MASTER OF TECHNOLOGY
(INTELLIGENT SYSTEMS)

PROJECT REPORT

VisionLab
(Facial Images Generator & Detector)

GROUP MEMBERS

YIN TIAN SHI

YANG LU YI

YU YU

Contents

1.0 Executive Summary

In the early stage of overall system design, we did a lot investigation and practical trials on model trainings with different deep learning algorithms. In order to make learnt technical knowledge into practice well, we designed an intelligent we system named “VisionLab”, which including Images Generator and Images Detector.

One the one hand, with adoption of unsupervised learning methods with GANs, we integrated two our own trained models (DCGAN & StyleGAN2) and one pre-trained StyleGAN2 into image generator. The DCGAN model can produce virtual human face for internet celebrities and cute cartoon faces. Although the StyleGAN2 model has been trained for 2,120,000 steps, its performance is still not good, this model will be an experiment for users to try. After integration, the pre-trained model is able to generate 4 types of virtual faces, such as Asian face, celebrity face, online celebrity faces and cartoon faces.

On the other hand, many kinds of supervised learning with Deep CNN algorithms have been used ensemble in model training for image detector to identify the authenticity of real/ virtual human and cartoon faces, just like VGG16/19, ResNet50/101/152, Xception, InceptionResNet-v2, MobileNet-v2, DenseNet and NASNet.

While interact with VisionLab system, users can revise system settings flexibly to change the no. of generated images per batch (range of values 0-20), single or mixed model types for generating fake faces and detecting uploaded photos. On Image Generator page, users are allowed to get more virtual faces via click “+” button, mark and download any favorite faces. When a photo is uploaded in Image Generator page, a cute robot responds to users whether that photo is real or fake, and now users are allowed to choose correct categorization through click “Agree” and “Disagree” buttons.

Furthermore, our system not only provides face detection and generation, but also offers more valuable features, such as the ability to upload new weight files if users train their own dataset with our proposed models, and users are also able to train detector models online as we have formed a model auto-training pipeline in our system.

If you want to learn more about the system VisionLab, please access to our GitHub link via clicking the project Site icon on system home page.

2.0 Business Problem Background

Artificial intelligence, as the current direction of technological development and innovation, has given rise to many emerging technologies. For instance, the face recognition technology, which is now widely used in various fields. It is getting closer to our daily lives, such as mobile phones unlock, community security upgrades and missing person search and rescue etc. However, at the same time, it has also triggered to the public’s unease and concerns. Because there are still many

uncertainties and risks associated with the application of facial recognition technology.

Out of interest, we would like to design an intelligent web system including facial images generator and detector. The generator can produce colorful virtual/ fake human and cartoon faces, while detector is able to detect the authenticity of facial images that users uploading.

The main purpose of generator is to provide many types of face materials for customers to use as will and no need to worry about face copyright issues. We can image that areas of legitimate use of facial photos are starting to emerge. Artwork and marketing departments can save large budget on expensive photo shoots, instead get enough faces of underrepresented groups to balance media and marketing materials. In addition, more diverse cartoon faces have important artistic and practical values since it can be precious pictures for creation. For example, it helps artists to portray the ultimate fantasy for people to see. General speaking, requesting professionals to draw exclusive cartoon avatars for everyone is impractical and limits creativity.

On the other hand, the detector can help people to distinguish dubious use of applications that have brought attention to AI-generated images: profile pictures for fake news and dating services. And it is also suitable for airport, customs and other ID verification scenarios when using face detection for identity verification.

While upload a clear facial photo to our system, users are able to verify its authenticity and generate corresponding cartoon face. In the meantime, getting lots of various AI-generated human faces for commercial or personal use.

Note,

The definition of "virtual/ fake" is that facial photos are AI-generated.
The definition of "real" is that facial photos are taken by camera or drawn by artists.

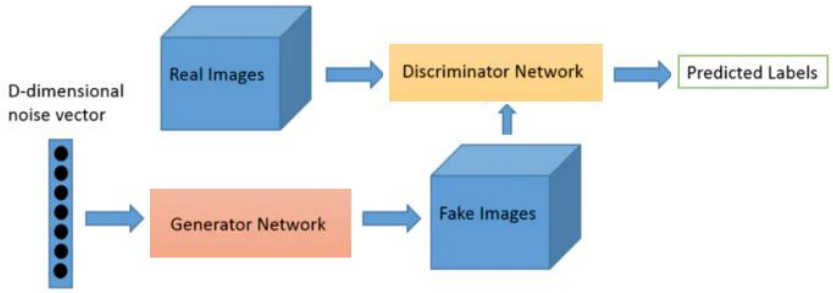
3.0 Market Research

3.1 Face Images (Human/ Cartoon) Generator

It's obvious that Face Recognition, Target Detection, Object Recognition and Video Analytics are quite popular in the current application of computer vision. Such trend is understandable and acceptable since enterprises require stable and industry-accepted algorithmic models with consideration on limited budget and resources. While the generated models are not recognized in industrial applications widely and still mostly at the theoretical level.

As times goes, the technologies on computer vision has changed dramatically. Traditional approaches based on artificially designed features and traditional machine learning techniques have recently been replaced by deep neural networks. To be specific, Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) are the most commonly used methods of deep learning. The main advantage of deep learning approaches is that large amounts of data can be used to train models and then learn robust face representations of the changes that occur in the training data. It is significant that these

large datasets need to contain enough variation/ features, so that they can be generalized to new samples.
Furthermore, GANs have made tremendous progress since it was introduced by Ian Goodfellow in 2014, with a wealth of results in terms of theoretical algorithms and applications. Of interest is that the evolution of face generation from its early blurred stage to its current level of realism.



The simple idea of GANs is to use two models, a generative (G) model and a discriminative (D) model. The D model is used to determine the authenticity of a given image (an image taken from a dataset), while the G model is to create an image that looks like a real image. In the beginning, both models are yet trained, instead they are together against training. G model generates a picture to cheat D model, then D model detect whether the picture is real or AI-generated. Finally, in the process of training these two models, they can become more and more capable till reach a steady state.

With the popularity of GANs, there are at least a hundred of available related derivative models. Please refer this blog to learn various different GANs [URL: https://deephunt.in/the-gan-zoo-79597dc8c347]. In this large family of GANs, the typical applications in computer vision are listed in right spreadsheet for reference.

Application Scenarios	Types of GANs
High-quality image generation	TripleGAN StackGAN Deep Convolutional GAN (DCGAN) StyleGAN etc.
Style transformation	CycleGAN StarGAN etc.
Feature extraction	InfoGAN VAEGAN BiGAN etc.

As the part of our project is targeting to build a human and cartoon facial photos generator, we mainly investigated the current application status of this field.
Listed some popular face generators for reference.

Types of Generators	Web Description	Features	Theories	Company/ Founder
Human Face Generator	ThisPerson DoesNot Exist.com	When refreshing the web, it can generate a new face image from a 512-dimensional vector.	Training model on a massive real dataset, then used GAN to product new images.	Philip Wang
	GENERATED PHOTOS	Users can select photos according to different filters (e.g. Random 100, All 30 on this page), Background Colors, Head Pose, Sex, Age, Ethnicity, Eye/ Hair colors and Emotion etc.	A large real faces dataset is taken in studio. Tagged and categorized photos before machine learning training and GAN producing new faces. Further ML applies on new faces to identify and remove flawed photos.	Ivan Braun, team of 20+ AI and Photography professionals

Cartoon Face Generator	Photo to Cartoon	Upload a photo locally/ Input the URL of a picture (<1M), wait for few seconds, a cartoon face will be generated. There is 10+ types of cartoon face with different filters.	Added Face ID loss. Proposed a new method named Soft-AdaLIN (Soft Adaptive Layer-Instance Normalization) that fuses both mean variance of encoder (photo features) and decoder (cartoon features) during denormalization. And addition of two hourglass modules before encoder and after the decoder improves model feature abstraction and reconstruction capabilities.	MINIVISION

After careful market research on applications of Face Images Generator, our summary is as follows.

1) The popularity of a new technology is related to many factors. As for an enterprise, cost control and profitability are its purpose. Both high threshold and large investment for face generation technology have limit its application scenarios in the short term, so many enterprises are hesitant in the layout of face generation. In addition, a good business model is crucial if stakeholders want face generation technology to enter thousands of households like online shopping.
2) Face generator has a high requirement on image datasets. Not only does it require large and diverse real faces to train models, but also each picture needs to be as clear and unobstructed as possible. How to get lots of real faces for commercial use may be a problem as it must solve copyright issue first.
3) The prospect of implementing face generation is really valuable, thus, it deserves to be explored by interested parties.

3.2 Face Images Detector

There is a Chinese saying, “seeing is believing”, which was the standard discernibility of right and wrong for our ancestors. But nowadays, with the introduction of various deep learning algorithms, what is "seen" may not be "true". For instance, some industry-leading face faking technologies (e.g. DeepFakes and Neural Textures) can produce faces that are hard to distinguish from the real ones. Thus, to avoid abuse of virtual faces and to protect the security of the social network, detecting and defending against fake faces has become a matter of urgency.

There are some good facial image detectors for reference.

1) Anti-Deepfake (Tencent)

Anti-Deepfake (ATDF) is dedicated to detect the authenticity of human faces in videos/ images. It can be widely used in a variety of scenarios for face detection, public figure identification and risk assessment. This product is designed to protect rights and interests of users and make AI-generated faces invisible, while it also helps company saving manpower costs and improving efficiency.

Application Scenarios	Features
This product is applicable on following scenes. • *Pornography domain:* With the rapid development of Deep learning algorithms, the number of pornographic contents with fake faces on the web has doubled, the views are more than 130 million. • *Social network domain:* The combination of fake videos, news and social networks has intensified the spread of Internet rumours and even influenced the direction of public opinions. As a result, we are always confused on the truth and falsehood. • *Online fraud domain:* The explosion of some face-swapping apps for entertainment has also raised concerns about the security of personal information (e.g. faces). If fraudulent methods and scenarios are used with AI face swap, perhaps the number of fraud victims will exponentially increase, which brings new challenges to online fraud management.	a) Efficient and rapid detection and analysis of faces in videos. b) ATDF analyses video contents hierarchically. If a face has been identified as fake by the detector, it will be sent to analysis module of the public figure for further identification, to ensure whether the public figure's face was used. Finally, ATDF will combine results of identification and assess the risk level of images/ videos.

2) Face X-Ray (Microsoft)

The technique was published in the paper "Face X-Ray for More General Face Forgery Detection", and according to the researchers, such tools can help prevent the misuse of face-swapping images.

Algorithms	Features
• *Main idea:* To identify the unique label of each image. There are a variety of reasons of label generations, such as software algorithms, hardware sensors etc. • *1st step:* Given a real face image, search for another image as a variant of the real one. Use face landmarks as matching criteria, then find out mixed area according to European distances. • *2nd step: G*enerate a mask to delimit the spliced area. • *3nd step:* With the training process, the boundary of blended image can dynamically generate the label data, and self-supervise the training of the framework based on CNN. Thus, a large amount of training data can be generated just by operating at the real image level. (Note, Input is an image, Output is a Face X-ray, Cross-entropy loss measures the accuracy of prediction.)	a) Whatever algorithm is used to synthesize facial images, can accurately detect them without targeted training. b) Not only can determine if a picture is composite, but also able to point out where the spliced area is. So, the technique supports both recognition and interpretation functions.

Overall, the new generation technology revolution represented by face detection has already started. These increasingly high requirements for the new technology require both data

accuracy and data security, and face detection has great potential in this regard. The technical strength and innovation ability of enterprises determine the direction of the whole industry, any a little bit of technical innovation on face detection may bring changes in the industry.

In the future, the main research direction of face detection will focus on some typical problems currently faced. Such as the similarity of facial structure, face posture, age changes, lighting changes in complex environments and face blocking by decorations etc.

Furthermore, relying on the rapid advancement of the Internet of Things and artificial intelligence, the application scenarios of face detection will become more widespread. With the development investment of national research institutions, the exploration of technology by enterprises and the promotion of the market, these will be the signs of a bright future for face detection.

4.0 Project Objectives & Success Measurements

Our team aims to create a web system named as "VisionLab" that supports both human & cartoon face generator and detector functionalities. On the one hand, facial images generator can produce different types of human faces (e.g. faces of Asian, celebrities) and cute cartoon faces. On the other hand, facial detector is able to identify those AI-generated human and cartoon faces as fake faces, while tell users those uploading facial images are real ones if they are products of cameras or drawn by artists.

