**Convolutional Neural Network Hand Emoji Detector with Tensorflow Extended (TFX)**

Hand emoji recognition by using convolutional neural network implemented with Keras, Theano, and OpenCV

**Team Members**

We are a five-member team:

* Ma Jia
* Yuanzhe Li
* Ying Liu
* Yuhua He
* Samuel Yang

**Project Prosal**

We are planning to use deep learning neural network model so that computers can detect hand emojis: punch, ok, peach, stop, nothing. We used OpenCV to process the input images for training with two modes- binary mode, and SkinMask Mode. The CNN model we plan to build has 12 layers. Our project is able to visualize the feature maps in various layers as specified by used in ther terminal’s UI. The project is deployed into Colab, and implemented with CPU, TPU, GPU as comparison of each other. Then we use Tensorboard to measure the accuracy of the model. Finally, we write a tensoflow extended (tfx) model, and implemented the same detection function again, and generate graphs from Airflow.

**Technical Requirements:**

* Python 3
* OpenCV
* Keras
* Tensorflow
* Theano
* Matplotlib
* tfx.component: SchemaGen, Trainer, Evaluator, Transform, ModelValidator, Pusher, Bulkinferrer
* Apache Airflow
* Package versions are specified in the requriements.txt file in the repository.

**Repository file contents**

* **handemoji\_utils.py** : This code file uses TFX and process the data, generate features, and build serving and other infrastructure for executing TFX.
* **handeomoji\_pipeline\_beam\_end2ent\_test.py** : This code file is for testing the TFX model.
* **handemoji\_pipeline\_beam.py** : This code file converted our main function that runs traditional deep learning models into Tensorflow Extended (TFX)
* **trackhandemoji.py** : This is our main function. This file contains all the code for user interface options and OpenCV code to capture camera contents. This file internally calls interfaces to HandEmojiCNN.py.
* **HandEmojiCNN.py** : This file holds all the CNN specific code to create CNN model, load the weight file (if model is pretrained), train the model using image samples present in **./imgfolder\_b**, visualize the feature maps at different layers of NN (of pretrained model) for a given input image present in **./imgs** folder.
* **imgfolder\_b** : This folder contains all the 4015 hand emoji images to train the model. You need to unzip the file inside.
* ***pretrained\_weights\_MacOS.hdf5*** : This is pretrained weight file on MacOS. Due to its large size (150 MB), its hosted seperately on this google driver link - <https://drive.google.com/file/d/1j7K96Dkatz6q6zr5RsQv-t68B3ZOSfh0/view>. You need to download this and save it to the application folder before running the app.
* ***imgs*** - This folder has some sample images to visualize feature layer maps at different layers along with App Demo images for ReadMe display. We have multiple layer images pasted below.

**Colab links**

for trackhandemoji.ipynb and HandEmojiCNN.ipynb : (SJSU accounts have view access to below links) <https://colab.research.google.com/drive/1_pPYctqgU4mS8Y33uYAuzh9jvhYrl1Fe> (HandEmojiCNN.ipynb) <https://colab.research.google.com/drive/1MjzqiI5bgT2Aw67Ety7d6L6TQMw3Ee1f> (Trackhandemoji.ipynb) This colab shows visulization of various image layers, and tensor board)

**Implementation from local terminal**

**On Mac only** (We tested this application on multiple mac laptops.)

With Theano as backend

$ KERAS\_BACKEND=theano pythonw trackhandemoji.py

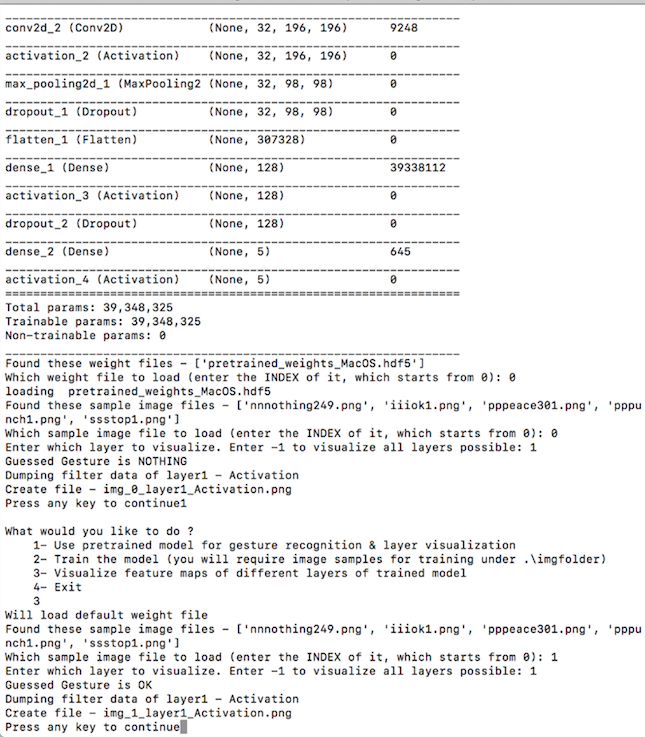
We are setting KERAS\_BACKEND to change backend to Theano.

