دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی اول یادگیری عمیق

Smile Recognition

پدیدآورنده:

محمد امین کیانی

4003613052

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد راهنما: جناب اقای دکتر کیانی

نیمسال اول تحصیلی 04-1403

فهرست مطالب

مستندات

1-مسئله و تحلیل کلی آن:

شناسایی لبخند از روی ویدیو یک مسأله مهم در حوزه بینایی کامپیوتری و یادگیری عمیق است. این پروژه می‌تواند در حوزه‌های مختلفی مانند امنیت، روان‌شناسی، و تعامل انسان و ماشین کاربرد داشته باشد. در اینجا ما می‌خواهیم با استفاده از تکنیک‌های یادگیری عمیق و اموزش لبخند و صورت برای مدل، لبخند و غیر لبخند را در ویدیوها شناسایی کنیم.

- هدف: شناسایی زمان لبخند زدن و نزدن افراد در ویدیوهای ورودی.

- ورودی: ویدیوهای حاوی چهره‌های افراد(مِثلا خودم).

- خروجی: تعیین کادر تشخیص و احتمال لبخند زدن در طول ویدیو.

می‌توان از معماری های مختلفی استفاده کرد. در اینجا، ما از چند معماری رایج برای این منظور استفاده خواهیم کرد تا نتیجه بگیریم کدام بهتر است:

* *EfficientNet-B7 برای* دسته‌بندی ← 77% _ 54% بسته به B?

* *MobileNet* برای دسته‌بندی ← 64% تا نسخه‌ی v2 73%

* *VGG16* برای دسته‌بندی ← 84%

ابتدا، باید داده‌ها را از سایت اصلی دانلود و پیش‌پردازش و ترنسفرلرن کنیم، سپس مدل‌های مورد نظر را آموزش دهیم و در نهایت، نتایج را ارزیابی کنیم.

2-ورود دیتاست و کتابخانه های مورد نیاز پروژه:

دیتاست Genki4K یک مجموعه داده بزرگ از تصاویر چهره است که بر روی شناسایی و تحلیل احساسات تمرکز دارد. این دیتاست شامل ۴۰۰۰ تصویر از چهره‌هایی است که در شرایط مختلف احساسی (شادی، خشم، ترس و...) ضبط شده‌اند.پس یک ابزار مفید برای تحقیقات در زمینه پردازش تصویر و یادگیری عمیق است.

ویژگی‌ها:

1. تنوع احساسی: این دیتاست احساسات مختلف را پوشش می‌دهد که می‌تواند به شناسایی و تجزیه و تحلیل دقیق‌تری کمک کند.

2. کیفیت بالا: تصاویر با کیفیت بالا (K۴) ضبط شده‌اند که برای مدل‌های یادگیری عمیق بسیار مناسب است.

3. ارتباطات اجتماعی: این دیتاست به محققان و توسعه‌دهندگان کمک می‌کند تا نرم‌افزارهایی بسازند که بتوانند تعاملات انسانی را بهتر شبیه‌سازی کنند.

```
Original file is located at
    https://colab.research.google.com/drive/1_omEumLTFO5MiZY1GM-SJvJxn_tHtGQi

# Smile Classification :)

#### Mohammad Amin Kiani 4003613052
###### Uni of Isfahan - Iran
###### Deep Learning - Dr.Kiani 1404-1403

------------------------------------------------------------------

## Requirements

#### Genki4K DataSet
"""

# Commented out IPython magic to ensure Python compatibility.
# %ls
from google.colab import files
uploaded = files.upload()

import zipfile
import io

zf = zipfile.ZipFile(io.BytesIO(uploaded['kaggle-genki4k.zip']), "r")
```

```
zf.extractall()

"""## All Libs"""

import numpy as np
import os
import cv2
# ----------------------------------------------------------------------
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.applications import VGG16

from tensorflow.keras import models, layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import dlib
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.optimizers import Adam
from keras.preprocessing import image
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau


# from google.colab import drive
# drive.mount('/content/drive')
```

3-برش چهره از تصاویر برای ورودی یادگیر:

مدل‌های تشخیص صورت انواع مختلفی دارند که می‌توان استفاده کرد:

- Haar Cascades: روش پیشنهادی برای تشخیص چهره، شامل مدل‌های مختلف مانند haarcascade_frontalface_alt.xml و haarcascade_profileface.xml (برای تشخیص صورت از کناره).

1. تشخیص صورت اصلی: این مدل به منظور شناسایی کلیه صورت‌ها در تصویر استفاده می‌شود.

2. تشخیص چشم: این مدل مخصوص شناسایی چشم‌ها در صورت است و معمولاً بعد از شناسایی صورت اعمال می‌شود.
3. تشخیص بینی: این مدل نیز برای شناسایی بینی در صورت به کار می‌رود.
4. تشخیص دهان: این مدل برای شناسایی ناحیه دهان و لب‌ها استفاده می‌شود.
5. تشخیص چهره در زاویه‌های مختلف: بعضی مدل‌ها برای شناسایی چهره‌ها از زوایای مختلف (مثلاً پرتره یا نیم رخ) بهینه‌سازی شده‌اند.

- DNN (Deep Neural Networks): برای کارایی بهتر و دقت بیشتر، می‌توانید از مدل‌های عمیق مانند cv2.dnn.readNetFromCaffe استفاده کنید، که معمولاً برای تشخیص چهره با دقیق‌تر استفاده می‌شود.

به کمک OpenCV و Haar Cascades به سادگی و با سرعت نسبتاً بالا می‌تواند صورت‌ها را شناسایی کند. این الگوریتم‌ها معمولاً برای برنامه‌های کاربردی در امنیت و تشخیص چهره‌ها و همچنین پروژه‌های ویدیویی به کار می‌روند.

- haarcascade_smile.xml
- haarcascade_mcs_mouth.xml
- haarcascade_mcs_nose.xml
- haarcascade_russian_plate_number.xml
- haarcascade_upperbody.xml
- haarcascade_eye.xml
- haarcascade_eye_tree_eyeglasses.xml
- haarcascade_frontalface_default.xml
- haarcascade_frontalface_alt.xml
- haarcascade_frontalface_alt2.xml
- haarcascade_frontalface_alt_tree.xml

برای استفاده از مدل تشخیص صورت با استفاده از OpenCV و کلاس CascadeClassifie، مراحل زیر را داریم:

1. نصب OpenCV

```
>> pip install opencv-python
```

2. بارگذاری مدل تشخیص صورت

```
import cv2
```

بارگذاری مدل تشخیص چهره

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

3. خواندن تصویر

سپس باید تصویری را که می‌خواهیم در آن صورت‌ها را تشخیص دهیم، بارگذاری کرده:

```
image = cv2.imread('path_to_your_image.jpg')
```

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # تبدیل تصویر به خاکستری برای تشخیص بهتر و شرط خود CV2
```

4. تشخیص صورت‌ها

حالا می‌توان از متد detectMultiScale برای تشخیص صورت‌ها استفاده کنیم:

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=?, minNeighbors=?)
```

- scaleFactor: مقدار تغییر اندازه که برای تشخیص صورت‌ها استفاده می‌شود. اگر مقدار آن کمتر از 1 باشد، باعث می‌شود که مدل مصوبات بیشتری را جدی بگیرد.

- minNeighbors: تعداد همسایگان مطلوب برای یک ناحیه تشخیص داده شده. مقدار بالاتر باعث می‌شود که نتیجه‌ها دقیق‌تر شوند. پس:

1. faces = face_cascade.detectMultiScale

- این خط یک متد به نام detectMultiScale را از شیء face_cascade فراخوانی می‌کند. این متد برای شناسایی اشیاء (در اینجا چهره‌ها) در تصویر استفاده می‌شود و نتیجه آن را در متغیر faces ذخیره می‌کند.

2. gray

- به این متد تصویر ورودی را می‌دهد که در اینجا تصویر خاکستری (gray) است. تبدیل تصویر به خاکستری معمولاً برای پردازش آسان‌تر و بهبود دقت شناسایی استفاده می‌شود.

3. scaleFactor=1.1,

- این پارامتر میزان افزایش مقیاس در هر مرحله از شناسایی را مشخص می‌کند. مقدار ۱.۱ به این معناست که اندازه تصویر در هر مرحله ۱۰ درصد افزایش می‌یابد. این کار به شناسایی چهره‌های با اندازه‌های مختلف کمک می‌کند.

4. minNeighbors=3,

- این پارامتر مشخص می‌کند که برای تأیید تشخیص هر چهره، چند چهره نزدیک به آن باید شناسایی شده باشند. مقدار ۳ به این معناست که حداقل باید سه همسایه شناسایی شده وجود داشته باشد تا یک ناحیه به عنوان چهره مشخص شود. این کار به کاهش اشتباهات کمک می‌کند.

5. minSize=(30, 30)

- حداقل اندازه‌ای که ناحیه ممکن است یک چهره باشد را مشخص می‌کند. در اینجا ناحیه‌ای که کمتر از ۳۰ پیکسل در عرض و ۳۰ پیکسل در ارتفاع باشد، شناسایی نمی‌شود.

6. flags=cv2.CASCADE_SCALE_IMAGE

- این پارامتر نشان می‌دهد که از چه نوع پرچم‌هایی استفاده شود. cv2.CASCADE_SCALE_IMAGE برای نشان دادن این است که مقیاس تصویر باید همواره با در نظر گرفتن ابعاد تصویر بدون تغییر باقی بماند.

