Sim2Real: Human Social Signal Detection

Hon Wing Eric Chan
PMP Big Data in Computer Science
Simon Fraser University
hwc9@sfu.ca

BenKun Chen
PMP Visual Computing in Computer Science
Simon Fraser University
bca96@sfu.ca

Abstract

Sim2Real refers to techniques that can be used to transfer knowledge from one environment (e.g in simulation) to another (e.g. real world). While Sim2Real has been applied to various fields such as object recognition, human pose estimation etc. it has never been applied to recognizing human social signals. We want to deliver a high accuracy machine learning model to our supervisor Angelica Lim (angelica@sfu.ca) that can accurately recognize the following 3 facial expressions:

(a) Angry

(b) Crying

(c) Happy

1 Introduction

With the increased interest in machine learning in recent years, there has been an explosion of human social signal detection tools like facial unlock or face mask filter across businesses and individuals. With the performance and reliability of these tools are heavily depend on both quantity and quality of the training data, it is a challenge for us to generate a set of synthetic data that can equip us with an optimal performance in the real-world environment after training. In this project, we explore how synthetically generated data can be beneficial for training as well as how the accuracy differ between two neural network models that we have created. At the end of this experiment, two major contributions we have made are 1). Generate a synthetic data set which mimics the images in a real-world environment 2). Create neural network models that can recognize all 3 facial expressions in a noisy environment with a good result. In this report, we cover all the technical details step-by-step for reaching the end result; a high accuracy facial detection model.

2 Data

With the word simulation (Sim), the question comes to why we are relying on generating our data by simulation rather than using faces from the real world. The main reasons why we choose simulation are cost, privacy, and testing. Using simulation is significantly more cost-effective and efficient than collecting real-world data. It is also anonymous as no real person's face will be used and the data cannot be traced back to the original owner, avoiding any possible copyright infringement. Rather than utilizing costly real-world data to test if the model is providing the desired output, we can plug

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

in the simulated data and analyze the results. By creating appropriately simulated data we can train a system on a multitude of scenarios that are not covered in the authentic data (e.g. children's faces), thus improving its diversity capability. While simulating data can be cost-effective and efficient, the challenge still remains to mimic the real-world data. we want to take in consideration of all the real world factors, we want to create a set of quality facial identities and have a full control over its variations such as background, pose and illumination.

2.1 Generate human model

We chose to use **MakeHuman** (1) as our facial identities generator. The application allows us quickly form and adjust facial and limb models of men and women of different ethnicity and ages. By using the "natural postural" system from the application we can easily adjust the emotion intensity level.

(a) Angry 20% intensity

(b) Angry 100% intensity

We have generated our human models based on 3 ethnicity groups (Asain, Black, White) with 2 sexes (Female, Male) and each has 3 facial expressions (Angry, Crying and Happy). So that, the total numbers of human models we have at the end are:

$$3\ ethnicity * 2\ sexes * 3\ expressions = 18\ models$$

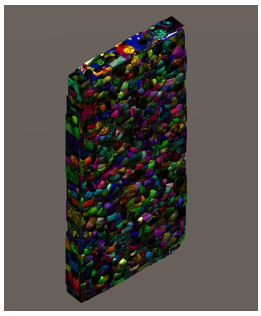
2.2 Generate randomized data

As we have mentioned in the Introduction, we want to consider different factors in which make an image diverse. The **Unity perception package** (2) comes handy when creating randomized factors as it is a toolkit for generating large-scale data set for perception-based machine learning training and validation by leveraging the power of **Unity Simulation** (3). The package comes with a handful of camera-based use cases which allow us to build a randomized scene inside Unity editor.

2.2.1 Background

Background is probably the most important factor for making an image diverse. In order to make the background random, we will create a wall behind the human model acting as the random background. To start with, we first created an empty scene and add the component ***Fixed Length Scenario*** provided by the package, which it controls the execution flow of our simulation by applying randomization parameters from top to bottom. Then we add ***BackgroundObjectPlacementRandomizer*** to our scenario, this script allowed us to generate a wall combined by the shapes prefabs we have selected(cube, cylinder, sphere etc). By using 2 layers with the separation distance of 0.4 between the center of each shapes, we got the result in (a). To attach and alter the colour of textures to our shapes in the wall, we added ***TextureRandomizer*** and ***HueOffsetRandomizer***. The last step is to add ***RotationRandomier*** so it will give all the shapes a new random rotation after each iteration, the end product will look like the image shown in (b) and our camera sees the image in (c). Until now, we have created a background in which it maximized the noise of an simulated image's background.

