



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 آقای دکتر کیانی

نیمسال اول تحصیلی 1403-04

فهرست مطالب

3	مستندات
3	1-مسئله و تحلیل کلی آن:
4	2-تولید ویژگی های جدید از تصاویر:
5	3-دیتاست نهایی بهبود یافته و پیش پردازش ها:
12	4-الگوریتم های خوشه بندی:
25	5-الگوریتم های کلاس بندی:
29	7-رسم نمودارها و ارتباطات:
31	7-خروجی نهایی و مقایسه ی کلاس بندی با خوشه بندی:
42	8-مراجع

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 آموزش 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 (مثلاً خودم).

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

می‌توان از معماری‌های 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 (۴K) 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_omEumLTF05MiZY1GM-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_profileface.xml` و `haarcascade_frontalface_alt.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 دقت‌تر 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 CascadeClassifier، 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.harcascades +  
                                     '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')
```

```
# تبدیل تصویر به  
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

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

    #
https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade\_eye.xml
    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:
    # مشخص نشده است، آن را پیدا می‌کنیم اگر RECTANGLE
    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)

        # RECTANGLE برای محدود کردن جستجوی چهره
        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

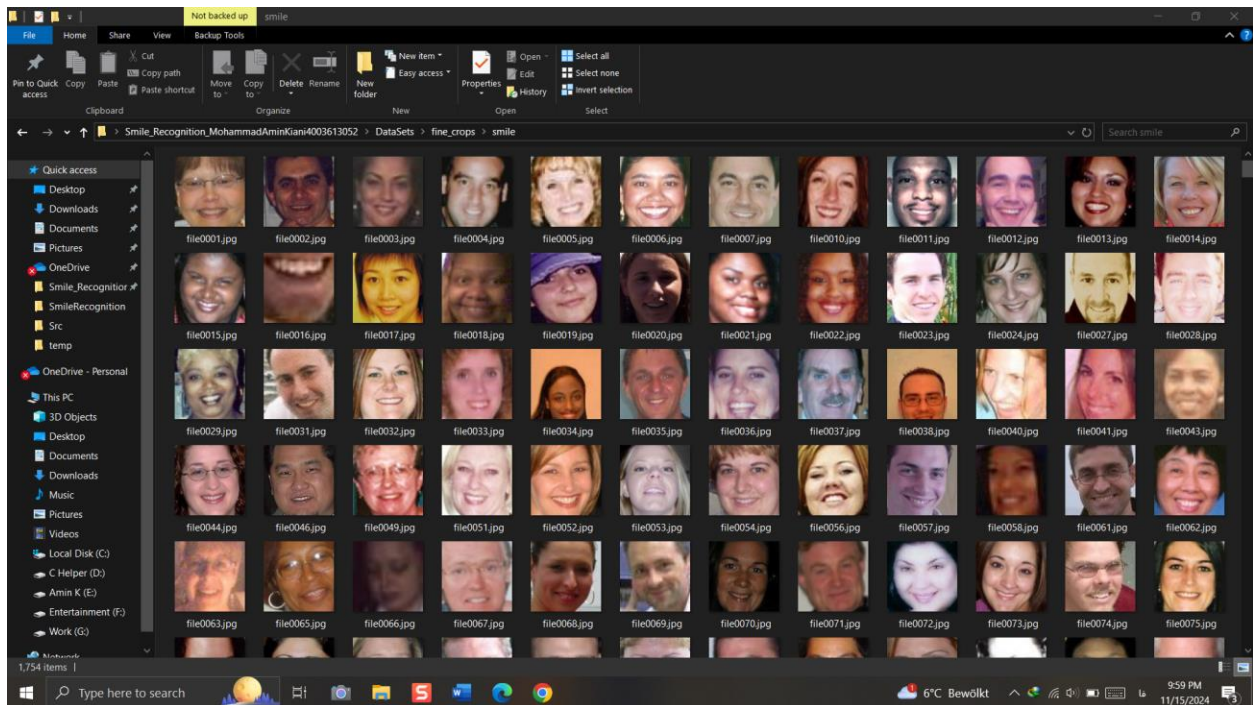
zf = zipfile.ZipFile(io.BytesIO(uploaded['fine_crops.zip']), "r")
zf.extractall()

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:

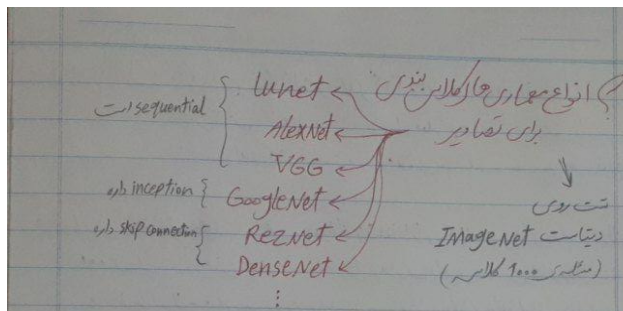
- ممکن است پیچیدگی بیشتری در پیاده‌سازی داشته باشد

نتیجه‌گیری

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

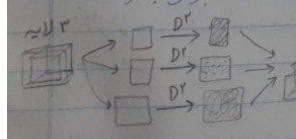
- انتخاب MobileNet: اگر به سرعت و کارایی در دستگاه‌های موبایل اهمیت می‌دهید.

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



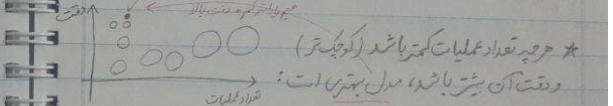
DenseNet: تشکیل شده از بلاک های dense که هر بلاک یک یا لایه های بعدی خود دارای skip connection است و این زنجار skip ها حتی با داشتن تعداد لایه کمتر نسبت به ResNet باز هم پیچیدگی بیشتری دارد و باعث دارد پس skip connection ها زیاد شده اما لایه کمتر! دلیل از همان منطق ResNet پیروی می کند اما اینجا اتصالات چگال داریم

MobileNet: از Depth separable convolutions (DSC) استفاده می کند که هر کلاسی از فیلترها روی یک لایه فقط اعمال می شود و نه عمیق آنها! این باید کانولوشن های جزایر و تعداد فیلترها و لایه ها را برابر یا بیشتر و حجم پردازش ها کمتر شده و پیچیدگی کاهش می یابد که خصوصاً در دستگاه های موبایل و وقت ترابا با منابع محاسباتی کم است و DSC با کاهش تعداد و با کمتر



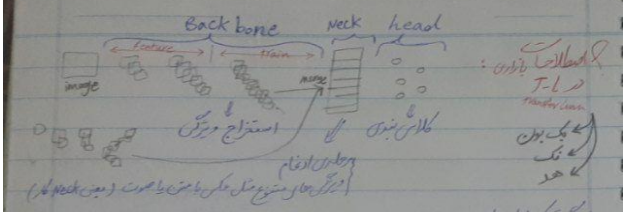
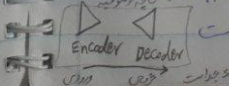
XceptionNet: به جای inception از Xception استفاده می کند که برعکس موبایل نت است اینجا اول داخل می رها می جزا ابتدا کاهش بعد می ده و سپس DSC را روی آن اجرا می کند پس inception نیست است که همان ماده بندی چند سری را دارد اما بهشت کاهش بعد و DSC در آن ترتیب انجام متفاوت دارد.

