In this paper, we will discuss four deep learning models' from the level of heavy training effort to no training effort (ready to use library). From below we address 4 examples of different deep learning approaches 1. Face Mask Detection and Object Detection, 2. DCGAN, 3. Face Recognition by VGG16 Transfer Learning and 4. Face Detector by Dlib. Each of these four examples will be analyzed and explained in below sections. In addition, we will discuss the migration of these four applications to a web platform by tensorflow.js. You can try our live demo at https://sirily11.github.io/ml-playground/#/

CCS CONCEPTS

• Computing methodologies → Neural networks;

KEYWORDS

Deep Learning, Neural Network, DCGAN, VGG16, Transfer Learning, HOG, SVM, TensorflowJS, Tensorflow, Keras.

Problem Definition

1.1 Face Mask Detection and Object Detection

In today's covid-19 situation, the face mask becomes a very crucial point in people's daily life. Most public places require the public to wear a face mask. However, it is hard for a human to monitor each visitor at everytime. So it is important for us to develop a system to automatically track people and send a warning when they don't wear a face mask at any time. There are many existing object detection models available. After comparing each model, we decide to use YOLO V5 [10] as the model for object detection. In the speed comparison with faster RCNN, YOLO V5 can achieve 20 times faster than faster RCNN in CPU only environment. YOLO V5 is the fifth version of the original YOLO's implementation. With YOLOv5s on Nvidia V100 GPU, we can even get 2ms speed on a 640 pixels size image, or 500 frames per second in equivalent. This surely meets our requirement for real time object detection. The below figure shows the speed comparison with YOLO V5.

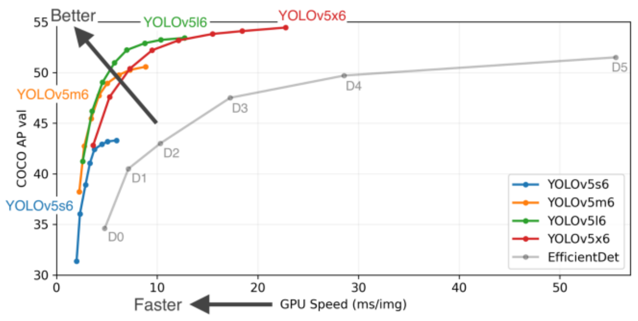


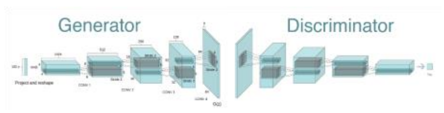
Figure 1.1 Speed comparison between different YOLOv5 models

In addition to our project's requirement, we will re-train this model targeting face mask detection only. And we will talk more about the final deployment of this model to the real application in the deployment section.

1.2 Deep Convolutional Generative Adversarial Network – DCGAN

Apart from Face mask Detection and Object Detection, this paper also will also discuss the implementation techniques, constraints of Deep Convolutional Generative Adversarial Network – DCGAN. Also, we will discuss its correlation between and model architecture with different datasets such as simple scene understanding and complicated scene understanding.

DCGAN was first started and proposed by Radford et al. (2015) [1] and is one of the most popular and successful network designs for GAN. It consists of convolution layers without max pooling or fully connected layers in between. It principally uses convolutional stride and transposed convolution for the down-sampling and the up-sampling. The figure below is the network design for the generator and discriminator.



The mathematical theory of DCGAN can be explained by below formula [6]:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

Let X be our true image dataset and Z be the normal distributed noise. Let p(z) be data from latent space Z. G and D are differentiable functions of the generator network and discriminator network respectively. D(x) represents probability that data come from real image dataset X. We train D to maximize the probability log(D(x)) and train G to minimize log(1 — D(G(z)).

DCGAN architecture can be easily understood by summarizing the below characteristics [2]:

- Use convolutional stride instead of max pooling
- Up-sampling by transposed convolution
- No fully connected layers
- Use Batch Normalization except for the output layer for generator and the input layer of the discriminator
- Use ReLU & LeakyReLU as the activation function

1.3 Face Recognition by Transfer Learning

Apart from the DCGAN which requires heavy network training, this paper will also discuss another face recognition deep learning application which requires only a medium level of model training by transfer learning of VGG16.