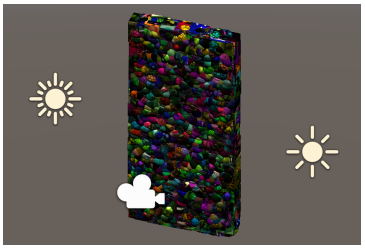
(a) Wall without texture



(b) Wall with texture

(c) Camera View

2.2.2 Directional Light



The directional light is the next important factor that contributes to the randomization of the simulated data. The light **direction** decides where the shadow will occur on the human body and the light **intensity** decides level of darkness or burnt-out on the human body. We have arranged two directional light sources, each placed at $45°$ to the left and right of the background wall. We have built a custom light script ***MyLightRandomzier*** for both of our directional lights by randomly selecting the light's intensity and color on each iteration of the Scenario. After a number of experiments we found the optimal minimum maximum light intensity are **0.5** & **3.0** to provide us with a typically nice lighting effect without excessive darkness or burnt-out highlights in ***MyLightRandomzier***.

2.2.3 Human

As we have finished setting up all the environment in our Senarios, we can now place our human model in the foreground. When we see the real world image of a person, they could face in different directions. We used ***RotationRamdomizer*** provided by the **Unity perception package** (2) and implement it to all 18 models. It gives the model a random rotation by x, y and z axes at each iteration. As the Randomizer picks the center pivot of our human model(near the belly) for rotation and makes the face sensitive to the rotation range, we created a custom pivot prob and moved the original pivot point to our model's neck. As a result, our camera will capture the model's face even with the extreme angle after the rotation. After a number of experiments we found that the best rotation range on x, y and z axes is $-30°$ to $30°$ since the human face will be hard to recognize even by human. In addition to the ***RotationRamdomizer***, we have also added the ***ForeGroundObjectPlacementRandomizer*** to randomly place our model on the xy plain within a safe boundary that can be captured by the camera. After we ran the simulation, some example results look like the following:

2.3 Simulation

By leveraging the power of **Unity Simulation** (3), we have generated 1800 images with 600 angry, 600 crying and 600 happy. Next we will discuss all the machine learning models we have trained and compare the accuracy of these models recognizing real life images.

3 Machine Learning Models

In this section we will discuss 2 machine learning models we have created and tested. At the end, we will compare the accuracy of these models by testing with images of real humans.

3.1 Facial Expression Recognition with OpenFace and Neural Network

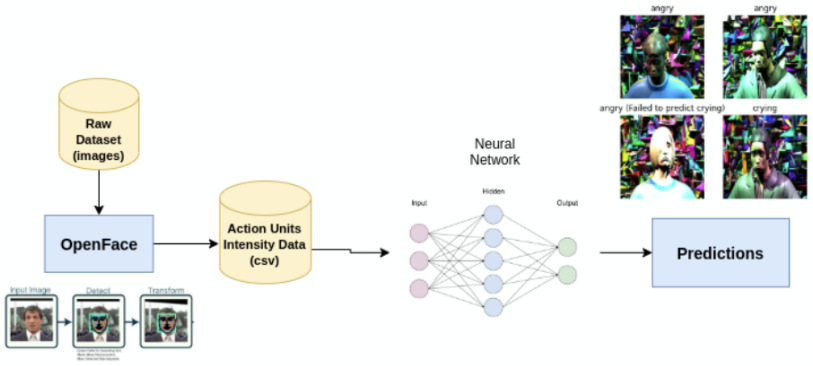


Figure 4: Baseline Model Pipeline

The first approach we have implemented is by leveraging the feature extraction power of **OpenFace** (4). We have fed our entire synthetic dataset (1800 images include angry, crying and happy) into the OpenFace, the **Facial Behavior Analysis** toolkit extracts action unit's intensity and saves numerical results to .csv file for each image. Next, we created a 3-layers neural network and use all the filtered output data from OpenFace to classify the facial expressions. This is the baseline model with 64 neurons in the input layer, 32 neurons in the hidden layer and 3 neurons in the output layer. Although OpenFace is a state of the art tool for facial feature extraction, it cannot locate all the action units in each image from our dataset correctly. As you can see (figure b), OpenFace fails to detect facial expressions in case of extreme angels and darkness but can still barely be recognized by a human. Therefore, we have decided to drop out all the images with the confidence rate < 80% and remove meaningless features from the feature dataset. This would reduce noise and speed up model training and testing.