پس Xception ترکیب از GoogLeNet و MobileNet است یعنی چند مسیر استخراج ویژگی را دارد که هر مسیر با DSC و کاهش بعد مورد بررسی قرار می گیرد البته باید یک Max Pooling تقریباً skip connection می تشکیل ده از درون هم خارج



EfficientNet: دارای بحث scaling به کرات است یعنی عرض یا طول یا عمق یک شبکه را از طریق دهم که در محاسبات scale از B1 تا B7 بوده و هر چه اشکال بیشتر، تعداد پارامتر بیشتر و پیچیده تر است یعنی می تواند در حالت های train کنیم

CNN Auto Encoder: ورودی و خروجی هر دو تصویر اند یعنی یک مابین است و ما به کلاس بندی نمی کنیم که فقط کار بردش کنیم که سبب افزایش کیفیت تصاویر و ناچیز بندی معنایی است



کلاس بندی: $T-L$ $Transfer Learning$ $Freeze$ کردن لایه ها: تا به $train$ را کوتاه تر کرده و سرعت اجرا بیشتر شود و معمولاً لایه های اولی که کل است را $freeze$ می کنند زیرا لایه های عمیق تر دارای ویژگی های جزئی تر است باید بهمانند! $(base_model.trainable = false)$ لایه های را خاموش کرد برای محاسبات ترتیب دستا مهم است مثل متن و سانسکری زمان، شبکات که تا الان خودنویس به درونی خود نیز ترتیب در آن ندارد

همه برابر بودن سانسکری و درونی شبکه با سانسکری تغییر داده شده بسیار مهم است حتی چنان که هم همین طور، یعنی شبکه ای که g_{v2} شده با g_{v1} را یکی RGB این که σ تعداد نوردهی برای 2 کلاس ها $= 1$ \Rightarrow فعال سازان: σ \Rightarrow تعداد نوردهی برای n کلاس ها $= n$ \Rightarrow فعال سازان: σ

یعنی برخی جاها به جای استخراج ویژگی های بیشتر، از $transfer learning$ استفاده می کنند که با آمدن $backbone$ آنی مثل 1000 می توان به کار برد و سانسکری را خاموش یا $backbone$ را از خاموش دریا

