دانشکده مهندسی

گروه کامپیوتر

تشخیص احساس صورت بر بستر عکس

پایان نامه کارشناسی در رشته مهندسی کامپیوتر

سعید بهمنی سنگسری

استاد پروژه

دکتر سید مجتبی روحانی

استاد داور

بهار 1400


بسم الله الرحمن الرحيم

قدردانی

از زحمات بی دریغ استاد راهنمای خود ، جناب آقای دکتر سید مجتبی روحانی کمال تشکر و قدردانی را داشته که بی تردید بدون حمایت ها و کمک های بی دریغ ایشان انجام این امر میسر نبود.

چکیده

امروزه با توجه به پیشرفت تکنولوژی،افراد جامعه به سمت تنها زندگی کردن پیش میروند، بنابراین یکی از وظایف مهم امروزه این دولت این است که به حالات روحی شهروندان توجه ویژه ای داشته باشد که بتواند از مشکلات پیش رو پیشگیری کنند.یکی از روش ها برای تشخیص این حالت ها،نگاه کردن به صورت افراد است.در این پروژه من با استفاده از دیتاست **fer2013** احساس صورت افراد را تشخصی میدهم.

فهرست

چالش ها در انتخاب دیتاست

وقت زیادی برای پیدا کردن دیتاست مناسب برای پروژه تعریف شده صرف شد که در ادامه با آن ها آشنا میشویم

AffectNet

این دیتاست یکی از محبوب ترین دیتاست برای تشخیص حالت صورت است.این دیتاست شامل نزدیک یک میلیون عکس هست.این عکس ها از طریق موتورهای جست و جو جمع آوری شده است و تقریبا حدود 450 هزار از این عکس ها به وسیله 12 ناظر حرفه ای طبقه بندی شده است

لینک دانلود

Extended Cohn-Kanade Dataset (CK+)

این دیتاست شامل 5876 عکس از 123 نفر میباشد.عکس ها در 7 گروه طبقه بنده شده اند و پس زمینه ثابت دارند.

لینک دانلود

FER-2013

این دیتاست به صورت خیلی وسیع برای دیتاست احساسی (emotion dataset) استفاده میشود.در این دیتاست عکس ها به 7 طبقه دسته بندی شده اند.این دیتاست شامل 35000 تا عکس هست.این عکس ها از گوگل جمع آوری شده اند و در عکس ها تنوع زیادی از نظر سن،محل قرار گیری صورت دارد.

لینک دانلود

EMOTIC

این دیتاست توسط مردم بوسیله گرفتن عکس از محیط واقعی جمع آوری شده است و در 26 دسته طبقه بندی شده است. این دیتاست شامل 18313 عکس که بوسیله 23788 مردم حاشیه نویسی شده است.بعضی از این عکس ها از گوگل جمع آوری شده است.

لینک دانلود

بعد از اینکه این دیتاست ها را پیدا کردم،نوبت به این رسید که کدام یکی مناسب تراست.حال دلیل رد کردن دیتاست ها را بیان میکنم.

- در دیتاست **EMOTIC** صورت افراد به درستی معلوم نبود و تعداد عکس ها برای هر دسته موقع ترین کردن کم بود.

- در دیتاست **Extended Cohn-Kanade Dataset (CK+)** اندازه دیتاست خیلی کوچک بود و پس زمینه عکس ها و زاویه صورت همه افراد در عکس ثابت بود.در ضمن تعداد افراد مختلف در دیتاست فقط 123 نفر بود که مناسب پروژه تعریف شده نبود.

پس از رد این 2 دیتاست تصمیم بر این شد که از دیتاست **AffectNet** یا **FER-2013** استفاده شود.

برای استفاده و دانلود کردن دیتاست **AffectNet** باید درخواست در سایت که لینک آن در قبل قرار داده شده بود ثبت میشد که متاسفانه پاسخی دریافت نشد.پس در نتیجه تصمیم بر این شد که از دیتاست **FER-2013** استفاده شود.

روش پیشنهادی و الگوریتم

حال به اینجا میرسیم که برای انجام این کار باید از چه راهی رفت، با توجه به اینکه کار بر روی عکس هست و باید به قسمت های ریز عکس توجه کرد از Convolutional neural networks یا به مخفف CNN استفاده شد.

برای دانلود و استفاده از از دیتاست **FER-2013** دو روش موجود بود.

1. استفاده از دیتاست که در یک csv گردآوری شده بود
2. استفاده از دیتاست به صورت عکس آماده شده

در این قسمت برای اینکه با ماهیت عکس آشنا شوم و بیشتر صورت مسئله را درک کنم از روش 2 استفاده کردم. بعد از انتخاب نوع دیتاست **FER-2013** حال به جزئیات میپردازیم.

برای استفاده از CNN از کتابخانه TensorFlow استفاده کردم، دلیل استفاده از این کتابخانه این بود که از رقبا در همه لحاظ بهتر وبهینه تر بود.