While investigation and practical trials on model trainings with different deep learning algorithms, we would like to make learnt theories into practice well, so we aligned internally to apply unsupervised learning with GANs and supervised learning with Deep CNN methods in generator and detector separately. As for the facial generator, we adopted deep convolutional GAN (DCGAN) and StyleGAN methods to build respective models that can produce similar human faces of famous persons, internet celebrities and ordinary Asian who do not exist in this world. While the DCGAN model can also generate cartoon faces for professionals' reference. In addition, many kinds of algorithms have been used ensemble in model training for the facial detector to identify the authenticity of real/ virtual human and cartoon faces, just like Visual Geometry Group (VGG) 16/19, ResNet50/101/152, Xception, InceptionResNet-v2, MobileNet-v2, DenseNet and NASNet etc. Within our project timeline, we aim to experiment as many model architectures as possible, and suggest a model that is adequate in performance yet compact in size.

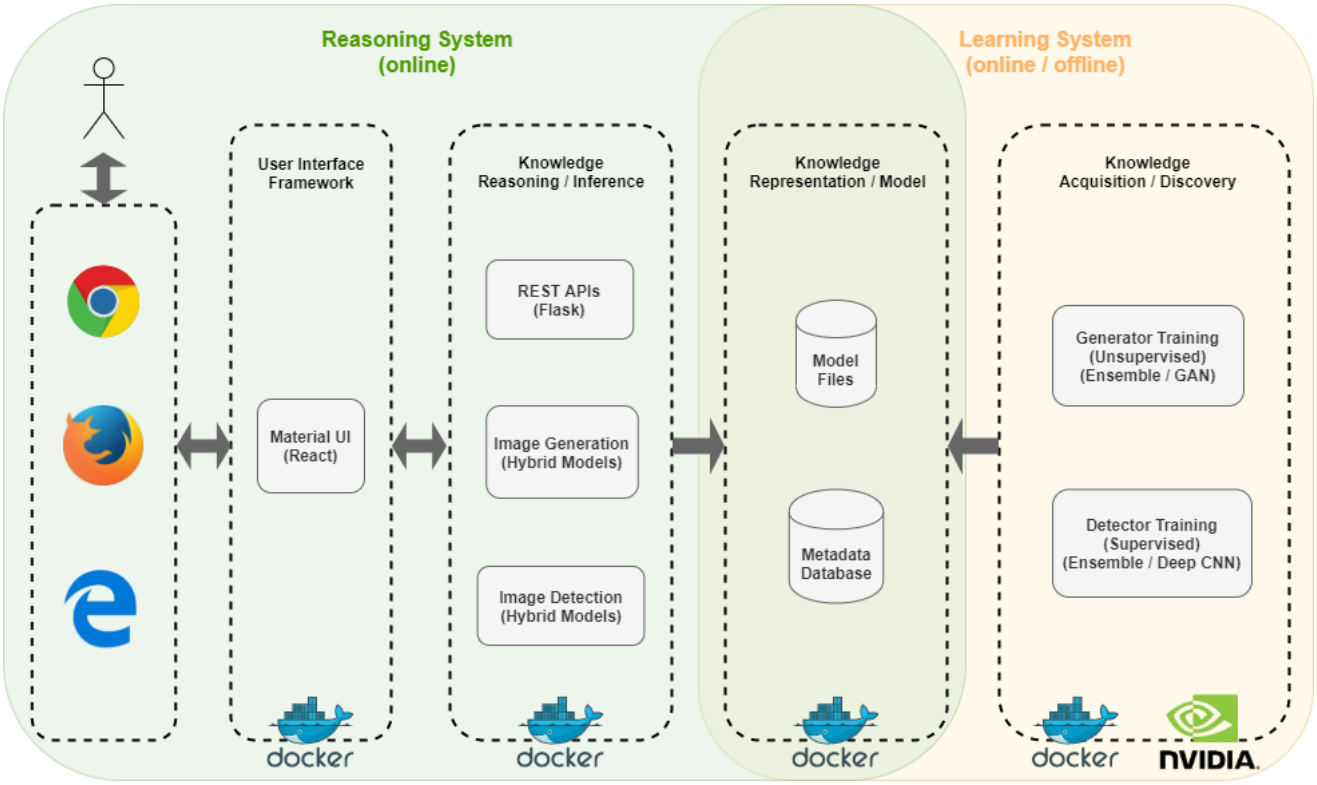
While interact with the intelligent web system, users can revise system settings flexibly to change the no. of generated images per batch (range of values 0-20), single or mixed model types for generating fake faces and detecting uploaded photos. Furthermore, users are able to get more fake faces via just click "+" button, mark and download favorite faces on Image Generator page. In addition, users are allowed to identify uploaded facial photo one by one. When a photo uploaded, there is a cute robot on the bottom right corner of the interface responds to users whether that photo is real or fake.

Use cases for image generator and image detector are as below.

No	Use Case	Functionality	User Input	System Output
1	Virtual Human Face Generator	End users are able to generate 10 number of random fake human faces each time. If needed, can generate more photos when click "+" button.	Users select AI-generated faces from drop list on Generator page.	A gallery view of human faces will show up on web UI, and user can download photos selectively or by batches.
2	Cartoon Face Generator	End users are able to generate 10 number of random fake cartoon faces each time. If needed, can generate more photos when click "+" button.	Users select AI-generated faces from drop list on Generator page.	A gallery view of cartoon faces will show up on web UI, and user can download photos selectively or by batches.
3	Real/ Virtual Human Face Detector	End users upload a photo via web UI, and a robot will suggest whether the photo is real or AI-generated.	Users upload photos via a drop-down box on detector page.	Classification result will print on screen, possibly with explanations.
4	Cartoon Face Detector	End users upload a photo via web UI, and a robot will suggest whether the photo is drawn by artists or AI-generated.	Users upload photos via a drop-down box on detector page.	Classification result will print on screen, possibly with explanations.
5	Hybrid Models & User Feedback	Web UI displays different models (e.g. cartoon face (DCGAN), Asian face (StyleGAN) & celebrity face (StyleGAN) etc.) for users to select through enabled/ disabled them on optional box of Image Generator page. Users are allowed to mark and download favourite photos.	Users select single or mixed models from optional box to generate photos, mark and download any photos.	The following info will show up on web UI. 1) *Types of models that generate images*. 2) *Users' preference of different models*. 3) *The history of image downloads from each model.*

5.0 Project Solution (To detail domain modelling & system design)

We built a web-based system for end user to interact with the intelligent system through uploading/ downloading facial images. Firstly, the User Interface design is rendered in the supported Web Brower (e.g. Google Chrome, Microsoft Eage and Firefox) by React framework in a Material style. Secondly, the Knowledge Reasoning/ Inference module integrates both hybrid models for Image generation and detection, as well as REST APIs for Front-end Service to perform requests or update to the whole application. Thirdly, all model files and Metadata Database (stores application data including training and users' data) are solid basics for Knowledge Representation. And all unstructured data are processed for the generator and detector trainings on the Knowledge Acquisition/Discovery stage. Finally, all modules are contained in Docker container.



5.1 Knowledge Acquisition/ Discovery

The purpose of knowledge discovery is to extract useful data from reliable sources and establish the basic foundation of overall information/ knowledge framework. In order to train models in good performance, we collected as much as possible various types of facial images from websites. In addition, we applied different algorithms for model trainings, for instance, unsupervised learning with GANs (e.g. DCGAN & StyleGAN2) and supervised learning with Deep CNN (e.g. VGG, Xception, InceptionResNet-v2 and ResNet etc.) methods have been used in generator and detector separately. And not only one algorithm has been used for several model trainings, so we also adopted ensemble approach to consolidate those outputs of individual good models, then to give a better final prediction.

1) Data Sources Preparation

These are face datasets we used for model trainings, and the requirement of all images is listed for reference.

Facial Photos Datasets	Requirements on Images
1) Real human faces • *Resolution: 128x128/ 256x256* • *No. of images: 200000+* 2) Virtual human faces • *Resolution: 128x128/ 256x256* • *No. of images: 10000+* 3) Cartoon faces draw by artists • *Resolution: 128x128/ 256x256* • *No. of images: 100000+* 4) AI-generated Cartoon faces • *Resolution: 128x128/ 256x256* • *No. of images: 10000+*	• The dataset came from Kaggle and GitHub, or any website that provides batch image downloads. • To collect 320000+ colorful human/ cartoon facial photos, the ratio of real and virtual photos is 1:1. • Human face images from the real world produced by a real camera with a real person as a subject in the age span of 0-116 years. • Real cartoon face images should draw by artists/professionals, those images are just like anime face from Japanese cartoon culture.

5) Realtime photos that user uploads • *Resolution: 128x128/ 256x256* • *Type of images: Any (balanced for real/virtual)*	• All facial images are unobstructed and clear without such phenomenon of motion blur (e.g. trailing shake), reflection and shadow. • The face in the photo is vertical, and angles of profile, pitch and deflection are less than 30°, 15° and 15° respectively. • The picture size of all photos can be 128x128/ 256x256. • The picture format is .jpg, .jpeg, .png.

2) Unsupervised learning with GANs for Generator Training

Since the limited resource and time duration, we not only trained our own models (DCGAN & StyleGAN2), but we also integrated a good pre-trained model (StyleGAN2) into our generator for user to produce human faces.

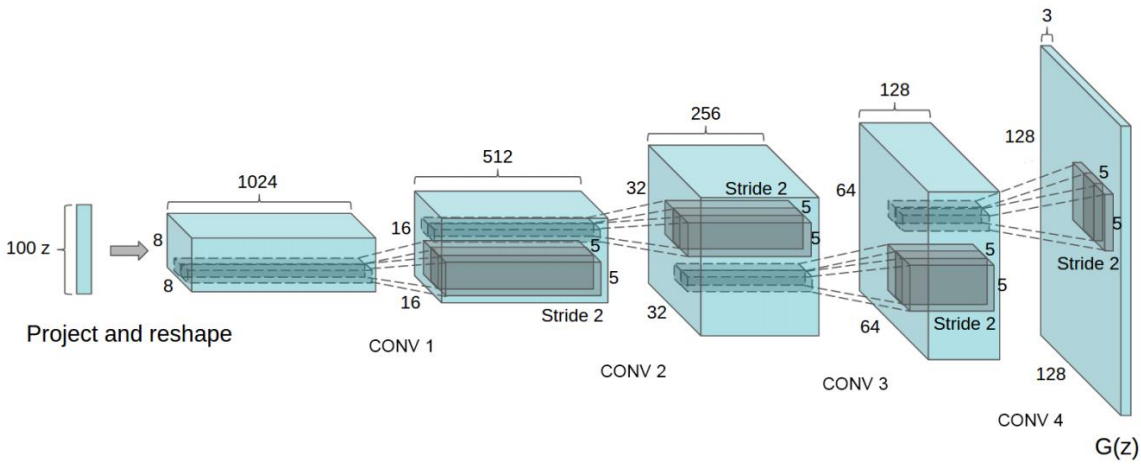
✓ ***DCGAN***

It is known that the best model for image processing in deep learning is CNN, so how do we combine CNN with another popular model GAN? DCGAN is one of the best attempts.

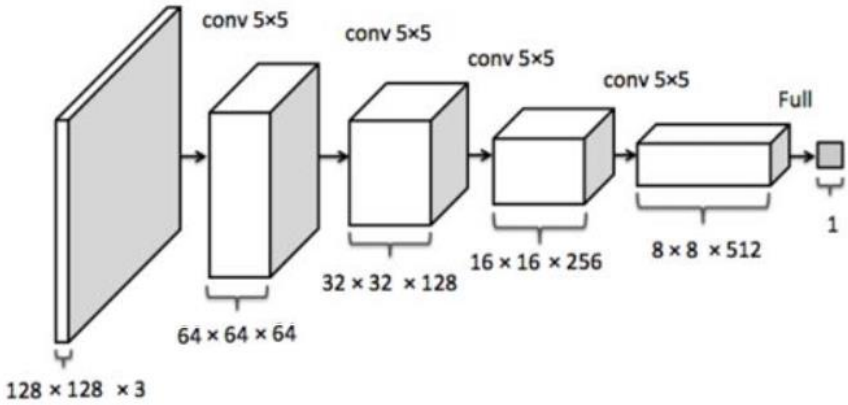
The generator and discriminator of DCGAN are replaced by two CNNs, but it is not just straightforward swap, DCGAN made some changes on the structure of CNN to improve the quality of samples and the speed of convergence. These changes are as follows.

