인공지능 프로그래밍 경진대회

컴퓨터정보공학과 김종휘, 박별이, 백종수

INDEX

개요 - 팀원과 인원 구성

20190706
김종휘

데이터 저장
전처리

20191797
박별이

딥러닝 모델

20190708
백종수

데이터와 훈련과정
예측 결과 시각화

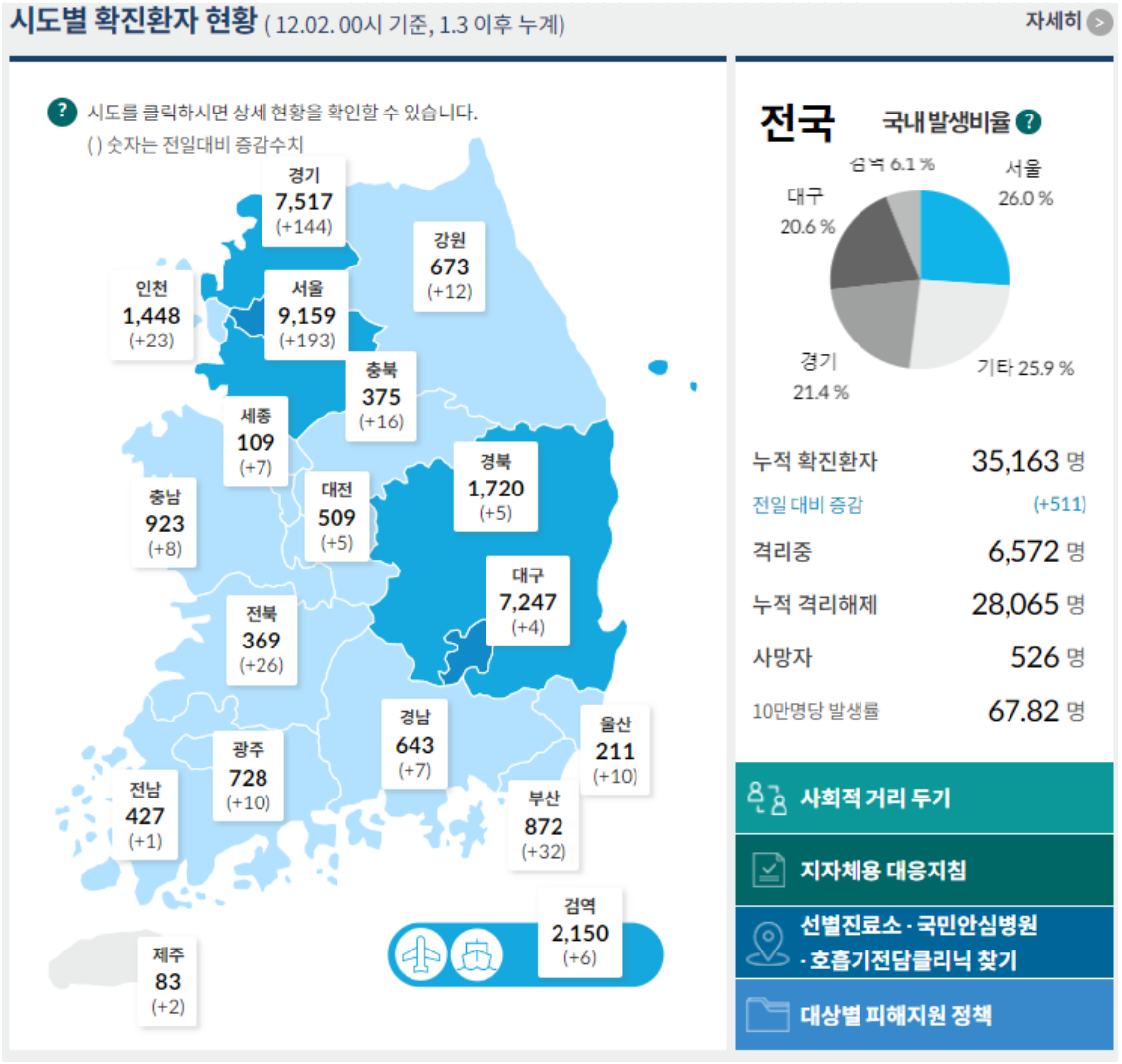
개요 - 딥러닝 목적과 분류

딥러닝 목적

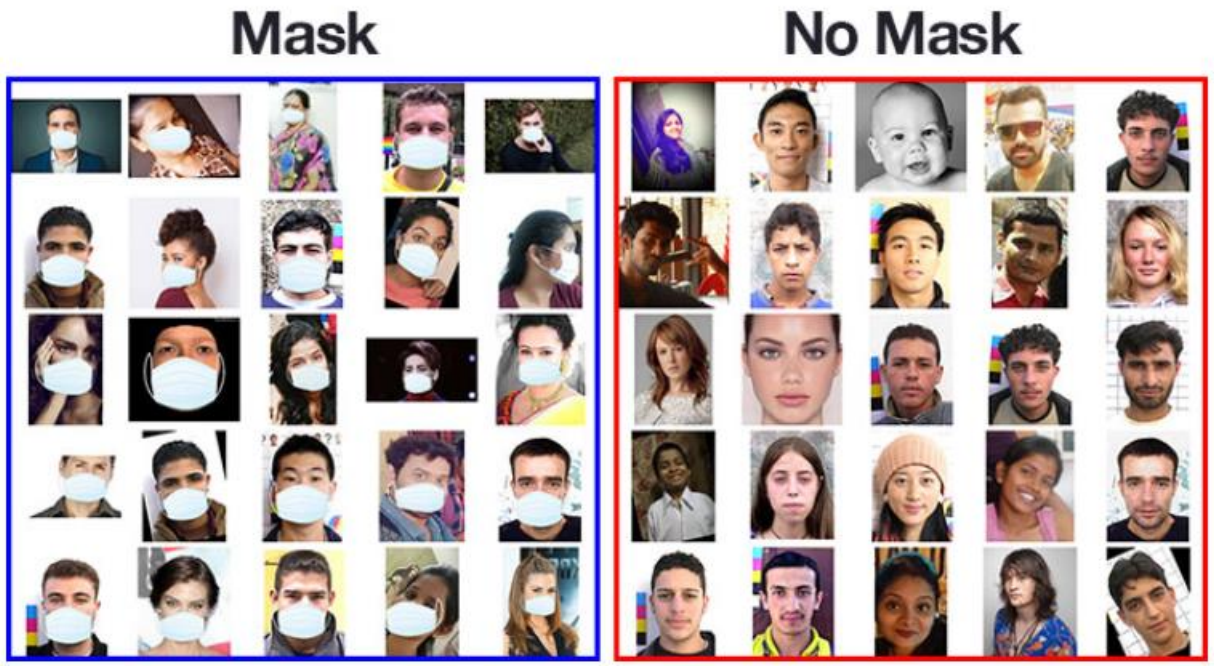
- 코로나 3차 대유행이란 문제점 확인
- 중앙재난안전대책본부의 방역 강화 표명