کتابخانه TensorFlow برای آموزش دادن شبکه هم میتواند از CPU یا GPU استفاده کند که با توجه به سیستم مورد استفاده کاربر،کاربر نسخه مربوطه را میریزد.در این جا باید اشاره کنم که برای استفاده از نسخه GPU با توجه به پردازنده گرافیکی سیستم باید چند نرم افزار خاص نصب شود تا بتواند کار کند.پردازنده گرافیکی بنده NVIDIA بود و برای استفاده از نسخه GPU-TensorFlow باید Cudnn و Cuda را نصب میکردم.

برای یادگیری CNN از کورس A Complete Guide on TensorFlow 2.0 using Keras API

استفاده کردم و پایه های پروژه را بر اساس آن ساختم.

در این جا به کارهای اولیه قبل معماری CNN میپردازیم.چون دیتاست رو به صورت روش 2 استفاده کردم برای تنظیم کردن داده های Train و Test از ImageDataGenerator استفاده شد در کانستراکتور آن مقادیر مختلفی را میتوان تنظیم کرد ولی من فقط rescale را تنظیم کردم.

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

در ادامه باید از متد flow_from_directory استفاده کرد که در زیر میتوان مقادیر پارامتر ها را مشاهده کرد.

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(input_target, input_target),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical',
    shuffle=True)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(input_target, input_target),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical',
    shuffle=True
)
```

- در train_dir مسیرپوشه داده های train تنظیم میشود
- در قسمت target_size سایز تصویر را وارد میکنیم که در اینجا برابر با 48 در 48 است.نکته جالبی که در اینجا هست،این است که وقتی target_size تنظیم میشود همه عکس های آن دایرکتوری به اندازه ست شده تغییر سایز داده میشوند که در بعضی جاها در وقت صرفه جویی میشود.

- چون عکس های خاکستری و تک لایه هستند در قسمت color_mode، grayscale ست شده است.
- در نهایت با توجه به اینکه داده ها در 7 کلاس هستند class_mode، caregorical ست شده است.

حال در این قسمت به معماری CNN میپردازیم.

معماری در کد به شکل زیر است:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=3, activation='relu',
input_shape=(input_target,input_target,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))



model.add(Flatten())

model.add(Dense(1024, activation='relu'))
```

```
model.add(Dropout(0.4))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(7, activation='softmax'))
```

و به صورت ریزتر به صورت زیر است:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 23, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 23, 23, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
```

```
conv2d_4 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 128)       147584
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 11, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              3277824
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 1799
=================================================================
Total params: 4,223,847
Trainable params: 4,223,015
Non-trainable params: 832
_________________________________________________________________
```

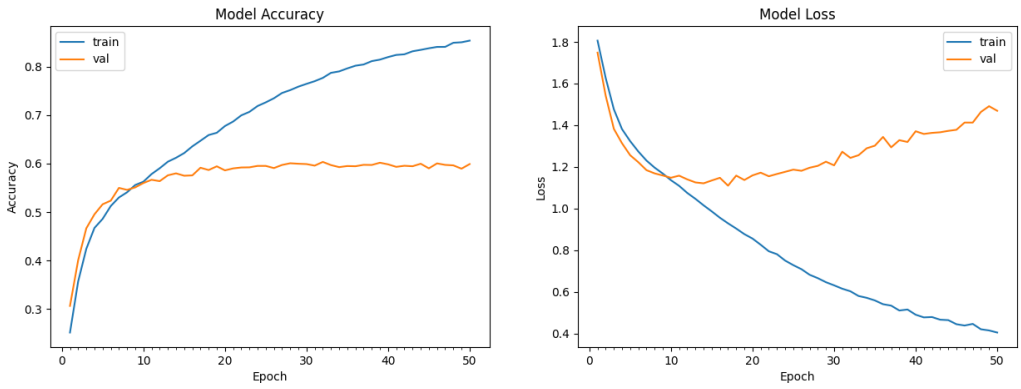
ایده اولیه معماری شبکه از کورس Udemy که در قبل اشاره کردم بود و در آنجا گفته شده بود برای لایه ها معمولا از ضریب 2 استفاده میکنیم.برای معماری شبکه از آزمون خطا استفاده شد در ادامه معماری های مختلف که استفاده شده را نشان میدهیم.

معماری با دقت 59.89%

معماری زیر دقت 59.89 بدست آورد.

