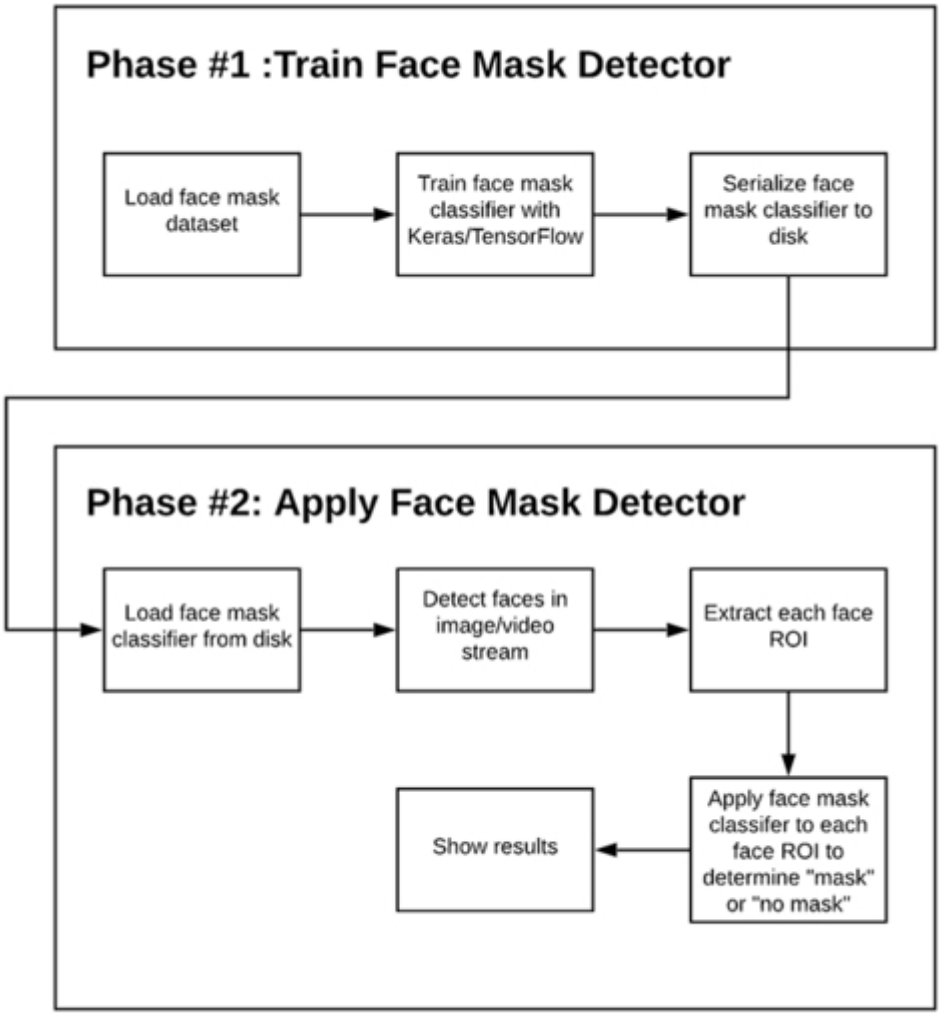
Deteksi Masker Wajah Menggunakan TensorFlow dalam Bahasa Python

Dalam artikel ini, kami akan membahas detektor masker wajah COVID-19 berbasis komputer vision/deep learning kami dalam dua tahap, menjelaskan bagaimana pipa pemrosesan vision komputer/deep learning kami akan diimplementasikan.