**Input and Output**

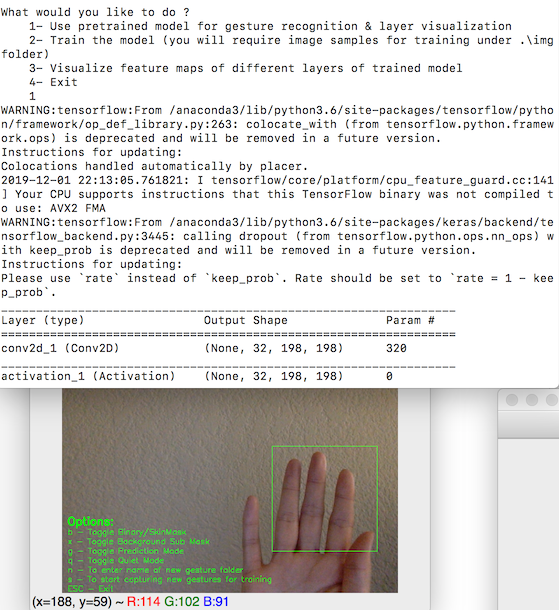
* We use the 4015 images to train the model.
* Users could use pre-trained weights without training the model. In this case, input is the hand emoji images captured from the computer camera.
* Output: application will predict the hand emoji's names. We have 5 names: OK, Peace, Stop, Punch, Nothing.

**App Demo**

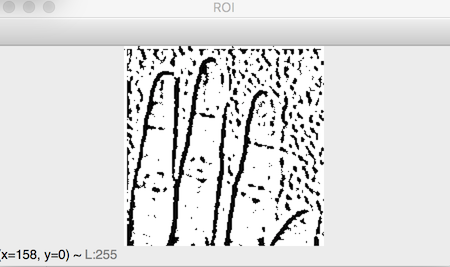
**local terminal**

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/Demo1.png)

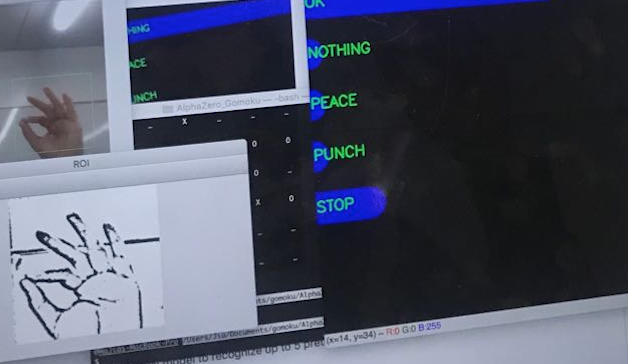
**camera capture**

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/Demo2.png)

**image processing**

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/Demo3.png)

**prediction result**

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/demo5.png)

**Features**

This application comes with CNN model to recognize up to 5 pretrained gestures:

* OK
* PEACE
* STOP
* PUNCH
* NOTHING (ie when none of the above gestures are input)

This application provides following functionalities:

* Prediction : this allows the app to guess the user's gesture against pretrained gestures. App can dump the prediction data to the console terminal or to a json file directly which can be used to plot real time prediction bar chart
* Training : this allows user to retrain the CNN model. User can change the model architecture or add/remove new gestures. This app has inbuilt options to allow users to create new image samples of user defined gestures if required.
* Visualization : this allows the user to see feature maps of different neural network layers for a given input gesture image. As illustrated in the layer images section below, each image will be saved with neural network layer name, such as activation, max pooling etc.

**Hand Emoji Input**

We are using OpenCV for capturing user's hand gestures. In order to simplify the prediction, we are doing post processing on the captured images to highlight the contours & edges, such as applying binary threshold, blurring, gray scaling.

There are two modes of image capturing:

* Binary Mode : we first convert the image to grayscale, then apply a gaussian blur effect with adaptive threshold filter. This mode is useful when you have an empty background like a wall, whiteboard etc.
* SkinMask Mode : we first convert the input image to HSV and put range on the H,S,V values based on skin color range. Then apply errosion followed by dilation. Then gaussian blur to smoothen out the noises. Using this output as a mask on original input to mask out everything other than skin colored things. Finally I have grayscaled it. This mode is useful when there is good amount of light and you dont have empty background.

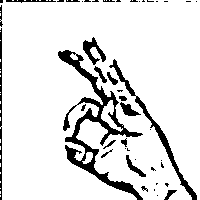
**Binary Mode**

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

blur = cv2.GaussianBlur(gray,(5,5),2)

th3 = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY\_INV,11,2)

ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/iiiok1.png)

**SkindMask Mode**

hsv = cv2.cvtColor(roi, cv2.COLOR\_BGR2HSV)

#Apply skin color range

mask = cv2.inRange(hsv, low\_range, upper\_range)

mask = cv2.erode(mask, skinkernel, iterations = 1)

mask = cv2.dilate(mask, skinkernel, iterations = 1)

#blur

mask = cv2.GaussianBlur(mask, (15,15), 1)

#cv2.imshow("Blur", mask)

#bitwise and mask original frame

res = cv2.bitwise\_and(roi, roi, mask = mask)

# color to grayscale

res = cv2.cvtColor(res, cv2.COLOR\_BGR2GRAY)