```python
# Create the model

model = Sequential()

model.add(Conv2D(32, kernel_size=3, padding="same", activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(32, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(32, kernel_size=3,padding="same", activation='relu'))
model.add(Conv2D(32, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(128, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(2028, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))
```

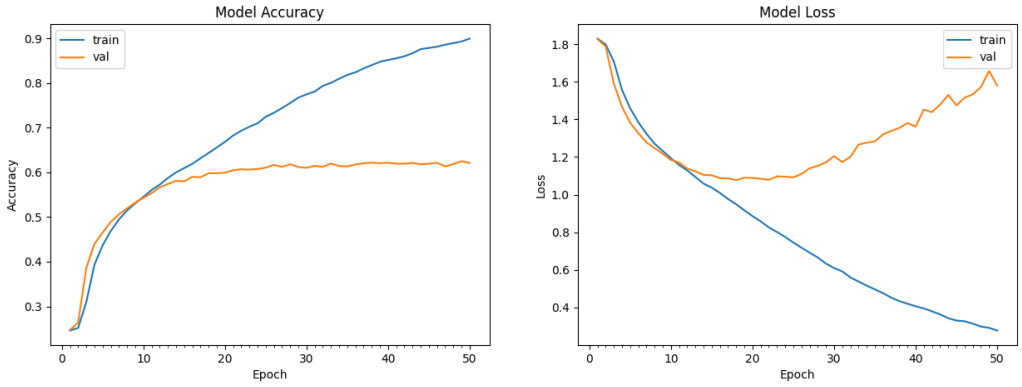
```
Epoch 50/50
448/448 [==============================] - 14s 31ms/step - loss: 0.4051 - accuracy: 0.8533 - val_loss: 1.4694 - val_accuracy: 0.5989
```



معماری با دقت 61.12%

معماری زیر دقت 61.12 را بدست آورد

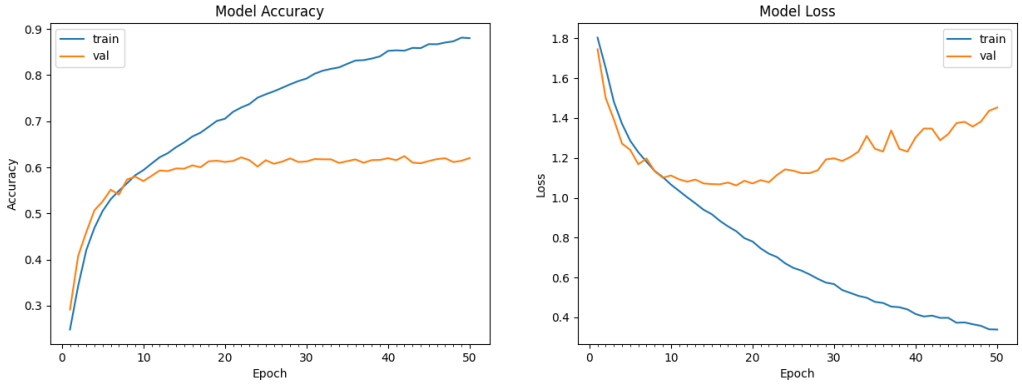
```python
# Create the model
model = Sequential()

model.add(Conv2D(32, kernel_size=3, padding="same", activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(128, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(256, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(256, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```



معماری با دقت 62%

معماری زیر دقت 62 درصد را بدست آورد

```python
# Create the model
model = Sequential()

model.add(Conv2D(32, kernel_size=3, padding="same", activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(32, kernel_size=3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=3,padding="same", activation='relu'))
model.add(Conv2D(64, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))


model.add(Conv2D(256, kernel_size=3,padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(2028, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))

#
```



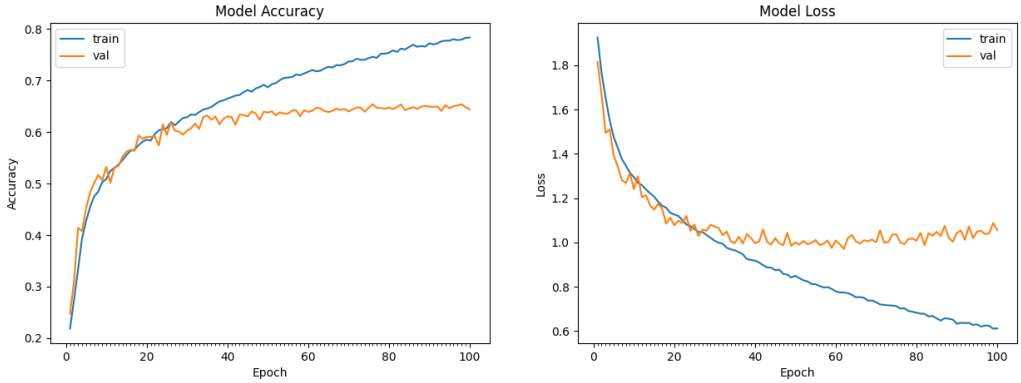
معماری با دقت 64.37%

معماری زیر دقت 64.37 را بدست آورد

```python
# Create the model

model = Sequential()

model.add(Conv2D(32, kernel_size=3, activation='relu',
input_shape=(48,48,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
```

```
model.add(Dropout(0.5))

model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(256, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(7, activation='softmax'))
```

```
poch 100/100
448/448 [==============================] - 13s 30ms/step - loss: 0.6124 - accuracy: 0.7833 - val_loss: 1.0554 - val_accuracy: 0.6437
```