Kami akan menggunakan skrip Python ini untuk melatih detektor masker wajah dan meninjau hasilnya. Dengan detektor masker wajah COVID-19 yang telah dilatih, kami akan melanjutkan untuk mengimplementasikan dua skrip Python tambahan yang digunakan untuk:

1. Mendeteksi masker wajah COVID-19 dalam gambar
2. Mendeteksi masker wajah dalam aliran video waktu nyata

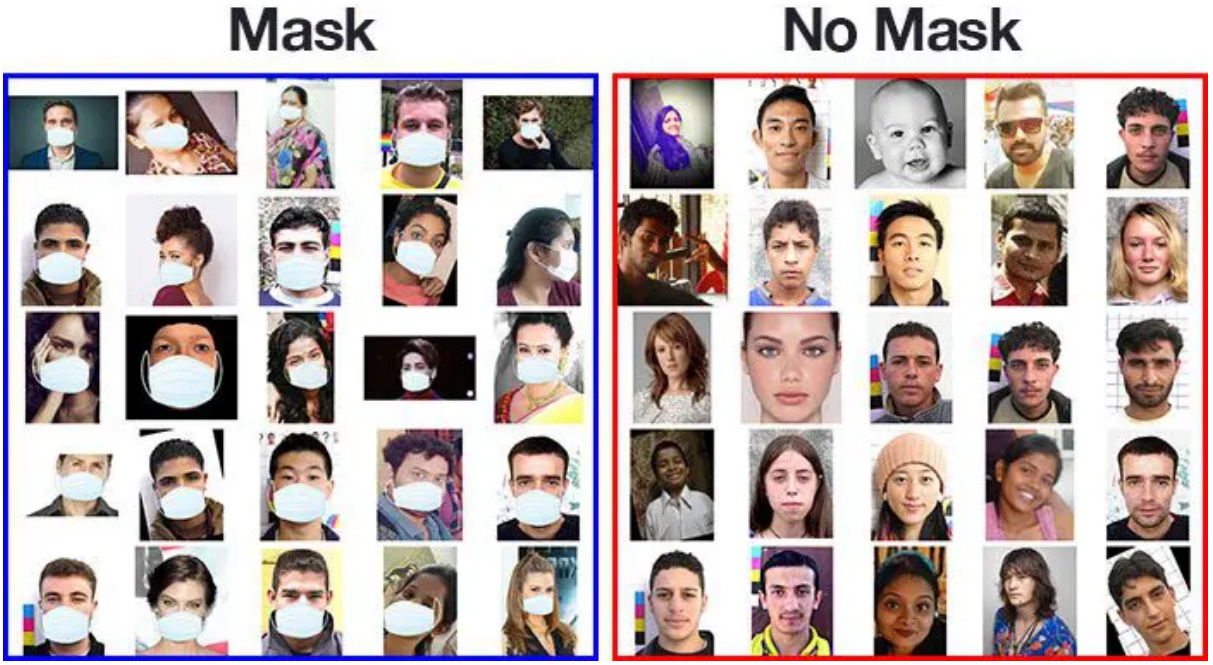
Diagram Alir Sistem Deteksi Masker Wajah



Untuk melatih detektor masker wajah kustom, kita perlu membagi proyek ini menjadi dua tahap yang berbeda, masing-masing dengan langkah-langkah sub-resmi (seperti yang

ditunjukkan oleh Gambar 1 di atas):

1. Pelatihan: Di sini kita akan fokus pada memuat dataset deteksi masker wajah kita dari disk, melatih model (menggunakan Keras/TensorFlow) pada dataset ini, dan kemudian menyerialkan detektor masker wajah ke disk.
2. Implementasi: Setelah detektor masker wajah terlatih, kita dapat melanjutkan dengan memuat detektor masker, melakukan deteksi wajah, dan kemudian mengklasifikasikan setiap wajah sebagai with_mask atau without_mask.



Kami akan menggunakan gambar-gambar ini untuk membangun model CNN menggunakan TensorFlow untuk mendeteksi apakah Anda mengenakan masker wajah dengan menggunakan webcam PC Anda. Selain itu, Anda juga dapat menggunakan kamera ponsel Anda untuk melakukan hal yang sama!

Langkah-demi-Langkah Implementasi

Langkah 1: Visualisasi Data

Pada langkah pertama, mari kita visualisasikan jumlah total gambar dalam dataset kita dalam kedua kategori. Kita dapat melihat bahwa ada 690 gambar dalam kelas 'yes' dan 686 gambar dalam kelas 'no'.