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/iiok44.png)

**CNN Model Architecture**

model = Sequential()

model.add(Conv2D(nb\_filters, (nb\_conv, nb\_conv),

padding='valid',

input\_shape=(img\_channels, img\_rows, img\_cols)))

convout1 = Activation('relu')

model.add(convout1)

model.add(Conv2D(nb\_filters, (nb\_conv, nb\_conv)))

convout2 = Activation('relu')

model.add(convout2)

model.add(MaxPooling2D(pool\_size=(nb\_pool, nb\_pool)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(nb\_classes))

model.add(Activation('softmax'))

This model has following 12 layers -

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 32, 198, 198) 320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 32, 198, 198) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 32, 196, 196) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_2 (Activation) (None, 32, 196, 196) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 32, 98, 98) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 32, 98, 98) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 307328) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 39338112

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_3 (Activation) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 5) 645

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

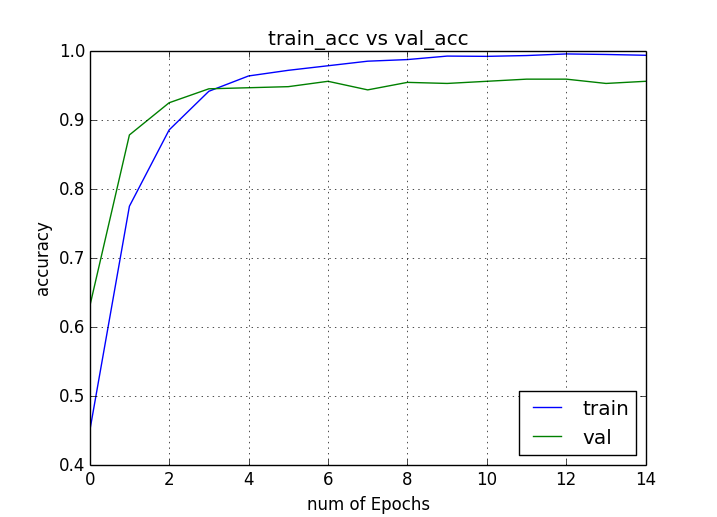
activation\_4 (Activation) (None, 5) 0

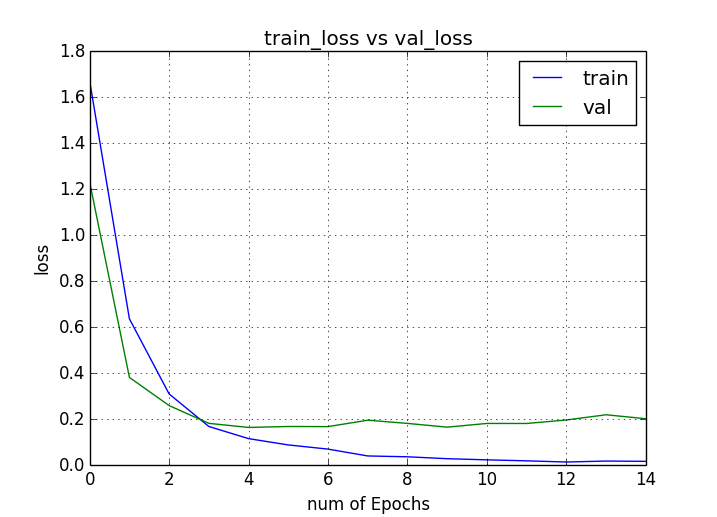
=================================================================

Total params: 39,348,325.0 Trainable params: 39,348,325.0

**Training**

We have trained the model for 15 epochs,and used loss & accuracy graphs for measurement. The results can be viewd in Colab links as well.

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/ori_4015imgs_acc.png)

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/ori_4015imgs_loss.png)

**Visualization**

After launching the main script, choose option 3 for visualizing different or all layer for a given image (currently it takes images from ./imgs, so change it accordingly)

What would you like to do ?

1- Use pretrained model for gesture recognition & layer visualization

2- Train the model (you will require image samples for training under .\imgfolder)

3- Visualize feature maps of different layers of trained model

3

Will load default weight file

Image number 7

Enter which layer to visualize -1

(4015, 40000)

