# Face Mask Detection for Covid-19 Pandemic

Team Name: Algorithm & Logarithm

Team Members: Fan Guo (A20473828)

Hong Yao (A20472837)

## Project Description and Improvement:

Ever since the outbreak of convid-19 pandemic, the world is heavily hit. The number of diagnosed infection case is still growing, and people are still suffering from this disease physically, economically, and spiritually. Right now, it seems the spreading momentum is slightly constrained, with the help of vaccine, potent drugs and other successful anti-virus experiences.

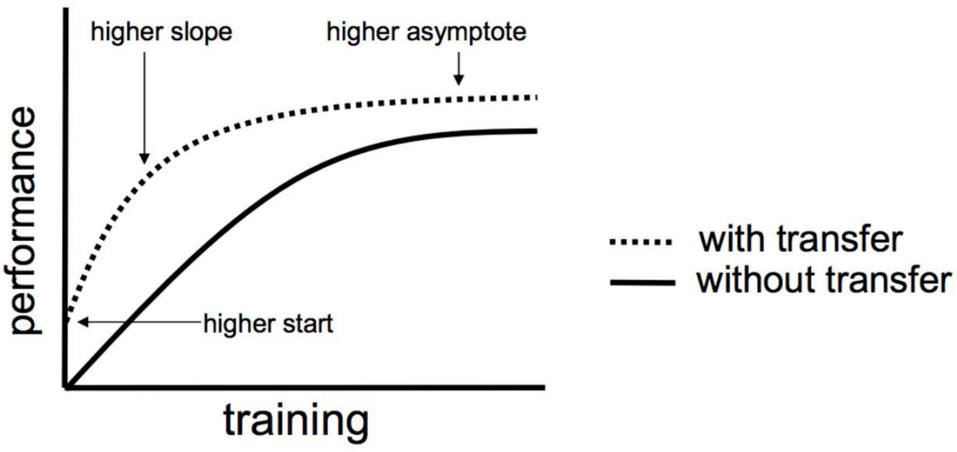
Among those experiences, wearing face-masks is proven one of them. However, putting a mask over the face is not a natural and comfortable habit to do so, even with the compulsive regulation from Government, many people are still emotionally against wearing masks in public. Fortunately, with the developed technology of human face detection, it is fairly easy to develop a system only detecting and warning those who are not wearing masks, but without violating human privacy.

Our project aims to optimize an existing face-mask detection system (open-sourced online with established dataset) with expectation of higher accuracy and correction of frequent errors, meanwhile learn the principle of SSD algorithm and integrate more function features to the system, such as detecting and alert abnormal body temperature synchronously.

## 1. Understand concepts and algorithms

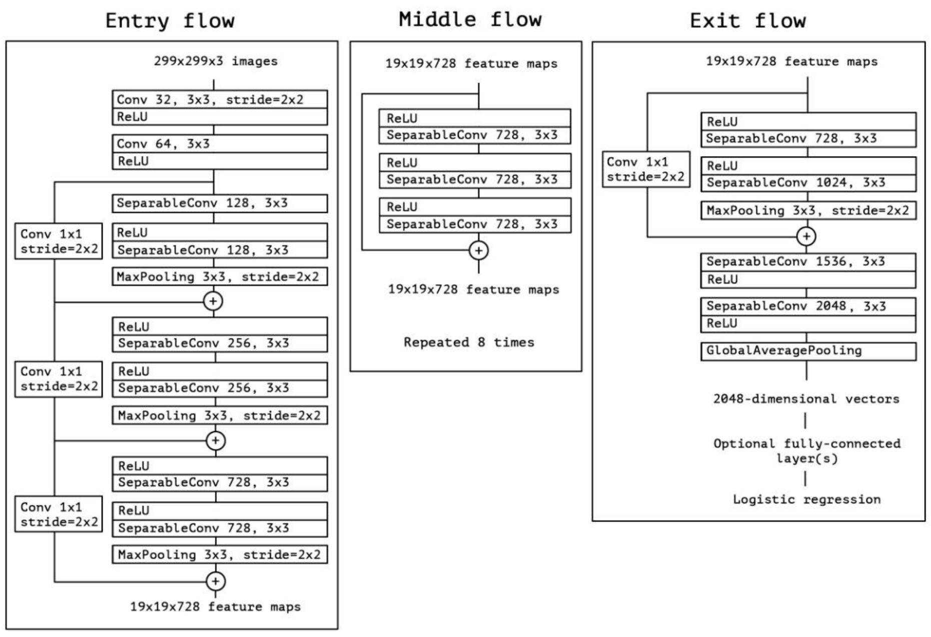
In the mask recognition project, we used transfer learning and Xception algorithms.

