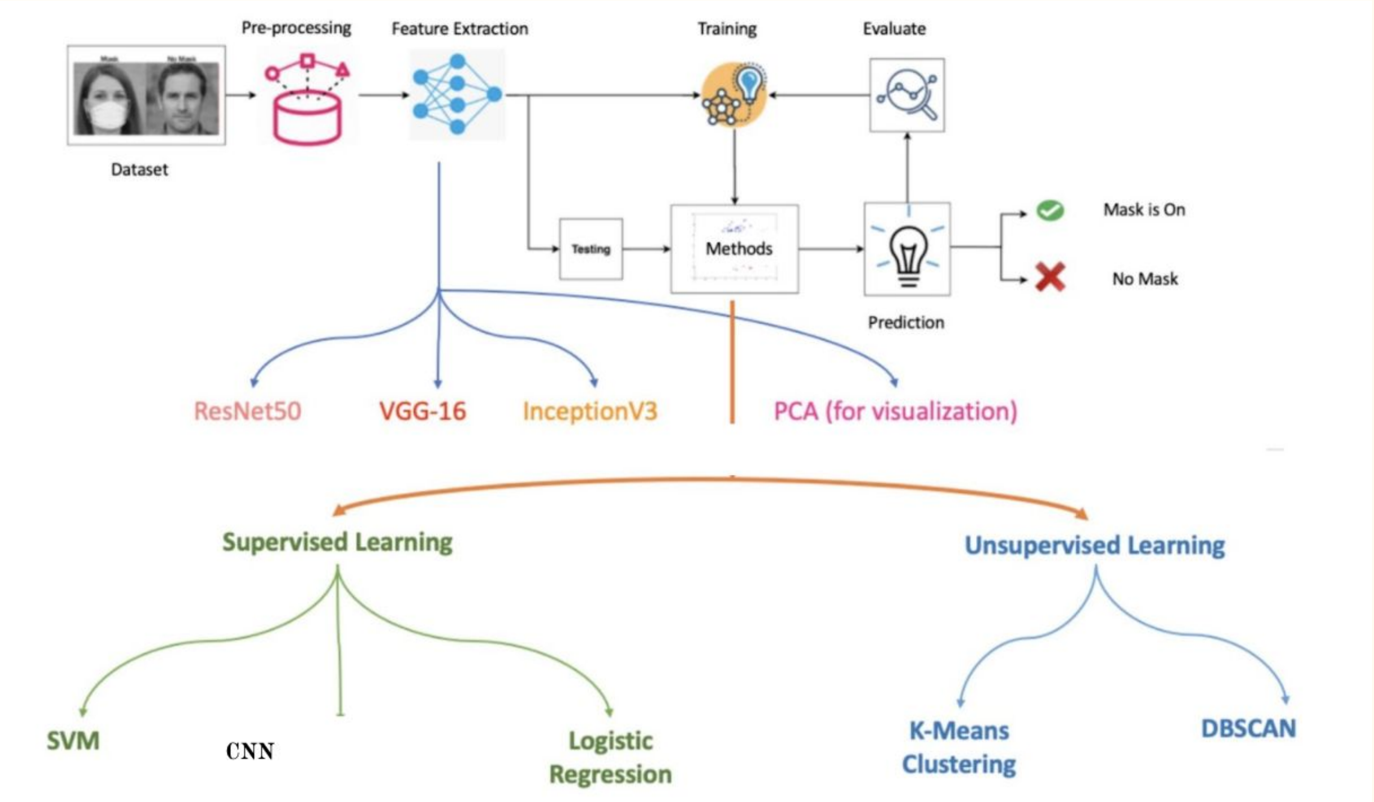
Face Mask Detection

By Shiyuan Liu, Ziyi Jiang, and Lexie Zhu

Abstract

- During the covid-19 pandemic, wearing face masks is becoming a mandatory requirement to protect people's safety. This project is thus inspired to perform face mask verification. Given a photograph, we will be categorizing whether people on the photo are wearing face masks correctly, incorrectly or not wearing at all. We implemented two methods to achieve this goal and they are convolutional neural network (CNN) and principal component analysis (PCA) with support vector classifier (SVC). We have studied the efficiency of both models and made a comparison. There are many applications where this project would be meaningful to implement such as mask detection before boarding a plane, or entering school, etc.