Press any key

samples\_per\_class - 803

Total layers - 12

Dumping filter data of layer1 - Activation

Dumping filter data of layer2 - Conv2D

Dumping filter data of layer3 - Activation

Dumping filter data of layer4 - MaxPooling2D

Dumping filter data of layer5 - Dropout

Can't dump data of this layer6- Flatten

Can't dump data of this layer7- Dense

Can't dump data of this layer8- Activation

Can't dump data of this layer9- Dropout

Can't dump data of this layer10- Dense

Can't dump data of this layer11- Activation

Press any key to continue

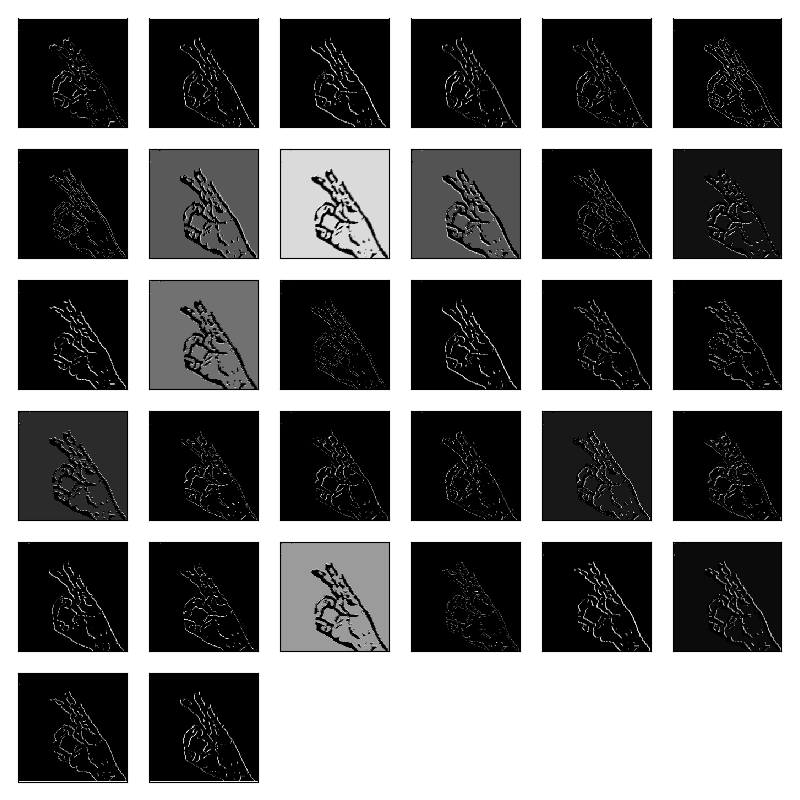
To understand how its done in Keras, check visualizeLayer() in gestureCNN.py

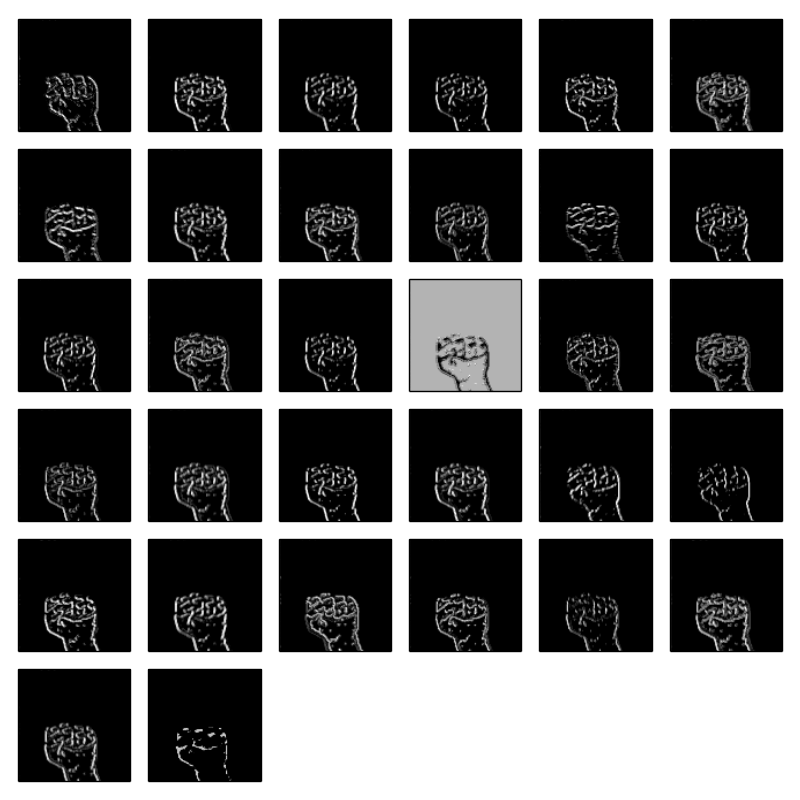
layer = model.layers[layerIndex]

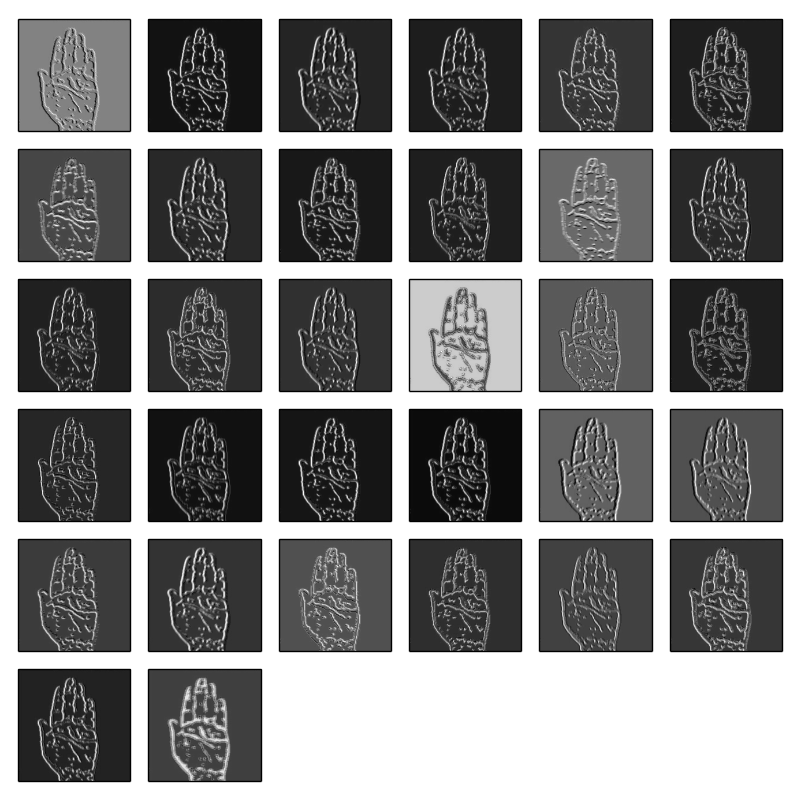
get\_activations = K.function([model.layers[0].input, K.learning\_phase()], [layer.output,])