5. برش و نمایش صورت‌ها

حال می‌توانید صورت‌های شناسایی شده را برش داده و نمایش دهید:

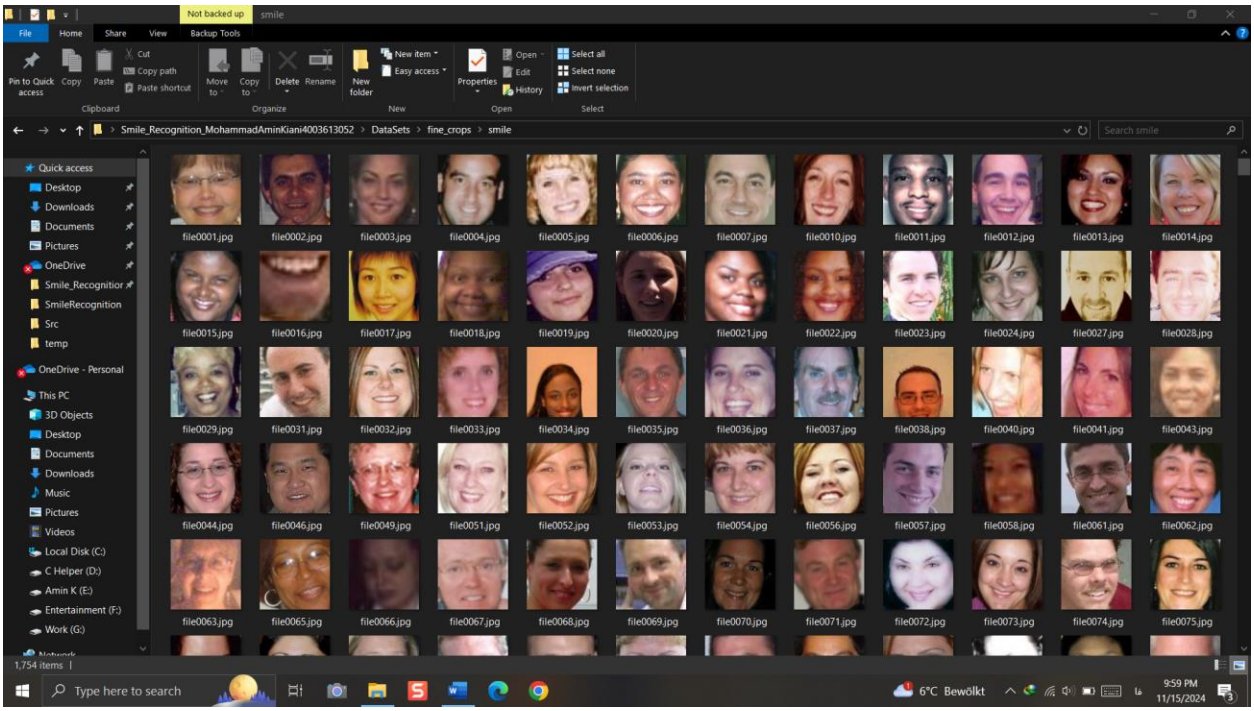
```
for (x, y, w, h) in faces:
    # برش صورت
    face = image[y:y+h, x:x+w]
    # نمایش صورت
    cv2.imshow('Face', face)
```

به طور کلی، این کد از مدل Cascade Classifier برای شناسایی چهره‌ها در یک تصویر خاکستری با تنظیمات خاصی استفاده می‌کند.

```
"""## Face Detect"""

with tf.device('/GPU:0'):

  # مسیر دیتاها
  data_dir = '/content/kaggle-genki4k'
  cropped_dir = '/content/crops'


  # تابع برش چهره‌ها از تصاویر
  # Cascade Classifier کتابخانه : برای تشخیص چهره
  face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

  #
https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_eye.xm
l
  eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye.xml')


  # cropped_dir = '/content/crops'
  os.makedirs(cropped_dir, exist_ok=True)

  def crop_face(image_path, save_path, rect=None):
      image = cv2.imread(image_path)
      if image is None:
```

```
        print(f"Error loading image: {image_path}")
        return

    # تصویر را به سیاه سفید تبدیل که برای الگوریتم‌های تشخیص چهره است
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    if rect is None:
        # اگر RECTANGLE مشخص نشده است، آن را پیدا می‌کنیم
        faces = face_cascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=3,
            minSize=(30, 30),
            flags=cv2.CASCADE_SCALE_IMAGE
        )

        if len(faces) == 0:

            print(f"No face detected in {image_path}")
            return

        # انتخاب اولین چهره
        rect = faces[0]

    (x, y, w, h) = rect
    face_crop = cv2.resize(image[y:y+h, x:x+w], (224, 224))
    cv2.imwrite(save_path, face_crop)


# data_dir = '/content/kaggle-genki4k'
# برش چهره‌ها و ذخیره‌سازی
for label in ['smile', 'non_smile']:
    label_dir = os.path.join(data_dir, label)
    cropped_label_dir = os.path.join(cropped_dir, label)
    os.makedirs(cropped_label_dir, exist_ok=True)

    for img_name in os.listdir(label_dir):
        img_path = os.path.join(label_dir, img_name)
        save_path = os.path.join(cropped_label_dir, img_name)

        # برای محدود کردن جستجوی چهره RECTANGLE
        image = cv2.imread(img_path)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(
            gray,
```

```
            scaleFactor=1.012,    # کاهش دادم تا صورت های کوچک تر را هم در بر بگیرد.
            minNeighbors=1,
            minSize=(10, 10),
            flags=cv2.CASCADE_SCALE_IMAGE
        )

        if len(faces) > 0:
            rect = faces[0]  # انتخاب اولین چهره
            crop_face(img_path, save_path, rect)

        else:
            print(f"No face detected in {img_path}")

"""## Download Crops to See"""

# files.download(cropped_dir)

import shutil

# Create a zip file of the crops directory
shutil.make_archive('/content/crops', 'zip', '/content/crops')

files.download('/content/crops.zip')

# Commented out IPython magic to ensure Python compatibility.
# %ls
from google.colab import files
uploaded = files.upload()

import zipfile
import io

zf = zipfile.ZipFile(io.BytesIO(uploaded['fine_crops.zip']), "r")
zf.extractall()

cropped_dir = '/content/fine_crops'

"""## Make & Compile AminNet_Model"""

# # this code helps to read the images faster
# AUTOTUNE = tf.data.AUTOTUNE
# train_generator = train_generator.prefetch(buffer_size=AUTOTUNE)
# validation_generator = validation_generator.prefetch(buffer_size=AUTOTUNE)
```



4-تست معماری های مختلف و ساخت مدل نهایی:

1. VGG %84

- معماری: VGG به دلیل سادگی و ساختار منظمش معروف است و عمدتاً از لایه‌های کانولوشن (Convolutional Layers) و لایه‌های Fully Connected تشکیل شده است.
- مزایا:
  - دقت بالا در تشخیص تصویر
  - ساده در طراحی و پیاده‌سازی
- معایب:
  - بسیار سنگین و نیاز به منابع محاسباتی بالا
  - زمان آموزش طولانی

2. MobileNet %64

- معماری: این معماری برای دستگاه‌های موبایل و سیستم‌های با توان محاسباتی کم طراحی شده است. از تکنیک Depthwise Separable Convolutions استفاده می‌کند.

- مزایا:

- سبک و سریع

- مصرف انرژی کم

- معایب:

- ممکن است دقت کمتری نسبت به VGG داشته باشد

3. EfficientNet %77

- معماری: EfficientNet با توازن بهینه‌ای بین دقت و پیچیدگی طراحی شده است. از تکنیک‌های مقیاس‌گذاری جدید استفاده می‌کند.

- مزایا:

- دقت بسیار بالا در مقایسه با حجم کم

- مقیاس‌پذیری خوب بر اساس نیاز

- معایب:

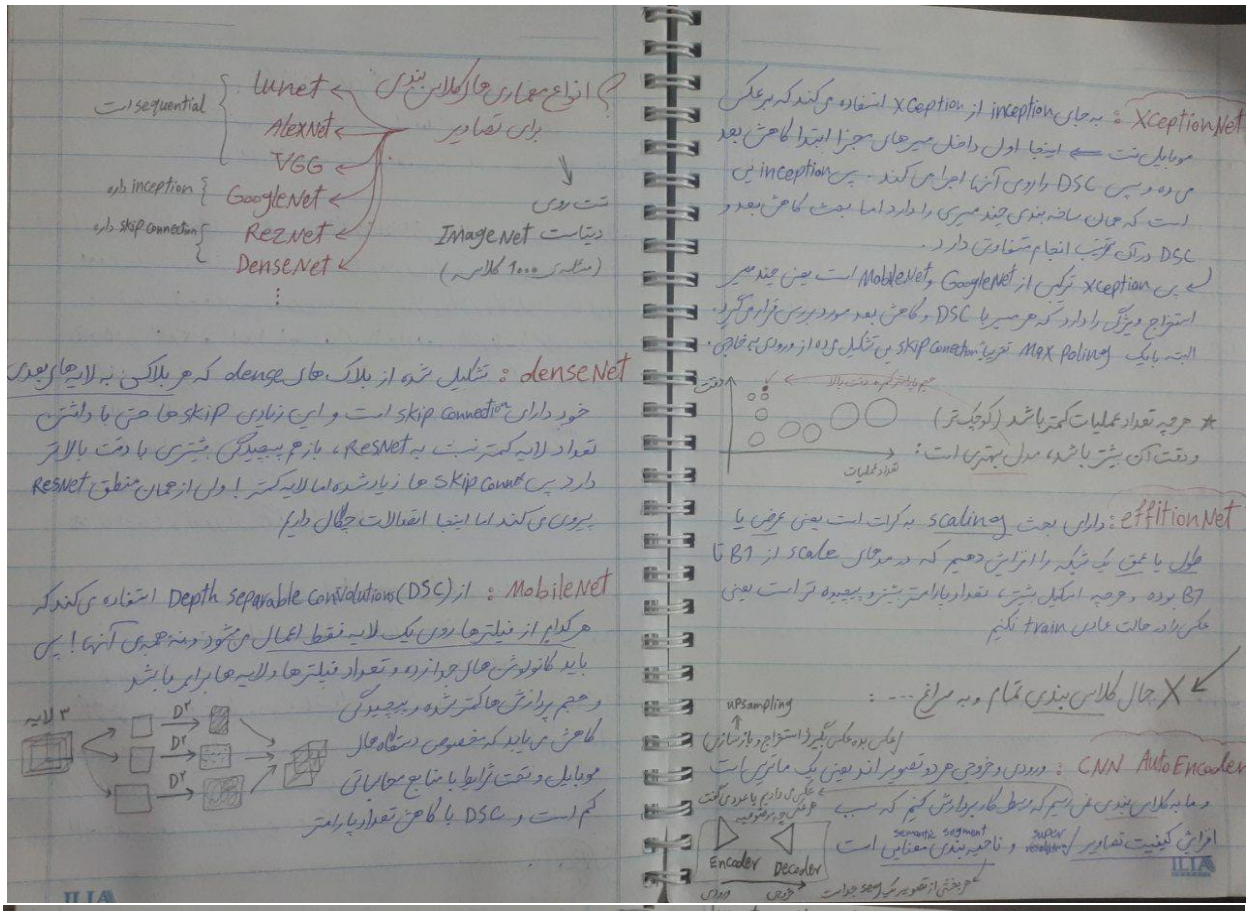
- ممکن است پیچیدگی بیشتری در پیاده‌سازی داشته باشد

نتیجه‌گیری

- انتخاب VGG: اگر به دقت بالا نیاز دارید و منابع محاسباتی کافی دارید.