- 공공시설 마스크 착용 여부 딥러닝 AI 학습

모델 구분

- 불연속적인 값을 예측하는 분류 모델
- 마스크를 썼다(긍정 : positive)., 마스크를 쓰지 않았다(부정 : negative).를 분류하는 이진 분류(binary classification) 사용
- Ex) 마스크를 착용 했나요?



데이터저장과 전처리 - 데이터 수



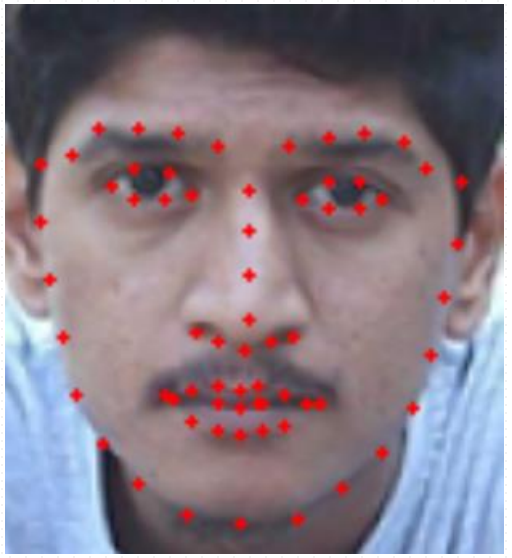
Prajna Bhandary의 데이터 세트는 두 클래스에 속하는 **1,376** 개의 **이미지** 로 구성됩니다.