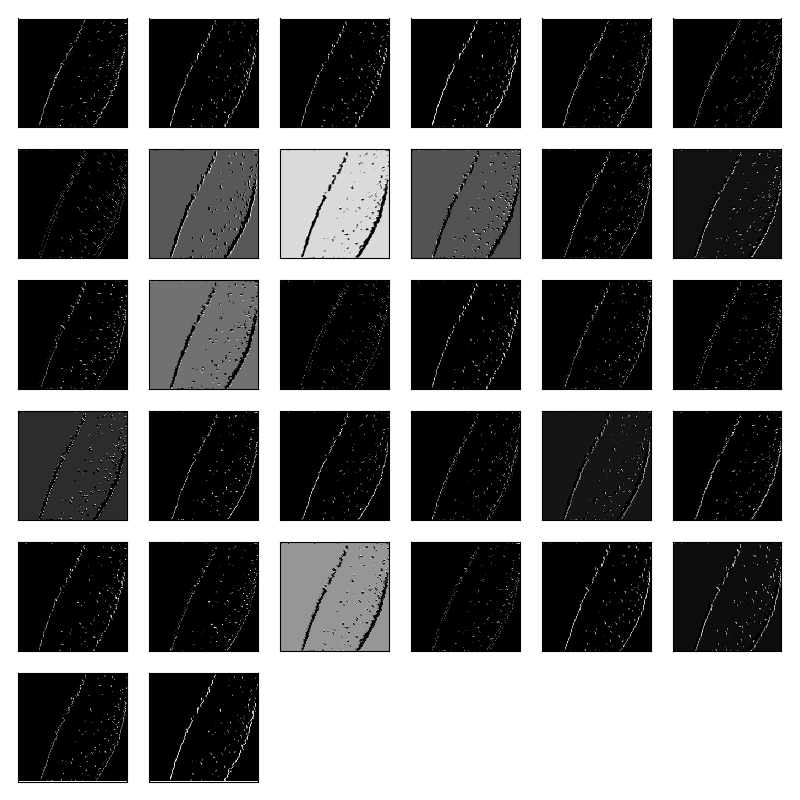
activations = get\_activations([input\_image, 0])[0]

output\_image = activations

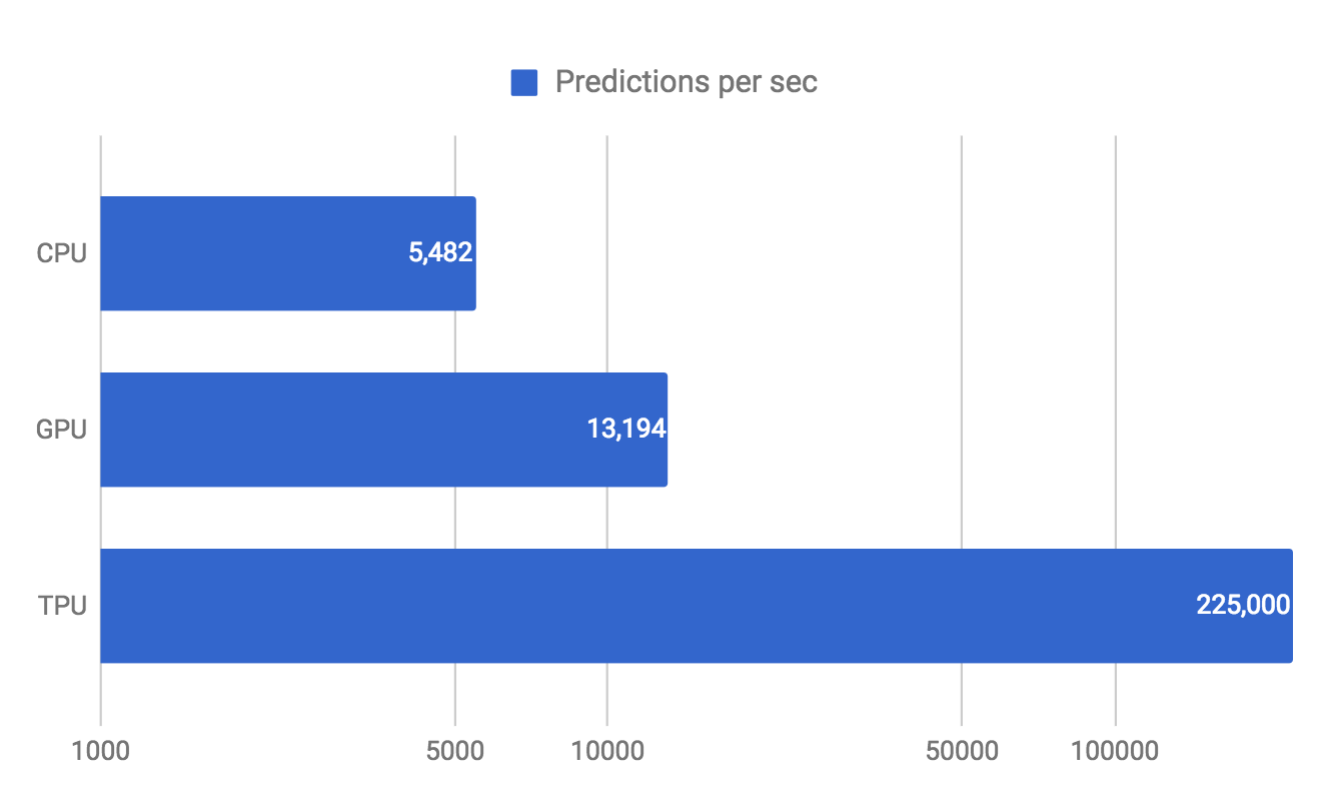
Layer 1 visualization for OK gesture [](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/img_1_layer1_Activation.png)