- انتخاب MobileNet: اگر به سرعت و کارایی در دستگاه‌های موبایل اهمیت می‌دهید.

- انتخاب EfficientNet: برای بهینه‌سازی دقت و پیچیدگی به شکل همزمان.


انواع معماری های کلاس بندی برای تصاویر
LuNet
AlexNet
VGG
GoogleNet
ResNet
DenseNet
sequential است
دارای inception
دارای skip connection
ست روی دیتاست ImageNet (مسئله ی 1000 کلاسه)
XceptionNet : به جای inception از xception استفاده می کند که برعکس موبایل نت ⟸ اینجا اول داخل میرهال مجزا ابتدا کاهش بعد می ده و بعد DSC را روی آنها اجرا می کند - پس inception این است که همان ساده بندی چند میری را دارد اما بعث کاهش بعد و DSC در آن ترتیب انجام متفاوتی دارد .
پس xception ترکیبی از GoogleNet و MobileNet است یعنی چند میر استوانج ویژگی را دارد که هر میر با DSC و کاهش بعد مورد بررسی قرار می گیرد .
البته بایک MaX Polinng تقریبا skip Conection ین تشکیل می ده از ورودی به خارجی .
★ هرچه تعداد عملیات کمتر باشد (کوچک تر) و دقت آن بیشتر باشد، مدل بهتری است :
دقت بالا
تعداد عملیات
effitionNet : دارای بحث scaling به کرات است یعنی عرض یا طول یا عمق یک شبکه را افزایش دهیم که در مدل های scale از B1 تا B7 بوده و هرچه اسکیل بیشتر، تعداد پارامتر بیشتر و پیچیده تر است یعنی عکس را در حالت عادی train نکنیم
X حال کلاس بندی تمام و به مراتع ... :
UPSampling
CNN AutoEncoder : ورودی و خروجی هر دو تصویر اند یعنی یک مانیز است و ما به کلاس بندی نمی رسیم که ربط کار برداش کنیم که سبب افزایش کیفیت تصاویر / Super resolution و ناحیه بندی معنایی است Semantic segment
Encoder Decoder
ورودی خروجی
ILIA
Back bone Neck head
feature
train
image
استخراج ویژگی
کلاس بندی
transfer learning
یک بدن
یک تنه
هد
★ اگر شبکه اولیه با تعداد کلاس head ما برابر باشد ⟸ head به درد می خورد .
★ ولی ۹۰٪ مواقع Back bone ها توسط transfer learn ها توسط pretrain برداشته می شوند .
★ freeze کردن لایه ها : تایم train را کوتاه تر کرده و سرعت اجرا بیشتر می شود و معمولا لایه های اول که کمل است را freeze می کنند زیرا لایه های عمیق تر دارای ویژگی های جزئی تر است باید بماند ! (base.model.trainable = False کل شبکه را خاموش کرد)
برای جابجایی ترتیب دیتاها هم است مثل متن و سایل بری زمان، شبکاتی که تا الان خوندیم بدردش نمی خورد زیرا ترتیب در آن ندارد .
★ برابر بودن سایز ورودی شبکه با سایز تصاویر داده شده بسیار مهم است حتی جنس آن هم همین طور ، مثلا شبکه اپکیل شده با grayscale را نمی RGB بزن کنه
★ تعداد نورون های لایه آخر : sigmoid ⟸ 1 = تعداد کلاس ها ⟸ فعال ساز آن
softMax ⟸ n = تعداد کلاس ها ⟸ فعال ساز آن
n>2
یعنی برخی جاها به جای استخراج ویژگی های بیشتر، از transfer learn استفاده کنیم یا آوردن back bone آنی شبکه
ILIA
lunet → AlexNet → VGG → GoogleNet
ulu → معماری آبجکت بندی
معماری های کلاس بندی بدون عمق
شبکه VGG : شکلش با کانولوشن های ۳×۳ و افزودن تعداد لایه ها با سایز فیلتر ۳ می باشد و دقت آن از قبل ها بیشتر و عمق دارتر است اما کند است . (معماری یک شبکه : یک بدن ها تا قبل فلت)
معماری GoogleNet : هر inception دارای ۴ لایه میر برای حرکت به لایه بعدی است یعنی چند میر استخراج ویژگی دارد که ویژگی های درشت و ریز را از لایه قبل به بعدی منتشر می کند و دارای کانولوشن های ۵×۵ و ۳×۳ و ۱×۱ است که (1x1) یعنی داریم لایه را ضرب در عدد اسکالر می کنیم و این ضرب های عددی ۱×۱ برای ما کاهش بعد هم ایجاد می کنند تا از پیچیدگی و افزایش عمق جلوگیری کند .
★ تعداد پارامتر کمتری نسبت به VGG دارد زیرا با اینکه چند میره است اما کاهش بعد سبب آسان تر شدن آن شده است .
★ تا اینجا و بعدی آن عمل برای کلاس بندی تصاویر بکاری روند .
معماری ResNet : ساختار sequential را کامل زیر سوال برد که اگر skip Connection ها را نداشت همان VGG بود با تعداد لایه بیشتر ؛ ولی حال برای اپکیت کردن loss اگر خواست برگردد یک میر میانبر داشته باشد و موانعی سازی حتی نخ دهد یعنی سرعت عملکرد بیشتر می شود و نورون ها استثنا به عقب دارند و احتمال overfit شدن کمتر است یعنی خروجی هر لایه برای لایه بعدی خود و دودا لایه جلوتر بعدی مستقیما میره و ۲ میر دارد و هر دو در لحظه اعمال می شود
یعنی لایه های بعدی اگر تغییرات خروجی را دیدند ⟸ $H(x) = x + F(x)$
تغییرات ورودی
ورودی
skip Connection
forward
backward
ILIA

```python
# from tensorflow.keras.applications import MobileNetV2

# # ترنسفر لرن بدون هد و فقط بک بون و تعیین اندازه ورودی
# # بارگذاری مدل MobileNet بدون لایه‌های بالایی و تعیین اندازه ورودی
# base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))


IMAGE_SHAPE = [224, 224]
batch_size=10
epochs = 12

base_model = VGG16(input_shape = (224,224,3), weights = 'imagenet', include_top =
False)

# from tensorflow.keras.applications import EfficientNetB7
# # Load EfficientNet-B7 model without top layers
# base_model = EfficientNetB7(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# Freeze layers  وزن‌ها در حین آموزش تغییر نمی‌کنند. برای جلوگیری از افت کیفیت ویژگی‌های استخراج شده،
# base_model.trainable = False
for layer in base_model.layers:
    layer.trainable = False

x = layers.Flatten()(base_model.output)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dense(2, activation='softmax')(x)  # number ofclasses = 2

model = models.Model(inputs=base_model.input, outputs=x)

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

#-------------------------------------------------------------------------------
--------------------


# global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

# prediction_layer = tf.keras.layers.Dense(1)
```

```
# inputs = tf.keras.Input(shape=(224, 224, 3))
# x = preprocess_input(inputs)
# x = base_model(x, training=False)
# x = global_average_layer(x)
# x = tf.keras.layers.Dropout(0.2)(x)
# outputs = prediction_layer(x)
# model = tf.keras.Model(inputs, outputs)

base_learning_rate = 0.001
#
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate
),
#               loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
#               metrics=['accuracy'])

with tf.device('/GPU:0'):

  #   # Prepare dataset
  # train_dataset = image_dataset_from_directory(
  #     cropped_dir,
  #     shuffle=True,
  #     seed=SEED,
  #     image_size=IMAGE_SIZE,
  #     batch_size=BATCH_SIZE,
  #     validation_split=VALIDATION_SPLIT,
  #     subset='training'
  # )

  # validation_dataset = image_dataset_from_directory(
  #     cropped_dir,
  #     shuffle=False,
  #     seed=SEED,
  #     image_size=IMAGE_SIZE,
  #     batch_size=BATCH_SIZE,
  #     validation_split=VALIDATION_SPLIT,
  #     subset='validation'
  # )

  # # Normalize pixel values
  # def normalize_images(image, label):
  #     image = tf.cast(image, tf.float32) / 255.
  #     return image, label

  # train_dataset = train_dataset.map(normalize_images)
```

```
  # validation_dataset = validation_dataset.map(normalize_images)

  # # Create data loaders
  # train_loader = tf.data.DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, seed=SEED)
  # validation_loader = tf.data.DataLoader(validation_dataset,
batch_size=BATCH_SIZE, shuffle=False, seed=SEED)

  # # Training
  # history = model.fit(
  #     train_loader,
  #     validation_data=validation_loader,
  #     epochs=EPOCHS,
  #     callbacks=[early_stopping, reduce_lr]
  # )

  # print(model.predict(train_loader))

  #-------------------------------------------------------------------------

  # آماده‌سازی داده‌ها با ImageDataGenerator
  # augmentation
  datagen = ImageDataGenerator(
      rescale=1./255,  #  0 255 ==> 0  1
      rotation_range=20,  # تصاویر تا ۲۰ درجه چرخش
      width_shift_range=0.2,  # تصاویر به اندازه ۲۰ درصد از عرض یا ارتفاع خود جابجا
      height_shift_range=0.2,
      shear_range=0.2,  # برش دهی تا ۲۰ درصد
      zoom_range=0.2,   # زوم بر روی تصاویر تا ۲۰ درصد.
      horizontal_flip=True,  #تصاویر می‌توانند به صورت افقی
      fill_mode='nearest',   # پر کردن پیکسل‌های جدید، نزدیکترین مقدار
      validation_split=0.2)  # 20 درصد داده‌ها برای تست

  train_generator = datagen.flow_from_directory(
      directory=cropped_dir,
      target_size=(224, 224),
      batch_size=10,
      class_mode='categorical',
      subset='training',
      shuffle=True                  # تصادفی کردن تصاویر برای جلوگیری از دسترسی‌های بلوکی به داده‌ها
  )

  validation_generator = datagen.flow_from_directory(
      directory=cropped_dir,
      target_size=(224, 224),
```

```
        batch_size=10,
        class_mode='categorical',
        subset='validation',
        shuffle=False
    )

#   #-------------------------------------------------------------------------

#   # استخراج ویژگی‌ها
#   train_features = model.predict(train_generator)
#   train_labels = train_generator.classes

#   validation_features = model.predict(validation_generator)
#   validation_labels = validation_generator.classes

#   #-------------------------------------------------------------------------

    early_stopping = EarlyStopping(monitor='val_loss',    #معیار نظارت بر بهبود مدل.
                                   #  min_delta=0.03, # minimium amount of change to
count as an improvement
                                   patience=2,   # اگر تغییر در معیار بیشتر از ۲ دوره مشاهده نشود،
آموزش متوقف می‌شود
                                   restore_best_weights=True)  # وزن بهترین مدل ذخیره شده
بازگردانده می‌شود.

    reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                   factor=0.2,   # اگر بهبودی نباشد، نرخ یادگیری به ۲۰ درصد کاهش
                                   patience=2,
                                   min_lr=1e-6)  # حداقل نرخ

    history = model.fit(
        train_generator,
        validation_data=validation_generator,
        epochs=epochs,
        verbose=1, # نمایش خروجی بیشتر
        # verbose=0: Minimal output. Only shows epoch number and total time.
        #verbose=1: Moderate output. Shows epoch number, validation loss, and
accuracy.
        #verbose=2: Detailed output. Shows batch size, iteration number, loss,
accuracy, etc.

        callbacks=[early_stopping, reduce_lr]   # استراتژی‌های حضوری برای شرایط خاص
    )

    #برچسب‌های کلاس‌ها و تعداد نمونه‌های موجود در ژنراتور آموزشی
```

```
  print(train_generator.classes)
  print(train_generator.samples)

# #-------------------------------------------------------------------------------
----------------------

# # model = models.Sequential([
# #     # base_model,
# #     # layers.GlobalAveragePooling2D(),
# #     layers.Flatten(input_shape=train_features.shape[1:]),  # 1D : vector
# #     layers.Dense(128, activation='relu'),
# #     layers.Dropout(0.2),
# #     layers.Dense(1, activation='sigmoid')
# # ])

# #-------------------------------------------------------------------------------
----------------------


# # from tensorflow.keras import layers, models
# # from tensorflow.keras.models import Model
# # from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout,
Flatten
# # from tensorflow.keras.optimizers import Adam
# # def create_model():
# #     # base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# #     x = base_model.output
# #     x = GlobalAveragePooling2D()(x)
# #     x = Dense(1024, activation='relu')(x)
# #     predictions = Dense(1, activation='sigmoid')(x)

# #     model = Model(inputs=base_model.input, outputs=predictions)

# #     return model

# # model = create_model()

# # # کامپایل مدل
# # model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

Y_pred = model.predict(validation_generator, validation_generator.samples /
batch_size)
```

```
# axix ==> کدام محور (یا ابعاد) آرایه بررسی شود تا بزرگ‌ترین مقدار گرفته شود
# axis=0: بالاترین مقدار در هر ستون (کلاس‌ها)
# axis=1: بالاترین مقدار در هر ردیف (نمونه‌ها)
val_preds = np.argmax(Y_pred, axis=1) #  تعیین کلاس‌های پیش‌بینی شده

import sklearn.metrics as metrics
val_trues =validation_generator.classes  #برچسب‌ها برای مقایسه با پیش‌بینی‌ها

from sklearn.metrics import classification_report
print(classification_report(val_trues, val_preds))  #گزارشی شامل دقت، فراخوانی ، و متوسط دقت

# # آموزش مدل با داده‌های آموزشی و تست روی داده‌های اعتبارسنجی

# history = model.fit(train_features,
#                     train_labels,
#                     batch_size=32,
#                     epochs=initial_epochs,
#                     validation_data=(validation_features, validation_labels),
#                     callbacks=[early_stopping])


# # ارزیابی مدل روی داده‌های اعتبارسنجی
# loss, accuracy = model.evaluate(validation_features, validation_labels)

# print(f"Validation Accuracy: {accuracy * 100:.2f}%")
# print(f"Validation Loss: {loss:.4f}")

# test_loss, test_accuracy = model.evaluate(test_datagen.flow_from_directory(
#     'crops',
#     target_size=(224, 224),
#     class_mode='binary',
#     batch_size=32
# ))

# print(f'Test accuracy: {test_accuracy:.2f}')

# ذخیره مدل
model.save('AminNet_Model_SmileDetector.h5')

files.download('AminNet_Model_SmileDetector.h5')
#  قابلیت ذخیره‌سازی داده‌های بزرگ با سرعت بالا
#  سازگاری با اکثر کتابخانه‌های عصبی
#  امکان ذخیره‌سازی وزن‌ها، ساختار شبکه و متغیرهای حالت
```