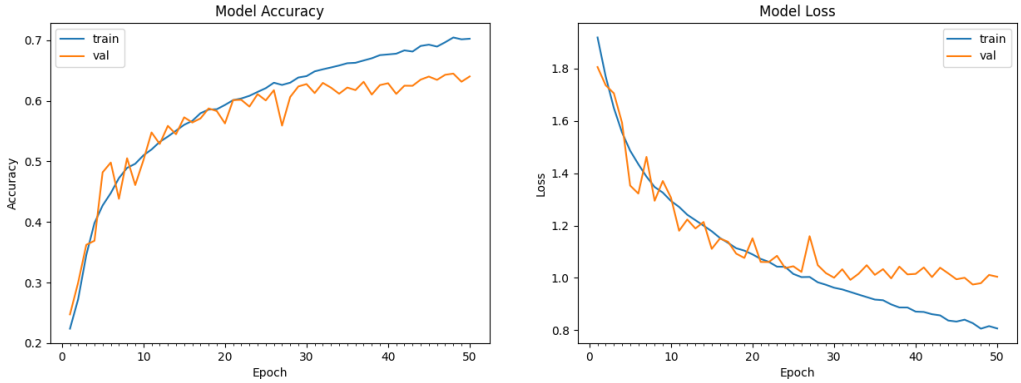


معماری با دقت 64.01%

معماری زیر دقت 64.01 بدست آورد

```
model = Sequential()

model.add(Conv2D(32, kernel_size=3, activation='relu',  input_shape=(48,48,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(256, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2048, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(7, activation='softmax'))
```

```
Epoch 50/50
448/448 [==============================] - 14s 30ms/step - loss: 0.8063 - accuracy: 0.7024 - val_loss: 1.0043 - val_accuracy: 0.6401
```



معماری با دقت 65%

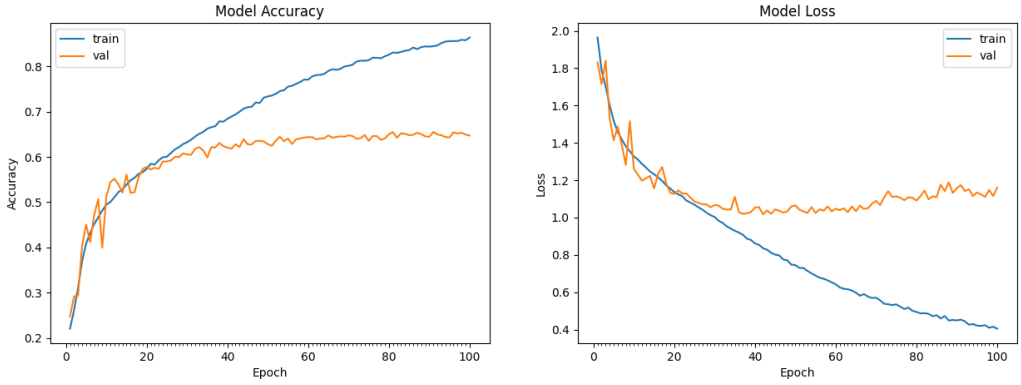
و در آخر معماری شبکه اصلی که در ابتدا معرفی شده بود توانست دقت 65 درصد را بدست آورد.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=3, activation='relu',
input_shape=(input_target,input_target,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))
```

```
model.add(Flatten())