Layer 4 visualization for PUNCH gesture [](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/img_4_layer4_MaxPooling2D.png)

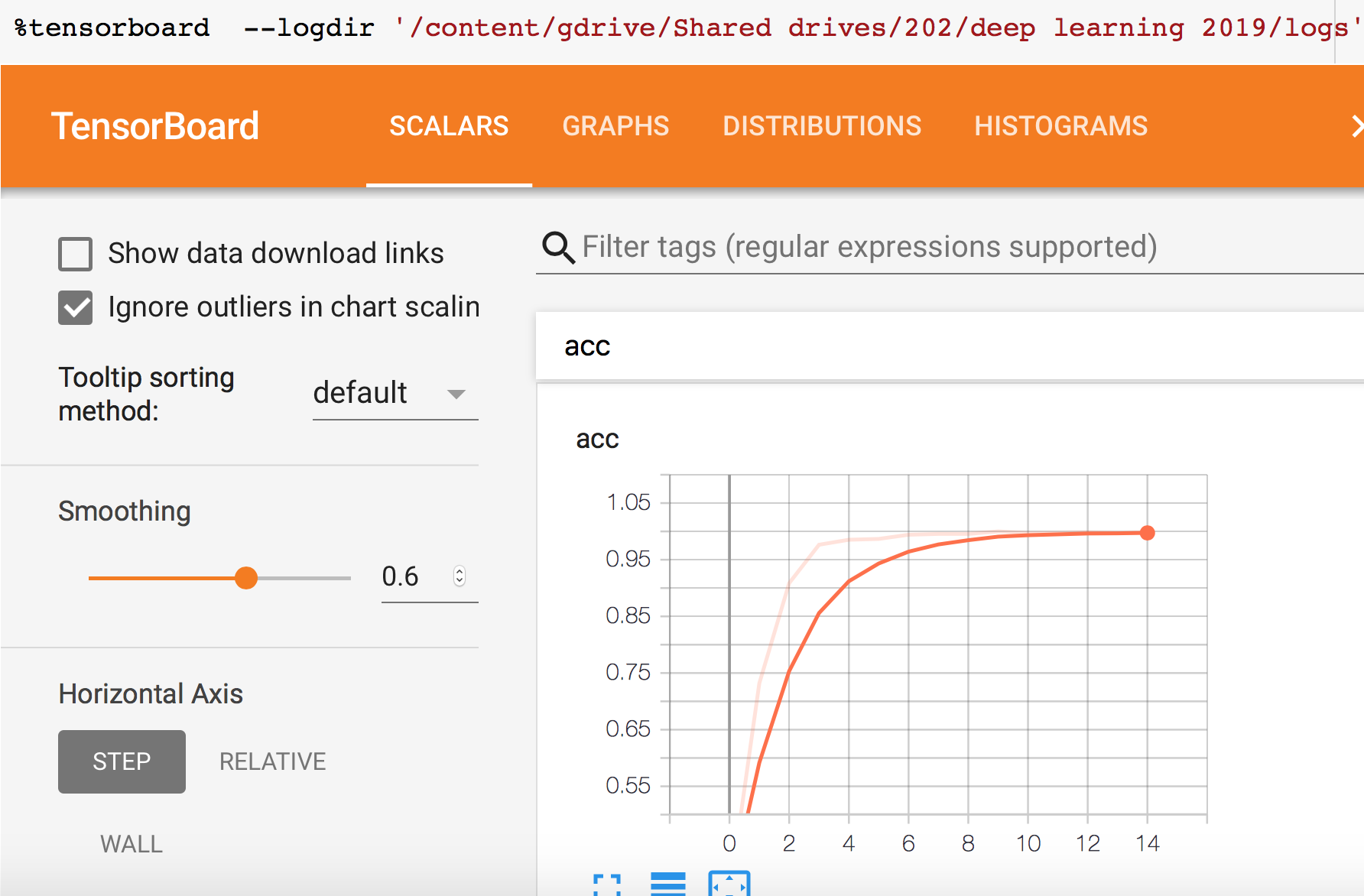
Layer 2 visualization for STOP gesture [](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/img_7_layer2_Conv2D.png)