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batch normalization in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

This is the network structure for generator of DCGAN. We can observe that a 100-dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions halve the number of stacked channels and double the width and length in turn, then convert the high-level representation into a 128×128×3 pixel image output.

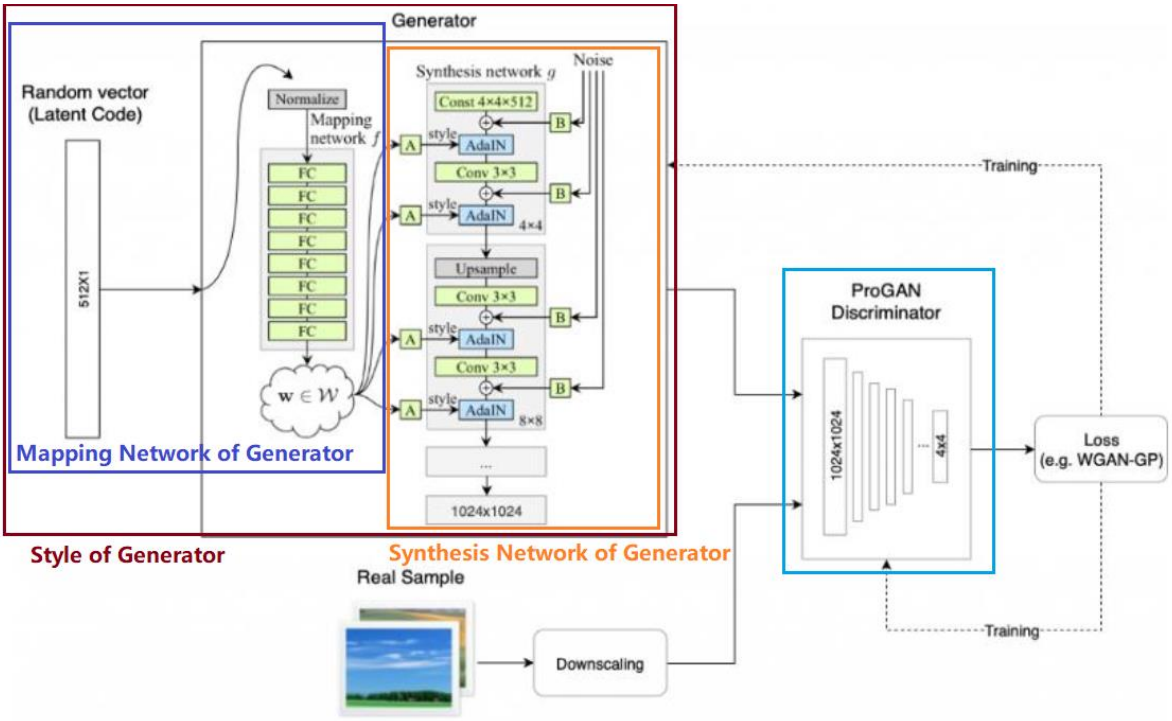


Additionally, the network structure for discriminator of DCGAN is as below. Discriminator can be seen as the reverse of the G-structure, and predict a binary (0, 1) output.



✓ ***StyleGAN & StyleGAN2***

StyleGAN not only generates high quality and realistic images, but also allows for better control and understanding of the generated image. The network architecture of StyleGAN is as below, which mainly includes four different modules, such as style network, mapping network & synthesis network of generator and discriminator. And StyleGAN2 focuses on fix artifacts and further improves the quality of generated images, especially in teeth, eye and other details.



- *Style Network of Generator*

  Style network is the whole structure of generator, it includes such components, parameter validation -> subnets setting -> variables setting-> mapped network output calculation -> update the average movement of W -> style blend regularization -> truncation tricks application -> synthetic network output

calculation. Particularly, the setting up subnetwork is the process of calling to build mapping and synthesis network of generator.

- *Mapping Network of Generator*
  It implements the mapping process from the initial generation code to the intermediate vectors and the mapping layer consists of 8 fully connected layers. This mapping network includes those components, such as input -> connection label -> normalized latent code -> mapping layer -> broadcast -> output.

- *Synthesis Network of Generator*
  It achieves the synthesis process from the intermediate vectors obtained from broadcast to generated images. Those components (pre-processing -> main input -> noisy input -> modulation function at the end of each layer -> early layer (4x4) structure -> blocks of the remaining layers -> the process of network growth and transformation -> output. To be specific, after the convolution, the modulation function at the end of each layer is targeting to incorporate noise and pattern arrangement (the process of above $\oplus$ and AdaIn). The process of network growth and transformation refers to the dynamically growing of the synthetic network during training period, in order to generate more higher resolution images.

- *Discriminator*
  Its function is to distinguish between real and synthetic images, which is basically what the generator looks like in reverse. Those component (pre-processing -> blocks setup-> the process of network growth and transformation -> label computation -> output) are defined in discriminator. And the process of network growth and transformation is identifying the dynamic growth of the network architecture, as the resolution of generated images increases during training period. In addition, when using dataset with labels for training, label calculation is setting the product of label value and discriminant score as the output.

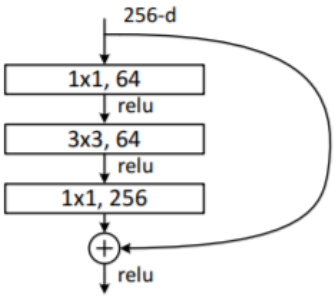
3) Supervised learning with Deep CNN for Detector Training

In order to train models with good performance and form a model auto-training pipeline, we referenced and researched many of the latest papers, and did some experiments with adoption of different algorithms. We have summarized all the used algorithms as follows, and attached links to original papers for reference.

✓ ***VGG16/19***

VGG is the first place in ImageNet's 2014 Targeting Contest. The biggest feature of VGG is the deepening of the entire neural network hierarchy by more thoroughly using 3x3 kernels to stack the neural network on AlexNet model. 16 and 19 correspond to the number of layers in the network that contains weights, such as the convolutional and fully connected layers. It is truth that VGG19 shows the best results, however, the parameters of VGG19 are much more than those of VGG-16, that's why everyone seems to prefer VGG16. Please refer to the original paper to learn more details. [URL: https://arxiv.org/pdf/1409.1556.pdf]

✓ ***ResNet50/101/152***

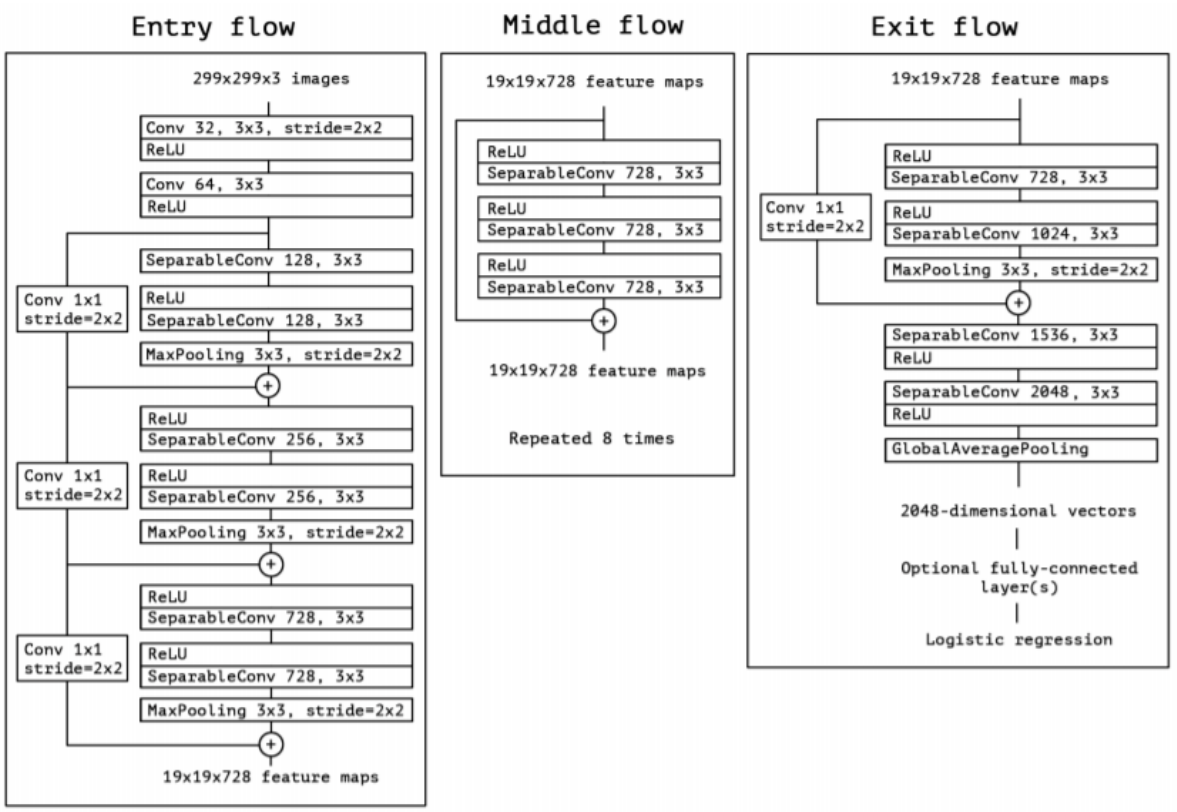


ResNet was presented in 2015 and won first place on the classification task of the ImageNet competition. The whole structure of ResNet50/101/152 is as shown on the right, it is also called as "building block" or "bottleneck design". The purpose of deeper ResNet is to reduce computation and no. of parameters. And 50, 101 and 152 network layers refer only to the convolutional/ fully connected layer, expect for the activation and pooling layers. We captured the following table of ResNet50/101/152 from original paper for reference. Please access to this link [URL: https://arxiv.org/pdf/1512.03385.pdf] to understand whole ideas of these three models.

layer name	output size	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times8$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times36$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$
	1×1	average pool, 1000-d fc, softmax		
FLOPs		3.8×10^9	7.6×10^9	11.3×10^9

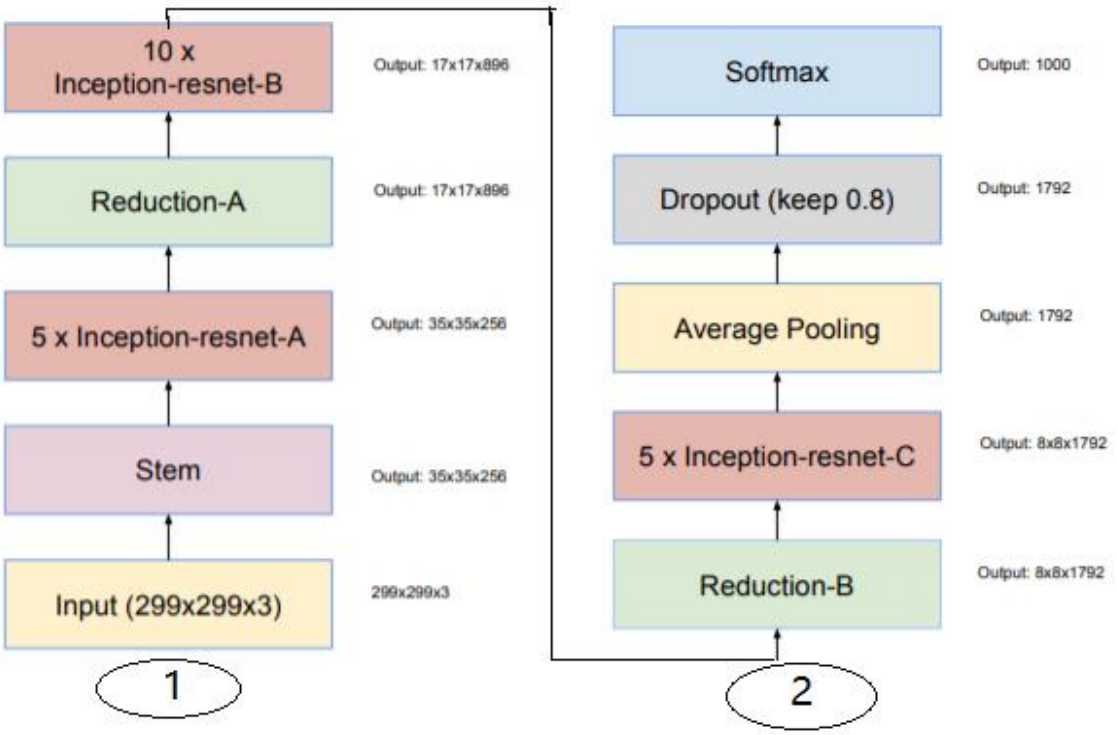
✓ ***Xception***

An Xception network consists of a series of Extreme Inception and some other conventional operations. The Inception module can learn more feature representations with fewer parameters and less computation by combining 1x1, 3x3, 5x5 convolutional kernels and pooling etc. Firstly, Inception module maps each channel of feature map to a new space using 1x1 kernels, the purpose is learning inter-channel correlations. Secondly, convolves through regular 3x3/ 5x5 kernels to learn both spatial and inter-channel correlations simultaneously. When all 3x3/ 5x5 convolution acts on the feature map of only one channel, the correlation between channels and spatial correlation is completely separated. In fact, by adjusting the ratio of the no. of branches of a 3x3 convolution to the no. of output channels of a 1x1 convolution, a series of states between traditional Inception and Extreme Inception modules can be achieved. Kindly refer to the original paper for more details. [URL: https://arxiv.org/pdf/1610.02357.pdf]



✓ ***InceptionResNet-v2***

The main idea of InceptionResNet-v2 is to combine residual and inceptive structures to get the advantage of residual, which accelerates the training of inceptive networks with a small improvement in accuracy. The residual inception block is as below. Please learn more details via this link [URL: https://arxiv.org/pdf/1602.07261.pdf].