Introduction

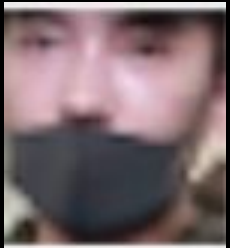


About the Dataset

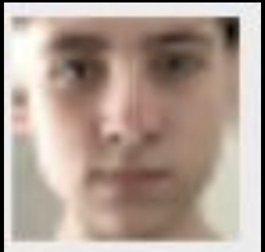
- This project uses face mask detection dataset by Larxel from Kaggle (Version I) which consists of 853 images belonging to 3 classes.
- Most images originally in the dataset consist of more than 1 face per image as illustrated by the example to the right.

Original Images Example

label: with mask



label: mask incorrectly worn



label: without mask

Dataset link:https://www.kaggle.com/andrewmvd/face-mask-detection

Data Extraction

	xmin	ymin	xmax	ymax	label	file	width	height	annotation_file	image_file
0	62	194	160	320	without_mask	maksssksksss299	301	400	maksssksksss299.xml	maksssksksss299.png
1	43	169	149	308	without_mask	maksssksksss528	301	400	maksssksksss528.xml	maksssksksss528.png
2	48	107	218	304	mask_incorrectly_worn	maksssksksss272	275	400	maksssksksss272.xml	maksssksksss272.png
3	28	78	43	99	with_mask	maksssksksss514	400	267	maksssksksss514.xml	maksssksksss514.png
4	160	66	176	83	with_mask	maksssksksss514	400	267	maksssksksss514.xml	maksssksksss514.png
...	...	...	...	...	...	...	...	...	...	...
4067	271	73	278	82	without_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png
4068	236	91	243	99	without_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png
4069	236	76	243	83	without_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png
4070	264	76	268	82	with_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png
4071	281	72	286	78	with_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png

4072 rows × 10 columns

- ➢ extract information from XML files with the location of the object (xmin,ymin,xmax,ymax)
- ➢ notice multiple labels in one image

Data Preprocessing

```
#Main processing and saving
for i in range(len(annotations_info_df)):
    image_filepath = annotations_info_df['image_file'].iloc[i]
    image = cv2.imread(images_directory + '/' + image_filepath)
    image = convert_to_RGB(image)
    xmin = annotations_info_df['xmin'].iloc[i]
    ymin = annotations_info_df['ymin'].iloc[i]
    xmax = annotations_info_df['xmax'].iloc[i]
    ymax = annotations_info_df['ymax'].iloc[i]
    new_cropped = image[ymin:ymax, xmin:xmax]
    new_cropped = Image.fromarray(new_cropped)
    new_cropped.save(input_path_rgb + '/' + str(i) + '.png')
```

- crop images to subimages that contain exactly 1 person per image

Method I - CNN - All the Packages We Use

```
import pandas as pd
import numpy as np

import cv2
import seaborn as sns

import tensorflow as tf
from tensorflow import keras

import os

import glob
from xml.etree import ElementTree

import matplotlib.pyplot as plt
from PIL import Image

import imageio

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
import numpy as np
```

Method I - CNN - Preparing the Input Image

```
x_lst_test = []
y_lst_test = []
x_lst_train = []
y_lst_train = []
test_files = []
train_files = []

test_num = 305
train_num = 2750

dim = (25,25)
count = 0
for i in range(len(annotations_info_df)):
    directory = input_path_rgb + '/' + str(i) + '.png'
    im = cv2.imread(directory)
    max_size = max(im.shape[0],im.shape[1])
    if 13 < im.shape[0] < 150 and 13 < im.shape[1] < 150:
        top = int((max_size - im.shape[0])*(1/2))
        bottom = max_size - im.shape[0] - top
        left = int((max_size - im.shape[1])*(1/2))
        right = max_size - im.shape[1] - left
        im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER_REPLICATE)
        resized = cv2.resize(im, dim, interpolation = cv2.INTER_AREA)
        cv2.imwrite(input_path_cnn + '/' + str(i) + '.png',resized)
        if count <= test_num:
            test_files.append(i)
        else:
            train_files.append(i)
        count += 1
```

- ❖ The cropped image are determined by the axis given in the dataset which make them vary greatly in size and shape.
- ❖ For better output of CNN, we excluded particularly small and large images and stacked the border pixel of each image to make them in square shape. Then we upscale and downscale images to make them equal in size: (25,25,3).
- ❖ There are 3055 images that can be trained and tested upon.