Jumlah gambar dengan masker wajah berlabel 'yes': 690

Jumlah gambar tanpa masker wajah berlabel 'no': 686

Langkah 2: Augmentasi Data

Pada langkah berikutnya, kita melakukan augmentasi dataset kita untuk memasukkan lebih banyak gambar dalam pelatihan kita. Pada langkah augmentasi data ini, kita memutar dan

membalikkan setiap gambar dalam dataset kita. Setelah augmentasi data, kita memiliki total 2751 gambar dengan 1380 gambar dalam kelas 'yes' dan 1371 gambar dalam kelas 'no'.
Jumlah contoh: 2751
Persentase contoh positif: 50,163576881134134%, jumlah contoh positif: 1380
Persentase contoh negatif: 49,836423118865866%, jumlah contoh negatif: 1371

Langkah 3: Pembagian Data

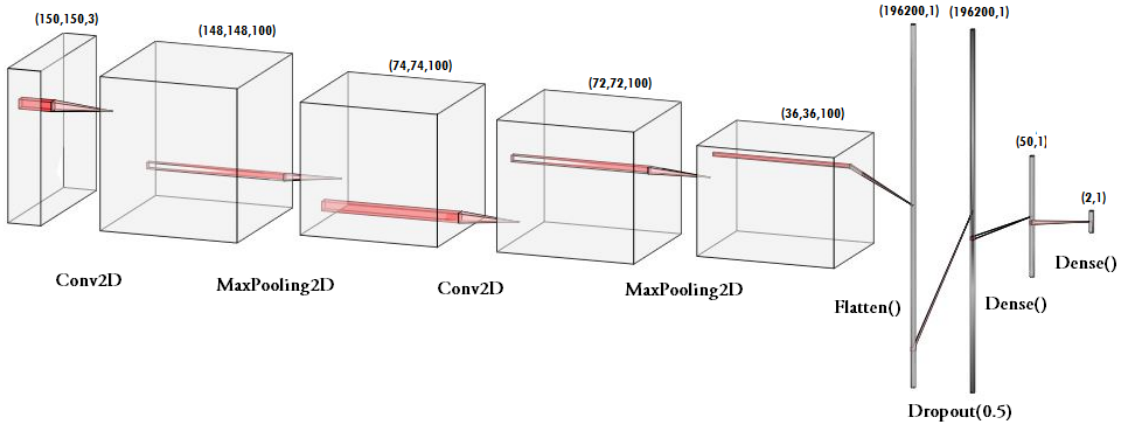
Pada langkah ini, kita membagi data kita menjadi set pelatihan yang akan berisi gambar-gambar yang akan dilatih oleh model CNN dan set pengujian dengan gambar-gambar di mana model kita akan diuji. Pada langkah ini, kita mengambil split_size = 0.8, yang berarti 80% dari total gambar akan masuk ke dalam set pelatihan dan 20% sisanya akan masuk ke dalam set pengujian.
Jumlah gambar dengan masker wajah dalam set pelatihan berlabel 'yes': 1104
Jumlah gambar dengan masker wajah dalam set pengujian berlabel 'yes': 276
Jumlah gambar tanpa masker wajah dalam set pelatihan berlabel 'no': 1096
Jumlah gambar tanpa masker wajah dalam set pengujian berlabel 'no': 275
Setelah pembagian, kita melihat bahwa persentase gambar yang diinginkan telah didistribusikan ke dalam set pelatihan dan set pengujian seperti yang disebutkan di atas.

Langkah 4: Membangun Model

Pada langkah berikutnya, kita membangun model CNN Sequential kita dengan berbagai lapisan seperti Conv2D, MaxPooling2D, Flatten, Dropout, dan Dense. Pada lapisan Dense terakhir, kita menggunakan fungsi 'softmax' untuk mengeluarkan vektor yang memberikan probabilitas masing-masing dari dua kelas.


```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(100, (3, 3), activation='relu',
                           input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(100, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['acc'])
```

Di sini, kami menggunakan pengoptimal 'adam' dan 'binary_crossentropy' sebagai fungsi kerugian kami karena hanya ada dua kelas. Selain itu, Anda bahkan dapat menggunakan MobileNetV2 untuk akurasi yang lebih baik.