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(7, activation='softmax'))
```

Classification Report

	precision	recall	f1-score	support
angry	0.55	0.56	0.55	958
disgusted	0.83	0.57	0.67	111
fearful	0.57	0.44	0.49	1024
happy	0.87	0.83	0.85	1774
neutral	0.51	0.68	0.59	1233
sad	0.53	0.52	0.53	1247
surprised	0.82	0.77	0.79	831
accuracy			0.65	7178
macro avg	0.67	0.62	0.64	7178
weighted avg	0.66	0.65	0.65	7178



در دیتاست FER-2013 دقت جهانی بدون اضافه کردن داده اضافی برابر با 71.2٪ است ودر شبکه معماری انتخاب شده درصد دقت برابر با 65٪ است.

با توجه به معماری های نشان داده شده در قبل،دلایل متعدی برای پیشرفت درصد دقت وجود داشت یکی از آن ها استفاده از شبکه ساده تر و استفاده از BatchNormalization بود.

در اینجا از same،padding استفاده شده چون نمیخواستم داده ای از خود عکس کم شه چون سایز عکس ها کوچک بود لازم بود که از تمام عکس استفاده شود.

در same عکس به صورت زیر دیده میشود

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

و درvalid به صورت زیر دیده میشود

35	19	25	6
13	22	16	53
4	3	7	10
9	8	1	3

در لایه آخر از activation، softmax استفاده کردیم چون مسائل classification خیلی استفاده داره و اونم به این خاطره که به فرم احتمالی خروجی میده و ما هم در این مسئله میخوایم احتمال تعلق داده به هر کلاس رو بدونیم و این تابع برای این کار مناسبه.

نمایش اطلاعات

در این قسمت درباره نمایش اطلاعات صحبت کاملی انجام میدهیم.

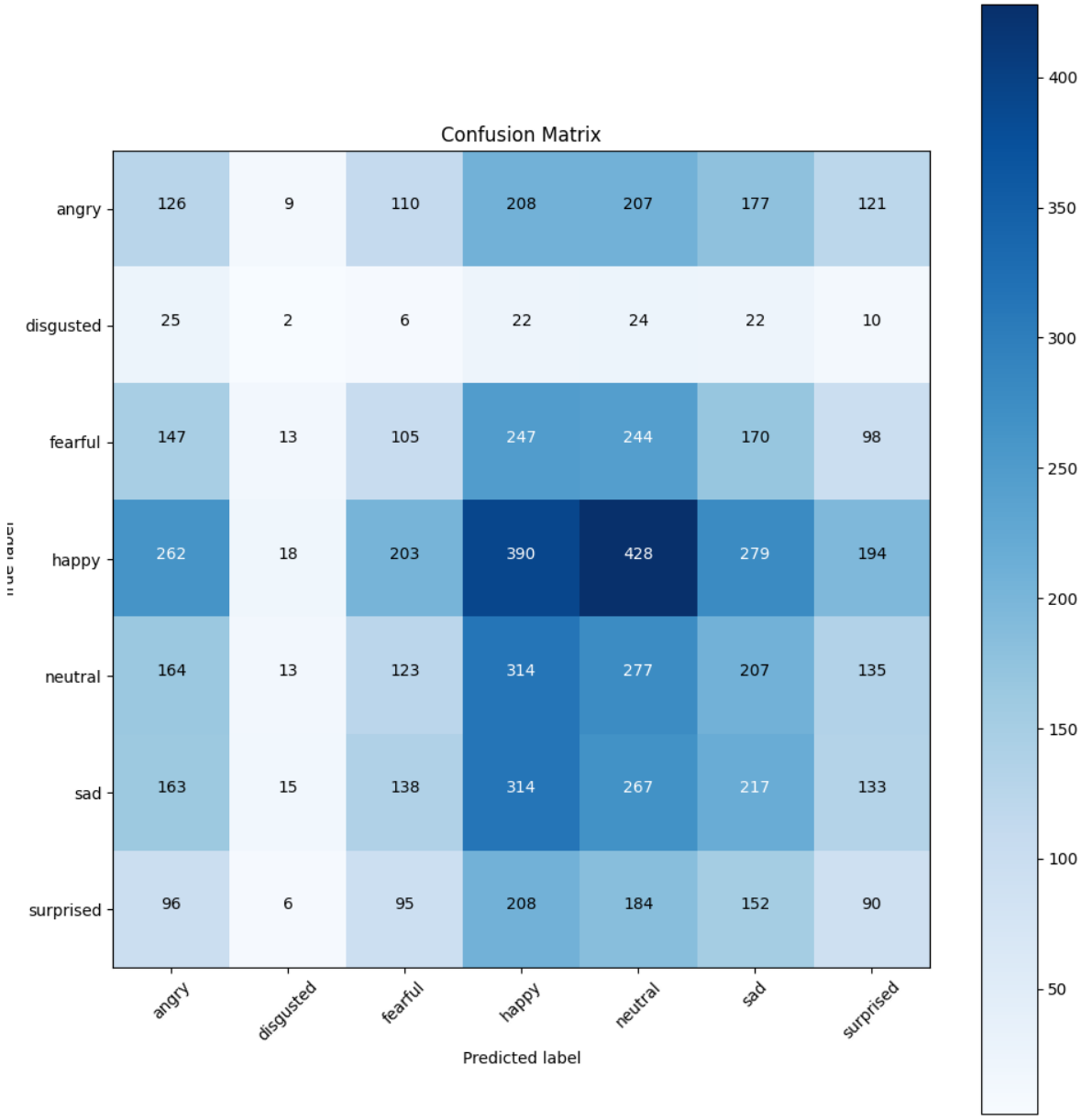
Confusion matrix

برای محسابه کانفیوژن ماتریکس اگر برای ترین داده از روش اول استفاده میکردیم بسیار ساده و راحت بود ولی چون من از روش دوم استفاده کرده بودم یک سری چالش ها درطول مسیر برخورد کردم.

برای بدست آوردن کانفیوژن ماتریکس از