Method I - CNN - Preparing the Input Image

```
directory = revised_input_path
for filename in test_files:
    im = imageio.imread(directory + '/' + str(filename) + '.png')
    im = np.array(im)
    im = np.reshape(im,(25,25,3))
    x_lst_test.append(im)
    if annotations_info_df['label'].iloc[filename] == 'without_mask':
        clsf = np.array([1, 0, 0])
    elif annotations_info_df['label'].iloc[filename] == 'mask_incorrectly_worn':
        clsf = np.array([0, 1, 0])
    elif annotations_info_df['label'].iloc[filename] == 'with_mask':
        clsf = np.array([0, 0, 1])
    y_lst_test.append(clsf)


for filename in train_files:
    im = imageio.imread(directory + '/' + str(filename) + '.png')
    im = np.array(im)
    im = np.reshape(im,(25,25,3))
    x_lst_train.append(im)
    if annotations_info_df['label'].iloc[filename] == 'without_mask':
        clsf = np.array([1, 0, 0])
    elif annotations_info_df['label'].iloc[filename] == 'mask_incorrectly_worn':
        clsf = np.array([0, 1, 0])
    elif annotations_info_df['label'].iloc[filename] == 'with_mask':
        clsf = np.array([0, 0, 1])
    y_lst_train.append(clsf)
```

1. Convert the image to numpy array and store them in to the x_test/x_train.

2. Build the y_test/y_train in the format [1, 0, 0], [0, 1, 0] and [0, 0, 1] to show its class.

3. Convert the x/y train/test data set to numpy array to train (90% and 10%)

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(2749, 25, 25, 3)
(2749, 3)
(306, 25, 25, 3)
(306, 3)
```

Method I - CNN - Building the layers

```
#Build CNN
x_train = x_train/255

model = Sequential()

model.add(keras.layers.Conv2D(8, (3,3), padding='same', input_shape=(25,25,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=None))

model.add(keras.layers.Conv2D(32, (3,3), padding='same', input_shape=(25,25,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=None))

model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation="softmax"))
```

1. Standardized the data set to from 0 to 1 (/255 obviously for an image).

2. Build the 2 convolutional (8 filters and 32 filters)layers and 2 pooling layers to construct the training part.

3. Since there are **3** possible signs, we finish our network with a dense layer with **3** units.

4. 'Adam' optimizer

```
model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
```

5. Set **30** epochs to ensure the CNN fully trained, but can be decreased because the training set touch the top at **19-25** epochs in our analysis

```
batch_size = 32
epochs = 30

history = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, validation_split=0.2)
```

Method I - CNN - Processing Output

```
Epoch 24/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0156 - accuracy: 0.9973 - val_loss: 0.3124 - val_accu
racy: 0.9473
Epoch 25/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0156 - accuracy: 0.9955 - val_loss: 0.3463 - val_accu
racy: 0.9509
Epoch 26/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0108 - accuracy: 0.9986 - val_loss: 0.3520 - val_accu
racy: 0.9491
Epoch 27/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0075 - accuracy: 0.9991 - val_loss: 0.3927 - val_accu
racy: 0.9491
Epoch 28/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0057 - accuracy: 0.9995 - val_loss: 0.4136 - val_accu
racy: 0.9473
Epoch 29/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0054 - accuracy: 0.9991 - val_loss: 0.3721 - val_accu
racy: 0.9455
Epoch 30/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0039 - accuracy: 0.9995 - val_loss: 0.3998 - val_accu
racy: 0.9455
```

The accuracy and loss(include validation part) are all converged after **24 epochs**, and seems like a good and full training.