Langkah 5: Pra-Pelatihan Model CNN

Setelah membangun model kami, mari buat 'train_generator' dan 'validation_generator' untuk menyesuaikannya dengan model kami pada langkah berikutnya. Kami melihat bahwa ada total 2200 gambar dalam set pelatihan dan 551 gambar dalam set pengujian.

Ditemukan 2200 gambar yang termasuk dalam 2 kelas.

Ditemukan 551 gambar yang termasuk dalam 2 kelas.

Langkah 6: Pelatihan Model CNN

Langkah ini adalah langkah utama di mana kami menyesuaikan gambar-gambar dalam set pelatihan dan set pengujian ke model Sequential yang kami bangun menggunakan perpustakaan keras. Saya telah melatih model selama 30 epoch (iterasi). Namun, kita dapat melatih untuk lebih banyak jumlah epoch untuk mencapai akurasi yang lebih tinggi tetapi perhatikan kemungkinan overfitting.

```
history = model.fit_generator(train_generator,
                              epochs=30,
                              validation_data=validation_generator,
                              callbacks=[checkpoint])
```

Kami melihat, bahwa setelah epoch ke-30, model kami memiliki akurasi 98,86% dengan set pelatihan dan akurasi 96,19% dengan set pengujian. Ini menunjukkan bahwa model ini telah terlatih dengan baik tanpa adanya overfitting.

Langkah 7: Pelabelan Informasi

Setelah membangun model, kami memberi label dua probabilitas untuk hasil kami. ['0' sebagai 'without_mask' dan '1' sebagai 'with_mask']. Kami juga mengatur warna kotak batas menggunakan nilai-nilai RGB. ['RED' untuk 'without_mask' dan 'GREEN' untuk 'with_mask']

labels_dict={0:'without_mask',1:'with_mask'}

color_dict={0:(0,0,255),1:(0,255,0)}

Langkah 8: Mengimpor Program Deteksi Wajah

Setelah ini, kami bermaksud menggunakannya untuk mendeteksi apakah kami mengenakan masker wajah menggunakan webcam PC kami. Untuk ini, pertama-tama, kami perlu mengimplementasikan deteksi wajah. Dalam hal ini, kami menggunakan Haar Feature-based Cascade Classifiers untuk mendeteksi fitur wajah.

face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

Klasifier kaskade ini dirancang oleh OpenCV untuk mendeteksi wajah frontal dengan melatih ribuan gambar. File .xml untuk itu perlu diunduh dan digunakan dalam mendeteksi wajah. Kami telah mengunggah file tersebut ke repositori GitHub.

Langkah 9: Mendeteksi Wajah dengan dan tanpa Masker

Pada langkah terakhir, kami menggunakan perpustakaan OpenCV untuk menjalankan loop tak terbatas untuk menggunakan kamera web kami di mana kami mendeteksi wajah menggunakan Cascade Classifier. Kode **`webcam = cv2.VideoCapture(0)`** menunjukkan penggunaan webcam.

Model akan memprediksi kemungkinan masing-masing dari dua kelas ([without_mask, with_mask]). Berdasarkan probabilitas yang lebih tinggi, label akan dipilih dan ditampilkan di sekitar wajah kami.

main.py

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os
```

```
def detect_and_predict_mask(frame, faceNet, maskNet):

	# grab the dimensions of the frame and
	# then construct a blob from it
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
	                                        (104.0, 177.0, 123.0))

	# pass the blob through the network
	# and obtain the face detections
	faceNet.setInput(blob)
	detections = faceNet.forward()
	print(detections.shape)

	# initialize our list of faces, their
	# corresponding locations, and the list
	# of predictions from our face mask network
	faces = []
	locs = []
	preds = []

	# 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 = frame[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)

                # add the face and bounding boxes
                # to their respective lists
                faces.append(face)
                locs.append((startX, startY, endX, endY))