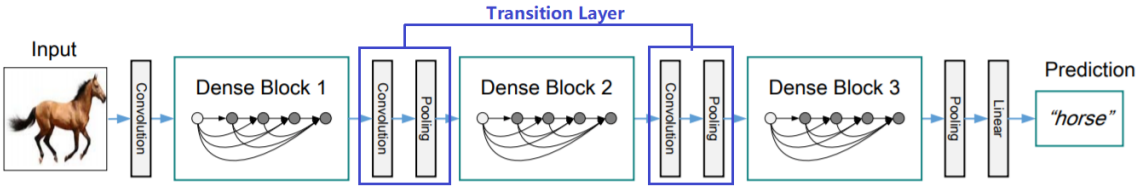
✓ ***MobileNet-v2***

MobileNet-v2 adopts depthwise separable convolutions (Xception variants) to build an efficient network based on streamlined architectures, it allows the direct construction of very small and low-latency models via two hyperparameters. The work of deep separable convolution is decomposing the standard convolution into depthwise convolution and point convolution. The advantage is the no. of parameters and the amount of computation can be greatly reduced. Furthermore, the comparison between MobileNet-v2 using the depthwise convolution shows a loss of 1% in terms of accuracy, but an order of magnitude reduction on computation and no. of parameters.

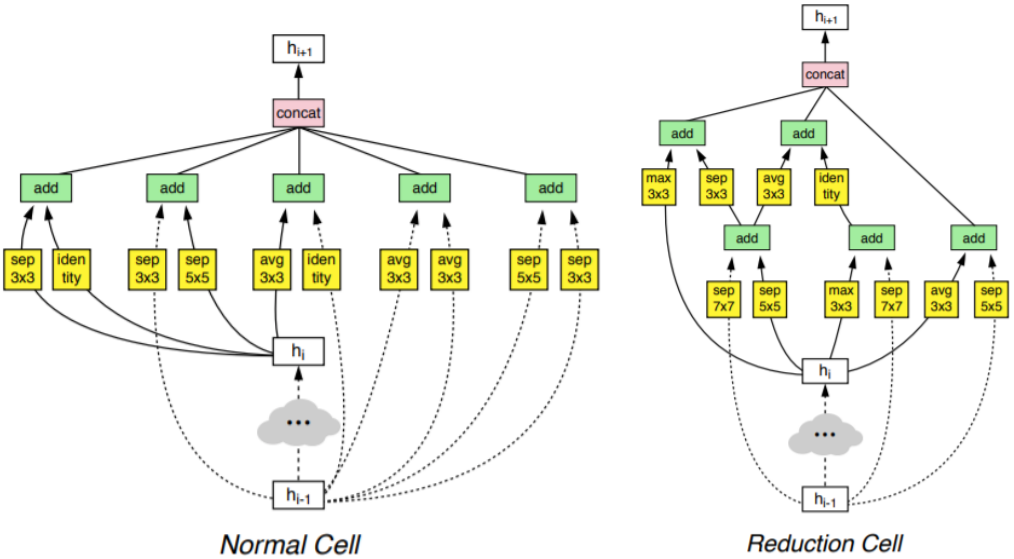
✓ ***DenseNet***

The paper of DenseNet was published in 2017 and won the CVPR 2017 Best Paper Award. This is a new convolutional network architecture that introduces a direct connection between any two layers with the same feature map size. The advantages of such dense connections are alleviating problems of vanishing gradients, enhancing feature propagation, encouraging feature reuse and greatly reducing the no. of parameters. In addition, DenseNet can be extended to hundreds of layers without optimization difficulties. As DenseNet's depth and kernel size increase, large parameters can still increase accuracy of models without overfitting.

This is a deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling. As for more info about DenseNet, please access to this link. [URL: https://arxiv.org/pdf/1608.06993.pdf]



✓ ***NASNet***



Google has introduced the NASNet architecture in 2018 for large-scale image classification and recognition, which features two AutoML designed layers, the Normal Layer and the Reduction Layer. So that experts are no longer required to perform this task. Human knowledge is used to build the convolutional network architecture, and the Hyperparameter is computed directly using RNN, which enables AI to learn automatically. Attached the original paper link for reference. [URL: https://arxiv.org/pdf/1707.07012.pdf]

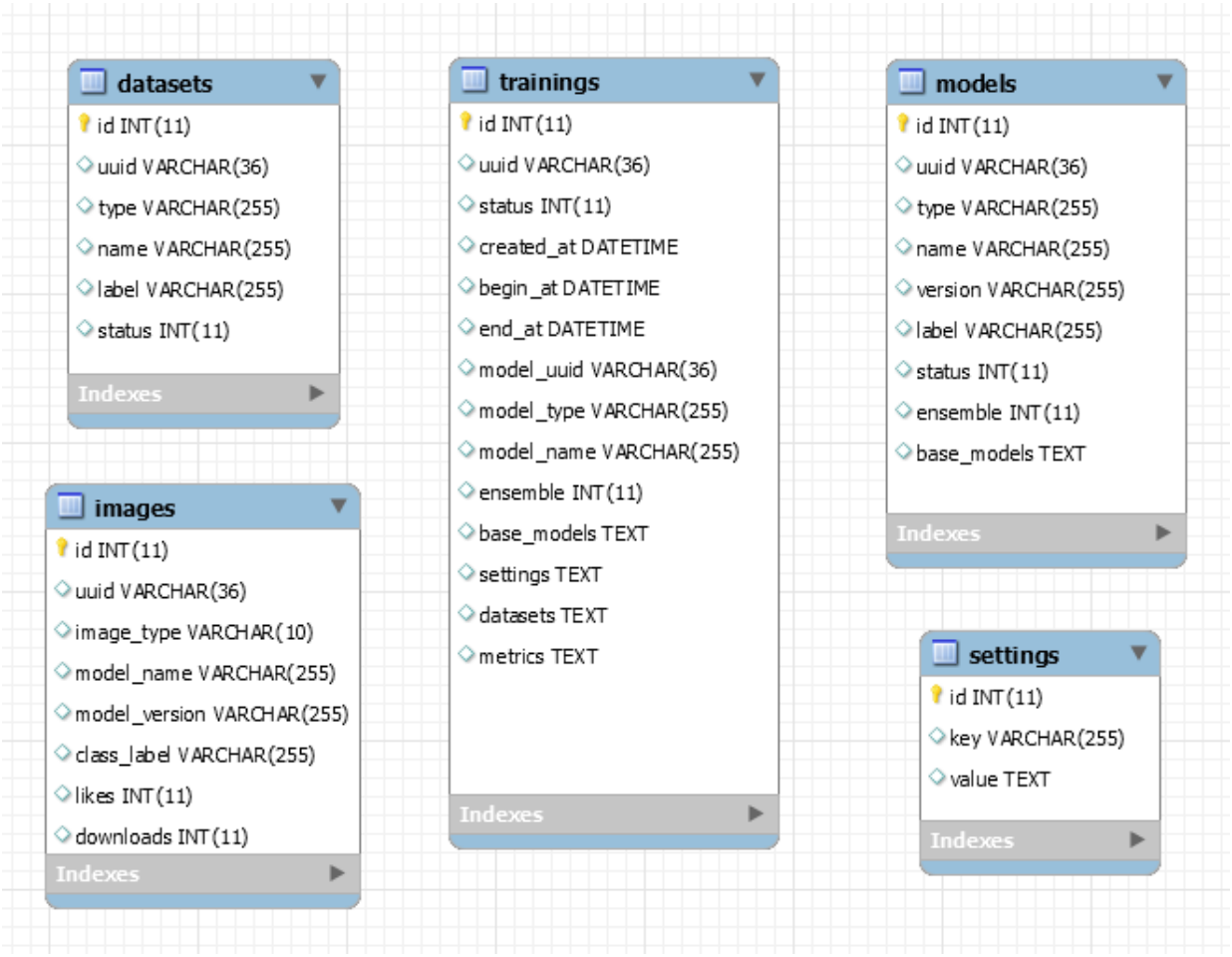
5.2 Knowledge Representation/ Modelling

5.2.1 Model Files

These model files include all network structures and weight files (e.g. .h5, .npz) for Image Generator and Image Detector.

5.2.2 Metadata Database

The following spreadsheet info used for our system are save in metadata database.



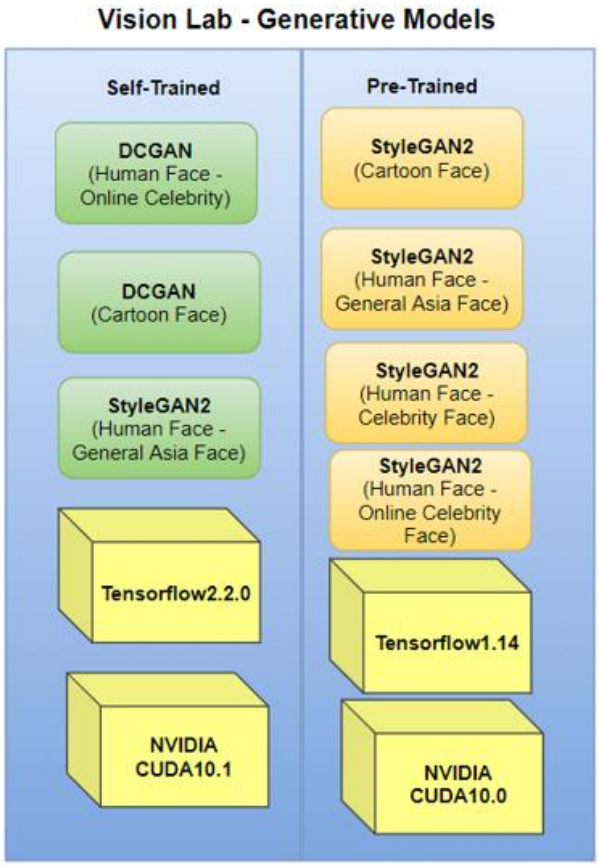
5.3 Knowledge Reasoning/ Inference

5.3.1 Virtual Facial Image Generator

We integrated two our own trained models (DCGAN & StyleGAN2) and one pre-trained model (StyleGAN2) for facial image generator, which is able to generate 4 types of face in total.

Model	Features	Remarks
DCGAN	· Generate fake human faces for social-media influencers. · Generate cute cartoon faces.	This model was trained by ourselves.
StyleGAN2	· Generate fake Asian faces.	This model was trained by ourselves.
StyleGAN2	· Generate fake human faces for Asian, celebrity, and social-media influencers; · Generate cute cartoon faces.	This is a pre-trained model.

1) DCGAN



As for DCGAN model, we have trained two networks, Generator and Discriminator according to the architecture in this slide, basically the image will be generated from a random noise vector by the Generator network, and the Discriminator will try to compare the generated fake images with real images and predict the scores of likeness between two images, the purpose of this comparison is to try make the generated fake image be like real image as much as possible, and the Generator network will adjust the weights in order to perform as good as possible in the fake image generation.