استفاده از ترنسفر لرنینگ و فاین تیون کردن در معماری مدل‌های یادگیری ماشین، به ویژه در حوزه‌های مانند بینایی کامپیوتری و پردازش زبان طبیعی، تأثیرات قابل توجهی دارد. بیایید این مفاهیم را به‌طور کامل بررسی کنیم:

۱. ترنسفر لرنینگ (Transfer Learning)

ترنسفر لرنینگ به فرایند استفاده از مدل‌های آموزش‌دیده شده در یک دامنه (مثلاً تشخیص اشیاء در عکس‌ها) و استفاده از آن‌ها در دامنه‌ای دیگر (مثلاً تشخیص اشیاء مخصوص در یک تصویر خاص) اشاره دارد. این کار به‌ویژه زمانی موثر است که داده‌های موجود برای دامنه جدید کم باشد.

مزایا:
- کاهش زمان آموزش: از آنجایی که مدل قبلاً بر اساس داده‌های متنوع آموزش دیده، سریع‌تر می‌تواند برای یک وظیفه خاص تنظیم شود.
- بهبود دقت: استفاده از ویژگی‌های استخراج‌شده از مدل‌های پیش‌آموزش‌دیده به افزایش دقت در وظایف جدید کمک می‌کند.
- مقاومت در برابر کمبود داده: در مواردی که داده‌های محدود داریم، ترنسفر لرنینگ می‌تواند نقطه قوتی باشد.

۲. فاین تیون کردن (Fine-tuning)

فاین تیون کردن به فرآیندی اطلاق می‌شود که در آن یک مدل پیش‌آموزش‌دیده برای یک وظیفه خاص به‌تدریج با استفاده از داده‌های مربوط به آن وظیفه تازه آموزش داده می‌شود. این کار معمولاً با کمی تغییر در لایه‌های نهایی مدل انجام می‌شود تا خروجی متناسب با نیاز خاص باشد.

چگونه کار می‌کند؟
- بخشی از لایه‌های مدل ثابت می‌مانند در حالی که لایه‌های میانی یا نهایی با داده‌های جدید آموزش می‌بینند.

- این کار به مدل اجازه می‌دهد تا درک عمیق‌تری از داده‌های جدید پیدا کند، بدون اینکه از پایه بسازید.

۳. تاثیرات بر روی مدل تابعی

- سرعت یادگیری: با استفاده از ویژگی‌های از پیش آموخته‌شده، مدل می‌تواند سریع‌تر و با کارایی بالاتری یاد بگیرد.
- افزایش دقت: با فاین تیون کردن، مدلی که قبلاً به دقت آموزش دیده است، می‌تواند به طرز محسوسی در وظایف جدید، خطای کمتری داشته باشد.
- کاهش اورفیتینگ: با استفاده از مدل‌های پایه‌ای که از برش بزرگ داده‌ها آموزش دیده‌اند، می‌توان ریسک اورفیتینگ (overfitting) را کاهش داد.

در دیپ لرنینگ، دو نوع ساختار اصلی برای معماری مدل وجود دارد:

ساختار سکونشال (Sequential)

- خطی بودن: در این نوع معماری، لایه‌ها به صورت خطی و پشت سر هم قرار می‌گیرند. هر لایه به لایه بعدی متصل است.
- سادگی: برای مدل‌های ساده و غیر پیشرفته، مانند شبکه‌های عصبی با چند لایه، مناسب است.
- کتابخانه‌ها: در بسیاری از کتابخانه‌های دیپ لرنینگ مانند Keras، این نوع ساختار برای ایجاد مدل به راحتی استفاده می‌شود.

ساختار تابعی (Functional)

- انعطاف‌پذیری: این نوع معماری اجازه می‌دهد تا لایه‌ها به صورت غیرخطی و با اتصالات پیچیده‌تری مثل چند ورودی و خروجی، بوجود آیند.
- معماری‌های پیشرفته: برای مدل‌هایی با ارتباطات پیچیده، مثل شبکه‌های عصبی با لایه‌های انفصال یا شاتد می‌تواند استفاده شود.
- قابلیت استفاده مجدد: می‌توانید لایه‌ها یا بلوک‌های مختلف را در ساختارهای مختلف دوباره استفاده کنید.