with_mask : 690 이미지

without_mask : 686 이미지

데이터저장과 전처리 - 데이터 읽어 오기

데이터저장과 전처리 – 데이터 읽어 오기



데이터저장과 전처리 – 데이터 읽어 오기

데이터저장과 전처리 - 데이터 읽어 오기

데이터저장과 전처리 - 데이터 읽어 오기

Import

In [2]:
```
import os
```

In [4]:
```
import argparse
from imutils import paths
```

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
```

In [7]:
```
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

데이터저장과 전처리 - 데이터 읽어 오기

매개 변수

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
```

```
DATA_PATH = "dataset"
```

초기화

```
# 데이터셋 디렉토리에 있는 이미지 목록을 가져온 다음 초기화
# 데이터 목록(즉, 영상) 및 클래스 이미지
images imagePaths
= list(paths.list_images(DATA_PATH))
data = []
labels = []
```

데이터저장과 전처리 – 훈련, 테스트 데이터 분할, 원 핫 인코딩, 전처리

데이터 분할

```
# 80%를 사용하여 데이터를 교육 및 테스트 분할
# 교육용 데이터 및 나머지 20% 테스트용 데이터
(trainX, testX, trainY, testY)
= train_test_split( data, labels,
test_size=0.20,
stratify=labels,
random_state=42
)
```

원 핫 인코딩

```
#라벨에 원 핫 인코딩 수행
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
```

[1, 0]

[0, 1]

전처리

```
for imagePath in imagePaths:
label = imagePath.split(os.path.sep)[-2]
image = load_img(imagePath, target_size=(224, 224))
image = img_to_array(image) image
= preprocess_input(image)
data.append(image) labels.append(label)
```

딥러닝 모델 - 입력, 중간(은닉), 출력 층, 패러미터 수

```
# MobileNetV2 네트워크를 로드하여 헤드 FC 계층 세트가 멈춤
baseModel = MobileNetV2(
        weights="imagenet",
        include_top=False,
        input_tensor=Input(shape=(224, 224, 3))
)
```

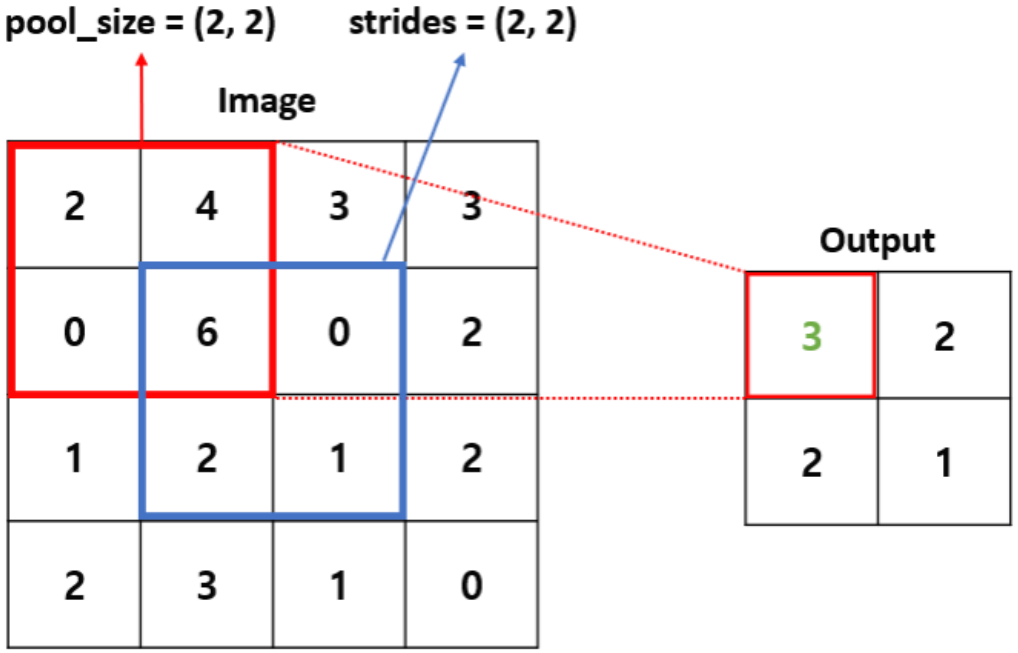
딥러닝 모델 – 입력, 중간(은닉), 출력 층, 패러미터 수

 # 기본 모델 구성

headModel = baseModel.output

 # AveragePooling2D: 평균 풀링 레이어 전체적인 평균 값을 사용하므로, 해당 이미지의 경향을 살린 채로 진행

headModel = AveragePooling2D(pool_size=(7, 7))(headModel)



딥러닝 모델 - 입력, 중간(은닉), 출력 층, 패러미터 수