        # only make a predictions if at least one
        # face was detected
        if len(faces) > 0:

                # for faster inference we'll make
                # batch predictions on *all*
                # faces at the same time rather
                # than one-by-one predictions
                # in the above `for` loop
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

        # return a 2-tuple of the face locations
        # and their corresponding locations
        return (locs, preds)


# load our serialized face detector model from disk
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model("mask_detector.model")
```

```
# initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
	# grab the frame from the threaded
	# video stream and resize it
	# to have a maximum width of 400 pixels
	frame = vs.read()
	frame = imutils.resize(frame, width=400)

	# detect faces in the frame and
	# determine if they are wearing a
	# face mask or not
	(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

	# loop over the detected face
	# locations and their corresponding
	# locations
	for (box, pred) in zip(locs, preds):

		# unpack the bounding box and predictions
		(startX, startY, endX, endY) = box
		(mask, withoutMask) = pred

		# 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 (0, 0, 255)

		# 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(frame, label, (startX, startY - 10),
				cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
		cv2.rectangle(frame, (startX, startY), (endX, endY), color,
2)

	# show the output frame
```

```
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
              break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Output:

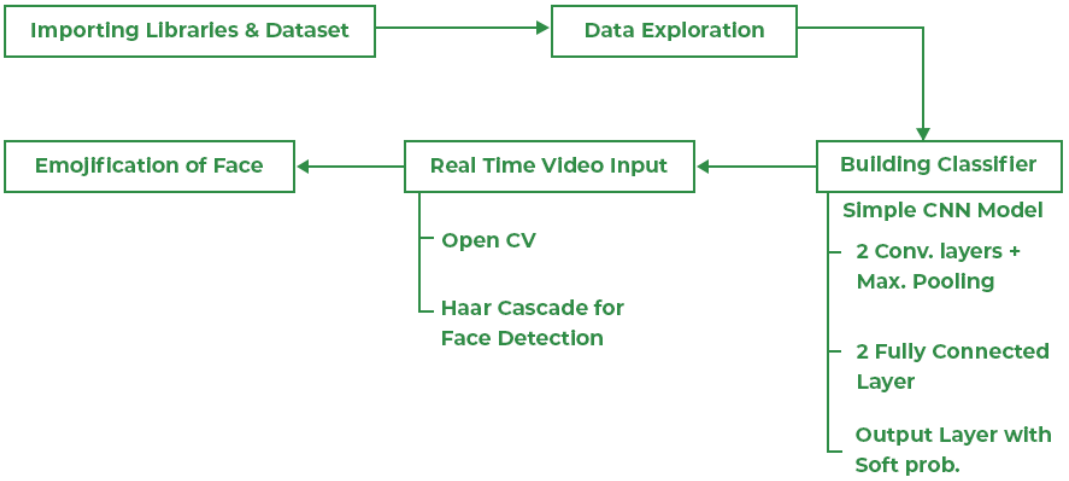
Klik link

Emojify Menggunakan Pengenalan Wajah dengan Machine Learning

Dalam artikel ini, kita akan mempelajari cara mengimplementasikan aplikasi modifikasi yang akan menampilkan emoji ekspresi yang menyerupai ekspresi wajah Anda. Ini adalah proyek yang menyenangkan berdasarkan computer vision di mana kita menggunakan model klasifikasi gambar untuk mengklasifikasikan berbagai ekspresi seseorang.

Proyek ini akan diimplementasikan dalam dua bagian:

1. Membangun model klasifikasi gambar yang dapat mengklasifikasikan gambar wajah dengan ekspresi yang berbeda.
2. Mengekstrak wajah dari gambar dan kemudian mengklasifikasikan ekspresinya menggunakan klasifikasi.



Aliran Data Proyek

Modul yang Digunakan

Pustaka Python memudahkan kita untuk mengelola data dan melakukan tugas-tugas khas dan kompleks dengan satu baris kode.