```
model.evaluate(x_test, y_test)
```

```
10/10 [==============================] - 0s 3ms/step - loss: 40.7221 - accuracy: 0.9085
[40.72206497192383, 0.9084967374801636]
```

Finally evaluate the model with test data set and find out the **loss is 40.7** and **accuracy is around 91%**

Method I - CNN - Optimizing parameters (Examples)

Training with **8 and 16** filters of convolutional layers:

```
Epoch 27/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0821 - accuracy: 0.9691 - val_loss: 0.2039 - val_accur
acy: 0.9382
Epoch 28/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0763 - accuracy: 0.9718 - val_loss: 0.2084 - val_accur
acy: 0.9400
Epoch 29/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0746 - accuracy: 0.9727 - val_loss: 0.2149 - val_accur
acy: 0.9382
Epoch 30/30
69/69 [==============================] - 0s 7ms/step - loss: 0.0688 - accuracy: 0.9754 - val_loss: 0.2190 - val_accur
acy: 0.9436
```

Training with **16 and 32** filters of convolutional layers:

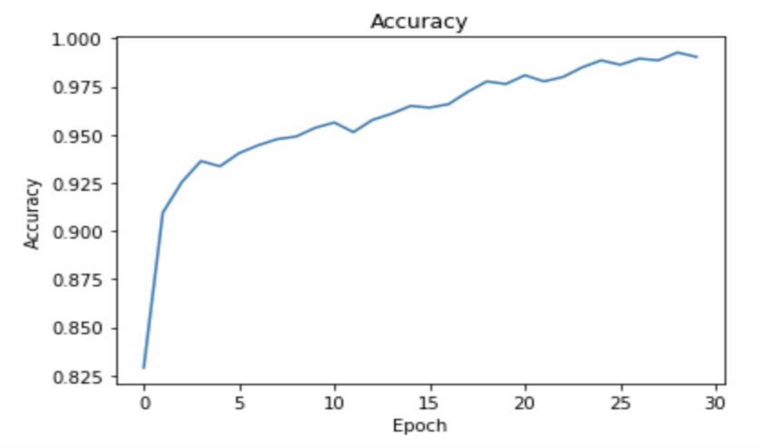
```
Epoch 27/30
69/69 [==============================] - 0s 7ms/step - loss: 0.2746 - accuracy: 0.8813 - val_loss: 0.3162 - val_accur
acy: 0.9018
Epoch 28/30
69/69 [==============================] - 0s 7ms/step - loss: 0.2407 - accuracy: 0.9109 - val_loss: 0.2543 - val_accur
acy: 0.8964
Epoch 29/30
69/69 [==============================] - 0s 7ms/step - loss: 0.2076 - accuracy: 0.9354 - val_loss: 0.2108 - val_accur
acy: 0.9400
Epoch 30/30
69/69 [==============================] - 0s 7ms/step - loss: 0.1982 - accuracy: 0.9363 - val_loss: 0.2016 - val_accur
acy: 0.9400
```

Overall, we change the number of layers; the order of pooling and convolutional layers; the number of filters, the batch_sizes and so on!

Method I - CNN - Measure Performance as Graphs

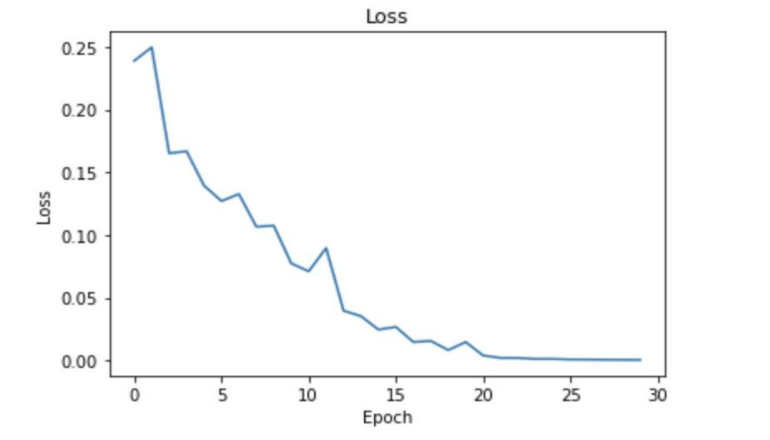
Accuracy :