The reason for using Transfer Learning for the demonstration is that we would like to introduce another way of deep learning which does not need to heavily train the entire network from scratch and instead, we could reuse a pretrained model on a new problem.

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. [7]

The model architecture is described as below:

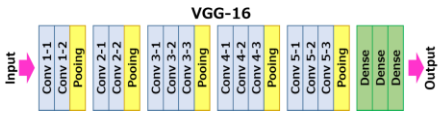


Figure 1.3 Model Architecture of VGG16

In this demonstration, we will create a simple application to demonstrate the implementation overview of Face Recognition by the use of Transfer Learning from VGG16.

1.4 Face Detector by Dlib

Apart from the VGG16 transfer learning which requires medium level of training effort when compared with DCGAN, this paper will also discuss another example of face detection deep learning application which does not require any training effort, we can just use the ready-to-use API, Dlib, to solve the problem.

Dlib is a python package wraps face detection functions into a simple, easy to use API

Two face detection method were built into the library

- HOG & Linear SVM face detector - for frontal face detection
- Max-Margin (MMOD) CNN face detector - for vary angle face detection

And, in the paper, we will demonstrate the implementation of HOG & Linear SVM face detector.

2. Data

2.1 Face Mask Detection and Object Detection

The training data for the face detection model are coming from the kaggle dataset https://www.kaggle.com/andrewmvd/face-mask-detection. This dataset contains 853 annotations for both faces with face masks and faces without face masks. In order to train the model in YOLO setup, we need to convert the annotation from the current format to the YOLO format.

2.2 Deep Convolutional Generative Adversarial Network – DCGAN

Two datasets are used in this project to show the correlation between DCGAN and the nature of datasets, simple and complicated image scene.

The first dataset was called Fruits 360 and was downloaded from https://www.kaggle.com/moltean/fruits [3]. The repository provides 131 classes of fruits which totally contain 90483 images. For this project, to reduce the training burden, 11 classes of fruits had been used which contained 5000 fruit images. It serves as the dataset of simple image scene.

The second datasets called Hard Hat Worker Dataset and was downloaded from https://makeml.app/datasets/hard-hat-workers [4]. The repository provides 5000 images of the scene of a construction site in which contain helmet, workers, site environment, equipment and etc. It serves as the dataset of complicated image scene.

2.3 Face Recognition by Transfer Learning

The dataset was prepared and processed by openCV library which takes 100 photos at a time from the computer camera.

In this demonstration, we take 3 human faces images as the dataset to train the custom layers of VGG16 for the face recognition application.

2.4 Face Detector by Dlib

In this demonstration, we take 3 worker qualification ID card images as the input image for the face detection, crop the face and save as a jpeg file. For the input image, please refer to our code repository. [9]

3. Methodologies

3.1 Face Mask Detection and Object Detection

In the process of training the face mask detection model, we first try the Faster R-CNN model on top of the Resnet 50. However, it has two drawbacks. (1) The model generated is too large for the deployment of the web platform. (2) The model runs very slow and almost unusable on the CPU only machine (less than 2 frames per second). After this experiment, we focus on the SSD model and YOLO model. And after the comparison from multiple web sources [11], we think YOLO can provide better performance for the task. Also YOLO has better internet resources for us to reference when we have a problem.

3.2 Deep Convolutional Generative Adversarial Network – DCGAN

Three different approaches were used to implement the DCGAN with two different datasets of simple (fruits datasets) understanding and complicated image (worker dataset) scene understanding.

The reason for designing these 3 experiments is to identify the constraints of the original DCGAN which was proposed by Radford et al (2015) and the demonstration of some training tricks to improve the performance of the DCGAN.

Methodologies of each experiments are as below:

3.2.1 Experiment 1 - Simple scene understanding with the original DCGAN architecture

For this experiment, it is used to prove the original DCGAN is good for simple scene understanding which the image contains only limited features apart from CIFAR10, SIMPSON, MNIST, MNIST FASHION etc.