معادله های لایه ها: $u_{11} \rightarrow u_{12}$ $u_{12} \rightarrow u_{13}$ $u_{13} \rightarrow u_{14}$ $u_{14} \rightarrow u_{15}$ $u_{15} \rightarrow u_{16}$ $u_{16} \rightarrow u_{17}$ $u_{17} \rightarrow u_{18}$ $u_{18} \rightarrow u_{19}$ $u_{19} \rightarrow u_{20}$ $u_{20} \rightarrow u_{21}$ $u_{21} \rightarrow u_{22}$ $u_{22} \rightarrow u_{23}$ $u_{23} \rightarrow u_{24}$ $u_{24} \rightarrow u_{25}$ $u_{25} \rightarrow u_{26}$ $u_{26} \rightarrow u_{27}$ $u_{27} \rightarrow u_{28}$ $u_{28} \rightarrow u_{29}$ $u_{29} \rightarrow u_{30}$ $u_{30} \rightarrow u_{31}$ $u_{31} \rightarrow u_{32}$ $u_{32} \rightarrow u_{33}$ $u_{33} \rightarrow u_{34}$ $u_{34} \rightarrow u_{35}$ $u_{35} \rightarrow u_{36}$ $u_{36} \rightarrow u_{37}$ $u_{37} \rightarrow u_{38}$ $u_{38} \rightarrow u_{39}$ $u_{39} \rightarrow u_{40}$ $u_{40} \rightarrow u_{41}$ $u_{41} \rightarrow u_{42}$ $u_{42} \rightarrow u_{43}$ $u_{43} \rightarrow u_{44}$ $u_{44} \rightarrow u_{45}$ $u_{45} \rightarrow u_{46}$ $u_{46} \rightarrow u_{47}$ $u_{47} \rightarrow u_{48}$ $u_{48} \rightarrow u_{49}$ $u_{49} \rightarrow u_{50}$ $u_{50} \rightarrow u_{51}$ $u_{51} \rightarrow u_{52}$ $u_{52} \rightarrow u_{53}$ $u_{53} \rightarrow u_{54}$ $u_{54} \rightarrow u_{55}$ $u_{55} \rightarrow u_{56}$ $u_{56} \rightarrow u_{57}$ $u_{57} \rightarrow u_{58}$ $u_{58} \rightarrow u_{59}$ $u_{59} \rightarrow u_{60}$ $u_{60} \rightarrow u_{61}$ $u_{61} \rightarrow u_{62}$ $u_{62} \rightarrow u_{63}$ $u_{63} \rightarrow u_{64}$ $u_{64} \rightarrow u_{65}$ $u_{65} \rightarrow u_{66}$ $u_{66} \rightarrow u_{67}$ $u_{67} \rightarrow u_{68}$ $u_{68} \rightarrow u_{69}$ $u_{69} \rightarrow u_{70}$ $u_{70} \rightarrow u_{71}$ $u_{71} \rightarrow u_{72}$ $u_{72} \rightarrow u_{73}$ $u_{73} \rightarrow u_{74}$ $u_{74} \rightarrow u_{75}$ $u_{75} \rightarrow u_{76}$ $u_{76} \rightarrow u_{77}$ $u_{77} \rightarrow u_{78}$ $u_{78} \rightarrow u_{79}$ $u_{79} \rightarrow u_{80}$ $u_{80} \rightarrow u_{81}$ $u_{81} \rightarrow u_{82}$ $u_{82} \rightarrow u_{83}$ $u_{83} \rightarrow u_{84}$ $u_{84} \rightarrow u_{85}$ $u_{85} \rightarrow u_{86}$ $u_{86} \rightarrow u_{87}$ $u_{87} \rightarrow u_{88}$ $u_{88} \rightarrow u_{89}$ $u_{89} \rightarrow u_{90}$ $u_{90} \rightarrow u_{91}$ $u_{91} \rightarrow u_{92}$ $u_{92} \rightarrow u_{93}$ $u_{93} \rightarrow u_{94}$ $u_{94} \rightarrow u_{95}$ $u_{95} \rightarrow u_{96}$ $u_{96} \rightarrow u_{97}$ $u_{97} \rightarrow u_{98}$ $u_{98} \rightarrow u_{99}$ $u_{99} \rightarrow u_{100}$ $u_{100} \rightarrow u_{101}$ $u_{101} \rightarrow u_{102}$ $u_{102} \rightarrow u_{103}$ $u_{103} \rightarrow u_{104}$ $u_{104} \rightarrow u_{105}$ $u_{105} \rightarrow u_{106}$ $u_{106} \rightarrow u_{107}$ $u_{107} \rightarrow u_{108}$ $u_{108} \rightarrow u_{109}$ $u_{109} \rightarrow u_{110}$ $u_{110} \rightarrow u_{111}$ $u_{111} \rightarrow u_{112}$ $u_{112} \rightarrow u_{113}$ $u_{113} \rightarrow u_{114}$ $u_{114} \rightarrow u_{115}$ $u_{115} \rightarrow u_{116}$ $u_{116} \rightarrow u_{117}$ $u_{117} \rightarrow u_{118}$ $u_{118} \rightarrow u_{119}$ $u_{119} \rightarrow u_{120}$ $u_{120} \rightarrow u_{121}$ $u_{121} \rightarrow u_{122}$ $u_{122} \rightarrow u_{123}$ $u_{123} \rightarrow u_{124}$ $u_{124} \rightarrow u_{125}$ $u_{125} \rightarrow u_{126}$ $u_{126} \rightarrow u_{127}$ $u_{127} \rightarrow u_{128}$ $u_{128} \rightarrow u_{129}$ $u_{129} \rightarrow u_{130}$ $u_{130} \rightarrow u_{131}$ $u_{131} \rightarrow u_{132}$ $u_{132} \rightarrow u_{133}$ $u_{133} \rightarrow u_{134}$ $u_{134} \rightarrow u_{135}$ $u_{135} \rightarrow u_{136}$ $u_{136} \rightarrow u_{137}$ $u_{137} \rightarrow u_{138}$ $u_{138} \rightarrow u_{139}$ $u_{139} \rightarrow u_{140}$ $u_{140} \rightarrow u_{141}$ $u_{141} \rightarrow u_{142}$ $u_{142} \rightarrow u_{143}$ $u_{143} \rightarrow u_{144}$ $u_{144} \rightarrow u_{145}$ $u_{145} \rightarrow u_{146}$ $u_{146} \rightarrow u_{147}$ $u_{147} \rightarrow u_{148}$ $u_{148} \rightarrow u_{149}$ $u_{149} \rightarrow u_{150}$ $u_{150} \rightarrow u_{151}$ $u_{151} \rightarrow u_{152}$ $u_{152} \rightarrow u_{153}$ $u_{153} \rightarrow u_{154}$ $u_{154} \rightarrow u_{155}$ $u_{155} \rightarrow u_{156}$ $u_{156} \rightarrow u_{157}$ $u_{157} \rightarrow u_{158}$ $u_{158} \rightarrow u_{159}$ $u_{159} \rightarrow u_{160}$ $u_{160} \rightarrow u_{161}$ $u_{161} \rightarrow u_{162}$ $u_{162} \rightarrow u_{163}$ $u_{163} \rightarrow u_{164}$ $u_{164} \rightarrow u_{165}$ $u_{165} \rightarrow u_{166}$ $u_{166} \rightarrow u_{167}$ $u_{167} \rightarrow u_{168}$ $u_{168} \rightarrow u_{169}$ $u_{169} \rightarrow u_{170}$ $u_{170} \rightarrow u_{171}$ $u_{171} \rightarrow u_{172}$ $u_{172} \rightarrow u_{173}$ $u_{173} \rightarrow u_{174}$ $u_{174} \rightarrow u_{175}$ $u_{175} \rightarrow u_{176}$ $u_{176} \rightarrow u_{177}$ $u_{177} \rightarrow u_{178}$ $u_{178} \rightarrow u_{179}$ $u_{179} \rightarrow u_{180}$ $u_{180} \rightarrow u_{181}$ $u_{181} \rightarrow u_{182}$ $u_{182} \rightarrow u_{183}$ $u_{183} \rightarrow u_{184}$ $u_{184} \rightarrow u_{185}$ $u_{185} \rightarrow u_{186}$ $u_{186} \rightarrow u_{187}$ $u_{187} \rightarrow u_{188}$ $u_{188} \rightarrow u_{189}$ $u_{189} \rightarrow u_{190}$ $u_{190} \rightarrow u_{191}$ $u_{191} \rightarrow u_{192}$ $u_{192} \rightarrow u_{193}$ $u_{193} \rightarrow u_{194}$ $u_{194} \rightarrow u_{195}$ $u_{195} \rightarrow u_{196}$ $u_{196} \rightarrow u_{197}$ $u_{197} \rightarrow u_{198}$ $u_{198} \rightarrow u_{199}$ $u_{199} \rightarrow u_{200}$ $u_{200} \rightarrow u_{201}$ $u_{201} \rightarrow u_{202}$ $u_{202} \rightarrow u_{203}$ $u_{203} \rightarrow u_{204}$ $u_{204} \rightarrow u_{205}$ $u_{205} \rightarrow u_{206}$ $u_{206} \rightarrow