(a) Successful

(b) Failed (dropped)

Table 1: Filtered images with confidence rate > 80%

Expression	# of images before filtering	# of images after filtering
Angry	600	430
Happy	600	459
Crying	600	419
Total	1800	1308

3.1.1 Image Resizing

During the process of generating our synthetic dataset, we have found that it is inefficient to distribute all the images due to the large image size and redundant space around image borders. Due to the lack of hardware performance in our team, we need to find a balance point to resize our images that gives faster storing and consumes least processing power with minimum sacrifice on OpenFace's accuracy. An experiment has been made for comparing the detectable rate by feeding groups of 100 identical images with different pixel sizes (or memory sizes) into OpenFace. The result shows that 820 * 580 pixels images produce the most balanced result.

Table 2: OpenFace detectable rate with different image sizes

	Detectable pictures rate (%)	300 images in Memory (MB)
1792 * 1267 (original image size)	74.38	790.8
820 * 580 (balance point)	70.21	195.5
256 * 181	60.32	27.9

3.1.2 Training the Model

Now we have a total of 1308 images left in our dataset after filtering (Table 1). We split our dataset to 20% testing, 80% for training and 20% of the validation set among the training batch. The training process repeats 1000 epochs to get the best result. The validation loss is around 0.25. The validation loss decay to a minimum but bounce up after epochs 200, this shows unnecessary training epochs lead to overfit the training data.

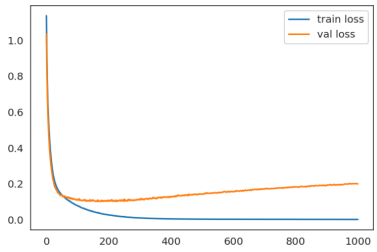


Figure 6: Validation Loss 1000 epochs

3.1.3 Early Stopping

Early Stopping is a method that allows us to stop the training when the performance stops to increase on the validation dataset. After testing different settings, we decided to stop the training if the minimum validation loss does not decrease for 50 more epochs. The model with the best performance is also saved while the training is done. As a result, the final validation loss has been improved and the training saved computing power by stopping the training epoch early.

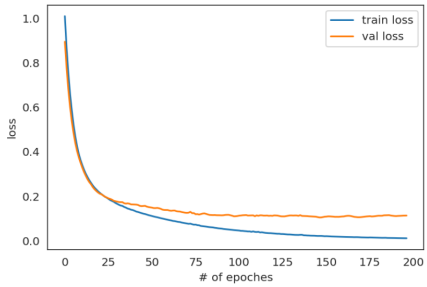


Figure 7: Validation Loss with Early Stopping

3.1.4 Result

In the baseline model training step, we found out that downsizing images data to 820 * 580 is better for data distribution in the performance wise. Features extracted by OpenFace have been cleaned and fed to a neural network and the Early Stopping method applied to the training. Consequently, test loss is 0.198 and accuracy is 94.275% on the synthetic human test set. It proves that the model is capable to classify most of the angry, happy and crying facial expressions in the synthetic human models. This testing result on synthetic data is hopeful. However, the loss is 1.09 and accuracy is 77.78% on the real human dataset. The loss depicts the baseline model misclassified facial expressions on real human images. Moreover, OpenFace could not extract features from 27.33% of images in the dataset. An end to end model is needed to directly extract features and train the whole image dataset, including very dark or bright lighting, and faces shown in extreme angles.

Table 3: Accuracy for the test set

	Loss	Accuracy (%)
Synthetic Human Testset	0.197599	94.274807
Real Human Testset	1.091273	77.777779