```
from sklearn.metrics import confusion_matrix
```

استفاده شد.در ابتدا جواب های متفاوت میگرفتم و بسیار شکه شده بودم و فکر میکردم که کد من درکل اشتباه است به عنوان نمونه کانفیوژن ماتریکس بر روی داده تست به صورت زیر میشد



که دقت آن 17٪ میشد،در جست و جو در اینترنت به جواب خاصی نرسیده بودم،پس باید کد خودم و کتابخانه رو میخوندم تا بفهمم مشکل از کجاس.با کلی وقت فهمیدم در قدم اول ترین داده ها که به این صورت بود

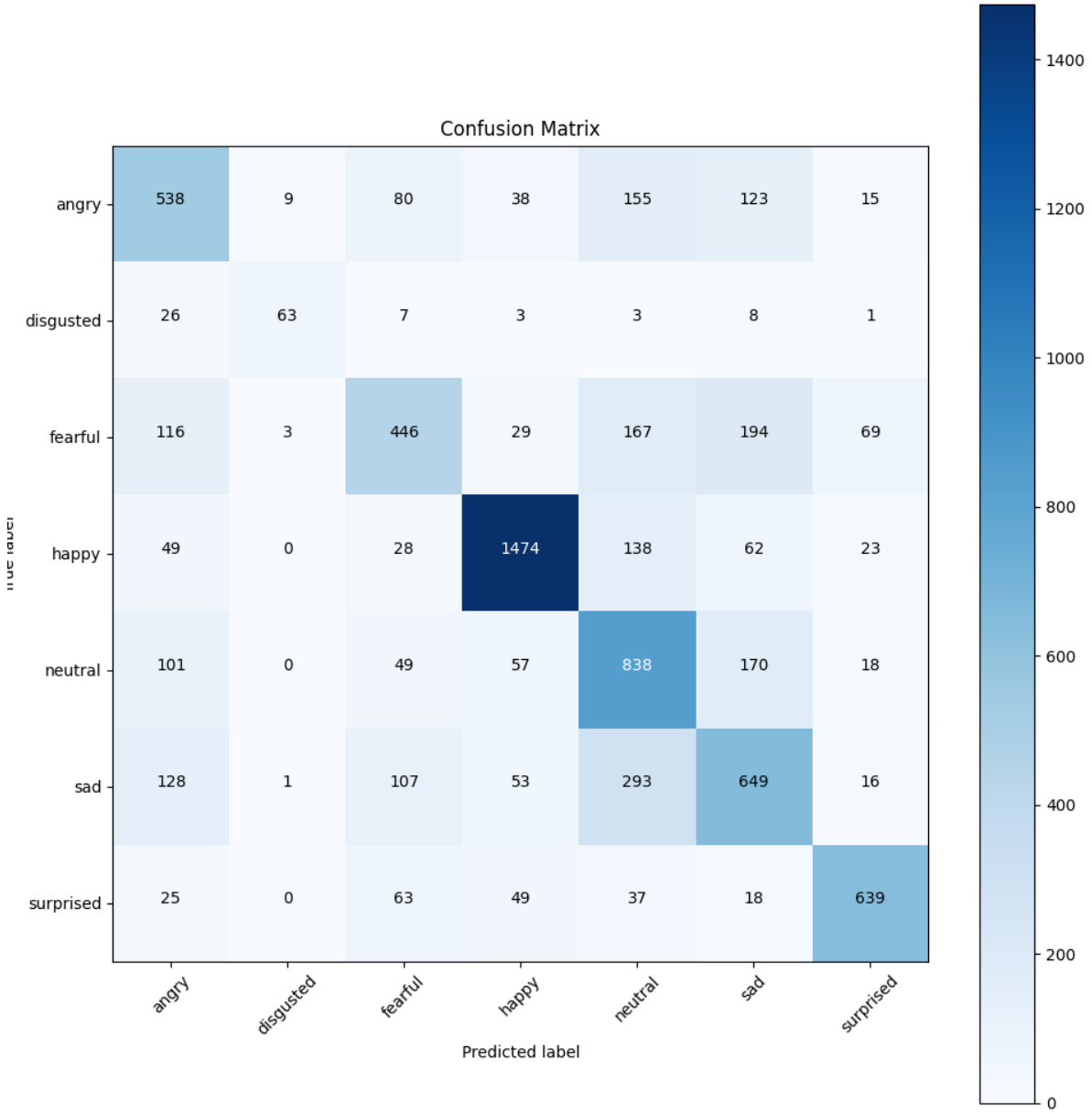
```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(input_target, input_target),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical',
```

```
    shuffle=True)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(input_target, input_target),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical',
    shuffle=True
)
```

و هیچ مشکلی هم برای ترین نبود ولی برای کانفیوژن ماتریکس باید shuffle مقدار False میگرفت تا ترتیب کلاس ها به هم نمیریخت پس برای محاسبه کانفیوژن ماتریکس از کد زیر استفاده کردم

```
if train==False:
    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(input_target, input_target),
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical',
        shuffle=False)

    validation_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(input_target, input_target),
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical',
        shuffle=False
        )
```

که مقدار shuffle برابر با False مقدار دهی شده است.سپس کانفیوژن ماتریکس بر روی داده تست به صورت زیر شد



که درست است.

حال به قسمت نحوه تست عکس خودم یا دیگران میرسیم.

برای این کار من از دوربین خود لپتاپ استفاده کردم که با استفاده از کتابخانه open-cv امکان پذیر هست.