```
plt.plot(history.history['accuracy'])
plt.title('Accracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```



Loss :

```
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



Method II - SVC with PCA - Packages we use

1. Numpy(ndarray)
2. Pandas(dataframe)
3. Pyplotlib for visualization
4. Scikit Learn

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from time import time
```

Method II - SVC with PCA - Data Washing

1. Drop the data we don't need:

   i). The size is smaller than 20 pixels

   ii). The image with label 'masks incorrectly worn'

```
annotations_info_df_svc.drop(annotations_info_df_svc[annotations_info_df_svc.label=='mask_incorrectly_worn'].index
annotations_info_df_svc.dropna(inplace = True)
annotations_info_df_svc.reset_index(drop=True, inplace=True)
```

2. Padding the images into same shape

```
def padding_image(image):
```

3. Upscale or downscale the images in to same size (30x34). This data is from the average size of all images.

```
def get_average_size():
```

```
def filter_images(image,i):
```

Method II - SVC with PCA - Data Washing

```
1 annotations_info_df_svc
```

	xmin	ymin	xmax	ymax	label	file	width	height	annotation_file	image_file	new_img_file
0	62	194	160	320	without_mask	maksssksksss299	301	400	maksssksksss299.xml	maksssksksss299.png	0.png
1	43	169	149	308	without_mask	maksssksksss528	301	400	maksssksksss528.xml	maksssksksss528.png	1.png
2	113	230	216	368	with_mask	maksssksksss500	301	400	maksssksksss500.xml	maksssksksss500.png	2.png
3	74	205	180	330	with_mask	maksssksksss266	301	400	maksssksksss266.xml	maksssksksss266.png	3.png
4	101	174	209	299	without_mask	maksssksksss716	301	400	maksssksksss716.xml	maksssksksss716.png	4.png
...	...	...	...	...	...	...	...	...	...	...	...
2051	362	89	399	142	with_mask	maksssksksss257	400	267	maksssksksss257.xml	maksssksksss257.png	2051.png
2052	342	34	379	71	with_mask	maksssksksss257	400	267	maksssksksss257.xml	maksssksksss257.png	2052.png
2053	197	84	218	107	with_mask	maksssksksss280	400	267	maksssksksss280.xml	maksssksksss280.png	2053.png
2054	202	78	223	100	with_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png	2054.png
2055	293	72	315	95	with_mask	maksssksksss294	400	241	maksssksksss294.xml	maksssksksss294.png	2055.png

2056 rows × 11 columns

Method II - SVC with PCA - PCA

1. Build a np matrix to store the image information.

   i). it’s RGB image, but one channel is enough for us, we only choose the frist R channel.

   ii) we flatten the first 2D image into 1D at put it in the first row of our matrix

   ii) do the same thing iteratively until finishing.

Method II - SVC with PCA - PCA

1. Build a np matrix to store the image information.
2. We then passed our flattened image matrix into a PCA function provided by scikit-learn for decomposition and feature extraction.

Question:

How to choose the quantum of components?

Method II - SVC with PCA -Comparing

	precision	recall	f1-score	support
0	0.00	0.00	0.00	20
1	0.84	1.00	0.91	450
2	0.00	0.00	0.00	64
accuracy			0.84	534
macro avg	0.28	0.33	0.30	534
weighted avg	0.71	0.84	0.77	534

Predicting masks on the test set

	precision	recall	f1-score	support
0	0.95	0.77	0.85	438
1	0.38	0.79	0.51	76
accuracy			0.77	514
macro avg	0.66	0.78	0.68	514
weighted avg	0.87	0.77	0.80	514