Below model architecture and hyperparameters were used:

- Number of CNN Layers: 5 (stride 2 and kernel size 5)
- Input image size: 64 x 64
- Batch Size: 64
- Optimizer: tf.compat.v1.train.AdamOptimizer
- Learning Rate: 1e-4
- Activation Function: LeakyReLU (alpha = 0.2) for Generator and ReLU for Discriminator
- Regularization: Dropout (0.3)
- Loss Function: tf.compat.v1.losses.sigmoid_cross_entropy
- Epoch: 1000

3.2.2 Experiment 2 - Complicated scene understanding with the original DCGAN architecture

For this experiment, the purpose is to prove the original DCGAN IS NOT GOOD for complicated scene understanding which the image contains many visual features or objects. Both the model architecture and hyperparameters are the same as the set used in experiment 1.

Below model architecture and hyperparameters were used:

- Number of CNN Layers: 5 (stride 2 and kernel size 5)
- Input image size: 64 x 64
- Batch Size: 64
- Optimizer: tf.compat.v1.train.AdamOptimizer
- Learning Rate: 1e-4
- Activation Function: LeakyReLU (alpha = 0.2) for Generator and ReLU for Discriminator
- Regularization: Dropout (0.3)
- Loss Function: tf.compat.v1.losses.sigmoid_cross_entropy
- Epoch: 800

3.2.3 Experiment 3 - Complicated scene understanding with the re-designed DCGAN architecture and training tricks

For this experiment, we take the result from experiment 2 and progressively tuning the model architecture and training hyperparameters and finally work out the below optimal model architecture and hyperparameters:

- Number of CNN Layers: 6 (stride 2 and kernel size 5)
- Input image size: 64 x 64
- Batch Size: 8
- Optimizer: tf.compat.v1.train.AdamOptimizer
- Learning Rate: 0.0002
- Activation Function: LeakyReLU (alpha = 0.2) for Generator and ReLU for Discriminator
- Regularization: Dropout (0.3)
- Loss Function: tf.compat.v1.losses.sigmoid_cross_entropy
- Epoch: 200
- Training Tricks:
  - ✓ Add more filters and increase the conv layers
  - ✓ Use smaller batch size (from 64 to 8)
  - ✓ Use ReLU instead of Leaky ReLU
  - ✓ Increase lr from 1e-4 to 0.0002
  - ✓ Add Gaussian Noise to both Real & Fake image
  - ✓ Gaussian Weight Initialization for G & D network
  - ✓ Label Smoothing for both real and fake image

3.3 Face Recognition by Transfer Learning

The implementation of face recognition by VGG16 transfer learning is very convenient and easy to use. The below implementation steps are used for the demonstration. For implementation details, please refer to our code repository. [9]

3.3.1 Data Preparation and pre-processing

OpenCV library is used to prepare the training and testing data from the computer camera, which takes 3 human face data and each human face will take 100 images as the training data to train the custom layers of VGG16.

3.3.2 Define the customer layers

A python method was defined to establish the three customer dense layers of VGG16

3.3.3 Add custom layers to pretrained layers

The below code first construct the VGG16 model object, and then add the defined custom layers in previous step on the top of the VGG16 model, implementation code is shown below:

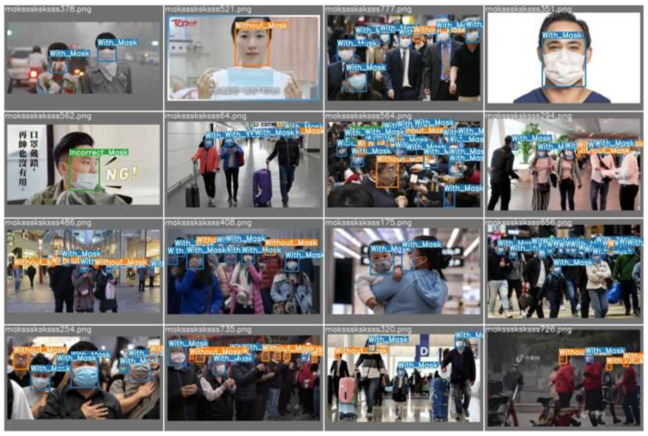
3.4 Face Detector by Dlib

In the library, a function called get_frontal_face_detector can be used to detect the front face from an image, it does not accept any parameters. A call to it returns the pre-trained HOG & Linear SVM face detector included in the Dlib library. For implementation details, please refer to our code repository. [9]

4. Discussion

4.1 Face Mask Detection and Object Detection

The training for 30 epochs on a Nvidia v100 machine takes roughly 45 minutes to finish. After 30 epochs training, we reached 0.103 loss at the end. That is enough for the task we proposed. The following picture shows how the final result looks like.