فرمت‌های ذخیره‌سازی مدل‌های عصبی شامل چندین نوع مختلف است که هر کدام مزایا و معایب خاص خود را دارند. در اینجا به برخی از انواع رایج‌ترین فرمت‌های ذخیره‌سازی مدل‌ها اشاره می‌کنیم:

1. H5 (HDF5)

فرمت H5 یکی از رایج‌ترین و قدرتمندترین روش‌های ذخیره‌سازی مدل‌ها است.

- مزایا:
  - قابلیت ذخیره‌سازی داده‌های بزرگ با سرعت بالا
  - سازگاری با اکثر کتابخانه‌های عصبی
  - امکان ذخیره‌سازی وزن‌ها، ساختار شبکه و متغیرهای حالت
- نحوه استفاده:

```
model.save('smile.h5')
```

2. ONNX (Open Neural Network Exchange)

ONNX یک فرمت استاندارد برای تبادل مدل‌های عصبی است.

- مزایا:
  - سازگاری با اکثر کتابخانه‌های عصبی مختلف
  - امکان تبدیل مدل‌ها به فرمت‌های دیگر
  - قابلیت استفاده در محیط‌های مختلف
- نحوه استفاده:

```
model.save('smile_detection_model.onnx')
```

3. TensorFlow SavedModel

این فرمت مخصوص استفاده با TensorFlow است.

- مزایا:
  - سازگاری کامل با TensorFlow
  - شامل متغیرهای حالت و وزن‌ها
  - قابلیت ذخیره‌سازی ساختار شبکه
- نحوه استفاده:

```
model.save('smile_detection_model', save_format='tf')
```

4. PyTorch SavedModel

فرمت ذخیره‌سازی مخصوص استفاده با PyTorch است.

- مزایا:
  - سازگاری کامل با PyTorch
  - شامل متغیرهای حالت و وزن‌ها
  - قابلیت ذخیره‌سازی ساختار شبک
- نحوه استفاده:

```
torch.save(model.state_dict(), 'smile_detection_model.pth')
```

5. Keras H5

این فرمت مخصوص استفاده با Keras است.

- مزایا:
  - سازگاری با Keras
  - شامل ساختار شبکه و وزن‌ها
  - قابلیت ذخیره‌سازی متغیرهای حالت
- نحوه استفاده:

```
model.save('smile_detection_model.keras')
```

چگونه عمل می‌کند؟

فرمات‌های ذخیره‌سازی معمولاً شامل موارد زیر هستند:

1. ساختار شبکه عصبی
2. وزن‌ها
3. متغیرهای حالت (در صورت موجود)
4. تنظیمات آموزش

در طول فرآیند ذخیره‌سازی، مدل به صورت یک فایل یا مجموعه‌ای از فایل‌ها ذخیره می‌شود. این فایل‌ها حاوی اطلاعات ضروری برای بازنویسی و استفاده مجدد از مدل هستند.

انتخاب بهترین فرمت برای ذخیره‌سازی مدل بستگی به موارد زیر دارد:
- کتابخانه مورد استفاده (TensorFlow, PyTorch, Keras)
- هدف نهایی (تبادل بین محیطها، استفاده در همان محیط)
- نیاز به ذخیره‌سازی متغیرهای حالت
- اندازه مدل و نیاز به سرعت ذخیره‌سازی

برای مدل ما که با TensorFlow ساخته شده است، استفاده از فرمت H5 یا SavedModel مناسب خواهد بود.

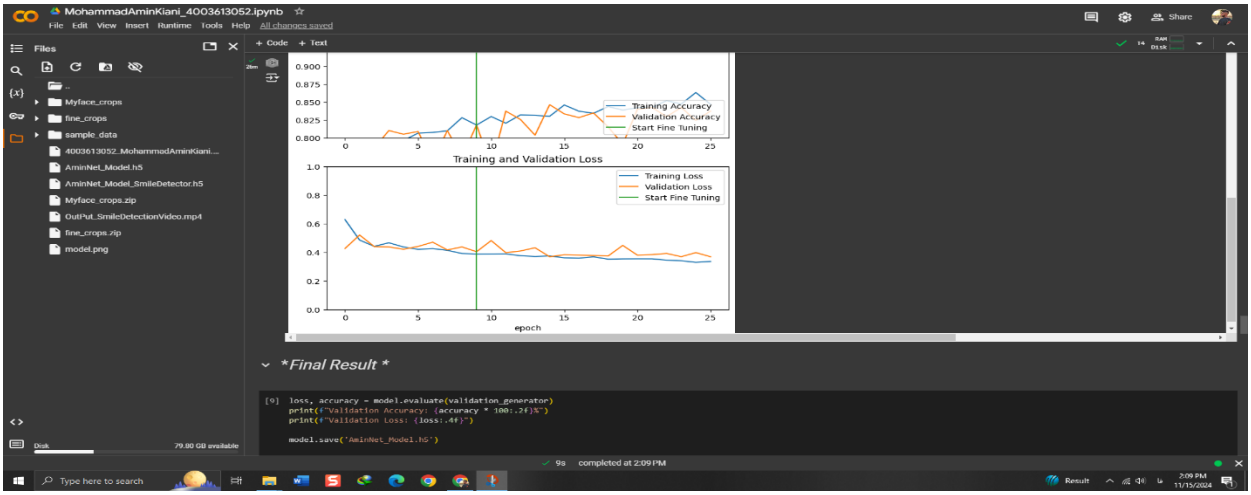
5-بهبود مدل و fine tuning:

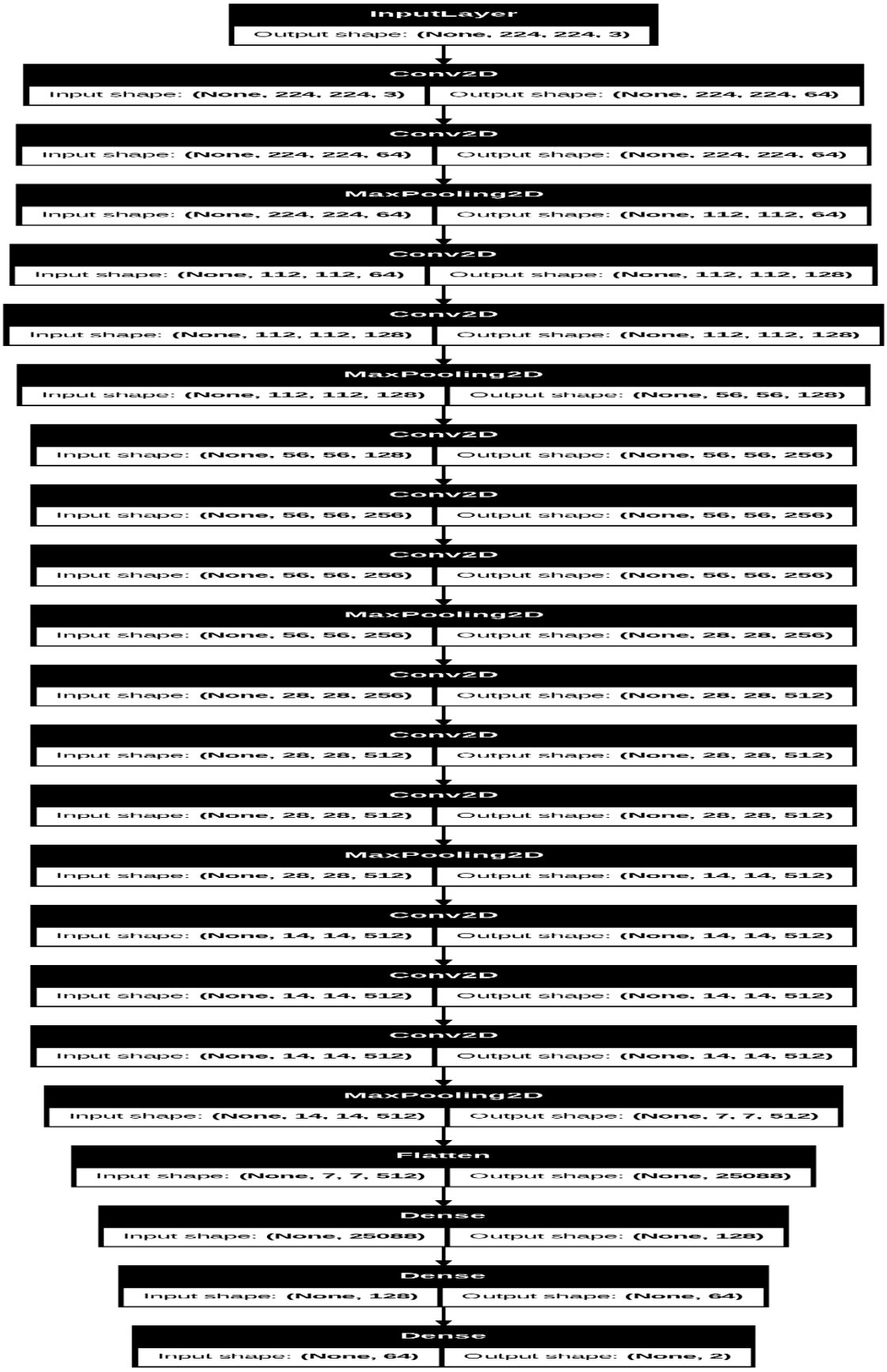
در بخش قبلی به طور کامل از فریز کردن اموزش بیس مدل تا ترنسفر لرن و فاین تیون کردن ان شهر داده شد و کد های ان به صورت زیر است که دقت مدل را از 79% به 84% رساند و به خوبی بهبود بخشید.

```
"""# Summary"""

base_model.summary()
model.summary()


import tensorflow as tf
tf.keras.utils.plot_model(model, show_shapes=True)


acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')
```

```
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('binary_crossentropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

# ---------------------------------------------------------------------------- tune
base_model.trainable = True  # می‌توانند وزن‌های خود را در طول فرایند آموزش به‌روزرسانی کنند.
fine_tune_at = 100

# آموزش Fine-Tuning (زیر ۱۰۰)، وزن‌ها قابل آموزش نباشند. این به ما کمک می‌کند تا فقط لایه‌های بالایی مدل در طول فرآیند
ببینند.
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001/10),
              metrics=['accuracy'])

initial_epochs = 10
fine_tune_epochs = 15
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_generator,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_generator)


acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
```