u_{207}$ $u_{207} \rightarrow u_{208}$ $u_{208} \rightarrow u_{209}$ $u_{209} \rightarrow u_{210}$ $u_{210} \rightarrow u_{211}$ $u_{211} \rightarrow u_{212}$ $u_{212} \rightarrow u_{213}$ $u_{213} \rightarrow u_{214}$ $u_{214} \rightarrow u_{215}$ $u_{215} \rightarrow u_{216}$ $u_{216} \rightarrow u_{217}$ $u_{217} \rightarrow u_{218}$ $u_{218} \rightarrow u_{219}$ $u_{219} \rightarrow u_{220}$ $u_{220} \rightarrow u_{221}$ $u_{221} \rightarrow u_{222}$ $u_{222} \rightarrow u_{223}$ $u_{223} \rightarrow u_{224}$ $u_{224} \rightarrow u_{225}$ $u_{225} \rightarrow u_{226}$ $u_{226} \rightarrow u_{227}$ $u_{227} \rightarrow u_{228}$ $u_{228} \rightarrow u_{229}$ $u_{229} \rightarrow u_{230}$ $u_{230} \rightarrow u_{231}$ $u_{231} \rightarrow u_{232}$ $u_{232} \rightarrow u_{233}$ $u_{233} \rightarrow u_{234}$ $u_{234} \rightarrow u_{235}$ $u_{235} \rightarrow u_{236}$ $u_{236} \rightarrow u_{237}$ $u_{237} \rightarrow u_{238}$ $u_{238} \rightarrow u_{239}$ $u_{239} \rightarrow u_{240}$ $u_{240} \rightarrow u_{241}$ $u_{241} \rightarrow u_{242}$ $u_{242} \rightarrow u_{243}$ $u_{243} \rightarrow u_{244}$ $u_{244} \rightarrow u_{245}$ $u_{245} \rightarrow u_{246}$ $u_{246} \rightarrow u_{247}$ $u_{247} \rightarrow u_{248}$ $u_{248} \rightarrow u_{249}$ $u_{249} \rightarrow u_{250}$ $u_{250} \rightarrow u_{251}$ $u_{251} \rightarrow u_{252}$ $u_{252} \rightarrow u_{253}$ $u_{253} \rightarrow u_{254}$ $u_{254} \rightarrow u_{255}$ $u_{255} \rightarrow u_{256}$ $u_{256} \rightarrow u_{257}$ $u_{257} \rightarrow u_{258}$ $u_{258} \rightarrow u_{259}$ $u_{259} \rightarrow u_{260}$ $u_{260} \rightarrow u_{261}$ $u_{261} \rightarrow u_{262}$ $u_{262} \rightarrow u_{263}$ $u_{263} \rightarrow u_{264}$ $u_{264} \rightarrow u_{265}$ $u_{265} \rightarrow u_{266}$ $u_{266} \rightarrow u_{267}$ $u_{267} \rightarrow u_{268}$ $u_{268} \rightarrow u_{269}$ $u_{269} \rightarrow u_{270}$ $u_{270} \rightarrow u_{271}$ $u_{271} \rightarrow u_{272}$ $u_{272} \rightarrow u_{273}$ $u_{273} \rightarrow u_{274}$ $u_{274} \rightarrow u_{275}$ $u_{275} \rightarrow u_{276}$ $u_{276} \rightarrow u_{277}$ $u_{277} \rightarrow u_{278}$ $u_{278} \rightarrow u_{279}$ $u_{279} \rightarrow u_{280}$ $u_{280} \rightarrow u_{281}$ $u_{281} \rightarrow u_{282}$ $u_{282} \rightarrow u_{283}$ $u_{283} \rightarrow u_{284}$ $u_{284} \rightarrow u_{285}$ $u_{285} \rightarrow u_{286}$ $u_{286} \rightarrow u_{287}$ $u_{287} \rightarrow u_{288}$ $u_{288} \rightarrow u_{289}$ $u_{289} \rightarrow u_{290}$ $u_{290} \rightarrow u_{291}$ $u_{291} \rightarrow u_{292}$ $u_{292} \rightarrow u_{293}$ $u_{293} \rightarrow u_{294}$ $u_{294} \rightarrow u_{295}$ $u_{295} \rightarrow u_{296}$ $u_{296} \rightarrow u_{297}$ $u_{297} \rightarrow u_{298}$ $u_{298} \rightarrow u_{299}$ $u_{299} \rightarrow u_{300}$ $u_{300} \rightarrow u_{301}$ $u_{301} \rightarrow u_{302}$ $u_{302} \rightarrow u_{303}$ $u_{303} \rightarrow u_{304}$ $u_{304} \rightarrow u_{305}$ $u_{305} \rightarrow u_{306}$ $u_{306} \rightarrow u_{307}$ $u_{307} \rightarrow u_{308}$ $u_{308} \rightarrow u_{309}$ $u_{309} \rightarrow u_{310}$ $u_{310} \rightarrow u_{311}$ $u_{311} \rightarrow u_{312}$ $u_{312} \rightarrow u_{313}$ $u_{313} \rightarrow u_{314}$ $u_{314} \rightarrow u_{315}$ $u_{315} \rightarrow u_{316}$ $u_{316} \rightarrow u_{317}$ $u_{317} \rightarrow u_{318}$ $u_{318} \rightarrow u_{319}$ $u_{319} \rightarrow u_{320}$ $u_{320} \rightarrow u_{321}$ $u_{321} \rightarrow u_{322}$ $u_{322} \rightarrow u_{323}$ $u_{323} \rightarrow u_{324}$ $u_{324} \rightarrow u_{325}$ $u_{325} \rightarrow u_{326}$ $u_{326} \rightarrow u_{327}$ $u_{327} \rightarrow u_{328}$ $u_{328} \rightarrow u_{329}$ $u_{329} \rightarrow u_{330}$ $u_{330} \rightarrow u_{331}$ $u_{331} \rightarrow u_{332}$ $u_{332} \rightarrow u_{333}$ $u_{333} \rightarrow u_{334}$ $u_{334} \rightarrow u_{335}$ $u_{335} \rightarrow u_{336}$ $u_{336} \rightarrow u_{337}$ $u_{337} \rightarrow u_{338}$ $u_{338} \rightarrow u_{339}$ $u_{339} \rightarrow u_{340}$ $u_{340} \rightarrow u_{341}$ $u_{341} \rightarrow u_{342}$ $u_{342} \rightarrow u_{343}$ $u_{343} \rightarrow u_{344}$ $u_{344} \rightarrow u_{345}$ $u_{345} \rightarrow u_{346}$ $u_{346} \rightarrow u_{347}$ $u_{347} \rightarrow u_{348}$ $u_{348} \rightarrow u_{349}$ $u_{349} \rightarrow u_{350}$ $u_{350} \rightarrow u_{351}$ $u_{351} \rightarrow u_{352}$ $u_{352} \rightarrow u_{353}$ $u_{353} \rightarrow u_{354}$ $u_{354} \rightarrow u_{355}$ $u_{355} \rightarrow u_{356}$ $u_{356} \rightarrow u_{357}$ $u_{357} \rightarrow u_{358}$ $u_{358} \rightarrow u_{359}$ $u_{359} \rightarrow u_{360}$ $u_{360} \rightarrow u_{361}$ $u_{361} \rightarrow u_{362}$ $u_{362} \rightarrow u_{363}$ $u_{363} \rightarrow u_{364}$ $u_{364} \rightarrow u_{365}$ $u_{365} \rightarrow u_{366}$ $u_{366} \rightarrow u_{367}$ $u_{367} \rightarrow u_{368}$ $u_{368} \rightarrow u_{369}$ $u_{369} \rightarrow u_{370}$ $u_{370} \rightarrow u_{371}$ $u_{371} \rightarrow u_{372}$ $u_{372} \rightarrow u_{373}$ $u_{373} \rightarrow u_{374}$ $u_{374} \rightarrow u_{375}$ $u_{375} \rightarrow u_{376}$ $u_{376} \rightarrow u_{377}$ $u_{377} \rightarrow u_{378}$ $u_{378} \rightarrow u_{379}$ $u_{379} \rightarrow u_{380}$ $u_{380} \rightarrow u_{381}$ $u_{381} \rightarrow u_{382}$ $u_{382} \rightarrow u_{383}$ $u_{383} \rightarrow u_{384}$ $u_{384} \rightarrow u_{385}$ $u_{385} \rightarrow u_{386}$ $u_{386} \rightarrow u_{387}$ $u_{387} \rightarrow u_{388}$ $u_{388} \rightarrow u_{389}$ $u_{389} \rightarrow u_{390}$ $u_{390} \rightarrow u_{391}$ $u_{391} \rightarrow u_{392}$ $u_{392} \rightarrow u_{393}$ $u_{393} \rightarrow u_{394}$ $u_{394} \rightarrow u_{395}$ $u_{395} \rightarrow u_{396}$ $u_{396} \rightarrow