```
cap = cv2.VideoCapture(0)
```

کد این قسمت به صورت زیر است

```
if display == True:



    cv2.ocl.setUseOpenCL(False)

    emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4:
"Neutral", 5: "Sad", 6: "Surprised"}

    # start the webcam feed
    cap = cv2.VideoCapture(0)
    while True:
        # Find haar cascade to draw bounding box around face
        ret, frame = cap.read()
        if not ret:
            break
        facecasc =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = facecasc.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)



        for (x, y, w, h) in faces:

            cv2.rectangle(frame, (x, y - 50), (x + w, y + h + 10), (255, 0,
0), 2)



            roi_gray = gray[y:y + h, x:x + w]

            cv2.imshow("face", roi_gray)
            cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray,
(input_target, input_target)), -1), 0)
            prediction = model.predict(cropped_img)
            print(prediction)
```

```
            maxindex1 = int(np.argmax(prediction))
            max1=prediction[0][maxindex1]
            print("max 1")
            print(max1)
            print(np.shape(prediction))
            prediction=np.delete(prediction,maxindex1)
            maxindex2 = int(np.argmax(prediction))
            print(np.shape(prediction))
            max2 = prediction[maxindex2]
            print("max 2")
            print(max2)
            neveshte=""


            if max1-max2<0.05:
                neveshte= "ya :"+emotion_dict[maxindex1]+", ya
:"+emotion_dict[maxindex2]
            else:
                neveshte = emotion_dict[maxindex1]
            cv2.putText(frame, neveshte, (x + 20, y - 60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255),
                        2, cv2.LINE_AA)
        cv2.imshow('Video', cv2.resize(frame, (1600, 960),
interpolation=cv2.INTER_CUBIC))
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

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

در ابتدا برای جلوگیری از از استفاده و لاگ کردن پیام های غیر ضروری Opencl آن را غیر فعال میکنیم.سپس دیکشنری از کلاس ها داریم که با توجه به خروجی شبکه به ما اسم کلاس را میگوید.بعد از دوربین لپتاپ استفاده کردیم.برای تشخیص صورت انسان درعکس با جست و جو در اینترنت با استفاده از یک

فایل xml و متد cv2.CascadeClassifier() که ورودی آن،فایل xml هست صورت انسان را تشخیص دادیم.

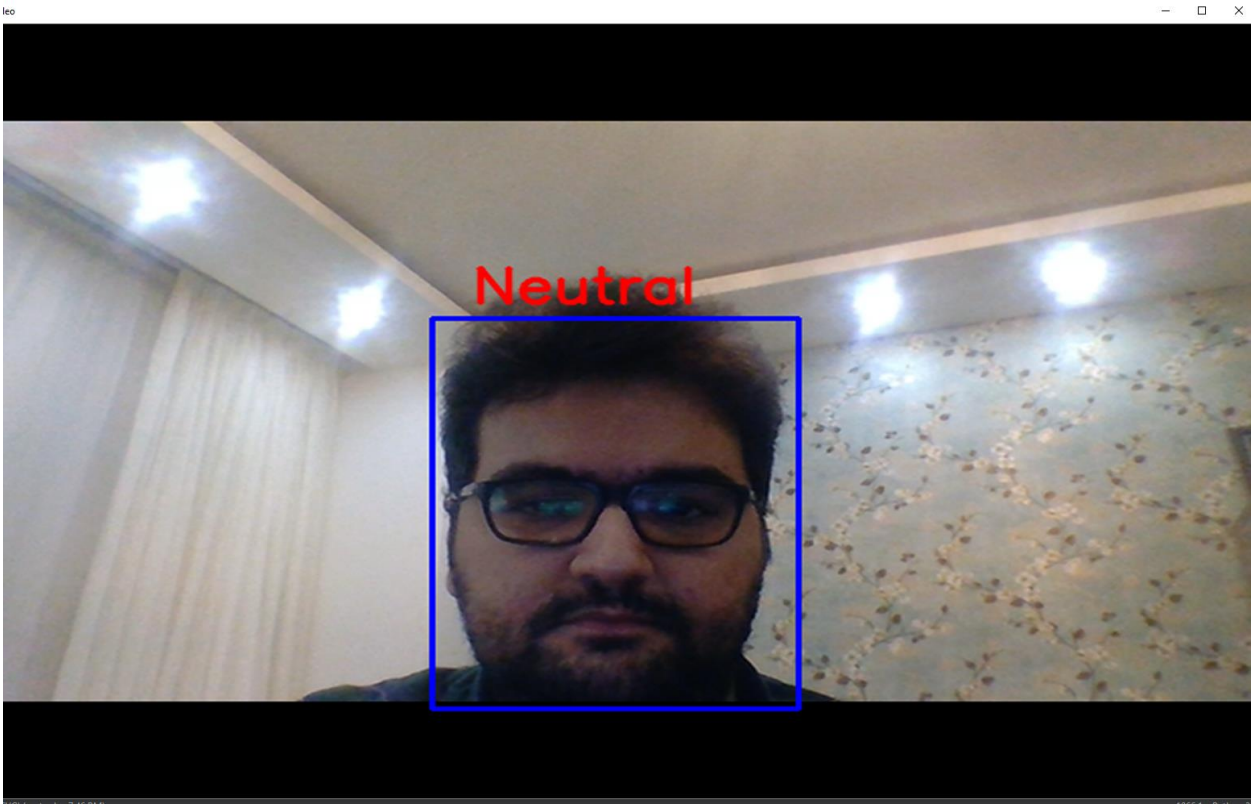
```
facecasc = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = facecasc.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
```