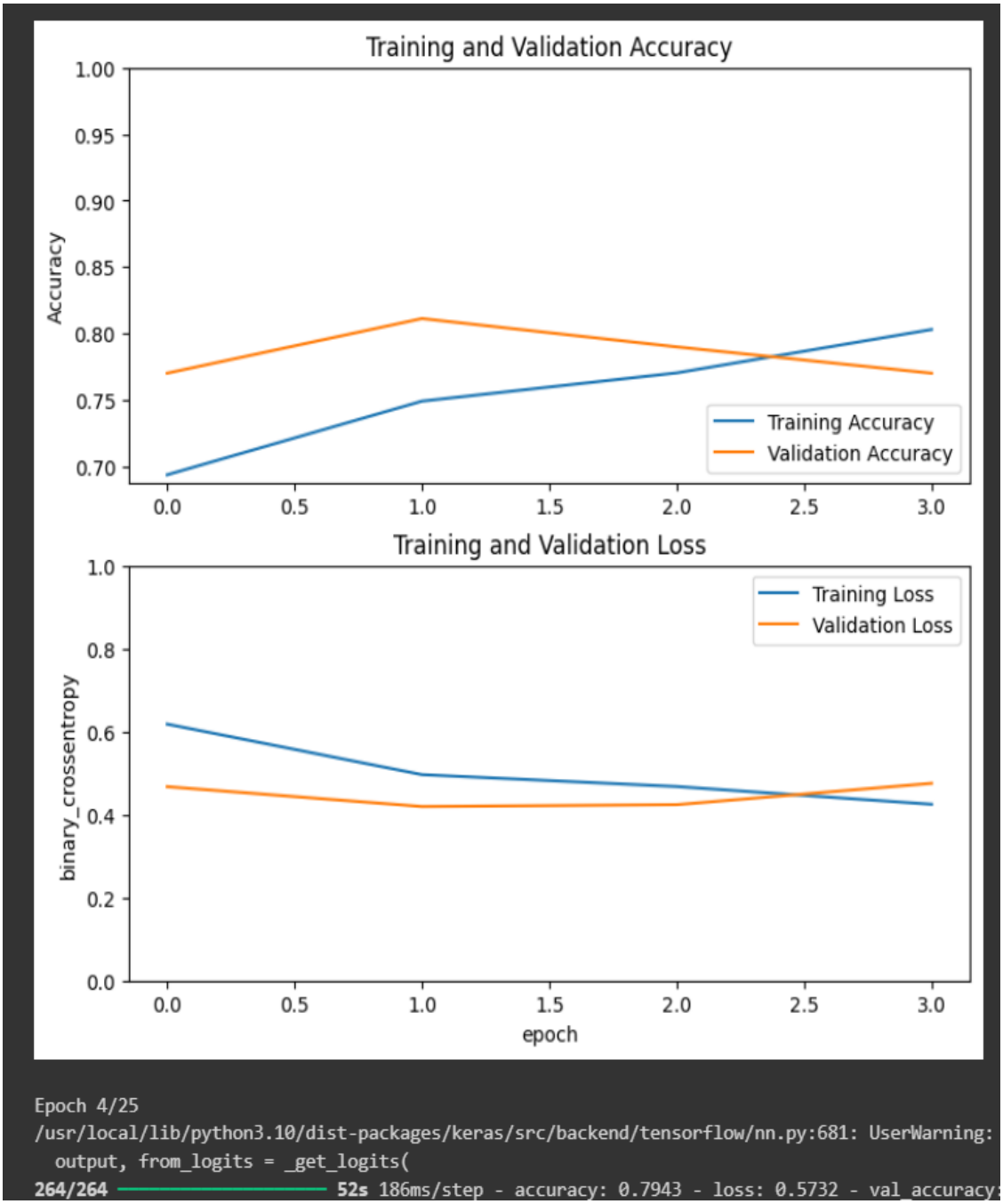
```
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

"""# **Final Result **"""

loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
print(f"Validation Loss: {loss:.4f}")

model.save('AminNet_Model.h5')

files.download('AminNet_Model.h5')

# from google.colab import drive
# drive.mount('/content')
# # Copy the file to Google Drive
# !cp 'AminNet_Model_SmileDetector.h5' '/content/drive/MyDrive/'
```



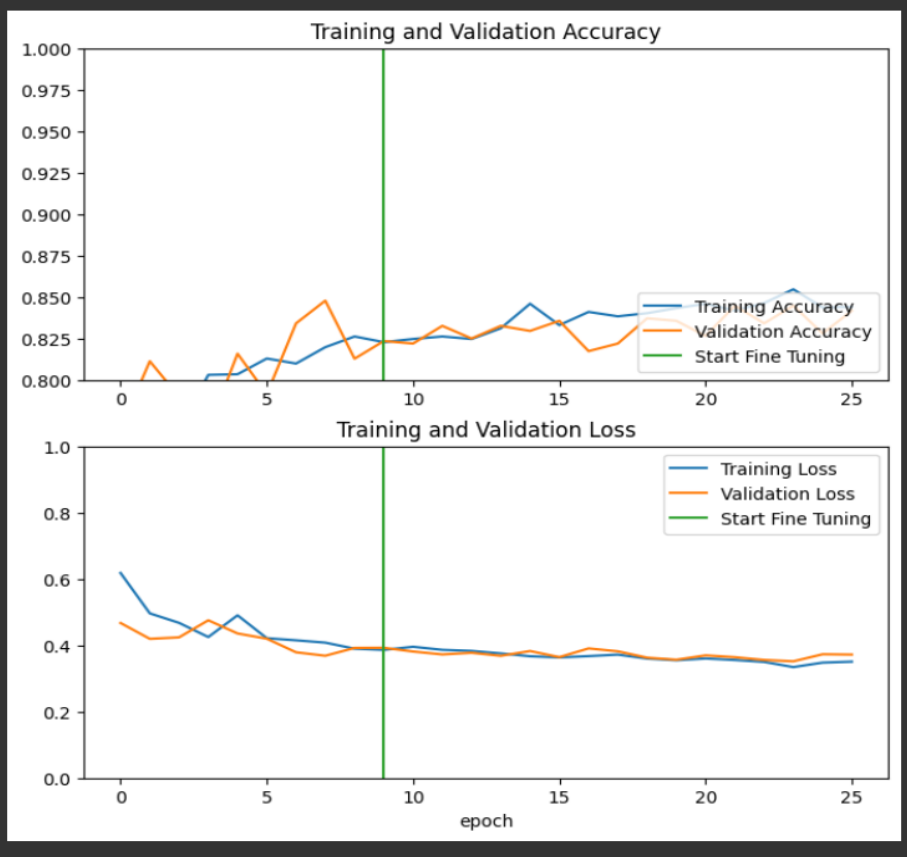

InputLayer
Output shape: (None, 224, 224, 3)
Conv2D
Input shape: (None, 224, 224, 3)
Output shape: (None, 224, 224, 64)
Conv2D
Input shape: (None, 224, 224, 64)
Output shape: (None, 224, 224, 64)
MaxPooling2D
Input shape: (None, 224, 224, 64)
Output shape: (None, 112, 112, 64)
Conv2D
Input shape: (None, 112, 112, 64)
Output shape: (None, 112, 112, 128)
Conv2D
Input shape: (None, 112, 112, 128)
Output shape: (None, 112, 112, 128)
MaxPooling2D
Input shape: (None, 112, 112, 128)
Output shape: (None, 56, 56, 128)
Conv2D
Input shape: (None, 56, 56, 128)
Output shape: (None, 56, 56, 256)
Conv2D
Input shape: (None, 56, 56, 256)
Output shape: (None, 56, 56, 256)
Conv2D
Input shape: (None, 56, 56, 256)
Output shape: (None, 56, 56, 256)
MaxPooling2D
Input shape: (None, 56, 56, 256)
Output shape: (None, 28, 28, 256)
Conv2D
Input shape: (None, 28, 28, 256)
Output shape: (None, 28, 28, 512)
Conv2D
Input shape: (None, 28, 28, 512)
Output shape: (None, 28, 28, 512)
Conv2D
Input shape: (None, 28, 28, 512)
Output shape: (None, 28, 28, 512)
MaxPooling2D
Input shape: (None, 28, 28, 512)
Output shape: (None, 14, 14, 512)
Conv2D
Input shape: (None, 14, 14, 512)
Output shape: (None, 14, 14, 512)
Conv2D
Input shape: (None, 14, 14, 512)
Output shape: (None, 14, 14, 512)
Conv2D
Input shape: (None, 14, 14, 512)
Output shape: (None, 14, 14, 512)
MaxPooling2D
Input shape: (None, 14, 14, 512)
Output shape: (None, 7, 7, 512)
Flatten
Input shape: (None, 7, 7, 512)
Output shape: (None, 25088)
Dense
Input shape: (None, 25088)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 64)
Dense
Input shape: (None, 64)
Output shape: (None, 2)

7-رسم نمودارها و ارتباطات:

مطابق پروژه ی قبلی از پلات پایتون برای رسم نتایج استفاده کرده تا ببنیم داده ها در کجا قرار دارند و دید کلی از قرار گیری ان پیش بینی ها داشته باشیم:



```
Epoch 25/25
264/264 ━━━━━━━━━━━━━━━━━━━━ 57s 211ms/step - accuracy: 0.8454 - loss: 0.3612 - val_accuracy: 0.8417 - val_loss
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



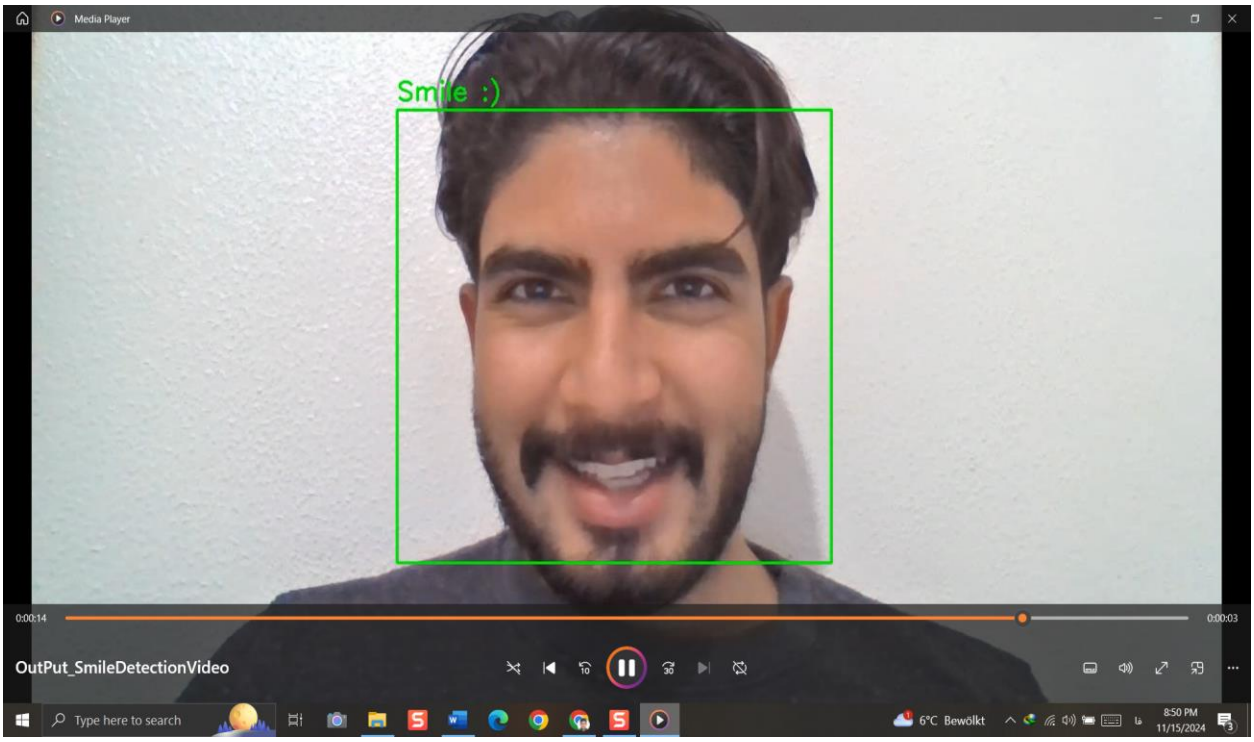
قبل از فاین تیون کردن:

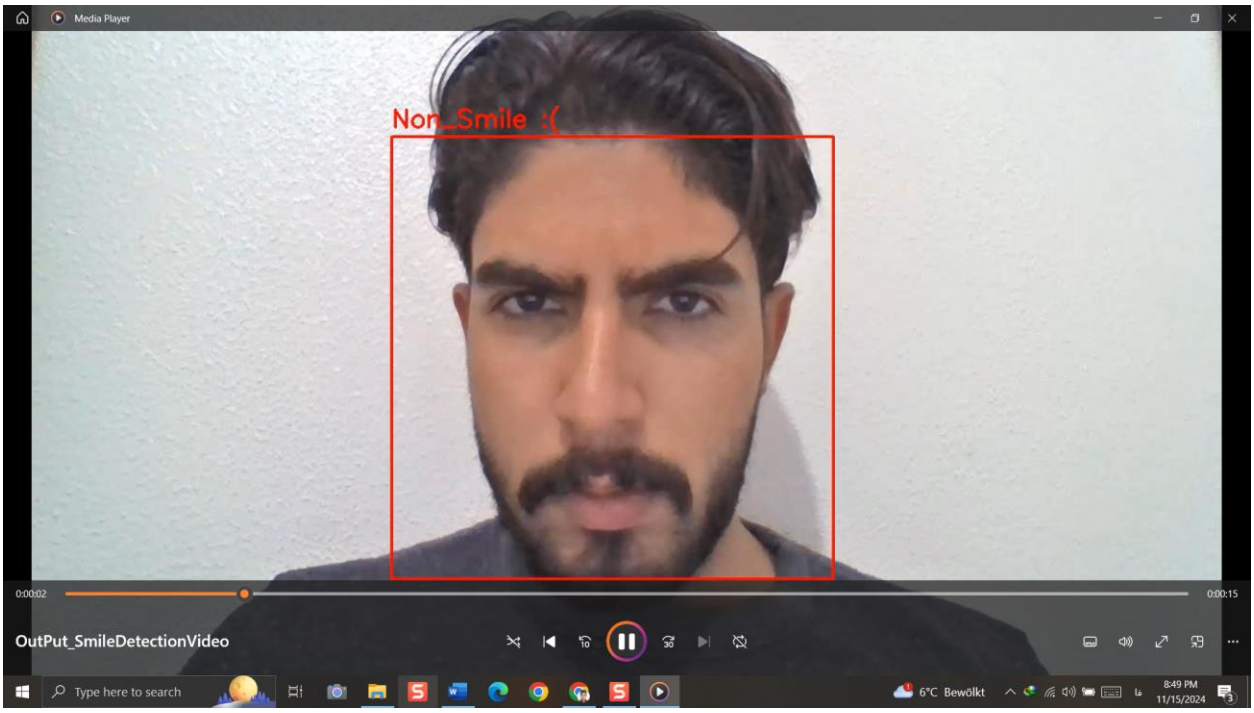
```
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 121ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       307
           1       0.80      0.80      0.80       350

    accuracy                           0.79       657
   macro avg       0.79      0.79      0.79       657
weighted avg       0.79      0.79      0.79       657
```

Summary

```
base_model.summary()
```

7-خروجی نهایی و تشخیص لبخند از ویدیو:





**Final Result **

```
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
print(f"Validation Loss: {loss:.4f}")

model.save('AminNet_Model.h5')

files.download('AminNet_Model.h5')

# from google.colab import drive
# drive.mount('/content')
# # Copy the file to Google Drive
# !cp 'AminNet_Model_SmileDetector.h5' '/content/drive/MyDrive/'
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 128ms/step - accuracy: 0.8276 - loss: 0.4199
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mo
Validation Accuracy: 84.17%
Validation Loss: 0.3572
```

```
"""# My Video"""

# ------------------------------------ model
from google.colab import files
uploaded = files.upload()

# ------------------------------------ video
from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

# Load the trained classifier model
model = load_model('AminNet_Model.h5')

# Load the feature extractor model (VGG16 without the top layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# face detector
```

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# directory to store face crops
if not os.path.exists('Myface_crops'):
    os.makedirs('Myface_crops')

# predict smile or not on a each frame and draw a rectangle
def predict_and_draw(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)   # Convert the image to
grayscale for better detection
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        # Crop and resize face to match input size 224*224
        face_crop = cv2.resize(frame[y:y+h, x:x+w], (224, 224))
        # Expand dimensions to match Keras input requirements
        face_crop_preprocessed = preprocess_input(np.expand_dims(face_crop,
axis=0))  # Preprocess the face crop

        # Save the face crop
        cv2.imwrite(f'/content/Myface_crops/face_{x}_{y}.jpg', face_crop)

        # Predict using the classifier model directly on the raw face crop
        prediction = model.predict(face_crop_preprocessed)

        # Draw rectangle based on prediction
        if prediction[0][0] > 0.5:  # treshold for 2 class
            color = (0, 0, 255)  # Red for non-smile
            label = "Non_Smile :( "
        else:
            color = (0, 255, 0)  # Green for smile
            label = "Smile :) "

        # Draw rectangle and label around the face
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2) #2: ضخامت خط مستطیل
        cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
color, 2)  # Display the label متن کمی بالاتر از بالای مستطیل

# process video frame by frame and save the output video
def process_video(video_path, output_path):
    cap = cv2.VideoCapture(video_path)  # Open the video

    # opened successfully?
    if not cap.isOpened():
```

```
        print("Error opening video file")
        return

    # Get video properties
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS)) #تعداد فریم‌هایی که در هر ثانیه نمایش داده

    # (Codec) در ویدیو به برنامه یا الگوریتمی اطلاق می‌شود که برای فشرده‌سازی و از بین بردن اطلاعات اضافی ویدیو
    # Define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # 'mp4v' به معنای استفاده از کدک MP4
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    while cap.isOpened():
        ret, frame = cap.read()   # ret : آیا فریم با موفقیت خوانده شده یا خیر

        # ret ==> true   : still has frame

        if not ret:   #  ret ==> False  (یعنی نتوانسته‌ایم فریم را بخوانیم)
            break

        # Predict smile and draw rectangle
        predict_and_draw(frame)

        # Write the frame to the output video
        out.write(frame)

    # Release video capture and writer منابع مربوط به ویدیو را آزاد می‌کنند، تا از نشت حافظه جلوگیری شود.
    cap.release()
    out.release()
    print("Processing completed. SmileDetection Video saved as:", output_path)


# ---------------------------------------------------- predict video by model:

process_video('/content/4003613052_MohammadAminKiani.mp4',
'/content/OutPut_SmileDetectionVideo.mp4')

"""Download Pics of me to see each frame & result predict"""

import shutil

# Create a zip file of the crops directory
shutil.make_archive('/content/Myface_crops', 'zip', '/content/Myface_crops')
```

```
files.download('/content/Myface_crops.zip')

files.download('/content/OutPut_SmileDetectionVideo.mp4')
```

```
#-------------------------------------------------------------------------------
-
# # Smile Detection with Webcam/Video

# # Define video capture
# cap = cv2.VideoCapture(0) # 0 for webcam, or specify video file path

# # Load the trained model
# model = load_model('AminNet_Model.h5')

# # Function for smile detection on a frame
# def detect_smile(frame):
#   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
#   faces = face_cascade.detectMultiScale(gray, 1.1, 4)
#   for (x, y, w, h) in faces:
#     face = frame[y:y + h, x:x + w]
#     face = cv2.resize(face, (224, 224))
#     face_input = preprocess_input(np.expand_dims(face, axis=0))
#     prediction = model.predict(face_input)
#     if prediction[0][0] > 0.5:
#       cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
#       cv2.putText(frame, 'Smile', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 255, 0), 2)
#     else:
#       cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
#       cv2.putText(frame, 'No Smile', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 255), 2)
#   return frame

# # Process video frames
# while True:
#   ret, frame = cap.read()
#   if not ret:
#     break
#   frame = detect_smile(frame)
#   cv2.imshow('Smile Detection', frame)
#   if cv2.waitKey(1) & 0xFF == ord('q'):
#     break
```

```
# # Release resources
# cap.release()
# cv2.destroyAllWindows()
```

برای پردازش تصویر و تشخیص لبخند با استفاده از وبکم یا آپلود ویدیو در پایتون و کتابخانه OpenCV (cv2)، می‌توانید مراحل زیر را دنبال کنید:

۱. نصب کتابخانه‌های مورد نیاز

ابتدا مطمئن شوید که OpenCV و سایر کتابخانه‌های مورد نیاز نصب شده‌اند.