u_{397}$ $u_{397} \rightarrow u_{398}$ $u_{398} \rightarrow u_{399}$ $u_{399} \rightarrow u_{400}$ $u_{400} \rightarrow u_{401}$ $u_{401} \rightarrow u_{402}$ $u_{402} \rightarrow u_{403}$ $u_{403} \rightarrow u_{404}$ $u_{404} \rightarrow u_{405}$ $u_{405} \rightarrow u_{406}$ $u_{406} \rightarrow u_{407}$ $u_{407} \rightarrow u_{408}$ $u_{408} \rightarrow u_{409}$ $u_{409} \rightarrow u_{410}$ $u_{410} \rightarrow u_{411}$ $u_{411} \rightarrow u_{412}$ $u_{412} \rightarrow u_{413}$ $u_{413} \rightarrow u_{414}$ $u_{414} \rightarrow u_{415}$ $u_{415} \rightarrow u_{416}$ $u_{416} \rightarrow u_{417}$ $u_{417} \rightarrow u_{418}$ $u_{418} \rightarrow u_{419}$ $u_{419} \rightarrow u_{420}$ $u_{420} \rightarrow u_{421}$ $u_{421} \rightarrow u_{422}$ $u_{422} \rightarrow u_{423}$ $u_{423} \rightarrow u_{424}$ $u_{424} \rightarrow u_{425}$ $u_{425} \rightarrow u_{426}$ $u_{426} \rightarrow u_{427}$ $u_{427} \rightarrow u_{428}$ $u_{428} \rightarrow u_{429}$ $u_{429} \rightarrow u_{430}$ $u_{430} \rightarrow u_{431}$ $u_{431} \rightarrow u_{432}$ $u_{432} \rightarrow u_{433}$ $u_{433} \rightarrow u_{434}$ $u_{434} \rightarrow u_{435}$ $u_{435} \rightarrow u_{436}$ $u_{436} \rightarrow u_{437}$ $u_{437} \rightarrow u_{438}$ $u_{438} \rightarrow u_{439}$ $u_{439} \rightarrow u_{440}$ $u_{440} \rightarrow u_{441}$ $u_{441} \rightarrow u_{442}$ $u_{442} \rightarrow u_{443}$ $u_{443} \rightarrow u_{444}$ $u_{444} \rightarrow u_{445}$ $u_{445} \rightarrow u_{446}$ $u_{446} \rightarrow u_{447}$ $u_{447} \rightarrow u_{448}$ $u_{448} \rightarrow u_{449}$ $u_{449} \rightarrow u_{450}$ $u_{450} \rightarrow u_{451}$ $u_{451} \rightarrow u_{452}$ $u_{452} \rightarrow u_{453}$ $u_{453} \rightarrow u_{454}$ $u_{454} \rightarrow u_{455}$ $u_{455} \rightarrow u_{456}$ $u_{456} \rightarrow u_{457}$ $u_{457} \rightarrow u_{458}$ $u_{458} \rightarrow u_{459}$ $u_{459} \rightarrow u_{460}$ $u_{460} \rightarrow u_{461}$ $u_{461} \rightarrow u_{462}$ $u_{462} \rightarrow u_{463}$ $u_{463} \rightarrow u_{464}$ $u_{464} \rightarrow u_{465}$ $u_{465} \rightarrow u_{466}$ $u_{466} \rightarrow u_{467}$ $u_{467} \rightarrow u_{468}$ $u_{468} \rightarrow u_{469}$ $u_{469} \rightarrow u_{470}$ $u_{470} \rightarrow u_{471}$ $u_{471} \rightarrow u_{472}$ $u_{472} \rightarrow u_{473}$ $u_{473} \rightarrow u_{474}$ $u_{474} \rightarrow u_{475}$ $u_{475} \rightarrow u_{476}$ $u_{476} \rightarrow u_{477}$ $u_{477} \rightarrow u_{478}$ $u_{478} \rightarrow u_{479}$ $u_{479} \rightarrow u_{480}$ $u_{480} \rightarrow u_{481}$ $u_{481} \rightarrow u_{482}$ $u_{482} \rightarrow u_{483}$ $u_{483} \rightarrow u_{484}$ $u_{484} \rightarrow u_{485}$ $u_{485} \rightarrow u_{486}$ $u_{486} \rightarrow u_{487}$ $u_{487} \rightarrow u_{488}$ $u_{488} \rightarrow u_{489}$ $u_{489} \rightarrow u_{490}$ $u_{490} \rightarrow u_{491}$ $u_{491} \rightarrow u_{492}$ $u_{492} \rightarrow u_{493}$ $u_{493} \rightarrow u_{494}$ $u_{494} \rightarrow u_{495}$ $u_{495} \rightarrow u_{496}$ $u_{496} \rightarrow u_{497}$ $u_{497} \rightarrow u_{498}$ $u_{498} \rightarrow u_{499}$ $u_{499} \rightarrow u_{500}$ $u_{500} \rightarrow u_{501}$ $u_{501} \rightarrow u_{502}$ $u_{502} \rightarrow u_{503}$ $u_{503} \rightarrow u_{504}$ $u_{504} \rightarrow u_{505}$ $u_{505} \rightarrow u_{506}$ $u_{506} \rightarrow u_{507}$ $u_{507} \rightarrow u_{508}$ $u_{508} \rightarrow u_{509}$ $u_{509} \rightarrow u_{510}$ $u_{510} \rightarrow u_{511}$ $u_{511} \rightarrow u_{512}$ $u_{512} \rightarrow u_{513}$ $u_{513} \rightarrow u_{514}$ $u_{514} \rightarrow u_{515}$ $u_{515} \rightarrow u_{516}$ $u_{516} \rightarrow u_{517}$ $u_{517} \rightarrow u_{518}$ $u_{518} \rightarrow u_{519}$ $u_{519} \rightarrow u_{520}$ $u_{520} \rightarrow u_{521}$ $u_{521} \rightarrow u_{522}$ $u_{522} \rightarrow u_{523}$ $u_{523} \rightarrow u_{524}$ $u_{524} \rightarrow u_{525}$ $u_{525} \rightarrow u_{526}$ $u_{526} \rightarrow u_{527}$ $u_{527} \rightarrow u_{528}$ $u_{528} \rightarrow u_{529}$ $u_{529} \rightarrow u_{530}$ $u_{530} \rightarrow u_{531}$ $u_{531} \rightarrow u_{532}$ $u_{532} \rightarrow u_{533}$ $u_{533} \rightarrow u_{534}$ $u_{534} \rightarrow u_{535}$ $u_{535} \rightarrow u_{536}$ $u_{536} \rightarrow u_{537}$ $u_{537} \rightarrow u_{538}$ $u_{538} \rightarrow u_{539}$ $u_{539} \rightarrow u_{540}$ $u_{540} \rightarrow u_{541}$ $u_{541} \rightarrow u_{542}$ $u_{542} \rightarrow u_{543}$ $u_{543} \rightarrow u_{544}$ $u_{544} \rightarrow u_{545}$ $u_{545} \rightarrow u_{546}$ $u_{546} \rightarrow u_{547}$ $u_{547} \rightarrow u_{548}$ $u_{548} \rightarrow u_{549}$ $u_{549} \rightarrow u_{550}$ $u_{550} \rightarrow u_{551}$ $u_{551} \rightarrow u_{552}$ $u_{552} \rightarrow u_{553}$ $u_{553} \rightarrow u_{554}$ $u_{554} \rightarrow u_{555}$ $u_{555} \rightarrow u_{556}$ $u_{556} \rightarrow u_{557}$ $u_{557} \rightarrow u_{558}$ $u_{558} \rightarrow u_{559}$ $u_{559} \rightarrow u_{560}$ $u_{560} \rightarrow u_{561}$ $u_{561} \rightarrow u_{562}$ $u_{562} \rightarrow u_{563}$ $u_{563} \rightarrow u_{564}$ $u_{564} \rightarrow u_{565}$ $u_{565} \rightarrow u_{566}$ $u_{566} \rightarrow u_{567}$ $u_{567} \rightarrow u_{568}$ $u_{568} \rightarrow u_{569}$ $u_{569} \rightarrow u_{570}$ $u_{570} \rightarrow u_{571}$ $u_{571} \rightarrow u_{572}$ $u_{572} \rightarrow u_{573}$ $u_{573} \rightarrow u_{574}$ u_{574}

```

# from tensorflow.keras.applications import MobileNetV2

# # ترنسفر لرن بدون هد و فقط بک بون و تعیین اندازه ورودی
# # بدون لایه‌های بالایی و تعیین اندازه ورودی MobileNet بارگذاری مدل
# 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 of classes = 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, # minimum 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)