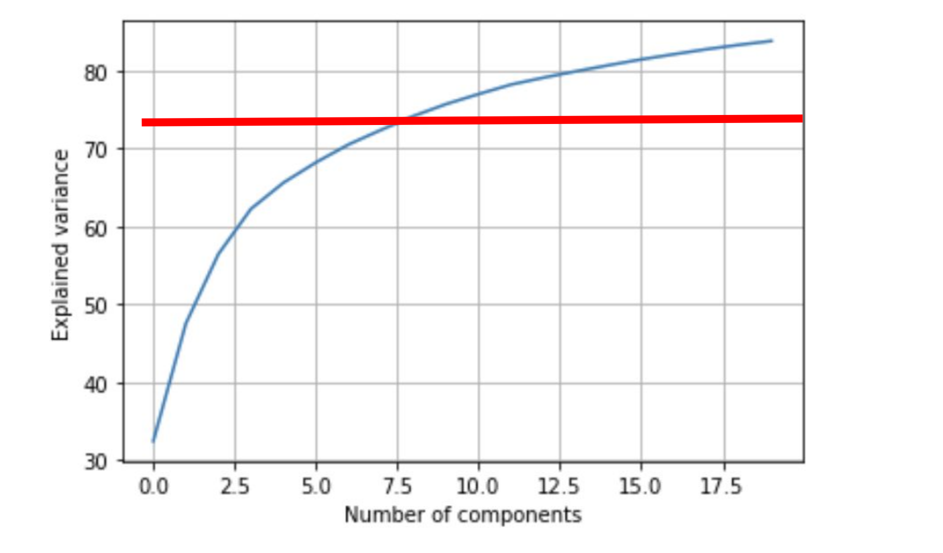
```
pca_100 = PCA(n_components=100)
img_pca_100_reduced = pca_100.fit_transform(img_data)
print("reduced shape:",img_pca_100_reduced.shape)
img_pca_100_recovered = pca_100.inverse_transform(img_pca_100_reduced)
print("recovered shape:",img_pca_100_recovered.shape)
# plt.show(temp)
```

```
pca = PCA(n_components=3,whiten=True)
img_pca_reduced = pca.fit_transform(img_data)
print("reduced shape:",img_pca_reduced.shape)
img_pca_recovered = pca.inverse_transform(img_pca_reduced)
print("recovered shape:",img_pca_recovered.shape)
# plt.show(temp)
```

Scary!!

Method II - SVC with PCA - PCA

1. Build a np matrix to store the image information.
2. We then passed our flattened image matrix into a PCA function provided by scikit-learn for decomposition and feature extraction.




```python
plt.grid()
plt.plot(np.cumsum(pca.explained_variance_ratio_ * 100))
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
plt.show()
plt.savefig('Scree plot.png')
```

Question:

How to choose the quantum of components?

Answer:

From the plot of Explained Variance, we know it's better to choose n around 12.5. Because if the **EV=80%** is best to avoid overfitting or underfitting.

Method II - SVC with PCA - PCA

After experiments,

we choose n_components =13.

```
pca = PCA(n_components=13,whiten=True)
img_pca_reduced = pca.fit_transform(img_data)
print("reduced shape:",img_pca_reduced.shape)
img_pca_recovered = pca.inverse_transform(img_pca_reduced)
overed.shape)
```

```
x = img_pca_reduced
y = np.array(revised_info_df["label"])
labelencoder = LabelEncoder()
y = labelencoder.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.25, random_state=42)
```

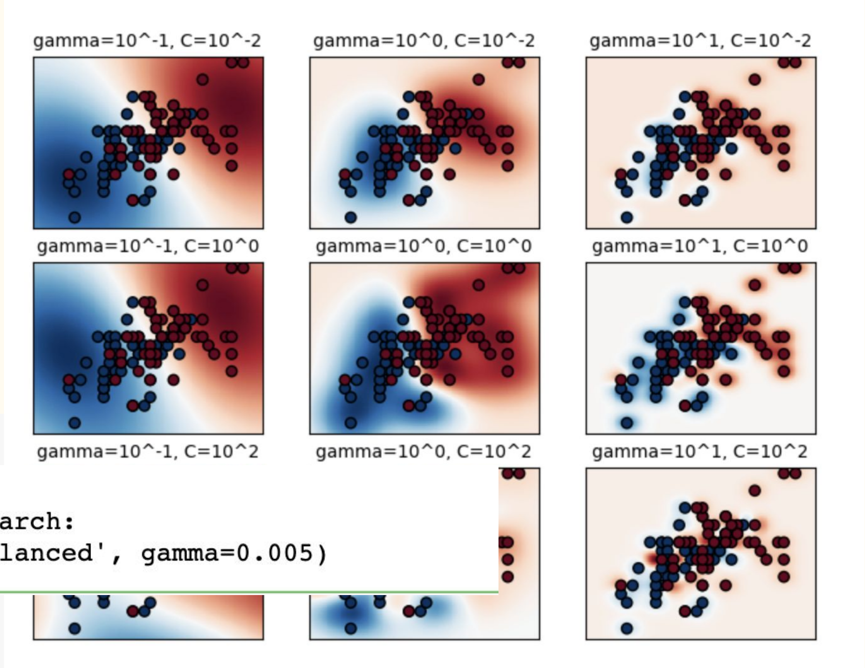
Encode label into 0 and 1 for convenience.