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.It is also an optimization, a shortcut to saving time or getting better performance.And It is common to perform transfer learning with predictive modeling problems that use image data as input.



Inspired by the concept of transfer learning, we make use of the Xception model to train ”whether or not to wear a mask”. It is supposed to be time-saving and should reduce the magnitude of the calculation.

Xception is a highly recognized efficient model for target detection, fine-grained classification, face attributes and large-scale geolocation scenarios. Initially, we tried another model, MobileNet, but the result isn’t quite what we expected, so we switched to Xception Model.



## 2. Prepare image dataset

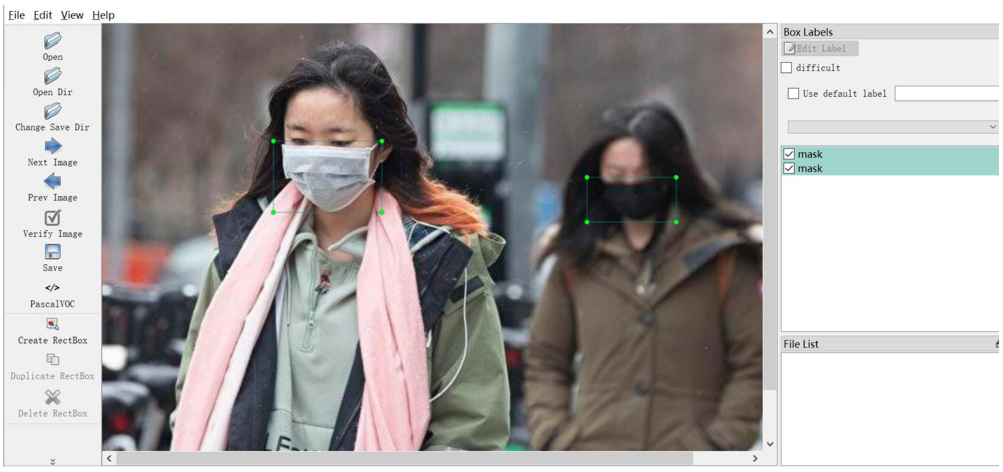
Xception is a highly recognized efficient model for target detection, fine-grained classification, face attributes and large-scale geolocation scenarios. Initially, we tried another model, MobileNet, but the result isn’t quite what we expected, so we switched to Xception Model.

## 1.1 Labeling image

There are two ways to get a dataset:

- download a dataset Kaggle
- tagged by the tool Labelimg

Labelimg is a graphical image annotation tool.It is written in Python and uses Qt for its graphical interface. The following are the steps to install and use Labelimg(Download).



## 1.2 View Datasets

All data sets are stored in the ./archive/ , and you can see that the image is identified as a binary category: masked and unmasked.

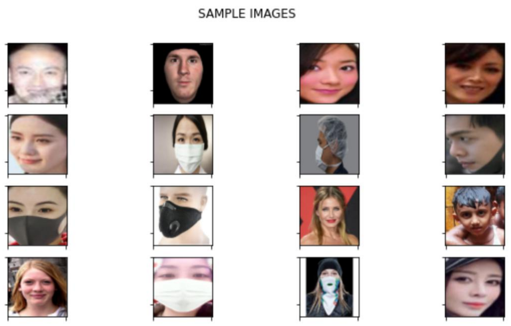
```
data_folder = './archive/'
categories = ['with_mask', 'without_mask']
len_categories = len(categories)
print('Mask Categories:', len_categories)
```

```
Mask Categories: 2
```

Then through the Matplotlib module, draw part of the image sample, used to view part of the data set.

```
# function to get an image
def read_img(filepath, size):
    img = image.load_img(os.path.join(data_folder, filepath), target_size=size)
    #convert image to array
    img = image.img_to_array(img)
    return img
```

```
nb_rows = 4
nb_cols = 4
fig, axs = plt.subplots(nb_rows, nb_cols, figsize=(10, 5));
plt.suptitle('SAMPLE IMAGES')
for i in range(0, nb_rows):
    for j in range(0, nb_cols):
        axs[i, j].xaxis.set_ticklabels([])
        axs[i, j].yaxis.set_ticklabels([])
        axs[i, j].imshow((read_img(df['filepath'].iloc[np.random.randint(998)], (255,255)))/255)
plt.show()
```



## 1.3 PREPROCESSING THE IMAGES

First, we blur the image. We know by smoothing an image we suppress most of the high-frequency components.And OpenCV provides cv2.gaussianblur function to apply Gaussian Smoothing on the input source image.

Then, we subtract this smoothed image from the original image(the resulting difference is known as a mask). Thus, the output image will have most of the high-frequency components that are blocked by the smoothing filter.