Layer 1 visualization for NOTHING gesture [](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/img_0_layer1_Activation.png)

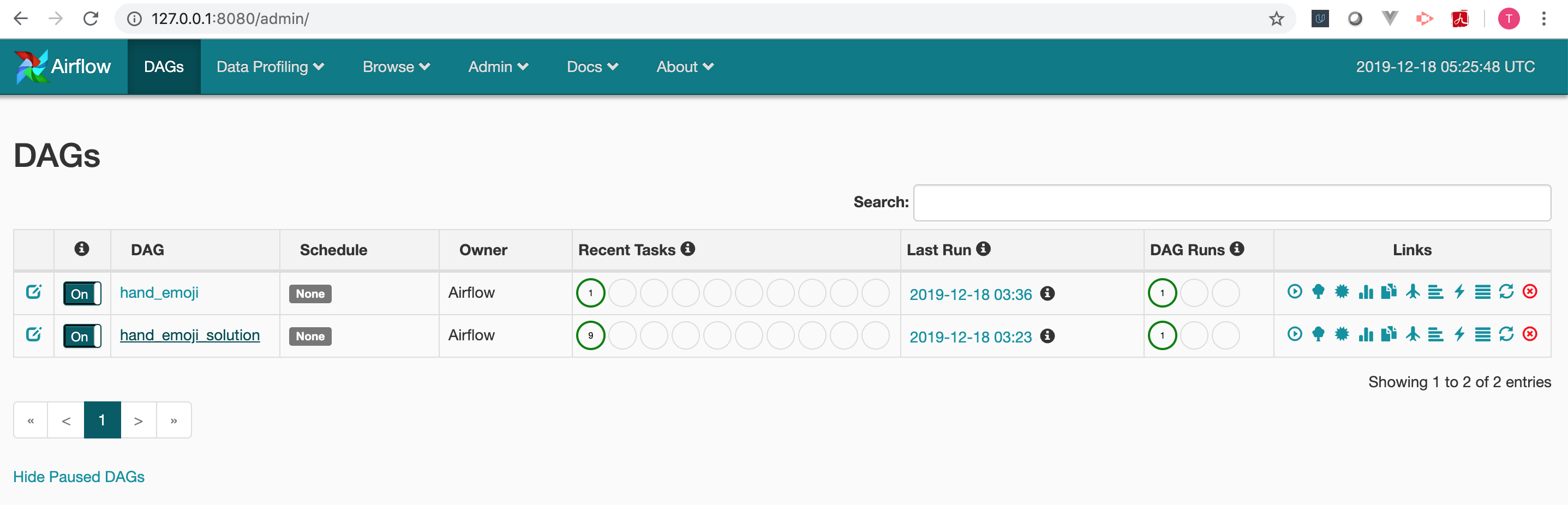
**TPU, CPU, GPU Comparision**

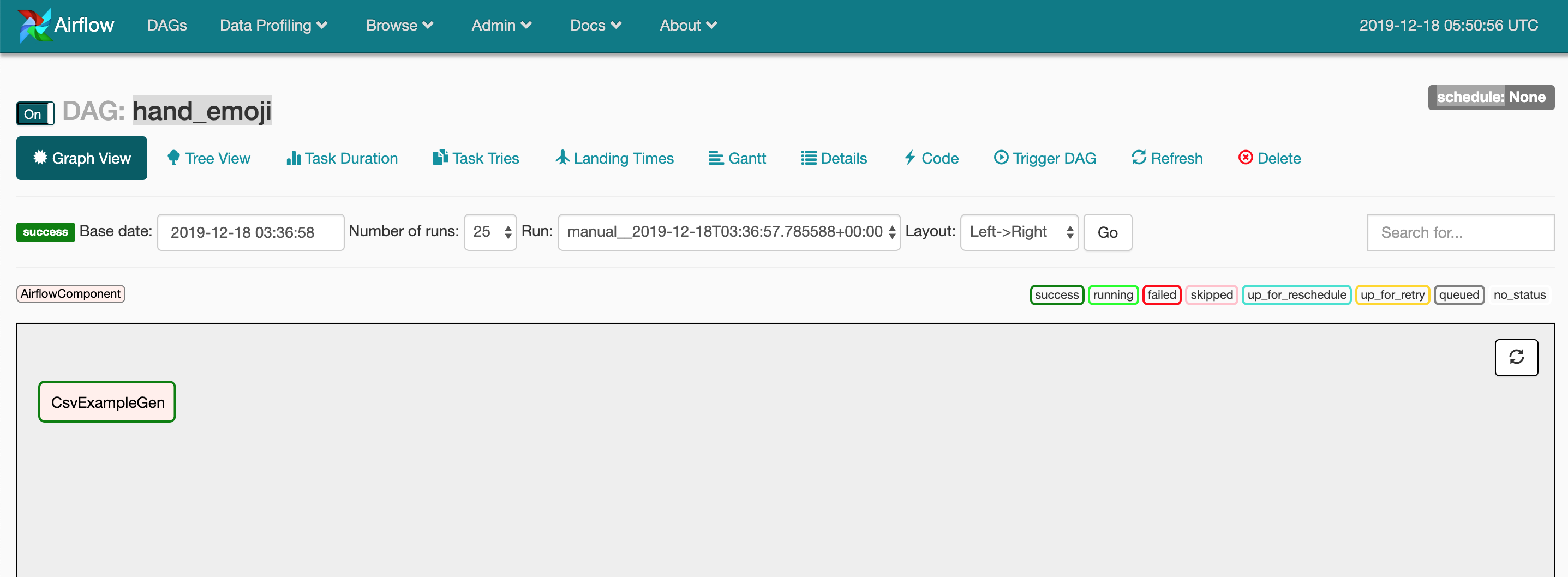
[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/CPU_TPU_GPU)