- **Pandas**: Pustaka ini membantu memuat kerangka data dalam format array 2D dan memiliki beberapa fungsi untuk melakukan tugas analisis dalam satu langkah.
- **Numpy**: Array Numpy sangat cepat dan dapat melakukan perhitungan besar dalam waktu singkat.
- **Matplotlib**: Pustaka ini digunakan untuk menggambar visualisasi.
- **Sklearn**: Modul ini berisi beberapa pustaka dengan fungsi-fungsi yang sudah diimplementasikan untuk melakukan tugas dari preprocessing data hingga pengembangan dan evaluasi model.

- **OpenCV**: Ini adalah pustaka open-source yang terutama fokus pada pengolahan dan penanganan gambar.
- **TensorFlow**: Ini adalah pustaka open-source yang digunakan untuk Machine Learning dan Artificial Intelligence dan menyediakan berbagai fungsi untuk mencapai fungsionalitas kompleks dengan satu baris kode.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from PIL import Image

from sklearn import metrics

import cv2
import os

from glob import glob

import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.preprocessing.image import ImageDataGenerator

import warnings
warnings.filterwarnings('ignore')
```

Mengimpor Dataset

Dataset yang akan kita gunakan berisi sekitar 30.000 gambar untuk tujuh kategori ekspresi yang berbeda. Meskipun gambar-gambar tersebut berukuran sangat kecil (48, 48) dalam format RGB. Karena gambar-gambar tersebut terlalu kecil untuk aplikasi dunia nyata. Selain itu, model kita akan menghadapi kesulitan dalam mempelajari pola ekspresi yang berbeda pada wajah manusia.

```
# Mengekstrak dataset yang dikompresi.
from zipfile import ZipFile
data_path = 'fer2013.zip'

with ZipFile(data_path,'r') as zip:
  zip.extractall()
  print('Dataset telah diekstrak.')
#output
#Dataset telah diekstrak.
```

Eksplorasi Data

Pada bagian ini, kita akan mencoba menjelajahi data dengan memvisualisasikan jumlah gambar yang disediakan untuk setiap kategori pada data pelatihan dan pengujian.

```
path = 'train'
classes = os.listdir(path)
classes
```

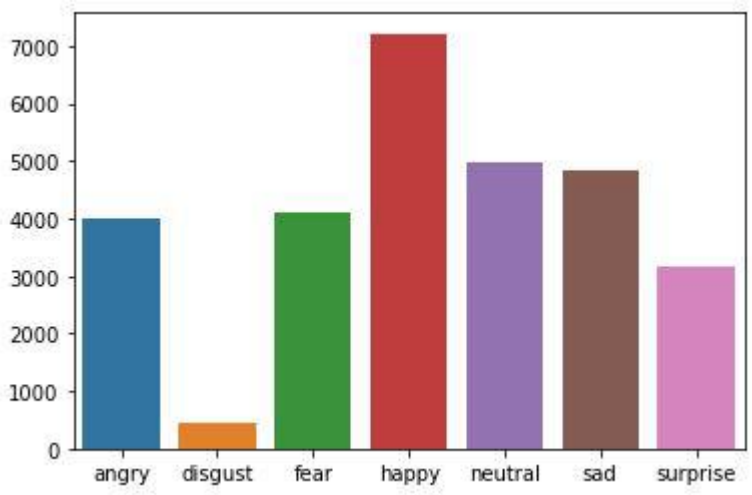
Output:

```
['angry', 'disgust', 'fear', 'happy', 'neutral',
'sad', 'surprise']
```

Ini adalah tujuh kelas ekspresi yang kita miliki di sini.