2) StyleGAN2 (trained by ourselves)

We trained a simplified version of StyleGAN2 model by ourselves for virtual Asian face generation. We have spent much time to train the model up to several million steps, but due to the resource and time constraints of our local laptop, the batch size can only be set to a very small value, and training progress is very slow. Even if the final result cannot be compared with the official pre-trained StyleGAN2 models which used very high-end GPU and took a few months to train, that's the best result we can achieve so far.

3) StyleGAN2 (pre-trained by others)

We incorporated the StyleGAN2 network framework created by NividiaLab, instead of using the original network models and weights from the StyleGAN2 official site. We used other pre-trained model to generate facial image of Asia styles, including general Asia face, Celebrity face, Online Celebrity face and Cartoon face. There are 3 models in StyleGAN2 network models, Generator for image generation, Discriminator for scoring the fake image against real image, and Mixture Model for style mixing.

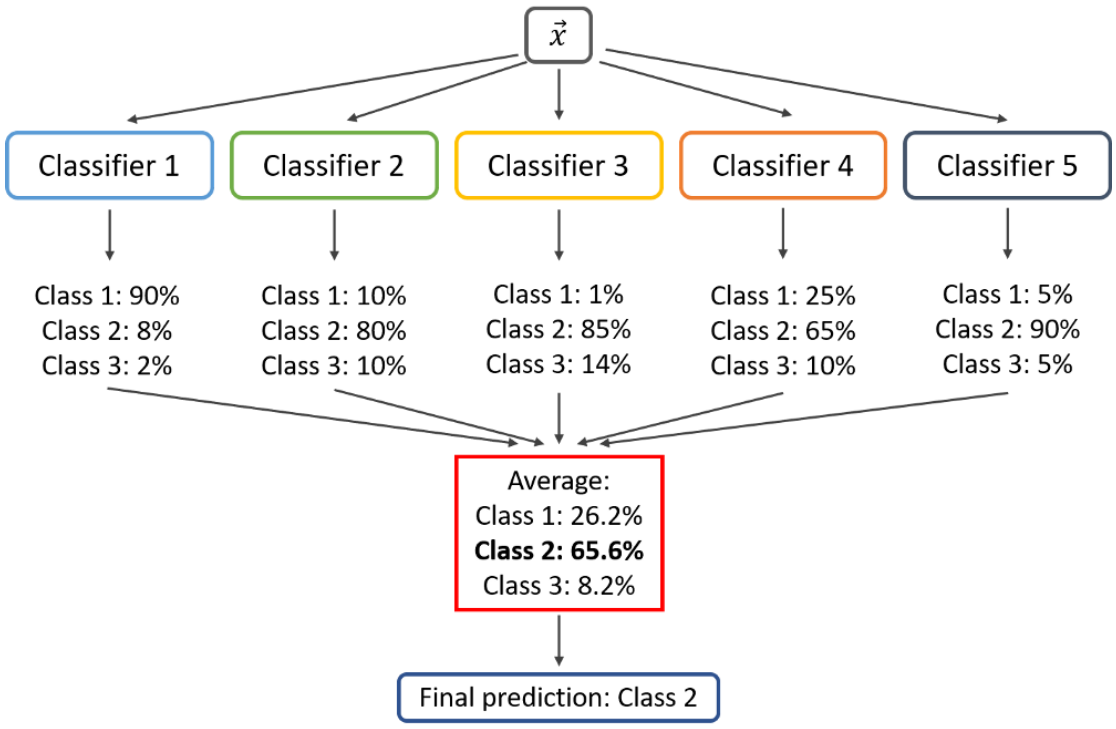
5.3.2 Real/ Virtual Facial Photo Detector

With various Deep CNN architectures that we studied and experimented in knowledge acquisition (learning) sub-system, we would have a collection of models or ensembles that are

capable of classifying images into one of 4 classes:

- Real human facial image
- Virtual human facial image generated by AI algorithm
- Cartoon faces drawn by artists
- Cartoon faces generated by AI algorithm

The facial image detector can be built with a hybrid models architecture, voting classifier, which consults all these pre-trained models. The voting classifier simply calculates average class probability based on predicted class probabilities inferred from pre-trained models. Such calculation does not require any training, so the voting classifier function can be implemented as one REST API, or embedded into front-end scripts.



5.4 Usability Design

An important aspect of our system design is to make it usable to ordinary users who do not have technical background. Also, for system developer and administrator, we would like it be easy to maintain and upgrade. To achieve these goals, we add following usability considerations into the design.

1) User Feedback

   No model is perfect. It is very important for us to know how users think of the trained models, and such feedback data can be used to guide future model training.

   For image generator, the system will record down total number of times when user likes or downloads an image produced by the generator model. Such statistic information is viewable on the web page, so that everyone knows the popularity of each model.

   For image detector, when the web page presents prediction result to a user, it also allows the user to express his opinion by agreeing or disagreeing with the result.

2) Automated Model Training

When the system administrator needs frequently upgrade various models loaded in the system, it could be tedious to keep tracking versions of each model. Also, when training a model offline and uploading to the system, training metadata is lost, e.g. training history and testing accuracy.

To simplify regular model training on updated image datasets, our system also supports scheduling training from web UI. These training requests are queued and processed automatically in the back-end, and resulted models and training metadata are kept for later use.

6.0 Project Implementation (To detail system development & testing approach)

6.1 Agile Development & Continuous Delivery

We adopted Agile software development approach in project implementation. With such approach, application development can start early while we were still brain-storming new use-cases and baking solution designs. Application is always in ready-to-deploy state, with incremental functionality updates being gradually merged into it. As a result, testing and experiment can be carried out against the real-time application, so that defects and improvements can be identified at early stage.

To facilitate the Agile development and support continuous delivery, we have followed below practices:

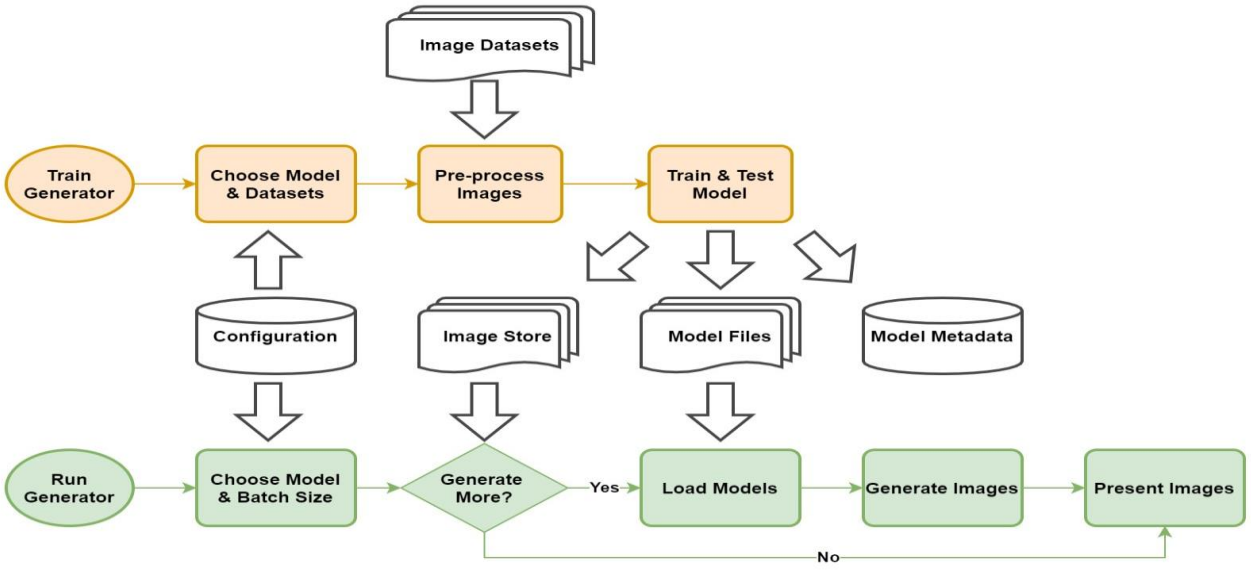
- Conduct weekly team meetings via Microsoft Teams to discuss solution designs, review development and model training progress and feedback testing results.
- Divide application into Front-end Service and multiple Back-end Services, enabling each service to be developed and tested independently.
- For local services hosted in Docker, regularly publish the Docker images to Docker Hub registry to allow all team members access and evaluate the latest version of application.
- For model trainings, which are resource and time consuming, we have shared Jupyter Notebooks so that team members can train multiple models in parallel on their PCs and on cloud platform like Colab.

As mentioned, the application is divided into multiple services and model trainings are run in parallel, following chapters will discuss implementation details for each of them.

6.2 Generator Model Training

6.2.1 GAN Models

For DCGAN and StyleGAN2 generative models, below flowchart shows the workflow for the process of training and generation.



1) *DCGAN*

For training of DCGAN generative model, we support offline training only as of now, and we follow steps below to train the model via Jupyter notebook.

a. Check OS Path to make sure the training program is under the correct path of the model source code.
b. Import the necessary libraries.
c. Define initial hyper parameters, such as epoch size, learning rate, momentum, batch size, input and output size, folders or directories required by the training program etc.
d. Create the folders or directories required by the training programme.
e. Define the function for data pre-processing, this function will load the real human or cartoon face images from the data directory, and do some data augmentation such as randomly flip left or right, and scale up the image, and return the pre-processed data via batches to the training method.
f. Define generator network model according to the network structure introduced in project design section above.
g. Define discriminator network model according to the network structure introduced in project design section above.
h. Load the previously trained weights file in order to continue the training from last time without losing the weights.
i. Define the epoch history saving method and model training method, those methods will train the models with pre-processed data and save the epoch history and weights into checkpoint folder regularly, so that the learning features will not be lost when the program

stopped. Gradient decent will also be performed in order to improve the generator and discriminator model's performance.

j. Start to train the model by passing in the input datasets, and print the performance of the generator and discriminator model for each step.
k. Load the test model and trained weights and perform the testing.
l. Display individual test image.
m. Save the generated images to output folder.
n. Below are our testing results so far for both human face generation and cartoon face generation.
    - Virtual human faces for Social-media influencers

    - Cartoon faces

2) *StyleGAN2 (trained by ourselves)*

Although we have trained the StyleGAN2 model till 2,120,000 steps for more than two weeks, however, the current performance is not very good. In our system, this model will be as an experiment for users to have a trial.

```
In [12]: gan.step
Out[12]: 2120102
```

- Virtual Asian faces

3) *StyleGAN2 (pre-trained by others)*

Because this pre-trained StyleGAN2 model can generate very diverse and realistic virtual faces, we aligned to integrate it into our system for users' selection.

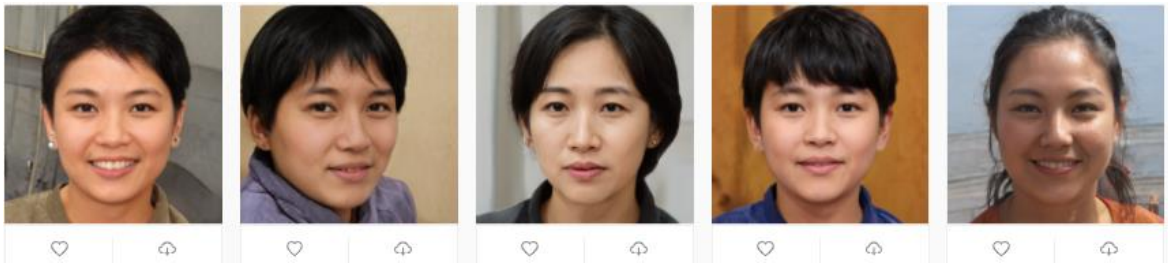
But there are some limitations of this model.

- Although both Linux and Windows are supported, but strongly recommend Linux for performance and compatibility reasons.