3.2 End-to-End Model

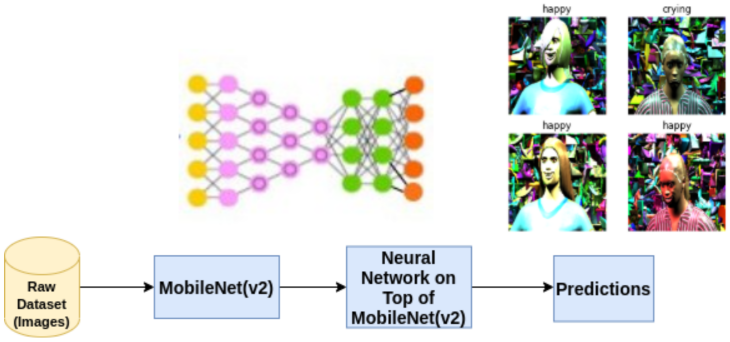


Figure 8: End-to-End model Pipeline

The End-to-End model aims to overcome the limitation of OpenFace which cannot extract features from complicated variations such as faces shown under extreme angles and lighting. With the ability to train and test on the whole dataset, the End-to-End model would have a better classification for angry, happy and crying expressions, including the variations. We used transfer learning to improve the performance of neural network trained on our small synthetic dataset. **MobileNet(V2)** (5) and VGG16 trained on **ImageNet** (6) learns similarly with image features extraction. A customized neural network added on top of the base models (MobileNet(v2) and VGG16) to collect features and make predictions specific to facial expression recognition.

3.2.1 Training Models

MobileNet(V2) (5):

MobileNet(v2) was created for mobile and embedded vision applications. MobileNet(v2) is based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. It is pre-trained on the ImageNet dataset which consists of 1.4M images and 1000 classes. The End-to-End model starts training with the initial weight trained on ImageNet and adds customized layers on top. We added an average pooling layer and a dropout layer with 1280 neurons. A regular densely connected neural network layer was added onto the dropout layer. The weights of customized layers updated based on our synthetic image training dataset. Early Stopping allowed the training process to stop training at the best validation loss remaining for 7 epochs.

VGG16 (7):

We used VGG16 to create another End-to-End model. It is a convolutional neural network model for image recognition proposed by the **Visual Geometry Group** in the **University of Oxford**, VGG16 refers to a VGG model with 16 weight layers. Same as MobileNet(v2), it is pre-trained on the ImageNet dataset. The input layer takes images in the size of (224 x 224 x 3), and the output layer is a softmax prediction on 1000 classes. From the input layer to the last max pooling layer (labeled by 7 x 7 x 512) is regarded as the feature extraction part of the model, while the rest of the network is regarded as the classification part of the model.

3.2.2 Feature Extraction

The convolution base (MobileNet(v2) and VGG16) is used as a feature extractor. The weight of the mobile net model froze to preserve the generic feature extraction ability. Customized trainable layers were added on top of the frozen layers to find a pattern of features in the convolution base for each class. From figure 9, we observe the validation accuracy and loss are improving slowly. For instance, Training with MobileNet(v2) stops at 98 epochs with a validation accuracy of 0.71622 and validation loss of 0.7453. The result statistics show that we should fine tune the model to achieve better performance. The trained network is not good enough to make accurate predictions for facial expression classification.

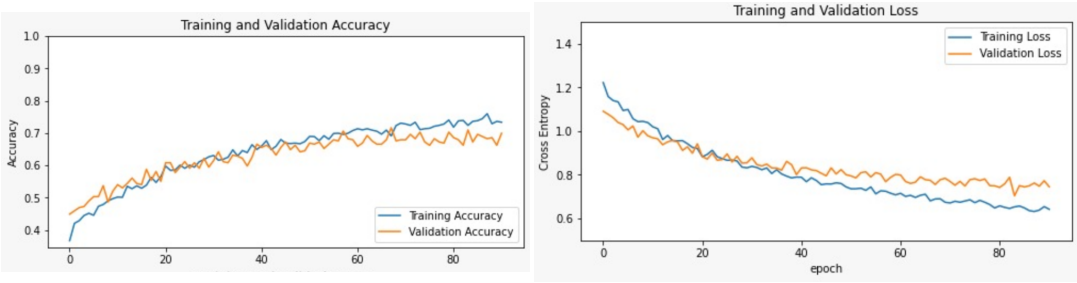


Figure 9: Validation Loss and Accuracy with Feature Extraction

3.2.3 Fine-Tuning

The training process continues based on the best weight update of the customized network. In fine-tuning part, we allow more weights from the last few layers of base models (MobileNet(v2) and VGG16) to be trainable. For instance, MobileNets(v2) contains 154 layers. All the parameters of the last 54 layers are trainable in this step to slightly adjust the pre-trained net's weights. The whole model will then adapt to our facial expression classification task by training data. The learning rate is lower compared with the feature extraction step since the features are already relatively good, so dramatic change is unnecessary. According to figure 10, the fine-tuning step improves the validation accuracy and loss and improvement by training is more significant. Maximum validation accuracy is 0.93581 and minimum validation loss is 0.2895.