As you can see, the model clearly identifies the position of the face mask and provides a very accurate label on it.

4.2 Deep Convolutional Generative Adversarial Network – DCGAN

4.2.1 Experiment 1 - Simple scene understanding with the original DCGAN architecture

We first explored the original DCGAN model proposed by Radford at a. (2015) [1] by the Fruits dataset. Since there are so many online resources to train DCGAN model on simple scene image such that there is only one simple object appearing in the image. For example, MNIST dataset, CIFAR10 dataset, Simpson dataset and MNIST Fashion dataset. We considered to use the fruit dataset which also contains only one simple object and possesses the same characteristics of the mentioned datasets mentioned before. As predicted, the final result and performance is good. As illustrated in Figure 4.2.1a, the shape and appearance of all the generated can be easily identified. Also, as illustrated in figure 4.2.1b the generator loss are very stable throughout 1000 epoch while the discriminator loss are quite fluctuated. It is not difficult to understand that the discriminator continuously distinguishes between the training real images (training set) and the generated fake images and hence, contributes to the fluctuation of discriminator loss. In every epoch, the generator is trying to maximize the chance of the discriminator to misclassify the image and at the same time, the discriminator is trying to maximize the chance of discriminator correctly classifying the real and generated fake image. And, observing the full set of generated images, some are in good quality and some are in bad quality, same as the reflection in figure 4.2.1b.

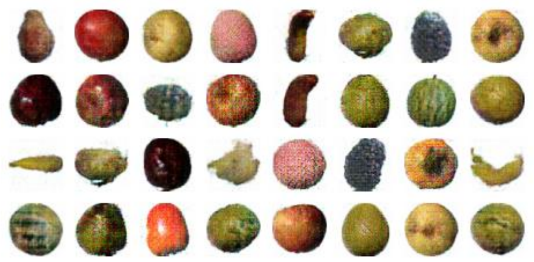


Figure 4.2.1a: Final Result of Experiment 1

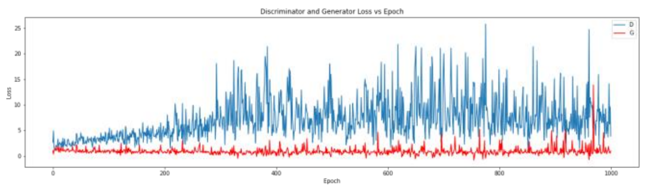


Figure 4.2.1b: D & G Loss vs Epoch of Experiment 1

4.2.2 Experiment 2 - Complicated scene understanding with the original DCGAN architecture

After the first exploration of DCGAN by Fruit Dataset, we then moved the same DCGAN model architecture and parameters for training the Worker Hard Hat Dataset. The purpose of this experiment is to prove there is correlation between DCGAN model architecture and the nature of datasets, simple and complicated image scene.

As illustrated in figure 4.2.2a and 4.2.2b, the final result of images is bad and also, both the generator and discriminator loss are very fluctuated. The fluctuation of generated loss implies that the Generator Network cannot learn the true distribution of the training data and perform poorly to generate good quality images. And, obviously, since the generator loss is not stable which causes the discriminator network also performed badly since it cannot guide the generator network with proper gradient to generate good quality image.