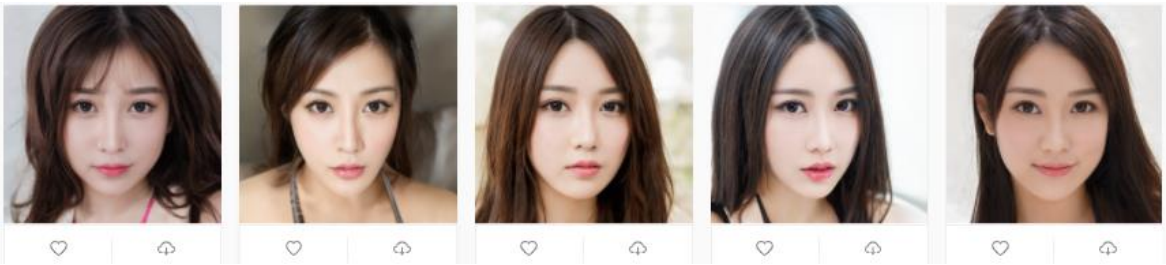
- For deployment on Windows, the Desktop development with C++ via Visual Studio Community 2017 must be installed.
- Only support the version 1.14.0 of Tensorflow.

These photos are the example result after our integration of this model.

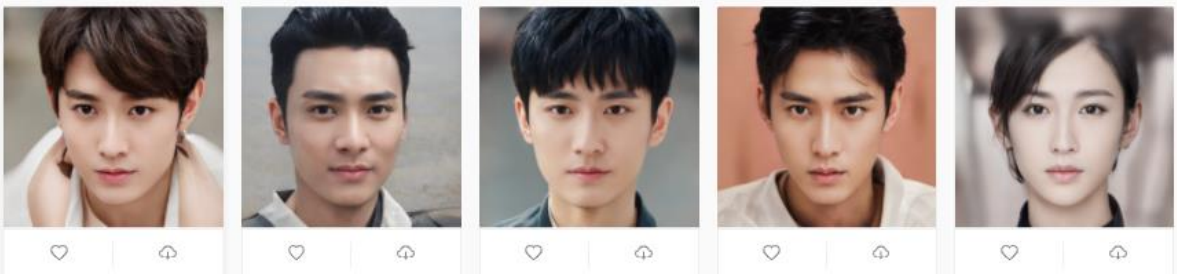
- Virtual Asian faces



- Virtual human faces for Social-media influencers



- Virtual human faces for Celebrity



- Cartoon faces

6.2.2 What we have done for GAN Models

1) We have explored several GAN models for facial image generation, experiment different GAN model's training and generation in our local PC, such as UGATIT, DCGAN, StyleGAN2, SAGAN, etc, and determine the feasibility of using it in our project, and scope out those models which require higher computing resource or training times than we can afford.

2) We have enhanced the models to generate bigger size of output image and tune the parameters to improve the quality of generated pictures. And reduced the batch size in order to avoid out of memory issue in our local PCs when training for self-trained models.

3) Experiment different versions of TensorFlow framework and NVIDIA GPU driver and toolkits to accommodate multiple models, and also try to align the version as much as we can, so finally we used 2 versions of Tensorflow to support various models, Tensorflow 1.14 (for StyleGAN2) and Tensorflow 2.2.0 for all the other models.

4) Train the DCGAN models days and nights with different datasets (both cartoon and real human face) to improve the generation quality, in both local and cloud, and fine tune the parameters, and create Jupyter notebook version of those models for step-by-step investigation and offline training.

6.3 Detector Model Training

Our detector (classifier) models are trained in two stages. In the first stage, we train base-models using various well-established Deep CNN architectures (mentioned in 'Project Solution'). We measure and record down each base-model's metrics, e.g. accuracy and confusion matrix.

Then, in the second stage, we compare and select the best performant base-models (one for each architecture) and form ensembles to be trained as final models.

In all the trainings, we have used some common setup in pipelines for datasets and performance measurements, so that the whole process can be easily repeated for different models and can be fully automated in Back-end Services. These common setups are listed below.

1) For the dataset, we use 5000 images for each of the 4 detector classes, i.e. *human* (real human face), *human_gen* (AI generated human face), *anime* (cartoon face drawn by artists), *anime_gen* (AI generated cartoon face). Originally, we tried to use 10000 images for each class, but later testing suggested that 5000 images per class are good enough to achieve comparable result while training time can be significantly reduced.


iv.human


iv.human_gen


iv.anime


iv.anime_gen

2) For pre-processing, all images are resized to 224x224. If the actual model architecture requires different sizing, additional resizing layer is added to the pipeline.
3) All trainings use batch size of 32.

4) Split 20% of dataset for validation, and models are saved only for epochs with best validation accuracy.
5) Use categorical cross-entropy as loss function, and RMSprop as optimizer.

6.3.1 Base Models

1) *VGG*

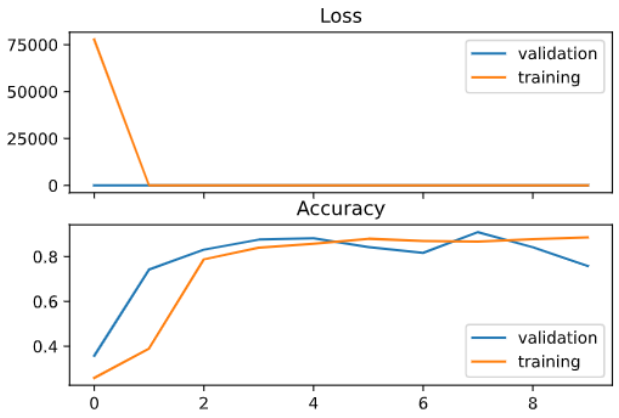
Both VGG16 and VGG19 have more than 100 million of trainable parameters, which makes them consume a lot of memory during training. We often encounter 'resource exhausted' (out of memory) errors during training, and eventually have to reduce batch size in order to make training proceed. Due to the large number of trainable parameters, the saved models are about 1GB in size, which also makes them inefficient to distribute and deploy.

The validation accuracy quickly saturates at around 90%, which is lower than other model architectures.

Accuracy: 90.93%

	precision	recall	f1-score	support
human	0.7609	0.9469	0.8437	1035
human_gen	0.9291	0.7006	0.7988	972
anime	0.9980	0.9840	0.9909	1001
anime_gen	0.9980	0.9980	0.9980	1006
accuracy			0.9093	4014
macro avg	0.9215	0.9074	0.9079	4014
weighted avg	0.9202	0.9093	0.9082	4014

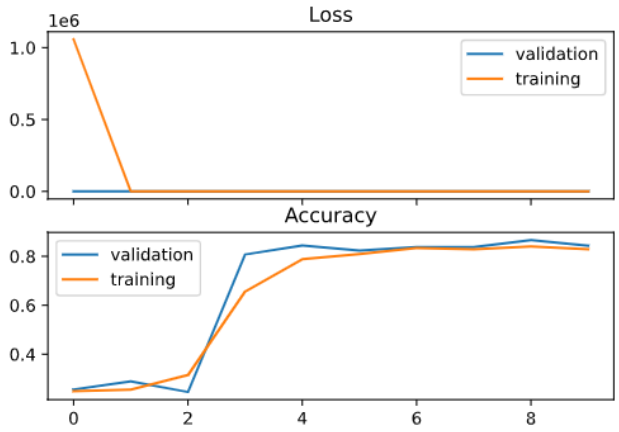
Base Model 1 - VGG16



Accuracy: 86.65%

	precision	recall	f1-score	support
human	0.8614	0.5700	0.6861	1014
human_gen	0.6826	0.9101	0.7801	990
anime	0.9779	0.9942	0.9860	1026
anime_gen	0.9990	0.9949	0.9969	984
accuracy			0.8665	4014
macro avg	0.8802	0.8673	0.8623	4014
weighted avg	0.8808	0.8665	0.8621	4014

Base Model 2 - VGG19



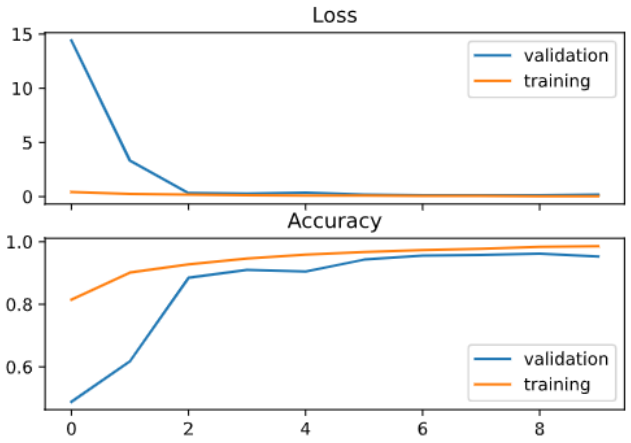
2) *ResNet v2*

Base models using ResNet architectures, i.e. ResNet50, ResNet101 and ResNet152, are relatively easier to train, and validation accuracy goes beyond 95% within a few epochs. And, for our case, ResNet50 is good enough, while deeper network in ResNet101 or ResNet152 does not produce better results.

Accuracy: 96.24%

	precision	recall	f1-score	support
human	0.9133	0.9383	0.9256	988
human_gen	0.9431	0.9241	0.9335	1041
anime	0.9969	0.9898	0.9934	984
anime_gen	0.9980	0.9990	0.9985	1001
accuracy			0.9624	4014
macro avg	0.9628	0.9628	0.9628	4014
weighted avg	0.9627	0.9624	0.9625	4014

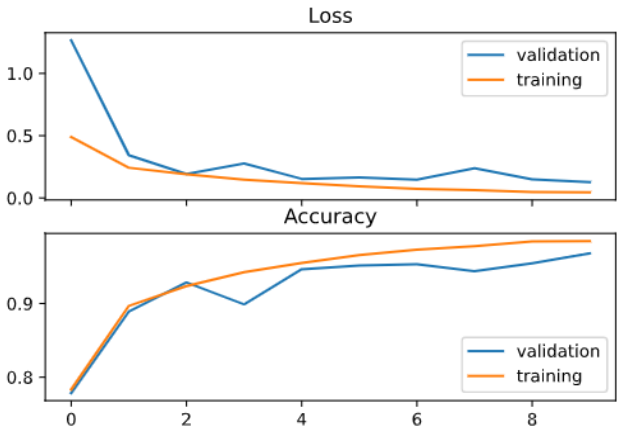
Base Model 3 - ResNet50 v2



Accuracy: 96.81%

	precision	recall	f1-score	support
human	0.9312	0.9456	0.9383	1030
human_gen	0.9461	0.9315	0.9387	1036
anime	0.9990	0.9990	0.9990	965
anime_gen	1.0000	1.0000	1.0000	983
accuracy			0.9681	4014
macro avg	0.9691	0.9690	0.9690	4014
weighted avg	0.9682	0.9681	0.9681	4014

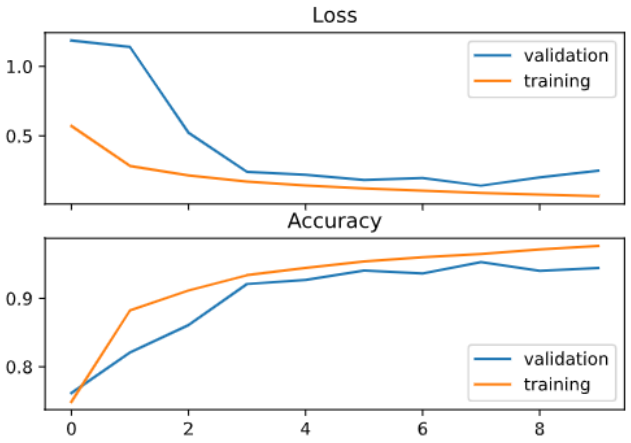
Base Model 4 - ResNet101 v2



Accuracy: 95.29%

	precision	recall	f1-score	support
human	0.9182	0.8967	0.9073	1026
human_gen	0.9032	0.9231	0.9130	1001
anime	0.9917	0.9969	0.9943	961
anime_gen	1.0000	0.9971	0.9985	1026
accuracy			0.9529	4014
macro avg	0.9533	0.9534	0.9533	4014
weighted avg	0.9530	0.9529	0.9529	4014

Base Model 5 - ResNet152 v2



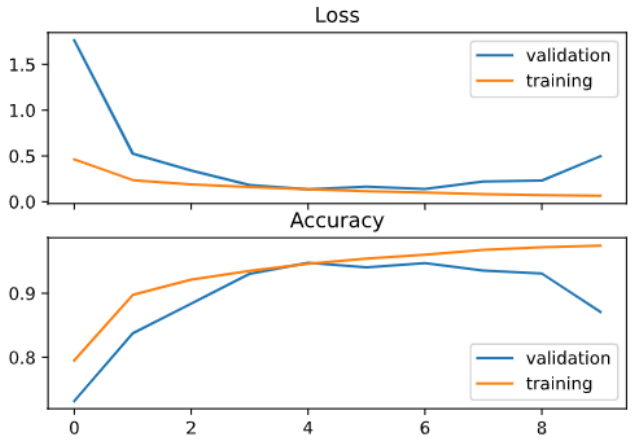
3) *DenseNet*

Base models with DenseNet architectures can achieve validation accuracy which is very close to ResNet. And it has less trainable parameters than ResNet, which is also the major reason that we select DenseNet121 as one of the base models to form final ensemble model. The resulting ensemble model has smaller size and is easier to deploy.