Method II - SVC with PCA -Model Training

Kernel = rbf(Gaussian Radial Basis Function)

$$k(x,y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$$

$$d \in Z^+, \sigma \in \mathcal{R} - \{0\}$$




```
param_grid = {
    "C": [1e3, 5e3, 1e4, 5e4
    "gamma": [0.0001, 0.0005
}

clf = GridSearchCV(SVC(kernel="rbf", class_weight="balanced"), param_grid)
clf = clf.fit(x_train, y_train)
```

```
done in 39.167s
Best estimator found by grid search:
SVC(C=10000.0, class_weight='balanced', gamma=0.005)
```

Gamma larger, C larger, the more complicated the model will be.

Performance Comparison

Model Performance — PCA

```
Epoch 24/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0156 - accuracy: 0.9973 - val_loss: 0.3124 - val_accu
racy: 0.9473
Epoch 25/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0156 - accuracy: 0.9955 - val_loss: 0.3463 - val_accu
racy: 0.9509
Epoch 26/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0108 - accuracy: 0.9986 - val_loss: 0.3520 - val_accu
racy: 0.9491
Epoch 27/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0075 - accuracy: 0.9991 - val_loss: 0.3927 - val_accu
racy: 0.9491
Epoch 28/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0057 - accuracy: 0.9995 - val_loss: 0.4136 - val_accu
racy: 0.9473
Epoch 29/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0054 - accuracy: 0.9991 - val_loss: 0.3721 - val_accu
racy: 0.9455
Epoch 30/30
69/69 [==============================] - 1s 12ms/step - loss: 0.0039 - accuracy: 0.9995 - val_loss: 0.3998 - val_accu
racy: 0.9455
```

	precision	recall	f1-score	support
0	0.96	0.95	0.96	438
1	0.73	0.79	0.76	76
accuracy			0.93	514
macro avg	0.85	0.87	0.86	514
weighted avg	0.93	0.93	0.93	514

Key: CNN outperforms the PCA in general speaking of the accuracy !

Conclusion and Possible Improvements

- During the period of the pandemic, doing a project around face mask detection is a rewarding experience for all of us. We used two models, convolutional neural network and principal component analysis with SVC, to detect the existence and condition of facial masks. Both models received sounding results for most of our experiments.
- If time permits, we could also extend this project to real time detection where users are tested for face masks in front of a webcam. A red frame will appear around the face if result is 1 (with mask). Otherwise, a green frame will appear.
- Another possible development is to integrate face mask detection with face recognition. That is, we could develop a strategy to detect faces correctly even when they are wearing masks. This project would have higher application value in this way.

Reference

Larxel. “Face Mask Detection.” Kaggle, 22 May 2020, https://www.kaggle.com/andrewmvd/face-mask-detection.

M. S. Ejaz, M. R. Islam, M. Sifatullah and A. Sarker, "Implementation of Principal Component Analysis on Masked and Non-masked Face Recognition," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019, pp. 1-5, doi: 10.1109/ICASERT.2019.8934543.

“RBF SVM Parameters”, Scikit Learn, https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

“Dealing with highly dimensional Data using Principal Component Analysis(PCA).” Isabella Lindgren, April 24, 2020. https://towardsdatascience.com/dealing-with-highly-dimensional-data-using-principal-component-analysis-pca-fea1ca817fe6