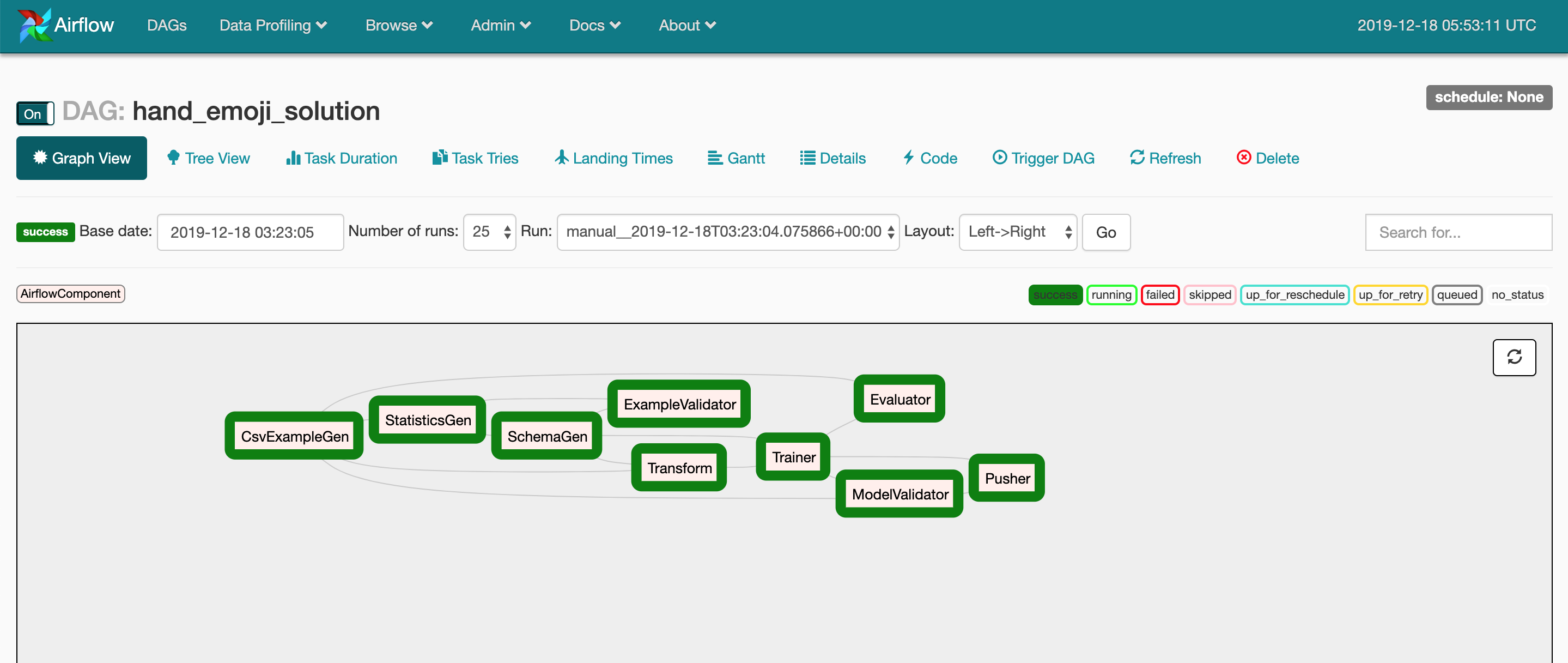
**Tensor Board**

[](https://github.com/wise-monk123/CMPE297DeepLearningProject/blob/master/imgs/Tensorboard.png)

**Tensorflow Extended (TFX)**

Airflow Dags [](https://user-images.githubusercontent.com/26746908/71058194-dcc1b900-2113-11ea-8411-1a479264cf5d.png)

Data Generator [](https://user-images.githubusercontent.com/26746908/71059301-66bf5100-2117-11ea-84bc-3d5182952611.png)

Graph [](https://user-images.githubusercontent.com/26746908/71059389-a423de80-2117-11ea-96c8-4bfe8cdb0e78.png)