```

```

# axis ==> 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 شبکه
- 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 آموزش بیس مدل تا ترنسفر لرن و فاین تیون کردن آن شهر داده شد و کدهای آن 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 فرایند
# بمانند.
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 **"""

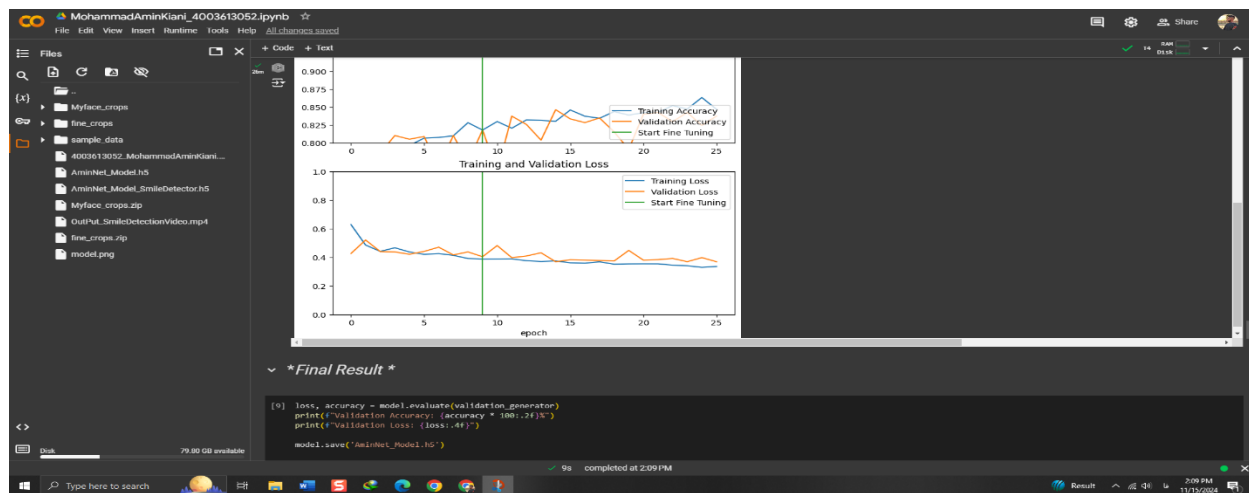
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
print(f"Validation Loss: {loss:.4f}")

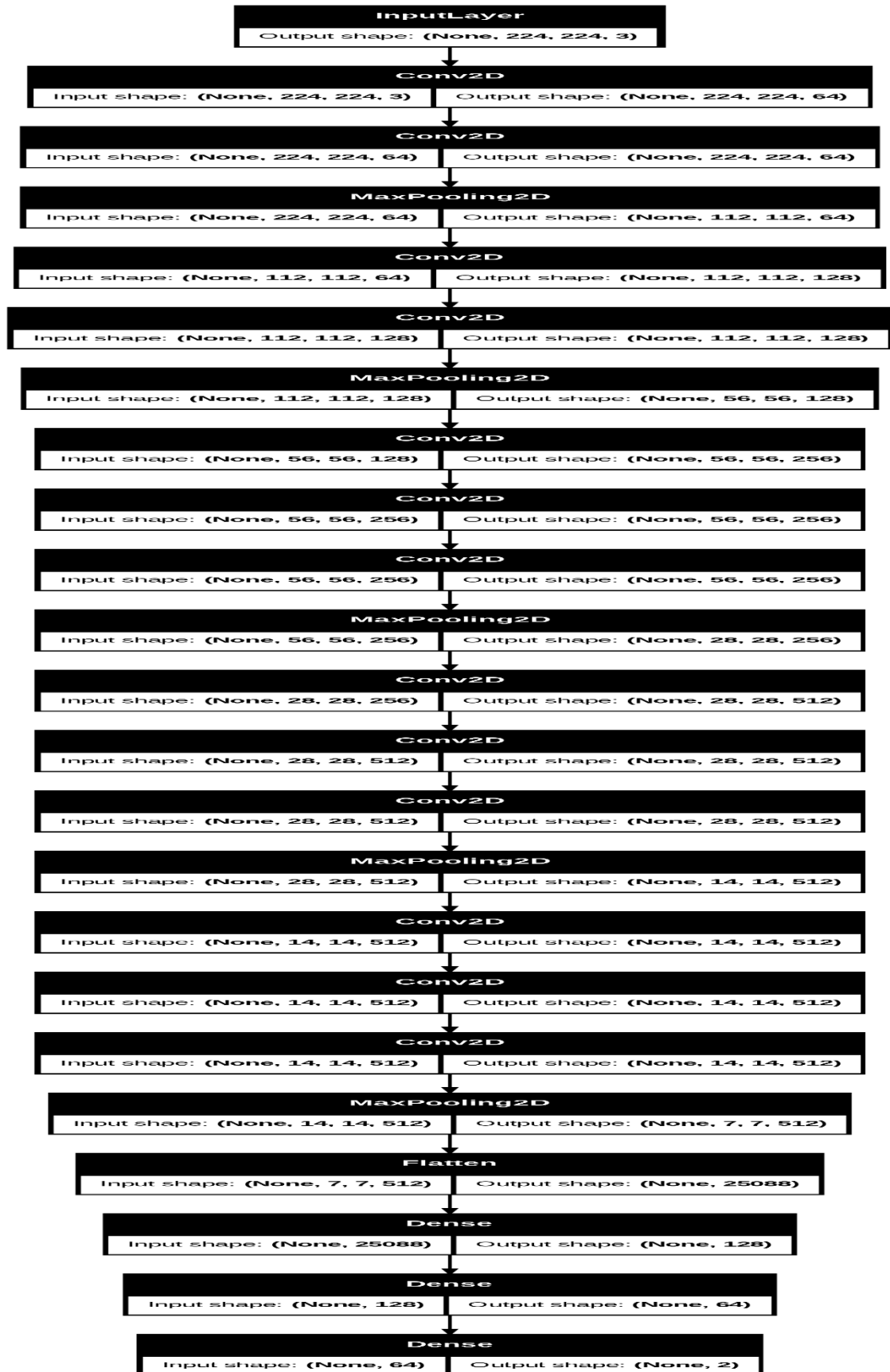
model.save('AminNet_Model.h5')

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/'

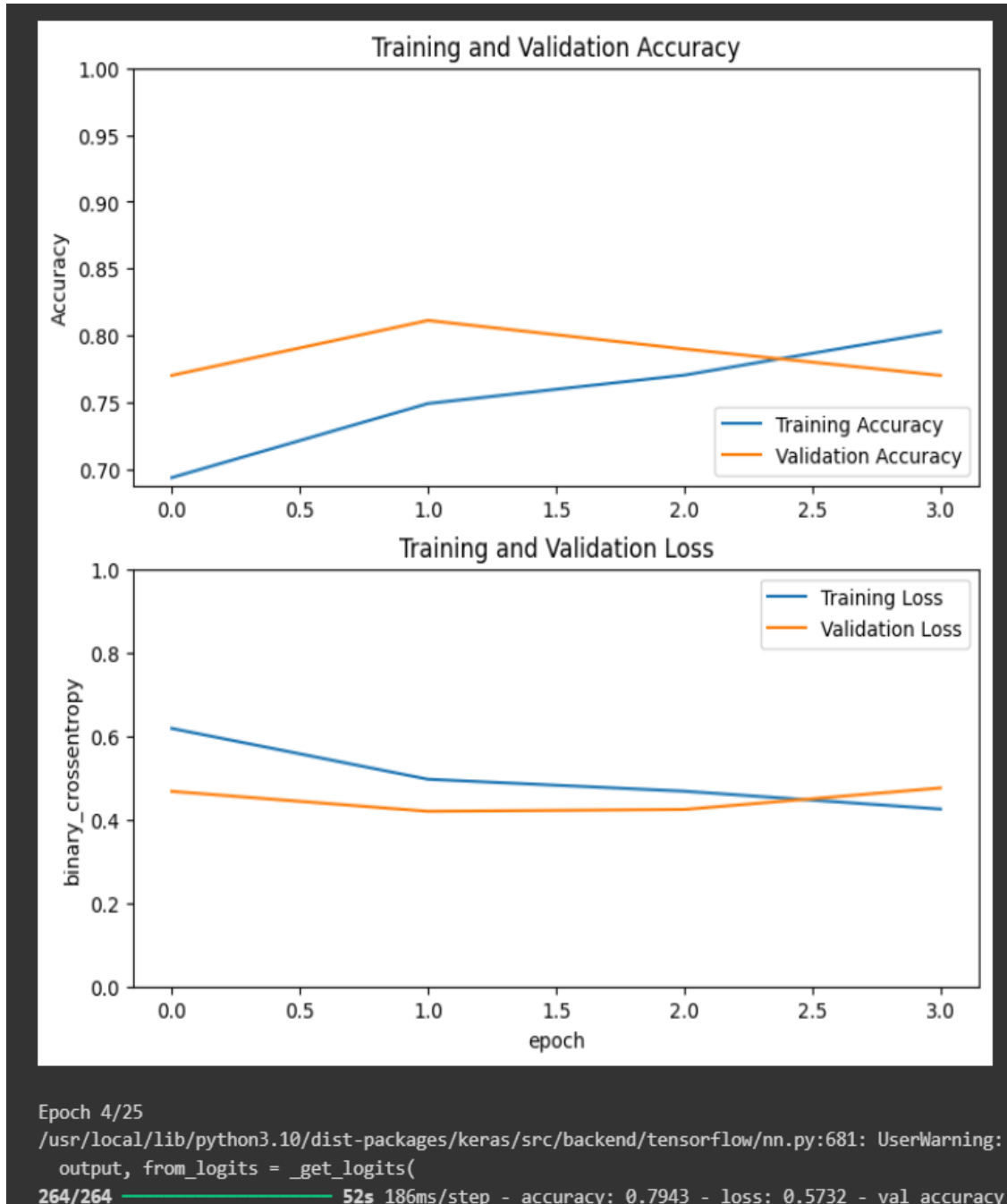
```