```
headModel = Flatten(name="flatten")(headModel)
# 완전 연결층이 128개, 활성화 함수는 relu 사용 (0 or 음수 -> 0, 양수 -> x값)
headModel = Dense(128, activation="relu")(headModel)
# Dropout으로 훈련 중에 50%를 중간에 끊음, 예측 때는 모두 사용
headModel = Dropout(0.5)(headModel)
# 출력이 2개, softmax n개의 요소를 갖는 확률 벡터로 각 요소 값이 0 ~ 1 사이, 전체 합은 1 -> 확률값이 큰 것이 결과
headModel = Dense(2, activation="softmax")(headModel)
# Total params: 2,422,210
```

딥러닝 모델 – 입력, 중간(은닉), 출력 층, 패러미터 수

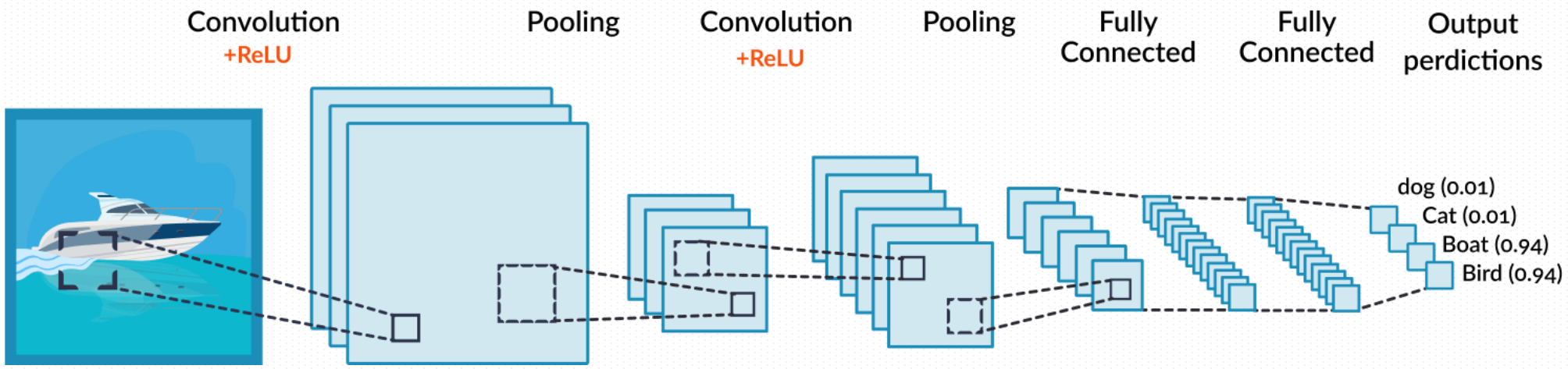
```
# 베이스 모델 위에 헤드 FC 모델을 배치 (실제 모델이 될 것)
model = Model(inputs=baseModel.input, outputs=headModel)