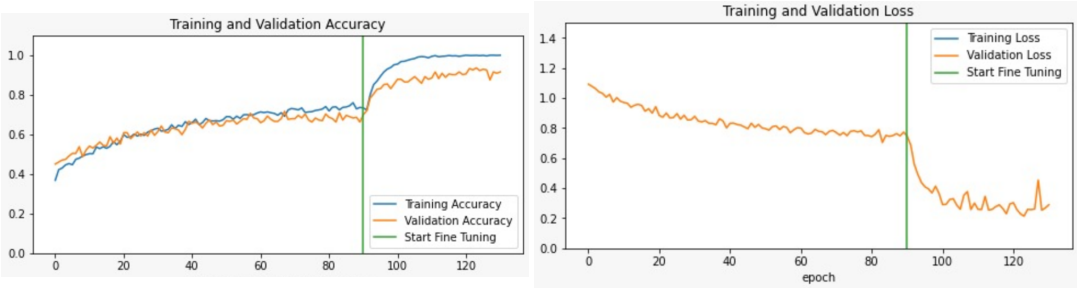


Figure 10: Validation Loss and Accuracy with Fine-Tunin

3.2.4 Result

We have utilized feature extraction and fine-tuning to train an end to end model with the synthetic dataset. The fine-tuning proves to have better performance on training speed (less epochs) and better performance overall. Both steps are necessary to transfer the generic knowledge of a base model in image recognition and learn from the labeled synthetic images. The test accuracy of synthetic dataset is 0.895, vgg-acc and test loss is 0.3497, vgg-loss with the End-to-End model trained on top of MobileNet(v2) and Vgg16 respectively.

4 Conclusion

Throughout this project, we have:

1. Synthetically generated our data that mimics the images in a real-world
2. environment by leveraging the power of Unity perception package and Simulation.
3. Found a balance point to resize our images that gives faster storing and consumes least processing power with minimum sacrifice on OpenFace's accuracy.
4. Created a baseline model with OpenFace and simple neural network, it cannot recoginize 27.33% images in synthetic dataset.
5. Built End-to-End model by transfer learning with MobileNetV2 and VGG16.
6. Tested on 18 images (6 per expression) has been made among three trained models (Table 4). The result shows that baseline model with openFace has the best performance which loss is 0.198, 1.0913 and accuracy is 0.9427, 0.7778 for synthetic dataset and real dataset respectively. MobileNet(V2) preforms better in both dataset among End-to-End models.

Table 4: Accuracy and Loss for the real human faces

	Synthetic Loss	Synthetic Accuracy (%)	Real Loss	Real Accuracy (%)
Baseline Model $openFace$	0.197599	0.9427	1.091273	0.7778
End-to-End Model $VGG16$	0.5525	1.0000	3.2421	0.2222
End-to-End Model $MobileNet(V2)$	0.3497	0.895	3.2633	0.3333

For the future work, we will:

1. Expand the dataset with more synthetic human models with different ethnicity and age groups.
2. Combine our simulated dataset with a good amount of data from real world.
3. Explore into more models in which it will has a high accuracy when recognizing facial expressions.

5 Code

`https://github.com/softMonkeys/Sim2Real`

6 Contributions

Eric

1. Complete the development of baseline Machine Learning Model with OpenFace and neural network.
2. Fine tune the Baseline model by early stopping.
3. Create 18 different human models for three facial expression classes.
4. Contribute to the development of transfer learning with mobileNet v2 model.
5. Create graphs and charts for direct comparison between Machine Learning methods.
6. Create an experiment to find the best image size for the baseline model.
7. Collect a dataset of real human faces for testing.

BenKun

1. Created customized Scenario and Randomizer in Unity by using it's Perception Package.
2. Generated synthetic dataset by using the Unity Simulation.
3. Contribute to the development of transfer learning with VGG16 model.
4. Create the Poster for the Poster Session

References

[1] Submitted by Joel Palmius on Fri, et al. Www.makehumancommunity.org, www.makehumancommunity.org/.

[2] Unity-Technologies. "Unity-Technologies/Com.unity.perception." GitHub, github.com/Unity-Technologies/com.unity.perception.

[3] Technologies, Unity. "Unity Simulation." Unity, unity.com/products/unity-simulation.

[4] B. Amos, B. Ludwiczuk, M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications,"CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861, Cornell University, 2017.

[6] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 2015.

[7] Team, Keras. "Keras Documentation: VGG16 and VGG19." Keras, keras.io/api/applications/vgg/.