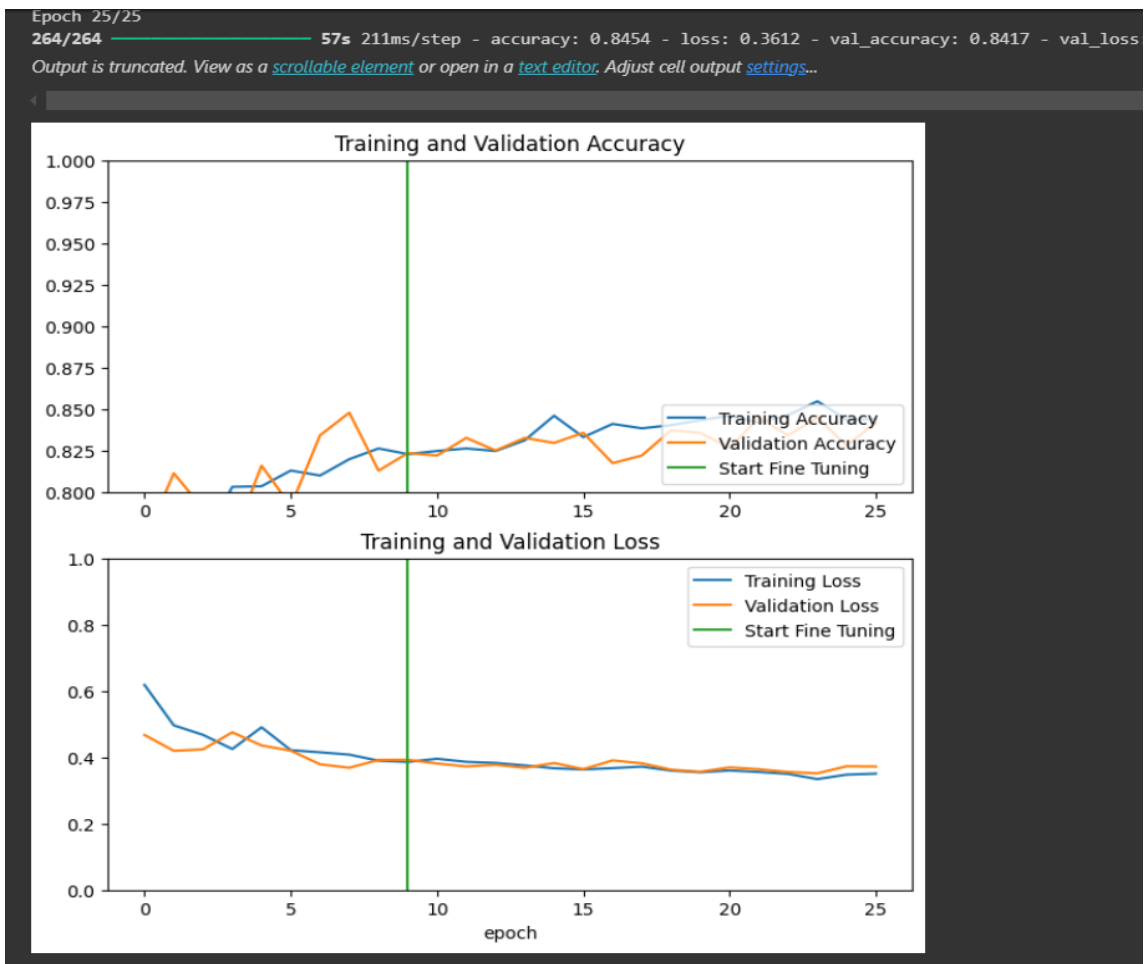




7- رسم نمودارها و ارتباطات:

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





قبل از فاین تیون کردن:

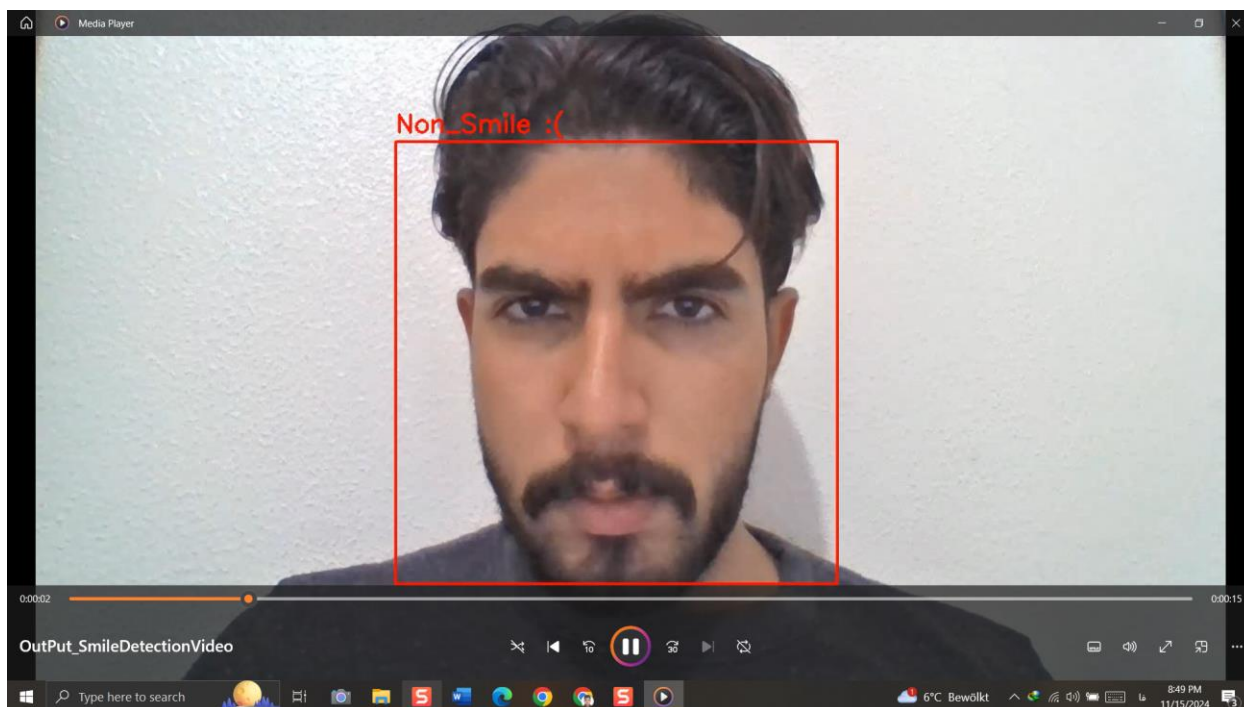
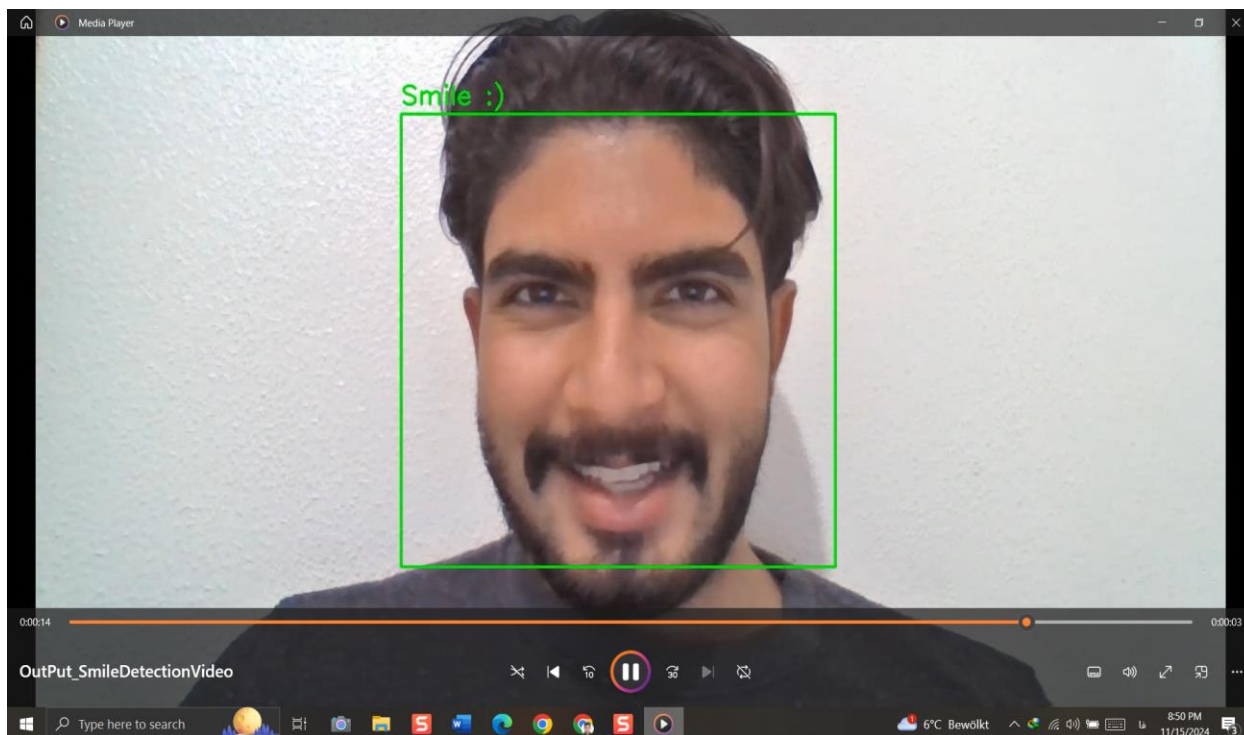
```
66/66 9s 121ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This
precision recall f1-score support
0 0.77 0.78 0.77 307
1 0.80 0.80 0.80 350

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(model)`
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.harcascades +
'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 آپلود برای تجزیه و تحلیل ویدیو

۲.۱. راه‌اندازی وبکم یا آپلود 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.

۲. 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 شناسایی شده، برش صورت انجام می‌شود و اندازه آن به 224x224 پیکسل تغییر می‌کند.
- پیش‌پردازش تصویر برش خورده 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 مشخص شده.

8. پیش‌بینی لبخند

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

- پیش‌بینی لبخند یا عدم لبخند با استفاده از مدل بارگذاری شده.

9. رسم مستطیل و برجسب