# 기본 모델의 모든 레이어를 학습하고 멈추기를 반복
# 첫 번째 교육 과정 동안 업데이트되지 않음
for layer in baseModel.layers:
        layer.trainable = False
```

딥러닝 모델 - 모델 종류(ANN, CNN, RNN 등)

CNN (합성곱신경망 : Convolution Neural Network)
-> 데이터의 특징을 추출하여 특징들의 패턴을 파악하는 구조

1. Convolution : 데이터의 특징을 추출하는 과정 (조사->특징 파악->압축)
2. Pooling : 위 과정 후 레이어의 사이즈 줄임, 노이즈 상쇄, 특징 제공

ex) 정보추출, 문장분류, 얼굴인식 등



딥러닝 모델 – 옵티마이저, 손실함수

```
# 모델 컴파일
# 옵티마이저 = Adam (0으로 편향된 것을 보정)
    lr = 초기 학습률, decay = 초기 학습률 / 훈련 수
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

# 손실함수 = binary_crossentropy (카테고리가 2개인 경우)
# metrics 훈련과 테스트 과정을 모니터링할 지표 (여기에서는 정확도만 고려)
model.compile(optimizer=opt, loss="binary_crossentropy",
metrics=["accuracy"])
```

딥러닝 모델 - Early Stopping – Callback 함수

사용 안 함

딥러닝 모델 - **모델 요약(summary)**

model.summary()

-> Total params: 2,422,210

데이터 훈련과정

훈련

```
# train the head of the network
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS
)
```

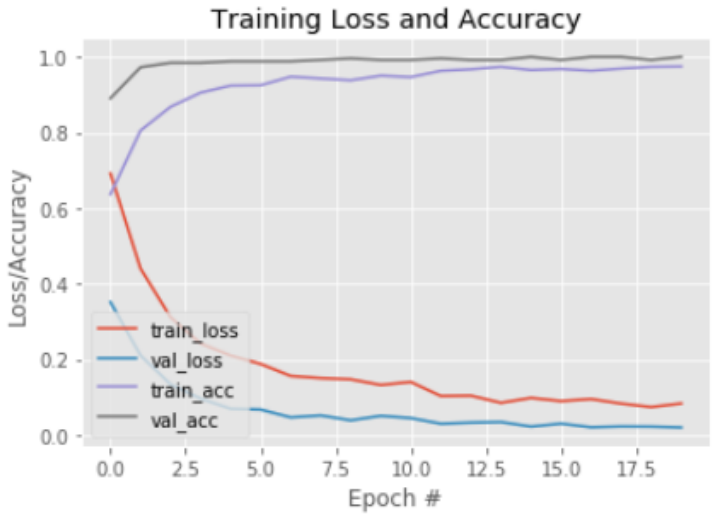
데이터 훈련과정

훈련