```
count = []
for cat in classes:
    count.append(len(os.listdir(f'{path}/{cat}')))
sb.barplot(classes, count)
plt.show()
```

Output:



Salah satu pengamatan penting yang bisa kita ambil di sini adalah ketidakseimbangan data yang tinggi dalam kategori "disgust". Karena alasan ini, mungkin model kita tidak akan berperforma dengan baik pada kelas gambar ini.

Pengembangan Model

Dari langkah ini ke depan, kita akan menggunakan pustaka TensorFlow untuk membangun model CNN kita. Kerangka kerja Keras dari pustaka tensorflow berisi semua fungsionalitas yang mungkin dibutuhkan seseorang untuk mendefinisikan arsitektur Convolutional Neural

Network dan melatihnya dengan data.

Arsitektur Model

Kita akan mengimplementasikan model Sequential yang akan berisi bagian-bagian berikut:

- Tiga Lapisan Konvolusi diikuti oleh Lapisan MaxPooling.
- Lapisan Flatten untuk meratakan keluaran lapisan konvolusi.
- Kemudian kita akan memiliki dua lapisan terhubung penuh diikuti oleh keluaran dari lapisan yang diratakan.
- Kami telah menyertakan beberapa lapisan BatchNormalization untuk memungkinkan pelatihan yang stabil dan cepat, serta lapisan Dropout sebelum lapisan terakhir untuk menghindari kemungkinan overfitting.
- Lapisan terakhir adalah lapisan output yang mengeluarkan probabilitas soft untuk tujuh kelas.

```
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
        'train',
        target_size=(48,48),
        batch_size=64,
        color_mode="grayscale",
        class_mode='categorical')

val_gen = val_datagen.flow_from_directory(
        'test',
        target_size=(48,48),
        batch_size=64,
        color_mode="grayscale",
        class_mode='categorical')
```

Output:

Output:

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

Mari simpan label dan yang ditugaskan ke berbagai kelas ekspresi.

```
emotions = list(train_gen.class_indices.keys())
```

Mari tentukan arsitektur model.

```
model = keras.models.Sequential([
    layers.Conv2D(32,(3,3),activation='relu',input_shape=(48, 48, 1)),
    layers.Conv2D(64,(3,3),activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(32,activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(7, activation='softmax')
])
```

Saat mengompilasi model, kita menyediakan tiga parameter penting:

optimizer: Ini adalah metode yang membantu mengoptimalkan fungsi biaya dengan menggunakan gradien descent.

loss: Fungsi kerugian dengan mana kita memantau apakah model ini memperbaiki performa dengan pelatihan atau tidak.

metrics: Ini membantu mengevaluasi model dengan memprediksi data pelatihan dan validasi.

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


Callback

Callback digunakan untuk memeriksa apakah model ini memperbaiki performa dengan setiap epoch atau tidak. Jika tidak, maka langkah-langkah yang diperlukan akan diambil, seperti ReduceLROnPlateau yang mengurangi tingkat pembelajaran lebih lanjut. Bahkan jika kinerja model tidak meningkat, pelatihan akan dihentikan oleh EarlyStopping. Kita juga dapat menentukan beberapa callback kustom untuk menghentikan pelatihan jika hasil yang diinginkan telah diperoleh lebih awal.

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('val_accuracy') > 0.90:
            print('\n Akurasi validasi telah mencapai 90%, sehingga
pelatihan dihentikan lebih lanjut.')
```

```
            self.model.stop_training = True

es = EarlyStopping(patience=3, monitor='val_accuracy',
restore_best_weights=True)
lr = ReduceLROnPlateau(monitor='val_loss', patience=2, factor=0.5,
verbose=1)
```

Sekarang kita akan melatih model kita:

```
history = model.fit(train_gen,
                    validation_data=val_gen,
                    epochs=5,
                    verbose=1,
                    callbacks=[es, lr, myCallback()])
```

Output:

```
Epoch 1/5
449/449 [==============================] - 732s 2s/step - loss: 1.9990 - accuracy: 0.2634 - val_loss: 1.7677 - val_accuracy: 0.
3406 - lr: 1.0000e-04
Epoch 2/5
449/449 [==============================] - 545s 1s/step - loss: 1.6626 - accuracy: 0.3887 - val_loss: 1.6183 - val_accuracy: 0.
3958 - lr: 1.0000e-04
Epoch 3/5
449/449 [==============================] - 345s 769ms/step - loss: 1.4614 - accuracy: 0.4610 - val_loss: 1.4016 - val_accuracy:
0.4806 - lr: 1.0000e-04
Epoch 4/5
449/449 [==============================] - 269s 600ms/step - loss: 1.3122 - accuracy: 0.5170 - val_loss: 1.4699 - val_accuracy:
0.4411 - lr: 1.0000e-04
Epoch 5/5
449/449 [==============================] - ETA: 0s - loss: 1.1924 - accuracy: 0.5626
Epoch 5: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
449/449 [==============================] - 238s 530ms/step - loss: 1.1924 - accuracy: 0.5626 - val_loss: 1.4294 - val_accuracy:
0.4928 - lr: 1.0000e-04
```

Bagian pertama dari proyek kita adalah membangun klasifikasi yang dapat mengklasifikasikan berbagai ekspresi.