```
:if prediction[0][0] > 0.5
```

```
(255 ,0 ,0) = color
```

"): label = "Non_Smile

```
:else
```

```
(0 ,255 ,0) = color
```

" (: 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.

- برجسب 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

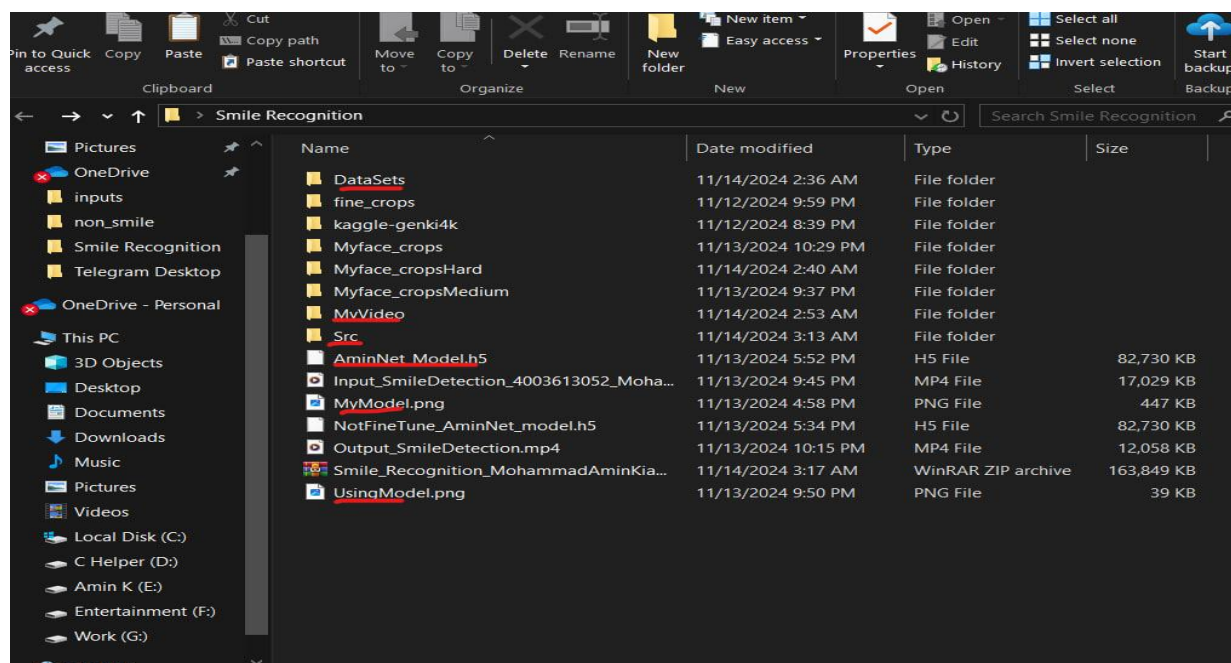
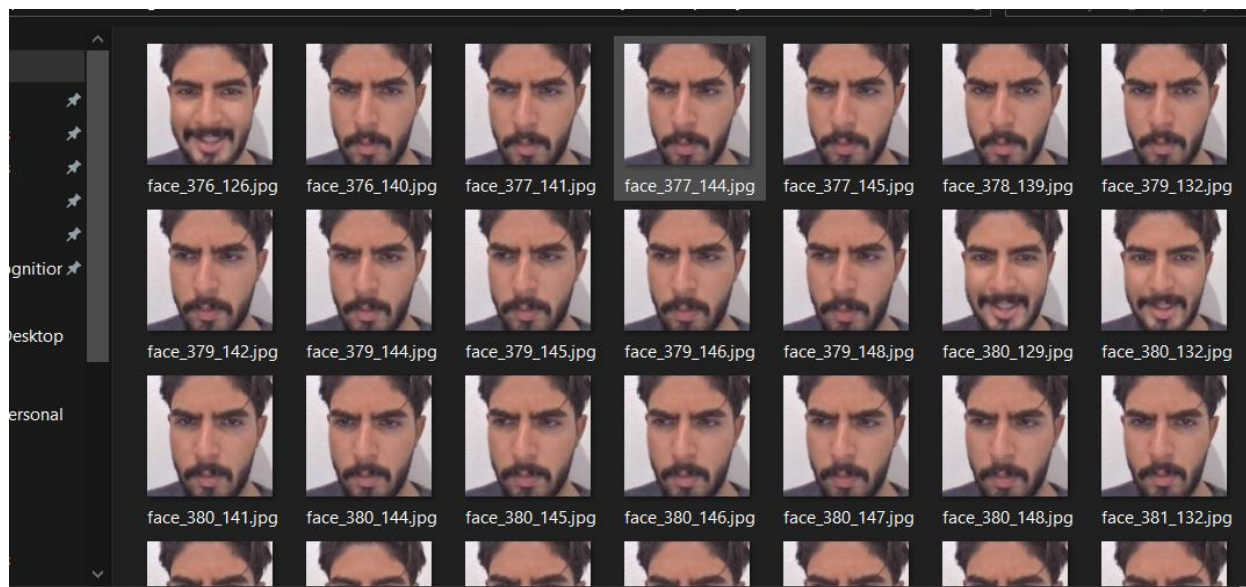
shutil.make_archive('/content/Myface_crops', 'zip', '/content/Myface_crops')

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.



- [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>