Adding this mask back to the original will enhance the high-frequency components.

```
def understand_gaussianblur(src_image):
    # read image
    image = cv2.imread(src_image)
    # apply guassian blur on src image
    image_blurred = cv2.GaussianBlur(image,(0, 0),3)
    image_sharp = cv2.addWeighted(image, 1.5, image_blurred, -0.5, 0)
    # display input and output image
    cv2.imshow("Gaussian + addWeighted",np.hstack((image,image_blurred, image_sharp)))
    cv2.waitKey(0) # waits until a key is pressed
    cv2.destroyAllWindows() # destroys the window showing image

understand_gaussianblur('archive/without_mask/Faceimg1,093.jpg')
understand_gaussianblur('archive/with_mask/image994.jpg')
```

Through the following images we can see the change trend of the image during preprocessing.

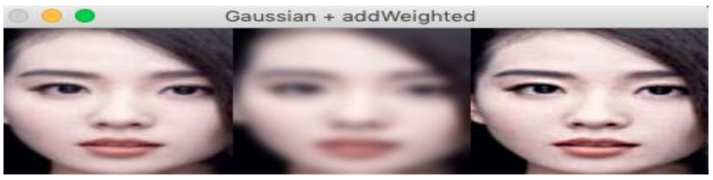
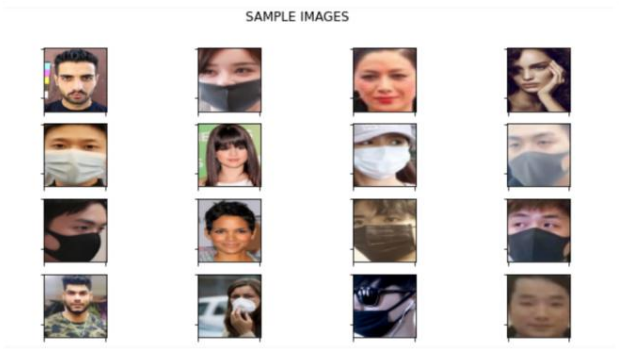
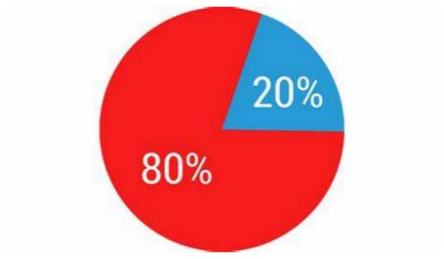


image                Blur the                Apply Unsharp masking

## 2. Train datasets and train datasets

When evaluate the predictive performance of model, its essential that the process be unbiased. Using train_test_split() from the data science library scikit-learn, We can split our dataset into subsets that minimize the potential for bias in evaluation and validation process.Here 80% of images were assigned for training and 20% for validating the ML models.





```
#split the data
y = df['label']
train_x, train_val, y_train, y_val = train_test_split(X_train, y, test_size=0.2, random_state=101)
```

```
print("all count:",X_train.shape[0])
print("train count:", train_x.shape[0])
print('test count:', train_val.shape[0])
```

```
all count: 1000
train count: 800
test count: 200
```

## 3. Bottleneck Feature Extraction

The basic technique to get transfer learning working is to get a pre-trained model (with the weights loaded) and remove final fully-connected layers from that model. We then use the remaining portion of the model as a feature extractor for our smaller dataset. These extracted features are called "Bottleneck Features" (the last activation maps before the fully-connected layers in the original model). We then train a small fully-connected network on those extracted bottleneck features in order to get the classes we need as outputs for our problem.

In [15]:
```
xception_bf = xception.Xception(weights='imagenet', include_top=False, pooling='avg')
bf_train_x = xception_bf.predict(train_x, batch_size=32, verbose=1)
bf_train_val = xception_bf.predict(train_val, batch_size=32, verbose=1)
```

```
25/25 [==============================] - 78s 3s/step
7/7 [==============================] - 20s 3s/step
```

In [16]:
```
#print shape of feature and size
print('Train Shape: ', bf_train_x.shape)
print('Train Size: ', bf_train_x.size)

print('Validation Shape: ', bf_train_val.shape)
print('Validation Size: ', bf_train_val.size)
```

```
Train Shape:  (800, 2048)
Train Size:  1638400
Validation Shape:  (200, 2048)
Validation Size:  409600
```