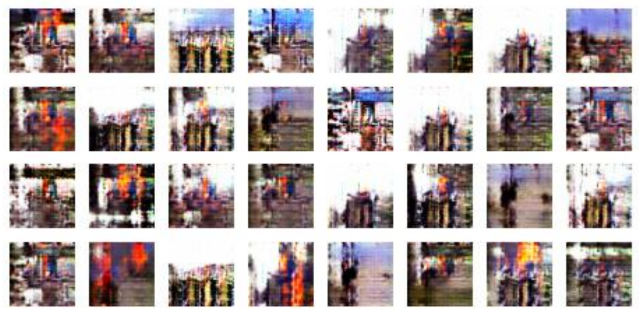


Figure 4.2.2a: Final Result of Experiment 2

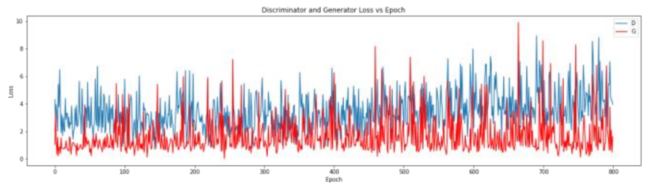


Figure 4.2.2b: D & G Loss vs Epoch of Experiment 2

4.2.3 Experiment 3 - Complicated scene understanding with the re-designed DCGAN architecture and training tricks

According to the result from Experiment 2, it clearly proved that the original DCGAN IS NOT GOOD for complicated image scene in which the image contains many visual features (multiple objects). And at the same time, we infer that there may have some causes of such a bad performance which are shown below:

- The capacity of DCGAN network is not strong enough for complicated scene image
- Batch size may be too large for the generator network to learn the distribution of training images well
- Lack of complexity of the discriminator training led to poor performance of discriminator [5]
- The fluctuation of Generator and Discriminator loss may be due to a random weight initialization.

To address the above uncertainties of the poor performance of DCGAN, we issued the below training tricks which target to improve the performance of DCGAN and the result of the generated image:

- Add more filters and increase the convolutional layers

  Reason: Apply more parameters for learning complicated image scene which require more feature map

- Use smaller batch size (from 64 to 8)

  Reason: Large batch size will cause poor generalization and so, we move to a smaller batch size so as to achieve a convergence.

- Use ReLU instead of Leaky ReLU

  Reason: for experiment only as training DCGAN is very unstable

- Increase learning rate from 1e-4 to 0.0002

  Reason: for experiment only as training DCGAN is very unstable

- Add Gaussian Noise to both Real & Fake image

  Reason: Give complexity to discriminator training since it helps giving some stability to the data distributions of the two competing networks. [5]

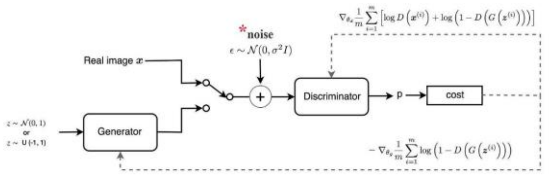
  
  $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
  *noise
  $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$
  Real image x
  Discriminator
  p
  cost
  $z \sim \mathcal{N}(0,1)$ or $z \sim U(-1,1)$
  Generator
  $-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$

  Figure 4.2.3a: Mathematical depiction of Gaussian Noise implementation

- Label Smoothing for both real and fake image

  Reason: Same as above

- Gaussian Weight Initialization for G & D network

  Reason: Improve the fluctuation of Generator and Discriminator loss

As illustrated in figure 4.2.3b, the quality of the final result got improved but still has room for further improvement. And, at the same time, both the generator and discriminator loss are stabilized after deepening the network architecture and apply the training tricks as described above, as shown in figure 4.2.3c.

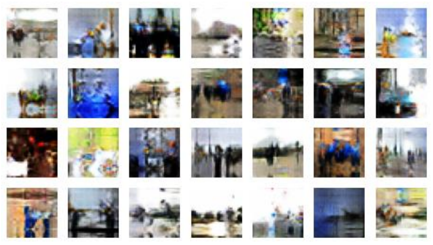


Figure 4.2.3b: Final Result of Experiment 3

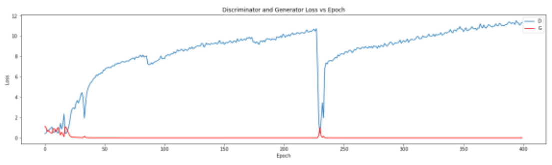


Figure 4.2.3c: D & G Loss vs Epoch of Experiment 3

4.2.3 Implications of the 3 experiments

It is hard and time consuming to train DCGAN. Even though a well trained DCGAN network works well in one dataset, it doesn't mean the same DCGAN network will be fit to another dataset. The reason is that both generator and discriminator networks are trained simultaneously, making improvement to one network will come at the expense from another network.