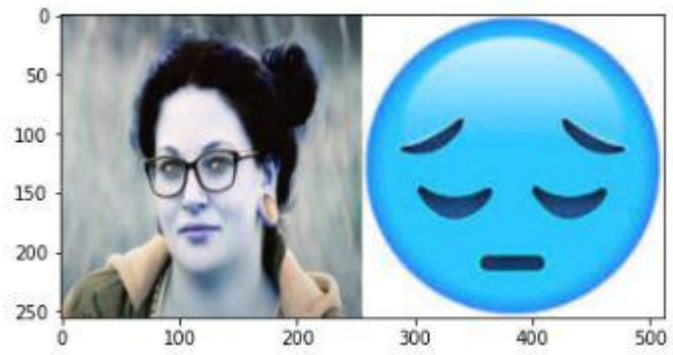
Memprediksi Emoji secara Real-Time

Ini adalah bagian kedua dari proyek ini di mana kita akan memprediksi ekspresi pada wajah seseorang dan menampilkan emoji berdasarkan ekspresi tersebut secara real-time. Untuk mendeteksi wajah dalam aliran video, kita akan menggunakan haar_cascade_classifier. Berikut adalah fungsi pembantu yang akan digunakan untuk memplot gambar.

```
def plot_image(img, emoj):
    wmin = 256
    hmin = 256

    emoj = cv2.resize(emoj, (wmin, hmin))
    img = cv2.resize(img, (wmin, hmin))
    cv2.imshow('Images', cv2.hconcat([img, emoj]))
```

Sekarang kita akan menangkap video dan dengan menggunakan haar_cascade_classifier kita akan memprediksi emoji untuk ekspresi tersebut.

```
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

img = cv2.imread('sad.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray)

for (x, y, w, h) in faces:
    gray = cv2.resize(gray[x:x+w-10,y:y+h+10], (48,48))
    gray = np.expand_dims(gray, axis=-1)
    gray = np.expand_dims(gray, axis=0)

    pred = model.predict(gray)
    idx = pred.argmax(axis=-1)[0]

    emoj = cv2.imread(f'emojis/{classes[idx]}.jpg')

    plot_image(img, emoj)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```



Berikut adalah kode untuk video:

```
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)

while(True):
    ret, img = cap.read()
```

```
        img = cv2.resize(img, (256, 256))
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray)

        if len(faces) > 0:
            for (x, y, w, h) in faces:
                try:
                    gray = cv2.resize(gray[x:x+w,y:y+h], (48,48))
                except:
                    break

                gray = np.expand_dims(gray, axis=-1)
                gray = np.expand_dims(gray, axis=0)

                pred = model.predict(gray)
                idx = pred.argmax(axis=-1)[0]

                emoj = cv2.imread(f'emojis/{emotions[idx]}.jpg')

                plot_image(img, emoj)
        else:
            emoj = cv2.imread('NofaceDetected.jpeg')
            plot_image(img, emoj)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```


Kesimpulan:

Mungkin ada beberapa ketidaksesuaian dalam hasil karena klasifikator yang kita bangun hanya dapat memprediksi emosi dengan akurasi 55%. Jika kita menggunakan dataset yang lebih baik dalam hal ukuran, maka akurasi model akan meningkat dan demikian pula emoji yang diprediksi oleh model. Yang ditunjukkan di atas adalah kasus penggunaan klasifikator untuk gambar statis. Cukup gantikan blok kode terakhir dengan kode berikut dan Anda akan dapat memprediksi emoji secara real-time untuk gambar Anda.