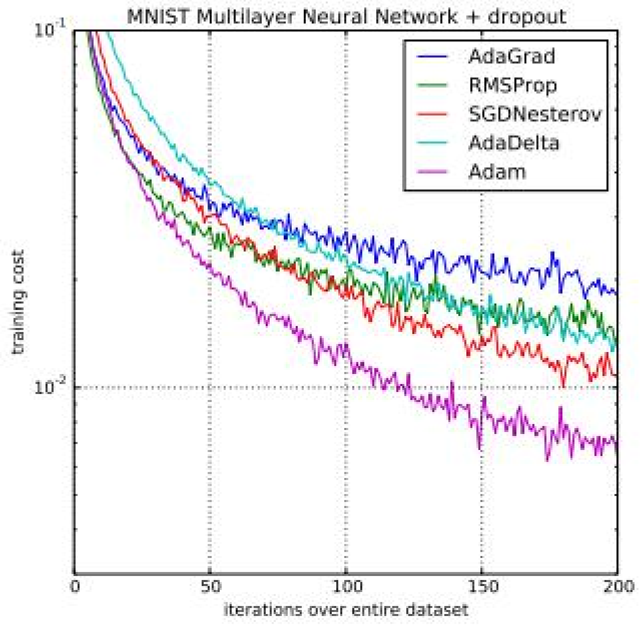
## 4. Modelling

The model consists of three convolution blocks with a average pool layer in each of them. There's not have fully connected layer with 256 units on top of it that is activated by a relu activation function and sigmod activation function. This model has not been tuned for high accuracy, the goal of this tutorial is to show a standard approach.

```
#optimizer
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-3,
    decay_steps=10000,
    decay_rate=0.9)
opt = keras.optimizers.Adam(learning_rate=lr_schedule)

#keras model
model = Sequential()
model.add(Dense(units = 256 , activation = 'relu', input_dim=bf_train_x.shape[1]))
model.add(Dense(units = 64 , activation = 'relu'))
model.add(Dense(units = 1, activation = 'sigmoid'))
model.compile(optimizer = opt,loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.summary()
```

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.



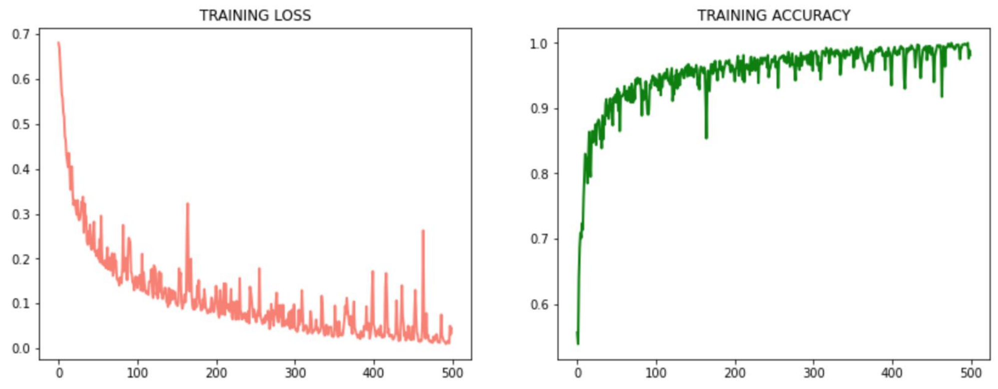
## 5. Loss And Accuracy

Loss can be seen as a distance between the true values of the problem and the values predicted by the model. Greater the loss is, more huge is the errors you made on the data.

Accuracy can be seen as the number of error you made on the data

```
fig, ax = plt.subplots(1,2,figsize=(14,5))
ax[0].set_title('TRAINING LOSS');
ax[1].set_title('TRAINING ACCURACY');


ax[0].plot(history.history['loss'], color= 'salmon',lw=2);
ax[1].plot(history.history['accuracy'], color= 'green',lw=2);
```



## 6. Real time detection using webcam and opencv

Now that our model is trained, we can modify the code so that it can detect faces and also tell us if the person is wearing a mask or not.

In order for our mask detector model to work, it needs images of faces.

## Reference List:

Papers:

[2016ISPL] Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks

[2016NIPS] Understanding the Effective Receptive Field in Deep Convolutional Neural Networks

[CVPR 2017] Xception: Deep Learning with Depthwise Separable Convolutions

Blogs:

https://blog.csdn.net/fengdu78/article/details/104489299/
https://blog.csdn.net/j879159541/article/details/101211358

https://blog.csdn.net/qq_34199326/article/details/88663242?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.control&dist_request_id=615a1c4d-6721-43c7-aafd-bc803f0b02ee&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.control

Video:

https://www.youtube.com/watch?v=_DJnWPOR5bg&t=0s
https://www.youtube.com/watch?v=96o4QcuVU4U&t=0s
https://www.youtube.com/watch?v=JM2IUTSIbwg