Accuracy: 94.77%

	precision	recall	f1-score	support
human	0.8813	0.9083	0.8946	981
human_gen	0.9078	0.8887	0.8981	997
anime	1.0000	0.9912	0.9956	1020
anime_gen	1.0000	1.0000	1.0000	1016
accuracy			0.9477	4014
macro avg	0.9473	0.9470	0.9471	4014
weighted avg	0.9481	0.9477	0.9478	4014

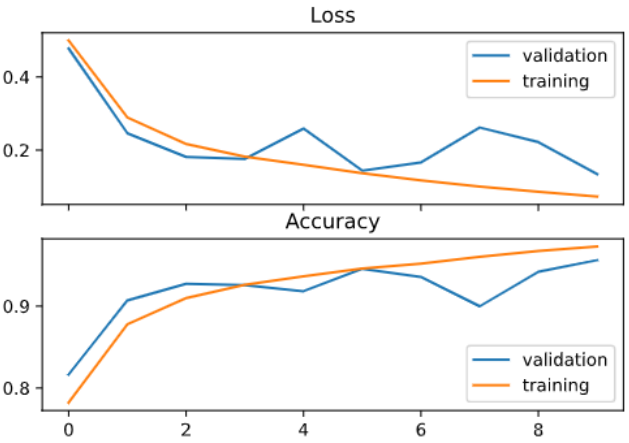
Base Model 6 - DenseNet121



Accuracy: 95.62%

	precision	recall	f1-score	support
human	0.8965	0.9376	0.9165	1025
human_gen	0.9366	0.8947	0.9152	1007
anime	0.9962	0.9952	0.9957	1042
anime_gen	1.0000	0.9989	0.9995	940
accuracy			0.9562	4014
macro avg	0.9573	0.9566	0.9567	4014
weighted avg	0.9567	0.9562	0.9562	4014

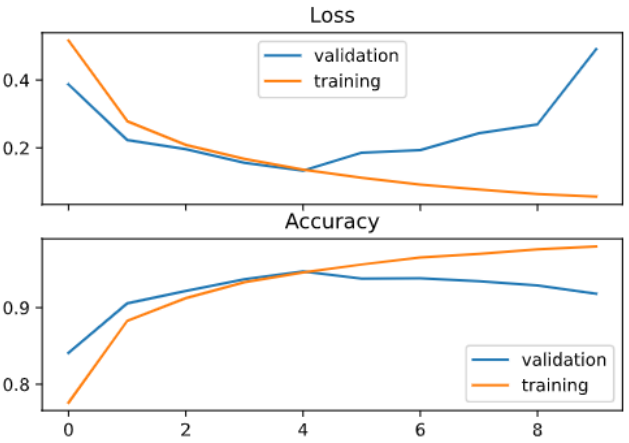
Base Model 7 - DenseNet169



Accuracy: 94.74%

	precision	recall	f1-score	support
human	0.8569	0.9463	0.8994	987
human_gen	0.9436	0.8496	0.8941	1024
anime	0.9969	0.9990	0.9980	978
anime_gen	1.0000	0.9971	0.9985	1025
accuracy			0.9474	4014
macro avg	0.9494	0.9480	0.9475	4014
weighted avg	0.9497	0.9474	0.9474	4014

Base Model 8 - DenseNet201



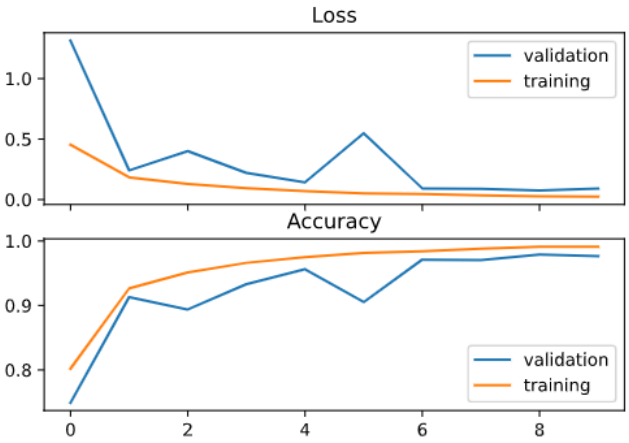
4) *Inception ResNet v2*

This base model is the second most accurate among all the base models that we train.

Accuracy: 97.88%

	precision	recall	f1-score	support
human	0.9487	0.9708	0.9596	1028
human_gen	0.9701	0.9458	0.9578	960
anime	0.9971	0.9981	0.9976	1042
anime_gen	1.0000	0.9990	0.9995	984
accuracy			0.9788	4014
macro avg	0.9790	0.9784	0.9786	4014
weighted avg	0.9790	0.9788	0.9788	4014

Base Model 9 - Inception ResNet v2



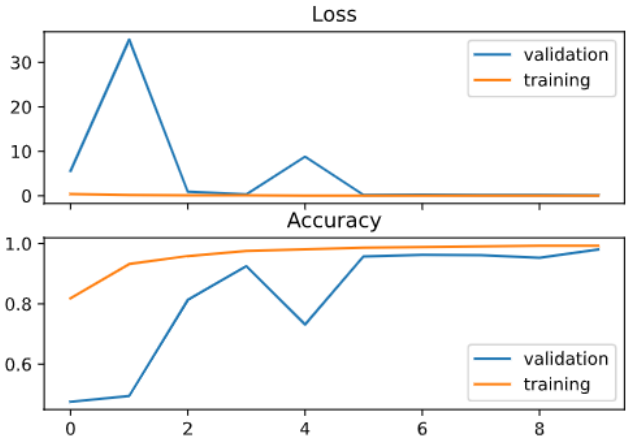
5) *Xception*

This base model achieves the highest accuracy among all the base models that we train.

Accuracy: 98.06%

	precision	recall	f1-score	support
human	0.9434	0.9820	0.9623	1002
human_gen	0.9815	0.9464	0.9636	1008
anime	0.9990	0.9949	0.9969	977
anime_gen	1.0000	0.9990	0.9995	1027
accuracy			0.9806	4014
macro avg	0.9810	0.9806	0.9806	4014
weighted avg	0.9810	0.9806	0.9806	4014

Base Model 10 - Xception



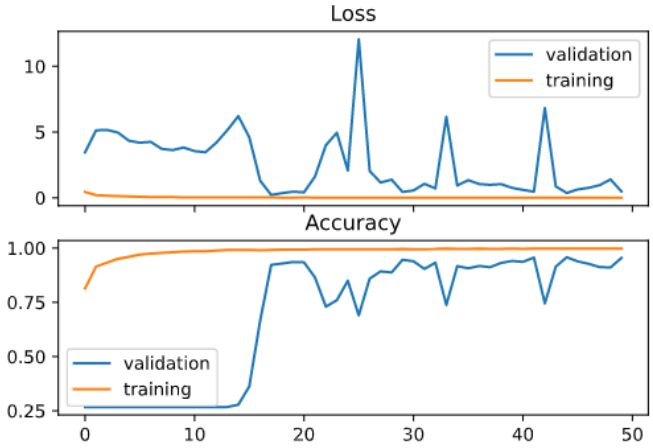
6) *MobileNet v2*

Like the name suggested, MobileNet base model is compact in size and is more suitable to be deployed to system with limited storage. However, as it has less trainable parameters, the training also progress slower than other base models. Unlike other base models which take less than 10 epochs to achieve accuracy higher than 90%, we have to train MobileNet models for more epochs to get comparable result.

Accuracy: 95.76%

	precision	recall	f1-score	support
human	0.9064	0.9469	0.9262	1074
human_gen	0.9449	0.8974	0.9206	975
anime	0.9990	0.9868	0.9929	986
anime_gen	0.9869	1.0000	0.9934	979
accuracy			0.9576	4014
macro avg	0.9593	0.9578	0.9583	4014
weighted avg	0.9581	0.9576	0.9576	4014

Base Model 11- MobileNet v2



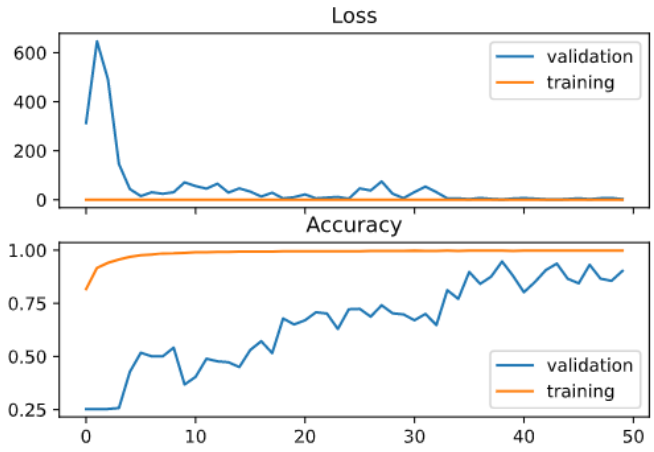
7) *NASNet Mobile*

Just like MobileNet, we also tried mobile version NASNet to get a compact base model.

Accuracy: 94.67%

	precision	recall	f1-score	support
human	0.8753	0.9380	0.9056	1033
human_gen	0.9399	0.8728	0.9051	1022
anime	0.9990	0.9823	0.9906	1016
anime_gen	0.9812	0.9979	0.9895	943
accuracy			0.9467	4014
macro avg	0.9489	0.9478	0.9477	4014
weighted avg	0.9480	0.9467	0.9467	4014

Base Model 12 - NASNet Mobile



6.3.2 Ensemble Models

Base model training shows that, although some of them may take longer time to train, most of them can achieve accuracy around 95% using the given dataset. As a result, when we choose base models to form final ensemble, our focus is more on the resulting model size.

In order to have a small ensemble model (to be pre-loaded into Back-end Service Docker image), eventually we choose DenseNet121, MobileNet and NASNet Mobile.

We used a simple stacking ensemble architecture, which concatenates the outputs of 3 base models and adds additional dense layers to train and learn the best combinational weights of the 3 base models.

```
Model: "functional_1"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 224, 224, 3) 0
_________________________________________________________________________________________________
bm0_functional_7 (Functional)  (None, 4)               7041604     input_1[0][0]
_________________________________________________________________________________________________
bm1_functional_1 (Functional)  (None, 4)               2263108     input_1[0][0]
_________________________________________________________________________________________________
bm2_functional_3 (Functional)  (None, 4)               4273944     input_1[0][0]
_________________________________________________________________________________________________
concatenate (Concatenate)    (None, 12)                0         bm0_functional_7[0][0]
                                                                 bm1_functional_1[0][0]
                                                                 bm2_functional_3[0][0]
_________________________________________________________________________________________________
dense (Dense)                (None, 12)                156       concatenate[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 4)                 52        dense[0][0]
=================================================================================================
Total params: 13,578,864
Trainable params: 208
Non-trainable params: 13,578,656
```

Because we fixed weights in base models, only weights in additional dense layers need be learned. The training is quite fast, and we get our final model having an accuracy of 99.15%.

6.4 Back-end Service

Base Docker image - https://hub.docker.com/r/tensorflow/tensorflow
Project Docker image - irs3y/vlab-backend-core

(URL: https://hub.docker.com/repository/docker/irs3y/vlab-backend-core)
The Back-end Service is a Flask application and it provides the REST APIs for all the inference and administrative functions:

- Use hybrid generator models to produce images
- Support file uploading (images, datasets and model files)
- Use hybrid detector models to classify uploaded images
- Support user feedback functions, e.g. image labelling
- Support query system status and manage system settings
- Manage online model trainings

6.4.1 REST APIs

1) Query Application Status
   Endpoint: HTTP GET /backend/status
   This API allows front-end to check back-end conditions, e.g. current system time and supported TensorFlow versions.

2) Query Application Settings
   Endpoint: HTTP GET /backend/settings
   This API allows front-end to get system-wide settings, e.g. image generator batch size.

3) Save Application Settings
   Endpoint: HTTP POST /backend/settings
   This API is for modifying system-wide settings.