```
[INFO] training head...
Train for 34 steps, validate on 276 samples
Epoch 1/20
34/34 [==============================] - 139s 4s/step - loss: 0.6867 - accuracy: 0.6367 - val_loss: 0.3510 - val_accuracy: 0.8906
Epoch 2/20
34/34 [==============================] - 121s 4s/step - loss: 0.4376 - accuracy: 0.8052 - val_loss: 0.2101 - val_accuracy: 0.9727
Epoch 3/20
34/34 [==============================] - 109s 3s/step - loss: 0.3103 - accuracy: 0.8680 - val_loss: 0.1337 - val_accuracy: 0.9844
Epoch 4/20
34/34 [==============================] - 135s 4s/step - loss: 0.2418 - accuracy: 0.9053 - val_loss: 0.0959 - val_accuracy: 0.9844
Epoch 5/20
34/34 [==============================] - 109s 3s/step - loss: 0.2110 - accuracy: 0.9242 - val_loss: 0.0691 - val_accuracy: 0.9883
Epoch 6/20
34/34 [==============================] - 110s 3s/step - loss: 0.1924 - accuracy: 0.9251 - val_loss: 0.0669 - val_accuracy: 0.9883
Epoch 7/20
34/34 [==============================] - 109s 3s/step - loss: 0.1563 - accuracy: 0.9476 - val_loss: 0.0462 - val_accuracy: 0.9883
Epoch 8/20
34/34 [==============================] - 102s 3s/step - loss: 0.1509 - accuracy: 0.9429 - val_loss: 0.0513 - val_accuracy: 0.9922
Epoch 9/20
34/34 [==============================] - 104s 3s/step - loss: 0.1452 - accuracy: 0.9382 - val_loss: 0.0385 - val_accuracy: 0.9961
Epoch 10/20
34/34 [==============================] - 104s 3s/step - loss: 0.1307 - accuracy: 0.9504 - val_loss: 0.0502 - val_accuracy: 0.9922
Epoch 11/20
34/34 [==============================] - 100s 3s/step - loss: 0.1397 - accuracy: 0.9466 - val_loss: 0.0443 - val_accuracy: 0.9922
Epoch 12/20
34/34 [==============================] - 100s 3s/step - loss: 0.1026 - accuracy: 0.9632 - val_loss: 0.0295 - val_accuracy: 0.9961
Epoch 13/20
34/34 [==============================] - 96s 3s/step - loss: 0.1030 - accuracy: 0.9672 - val_loss: 0.0324 - val_accuracy: 0.9922
Epoch 14/20
34/34 [==============================] - 97s 3s/step - loss: 0.0861 - accuracy: 0.9738 - val_loss: 0.0339 - val_accuracy: 0.9922
Epoch 15/20
34/34 [==============================] - 97s 3s/step - loss: 0.0973 - accuracy: 0.9654 - val_loss: 0.0219 - val_accuracy: 1.0000
Epoch 16/20
34/34 [==============================] - 97s 3s/step - loss: 0.0884 - accuracy: 0.9682 - val_loss: 0.0297 - val_accuracy: 0.9922
Epoch 17/20
34/34 [==============================] - 2250s 66s/step - loss: 0.0947 - accuracy: 0.9635 - val_loss: 0.0201 - val_accuracy: 1.0000
Epoch 18/20
34/34 [==============================] - 1899s 56s/step - loss: 0.0840 - accuracy: 0.9691 - val_loss: 0.0222 - val_accuracy: 1.0000
Epoch 19/20
34/34 [==============================] - 159s 5s/step - loss: 0.0734 - accuracy: 0.9738 - val_loss: 0.0218 - val_accuracy: 0.9922
Epoch 20/20
34/34 [==============================] - 118s 3s/step - loss: 0.0816 - accuracy: 0.9747 - val_loss: 0.0197 - val_accuracy: 1.0000
```

데이터 시각화 자료

```
# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```



예측

```python
def load_image(img_path):
    """
    load the input image from disk, clone it,
    and grab the image spatial dimensions
    """
    image = imread(img_path)
    (h, w) = image.shape[:2]
    return image, h, w
```

```python
def blob_from_image(image_file):
    """
    construct a blob from the image
    """
    blob = cv2.dnn.blobFromImage(image_file, 1.0, (300, 300), (104.0, 177.0, 123.0))
    return blob
```

```python
def prediction_preparation(blob):
    """
    pass the blob through the network and obtain the face detections
    """
    net.setInput(blob)
    detections = net.forward()
    return detections
```

예측

```
def predict(detections, image, h, w):

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the confidence is
        # greater than the minimum confidence
        if confidence > 0.5:
            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = image[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)
            face = np.expand_dims(face, axis=0)

            # pass the face through the model to determine if the face
            # has a mask or not
            (mask, withoutMask) = model.predict(face)[0]

            # determine the class label and color we'll use to draw
            # the bounding box and text
            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (255, 0, 0)

            # include the probability in the label
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

            # display the label and bounding box rectangle on the output
            # frame
            cv2.putText(image, label, (startX, startY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 1.5, color, 2)
            cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
    return image
```

예측

```
def predict_it(image_path):
    image, h,w = load_image(img_path=image_path)
    blob = blob_from_image(image_file=image)
    detections = prediction_preparation(blob=blob)
    image = predict(detections=detections, image=image, h=h, w=w)
    plt.figure(figsize=(10,10))
    plt.imshow(image)
    plt.axis('off')
    plt.show()
```

마스크

```
predict_it(image_path="examples/mask.jpg")
```

노 마스크

```
predict_it(image_path="examples/notmask.jpg")
```

예측