۲. استفاده از وبکم یا اپلود برای تجزیه و تحلیل ویدیو

۲.۱. راه‌اندازی وبکم یا اپلود ویدیو

۲.۲. تشخیص لبخند

بعد از خواندن فریم‌ها، می‌توانید برای تشخیص لبخند از مدل‌های یادگیری ماشین استفاده کنید. برای این کار می‌توانید از مدل‌های پیش‌آماده مثل Haar Cascades استفاده کنید.

- cv2: کتابخانه OpenCV برای پردازش تصویر و ویدیو.
- numpy: برای کار با آرایه‌ها و عملیات عددی.
- os: برای کار با سیستم فایل (ایجاد دایرکتوری، مدیریت فایل‌ها).
- tensorflow.keras: برای بارگذاری مدل یادگیری عمیق و استفاده از VGG16 به عنوان استخراج‌کننده ویژگی.

2. بارگذاری مدل‌های آموزش‌دیده

```
model = load_model('AminNet_Model.h5')

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

- load_model: بارگذاری مدل تشخیص لبخند که قبلاً آموزش دیده است.
- VGG16: مدل VGG16 به عنوان استخراج‌کننده ویژگی برای پردازش تصاویر صورت.

3. بارگذاری تشخیص‌دهنده چهره

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

بارگذاری مدل Haar Cascade برای تشخیص چهره.

4. ایجاد دایرکتوری برای ذخیره برش‌های صورت

```
if not os.path.exists('Myface_crops'):
    os.makedirs('Myface_crops')
```

بررسی وجود دایرکتوری Myface_crops و ایجاد آن در صورت عدم وجود.

5. تعریف تابع پیش‌بینی و رسم مستطیل

```
def predict_and_draw(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
```

- predictanddraw: تابعی برای پیش‌بینی لبخند و رسم مستطیل دور چهره.
- تبدیل فریم به تصویر خاکستری برای بهبود دقت تشخیص چهره.
- شناسایی چهره‌ها در تصویر.

6. برش و پردازش هر چهره شناسایی‌شده

```
for (x, y, w, h) in faces:
    face_crop = cv2.resize(frame[y:y+h, x:x+w], (224, 224))
    face_crop_preprocessed = preprocess_input(np.expand_dims(face_crop, axis=0))
```

• برای هر چهره شناسایی‌شده، برش صورت انجام می‌شود و اندازه آن به 224x224 پیکسل تغییر می‌کند.

• پیش‌پردازش تصویر برش‌خورده برای ورودی به مدل.

7. ذخیره برش‌های صورت

```
cv2.imwrite(f'/content/Myface_crops/face_{x}_{y}.jpg', face_crop)
```

• ذخیره برش صورت به عنوان یک تصویر JPEG در دایرکتوری مشخص‌شده.

8. پیش‌بینی لبخند

```
prediction = model.predict(face_crop_preprocessed)
```

• پیش‌بینی لبخند یا عدم لبخند با استفاده از مدل بارگذاری‌شده.

9. رسم مستطیل و برچسب

```
if prediction[0][0] > 0.5:
    color = (0, 0, 255)
    label = "Non_Smile :( "
else:
    color = (0, 255, 0)
    label = "Smile :) "
cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
cv2.putText(frame, label, (x, y- 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
```

• اگر پیش‌بینی نشان‌دهنده عدم لبخند باشد، مستطیل قرمز و اگر لبخند باشد، مستطیل سبز رسم می‌شود.

• برچسب مربوط به پیش‌بینی در بالای مستطیل نمایش داده می‌شود.

10. پردازش ویدیو فریم به فریم

```
def process_video(video_path, output_path):
    cap = cv2.VideoCapture(video_path)
```

• تابعی برای پردازش ویدیو به صورت فریم به فریم.

11. بررسی موفقیت در باز کردن ویدیو

```
if not cap.isOpened():
    print("Error opening video file")
    return
```

• بررسی اینکه آیا ویدیو به درستی باز شده است یا خیر.

12. دریافت ویژگی‌های ویدیو

```
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
```

• دریافت عرض، ارتفاع و تعداد فریم در ثانیه (FPS) ویدیو.

13. تعریف کدک و ایجاد شیء VideoWriter

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
```

• تعریف کدک و ایجاد شیء VideoWriter برای ذخیره ویدیو خروجی.

14. پردازش هر فریم و نوشتن آن در ویدیو خروجی

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

predict_and_draw(frame)

out.write(frame)

• حلقه‌ای برای خواندن هر فریم از ویدیو، پیش‌بینی لبخند، و نوشتن فریم پردازش‌شده در ویدیو خروجی.

15. آزادسازی منابع

cap.release()

ChatGPT 4 | Midjourney | Gemini | Claude, [11/15/2024 10:42 PM]

out.release()

print("Processing completed. SmileDetection Video saved as:", output_path)

• آزادسازی منابع مربوط به ویدیو و چاپ پیام اتمام پردازش.

16. فشرده‌سازی تصاویر برش‌خورده و دانلود آنها

import shutil

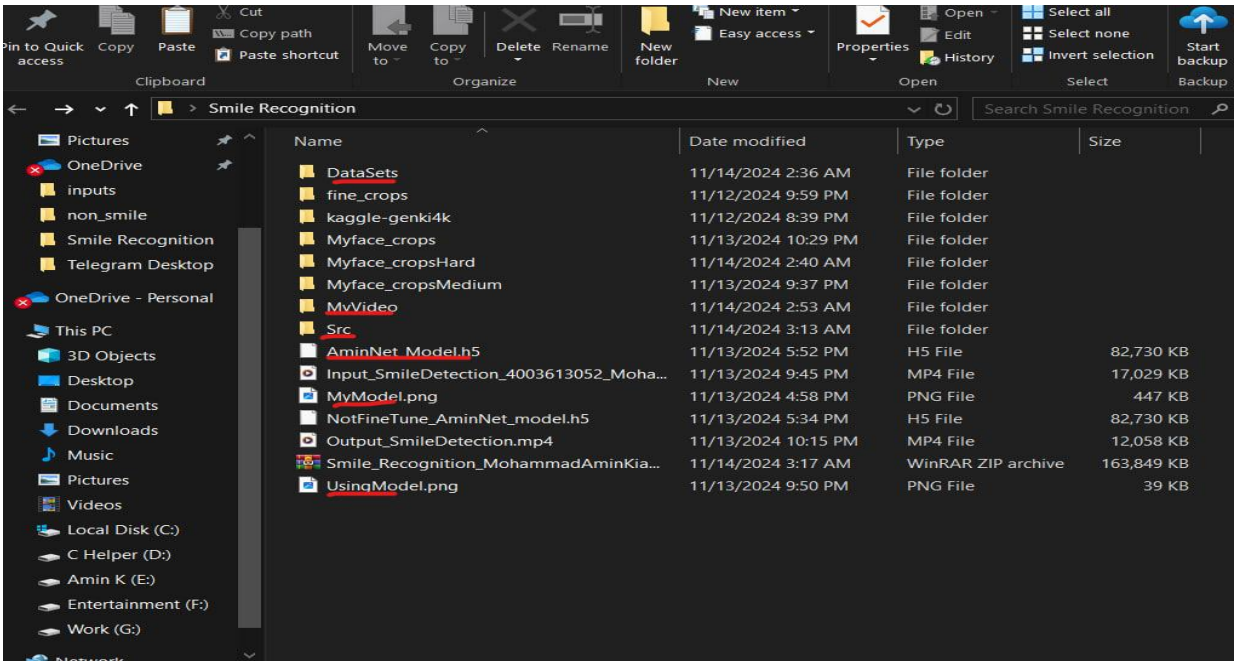
shutil.make_archive('/content/Myface_crops', 'zip', '/content/Myface_crops')

files.download('/content/Myface_crops.zip')

files.download('/content/OutPut_SmileDetectionVideo.mp4')

• ایجاد یک آرشیو ZIP از دایرکتوری برش‌های صورت و دانلود آن به همراه ویدیوی خروجی.

در کل یک برنامه کامل برای تشخیص لبخند در ویدیوها ساختیم که چهره‌ها را شناسایی کرده، لبخند را پیش‌بینی کرده، و نتایج را در قالب یک ویدیو خروجی ذخیره می‌کند.



8- مراجع

[1] https://github.com
[2] https://stackoverflow.com/questions
[3] https://www.wikipedia.org/
[4] https://colab.research.google.com/
[5] https://www.tensorflow.org/guide/
[6] https://keras.io/
[7] https://www.geeksforgeeks.org/opencv-python-program-face-detection/
[8] https://www.kaggle.com/datasets/talhasar/genki4k/data
[9] https://www.researchgate.net/figure/A-smiling-and-a-non-smiling-image-from-the-GENKI-4k-dataset_fig1_281991690
[10] https://ieeexplore.ieee.org/document/7727484