4) View Image
   Endpoint: HTTP GET /backend/image/<image_path>
   Images generated by generator or uploaded by end users are stored in back-end file system. For image to be displayed in front-end browser, this API will serve the image.

5) Upload Image
   Endpoint: HTTP POST /backend/image
   This API allows user to upload images which be used by subsequent API calls to perform image classification.

6) Classify Image (Trigger Detector Models)
   Endpoint: HTTP POST /backend/image/classify
   This API triggers detector models loaded in system and performs prediction on previously uploaded image.

7) Generate Image (Trigger Generator Models)
   Endpoint: HTTP POST /backend/image/generate
   This API triggers generator models loaded in system, creates images and stores them on the back-end file system. Subsequently, View Image API will be triggered from front-end to present generated images to user.

8) Label Image
   Endpoint: HTTP POST /backend/image/label

This API allows user to provide explicit labelling on images that they upload. So, when they disagree with the predication result from detector models, they can provide feedbacks to assist future model trainings.

9) Query Image Stats

Endpoint: HTTP GET /backend/image/stats

This API allows user to query image stats info, e.g. number of likes and downloads, grouped by generator models that produce the image.

10) Update Image Stats

Endpoint: HTTP POST /backend/image/stats

This API triggers when user likes or downloads generated images. Then image stats stored in metadata database shall be updated accordingly.

11)Upload Large Files (Models and Datasets)

Endpoint: HTTP POST /backend/file/upload

Model files and dataset files are quite large in nature (hundreds of MBs to a few GBs). This API allows uploading such files in small chunks.

12)Query Models

Endpoint: HTTP GET /backend/model/<model_type>

This API allows query model metadata, e.g. name, version, for either generator or detector. Then user can select models based on returned model list.

13)Add Models

Endpoint: HTTP POST /backend/model

This API adds uploaded model files (trained offline) to model repository. Then this model can used just like any pre-loaded models.

14)Update Model Status

Endpoint: HTTP PATCH /backend/model/<model_uuid>/status/<status>

This API changes model status, i.e. enabling or disabling a model.

15)Delete Models

Endpoint: HTTP DELETE /backend/model/<model_uuid>

This API removes a model from model repository.

16)Query Datasest

Endpoint: HTTP GET /backend/dataset/<model_type>

This API allows query datasets available (previously uploaded to system) for training the specified model type.

17)Add Datasets

Endpoint: HTTP POST /backend/dataset

This API adds uploaded dataset files to dataset repository. Then this dataset can be used for future model trainings.

18)Update Dataset Status

Endpoint: HTTP PATCH /backend/dataset/<dataset_uuid>/status/<status>

This API changes dataset status, i.e. enabling or disabling a dataset.

19)Delete Datasets

Endpoint: HTTP DELETE /backend/dataset/<dataset_uuid>

This API removes a dataset from dataset repository.

20)Query Trainings

Endpoint: HTTP GET /backend/training/<model_type>

This API allows query information on all the training related to the specified model type, which could be still pending inside training queue, currently running in progress or has already completed.

21)Add Trainings

Endpoint: HTTP POST /backend/training

This API schedules new training with provided training configuration. The scheduled training is pushed to back-end training queue, and will be picked up and processed by dedicated training thread.

22)Update Training Status

Endpoint: HTTP PATCH /backend/training/<training_uuid>/status/<status>

This API changes training status, e.g. pause or resume a pending training in the queue.

23)Delete Trainings

Endpoint: HTTP DELETE /backend/training/<training_uuid>

This API removes a training from training queue.

24) Process Trainings

Endpoint: HTTP POST /backend/training/process

This API signals dedicated training thread to start processing training requests in queue.

6.4.2 StyleGAN Variant

- Project Docker image - irs3y/vlab-backend-stylegan
- (URL: https://hub.docker.com/repository/docker/irs3y/vlab-backend-stylegan)

This variant of Back-end Service has the same set of APIs as core Back-end Service except that it installs additional packages and uses TensorFlow version 1.14 as base image, which is required for using StyleGAN image generator models.

6.4.3 DCGAN Variant

- Project Docker image - irs3y/vlab-backend-dcgan
- (URL: https://hub.docker.com/repository/docker/irs3y/vlab-backend-dcgan)

This variant of Back-end Service has the same set of APIs as core Back-end Service except that it installs additional packages and uses TensorFlow version 2.2 as base image, which is required for using DCGAN image generator models.

6.5 Front-end Service

- Base Docker image - nginx:1.15.12 (URL: https://hub.docker.com/_/nginx)
- Project Docker image - irs3y/vlab-frontend (URL: https://hub.docker.com/r/irs3y/vlab-frontend)

The Front-end Service is a Single-page Application (SPA) built with React and Material UI framework. The major components of UI are image generator page, image detector page, system settings page and model trainings page. These components will trigger Back-end Service APIs to perform various tasks based on user interaction with the web UI. For detailed instructions on using web UI and screen captures for each web pages, please refer to our User Guide document.

7 Project Performance & Validation

We designed following test cases to ensure the web system can work well, and all tests performed in the configuration below.

Configuration:
SW: Win10 64bit
Browser: Google Chrome 81.0.4044.113
Docker: Docker Desktop 19.03.8, Docker-Compose 1.25.4
Reference Documentation: Visionlab_User Guide

Test Flow	Step Description	Expected Result	Actual Result
Docker & Docker-compose Installation	Download Docker Desktop/ Toolbox and install it on laptop.	It can be installed successfully.	Pass
Environment setup for StyleGAN model deployment	1) Install Desktop development with C++ via Visual Studio Community 2017. 2) Install latest NVIDIA drivers, CUDA 10.0 toolkit and cuDNN 7.6.5. 3) Create an environment on Annaconda3 to install Python 3.6 (or above version), Tensorflow-gpu 1.14.0 and other necessary packages.	The environment can be created as referred to user guide.	Pass
Trigger Web System	1) Execute command to pull Docker images and trigger web system. 2) Open a supported browser, input the IP: http://localhost *Note*, if you installed Docker Toolbox, please check its IP with command "*docker-machine ip*" on a terminal.	The system page can show up without issues, and there are two Tensorflow backend versions (2.2.0 & 1.14.0) are running.	Pass
System Settings for Image Generator	1) Move blue circle on "Number of images generated per batch" bar. 2) Enable/ disable any displaying models, if a model is disabled, it will disappear on Image Generator page. 3) Click "Upload Model" button, and try to upload a demo model.	1) No. can be changed from 0 to 20 smoothly. 2) Models can be enabled/ disabled successfully. 3) The demo model is working.	Pass
System Settings for Image Detector	1) Enable/ disable any displaying models, if a model is disabled, it will disappear on Image Detector page. 2) Click "Upload Model" button, and try to upload a demo model.	1) Models can be enabled/ disabled successfully. 2) The demo model is working.	Pass
Image Generator	1) Click "Image Generator" icon on Home page. 2) Enable/ disable any displaying models, then click "+" button to generate images. 3) Click "+" button again with same enabled models.	1) Image Generator page can pop up quickly. 2) Only enabled single/ mix models can generate images, the default no of generated images per batch is 10.	Pass

	4) Click preference " ♡ " and download " ☁ " button at the bottom of generated photos.	3) Another 10 images can be generated. 4) Selected pictures have been marked and downloaded. These two categories for models have been accumulated.	
Image Detector	1) Click "Image Detector" icon on Home page. 2) Click "Drop image here" and upload a human face and cartoon face whether they are real or AI-generated. 3) Click "Agree" or "Disagree" button according to response from this robot and the authenticity of that image.	1) Image Detector page can pop up quickly. 2) There is a robot will respond whether that uploaded picture is real or AI-generated, and ask users' opinion on the response. 3) When click "Agree" button, that image will be added to predicted dataset, whereas, a feedback page will pop up to list all available dataset types for users to select.	Pass
Model Trainings	1) Click Model Trainings small icon on top left corner of Home page. 2) Click "Upload Dataset", fill up Name & Display Label info, and drop one dataset to upload. 3) Click "Add Training", choose dataset, include/exclude user uploaded images/ generator produced images, choose model, set batch size and maximum epochs. 4) Go to System Settings page, click Image Detector button and enable the new trained model.	1) Model Trainings page can pop up quickly. 2) All files of the dataset can be uploaded without issues. 3) All options and settings on Add Training page can be done without issues. 4) That model can be enabled successfully.	Pass
Project Site	Click the top left corner cat icon	The GitHub link where archive all VisionLab documents pops up.	Pass

8 Project Conclusions (Findings & Recommendation)

Out of interest, we aligned to build a toolbox system that supports both face generator and detector. In order to propose a good and unique web system, we did a lot of work on computer vision industry, not only did we research a variety of deep learning papers in recent years, but we also investigated both advantage and disadvantage of some existing model applications. Furthermore, it is quite significant and meaningful to put all studied theories of these three courses we learnt this semester into practice well. Hence, we applied various unsupervised GANs and supervised CNNs algorithms to image generator and detector respectively. And more than 10 different methods have been adopted into model trainings.

Since three of us worked together for practical project in the first semester, so we are working with each other even better on this semester. Although we all have to study at our own home without face-to-face communication, we still learnt a lot from each other as we were keen on sharing respective professions and skillsets during weekly meetings for this project. It is a good opportunity that not only allowed us to consolidate technical theoretical knowledge through practice, but all of us also enjoyed the wonderful process of experimenting and learning new things.

After the system goes live, it will continue to collect valuable data that users uploading. Since those are most significant and useful data for application scenarios.

1) Re-run the training process periodically with both pre-collected dataset photos and user uploaded photos. Compare the results and replace models when there is improvement.
2) Search and identify other possible model definitions (e.g. from newly published papers). Try them out and add suitable models to the collection of saved models.

Appendix A: Project Proposal

Please refer to file "VisionLab_Project Proposal_Group02_14Sept'20.pdf".

Appendix B: Mapped System Functionalities against knowledge of Courses

Knowledge of Courses	Facial Image Generator	Facial Image Detector
Supervised learning	With adoption of DCGAN and StyleGAN2 algorithms, we built 3 different models on Tensorflow framework to produce many types of fake human and cartoon faces. All those AI-generated images also can be the dataset source of detector for model trainings.	Those various algorithms (e.g. VGG16/ 19, ResNet50/ 101/ 152, Xception, Inception, ResNet, MobileNet, DenseNet and NASNet etc.) have been used in model trainings for facial detector to detect the authenticity of real/ virtual human and cartoon faces. In order to improve performance of the detector, we adopted ensemble method to combine several good individual model predictions, then vote a higher one for final prediction.
Unsupervised learning		
Deep learning · CNN · GANs · Tensorflow · ResNet		
Hybrid & Ensemble approaches		

Appendix C: Installation and User Guide

Please refer to file "VisionLab_User Guide_Group2.pdf" in details.

Appendix D: List of Abbreviations

Abbr.	Full Name
AI	Artificial Intelligence
ATDF	Anti-Deepfake
CNN	Convolutional Neural Network
DCGAN	Deep Convolutional GAN
FG	Face Generator
GAN	Generative Adversarial Network
ML	Machine Learning
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
Soft-AdaLIN	Soft Adaptive Layer-Instance Normalization
UI	User Interface
VGG	Visual Geometry Group

Appendix E: References

1. Face Dataset. [http://www.seeprettyface.com/mydataset_page3.html#dataset3]. Accessed on 15 Sept. 2020.
2. Silvia Man. March 6 2020. Using ML to detect fake face images created by AI [https://blog.jayway.com/2020/03/06/using-ml-to-detect-fake-face-images-created-by-ai/]. Accessed on 15 August 2020.
3. zsdonghao. 2019. DCGAN in TensorLayer. [https://github.com/tensorlayer/dcgan]. Accessed on 20 August 2020.
4. Real and fake face detection. [https://www.kaggle.com/ciplab/real-and-fake-face-detection]. Accessed on 18 July 2020.
5. Practical Deep Learning for Coders. [https://course.fast.ai/]. Accessed on 20 July 2020.
6. FaceForensics Benchmark. [http://kaldir.vc.in.tum.de/faceforensics_benchmark/documentation]. Accessed on 2 Oct. 2020.
7. FaceForensics++: Learning to Detect Manipulated Facial Images. [https://github.com/ondyari/FaceForensics]. Accessed on 2 Oct. 2020.
8. Soumith Chintala. Jan 7 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. [https://arxiv.org/pdf/1511.06434.pdf]. Accessed on 15 Oct. 2020.