The concept of DCGAN is not difficult but the implementation to generate quality fake images will be very tricky and it is hard to identify an exact way to do it. It requires the consideration of a combination of model tuning such as model architecture, hyperparameters, Label Switching, add Gaussian Noise to real and fake images etc. For the future work, we suggest applying Multi Scale Gradients (MSG GAN) to further improve the learning capability of the DCGAN network. And we summarized the Key Learning of DCGAN as below:

- The concept of GANs is not that hard to understand. But the implementation to produce quality images can be difficult.
- Different dataset required different DCGAN architecture and Training Tricks (combination of hyperparameters to handle the mode instability and quality)
- GPU is highly required
- Spend countless hours for a good outcome
- High learning rate will cause Mode Collapse
- Familiar with Tensorflow and Keras after the project

4.3 Face Recognition by Transfer Learning

The overall performance of transfer learning is good, 3 epochs was run and attained an training accuracy rate of 98.39% but validation accuracy only 50.97% due to limited training data.

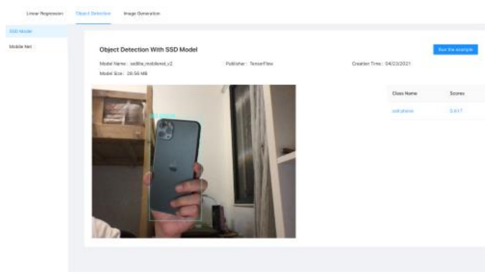
The result correctly predicts the class label of different test images. For result, please refer to our code repository. [9]

4.4 Face Detector by Dlib

Among the DCGAN and VGG16 transfer learning demonstration, Dlib's HOG + Linear SVM face detector is comparatively fast and efficient. By nature of how the Histogram of Oriented Gradients (HOG) descriptor works, but it is not invariant to changes in rotation and viewing angle [8]

5. Deployment

In order to deploy all the models we talked above to this platform, we use the toolkit provided by tensorflow.js. Since it only supports tensorflow's model, for those models trained by pytorch (YOLO V5), we will use the conversion tool to first transform the model into tensorflow model and then use the conversion tool to transform it to tensorflow.js model. The following screenshot shows the final deployment we have for the object detection model, GAN, and face mask detection.



Object Detection With SSD Model

Model Name : ssdlite_mobilenet_v2 Publisher : TensorFlow Creation Time : 04/23/2021

Model Size : 26.56 MB

Class Name	Scores
cell phone	0.617

The performance for this web deployment is pretty good especially for the devices with GPU built in which will have a better performance. This project shows the future of ML such that people can train their model in python or in any language they want and can easily convert their effort into a usable product.

6. Conclusion

In this report, we discussed a lot of interesting aspects where the ML meets the computer vision. From object detection to image generation, from face mask detection to face detection and from Python to Javascript. The evolution of Machine Learning surely helps a lot for people to make impossible possible and transform from prototype to a real product which runs everywhere.

ACKNOWLEDGMENTS

Thanks to Professor CK Tang and Teaching Assistant Rex of The Hong Kong University of Science and Technology for the support of this project.

REFERENCES

[1] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks.

[2] Medium 2018. GAN — DCGAN (Deep convolutional generative adversarial networks) from https://jonathan-hui.medium.com/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[3] Kaggle 2020. Fruits 360 dataset: A dataset of images containing fruits and vegetables from https://www.kaggle.com/moltean/fruits

[4] MakeML 2020. Hard Hat Workers Dataset from https://makeml.app/datasets/hard-hat-workers

[5] TowardsDataScience 2019. 10 Lessons I Learned Training GANs for one Year from https://towardsdatascience.com/10-lessons-i-learned-training-generative-adversarial-networks-gans-for-a-year-c9071159628

[6] TowardsDataScience 2018. Generative Adversarial Networks (GAN)- An AI—'Cat and Mouse Game' from https://towardsdatascience.com/art-of-generative-adversarial-networks-gan-62e96a21bc35

[7] NeuroHive 2018. VGG16 – Convolutional Network for Classification and Detection from https://neurohive.io/en/popular-networks/vgg16/

[8] PyImageSearch 2021. Face detection with dlib (HOG and CNN) from https://www.pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/

[9] Github 2021. 5016Project2021 from https://github.com/paddykcc/5016Project202

[10] Github 2021. Yolov5 https://github.com/ultralytics/yolov5

[11] One stop for object detection. https://medium.com/swlh/one-stop-for-object-detectors-2c99daa08c50