برای تشخیص صورت با توجه بع داکیومنت کتابخانه عکس ورودی باید grayscale باشد.

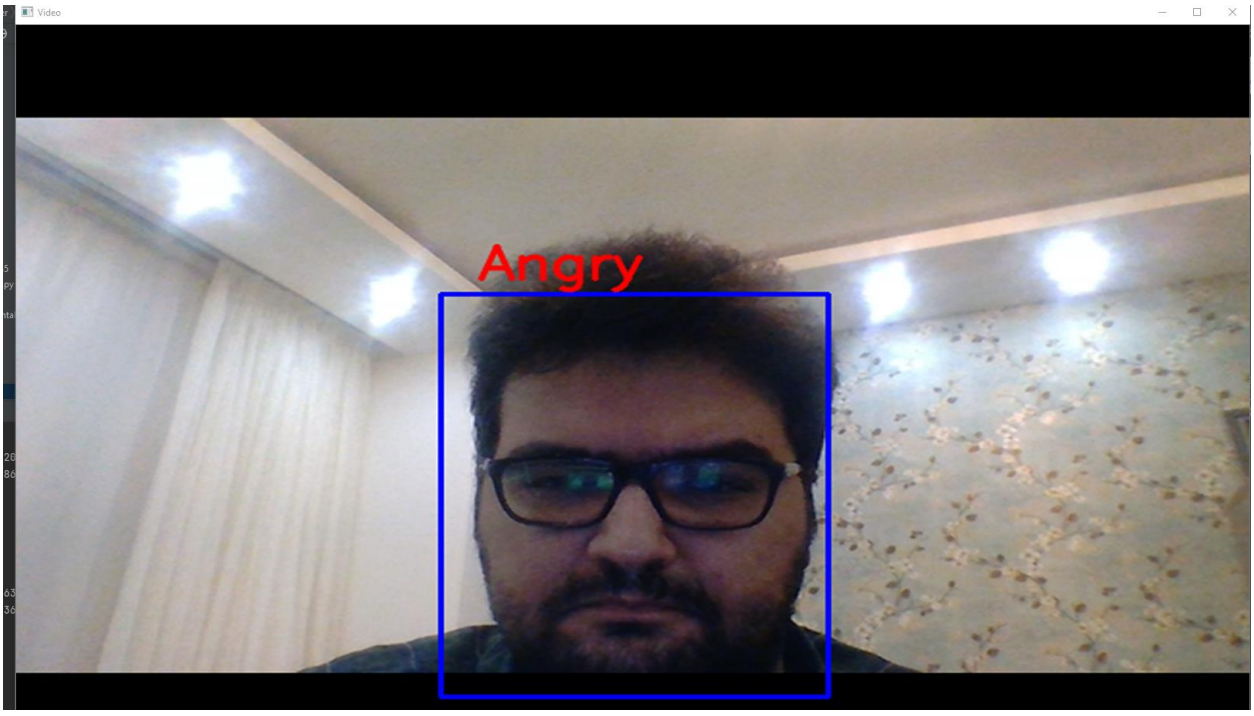
در انتها هر صورت را به شبکه دادیم و جواب گرفته شده را از شبکه را به دیکشنری دادیم تا بفهمیم کدام کلاس است و بالای هر چهره حالت صورتی آن را قرار دادیم.در بعضی اوقات شبکه جوابی که به صورت آرایه بر میگیرداند دو عدد که از همه بزرگرتر بودند خیلی به هم نزدیک بودند،سپس سنجیدیم که اگر تفاوت این دو عدد کم تر از 0.05 بود هر دو رو بالای چهره مینوشتیم و قرار میدادیم.

درانتها نمونه ای از خروجی هارا مشاهده میکنیم

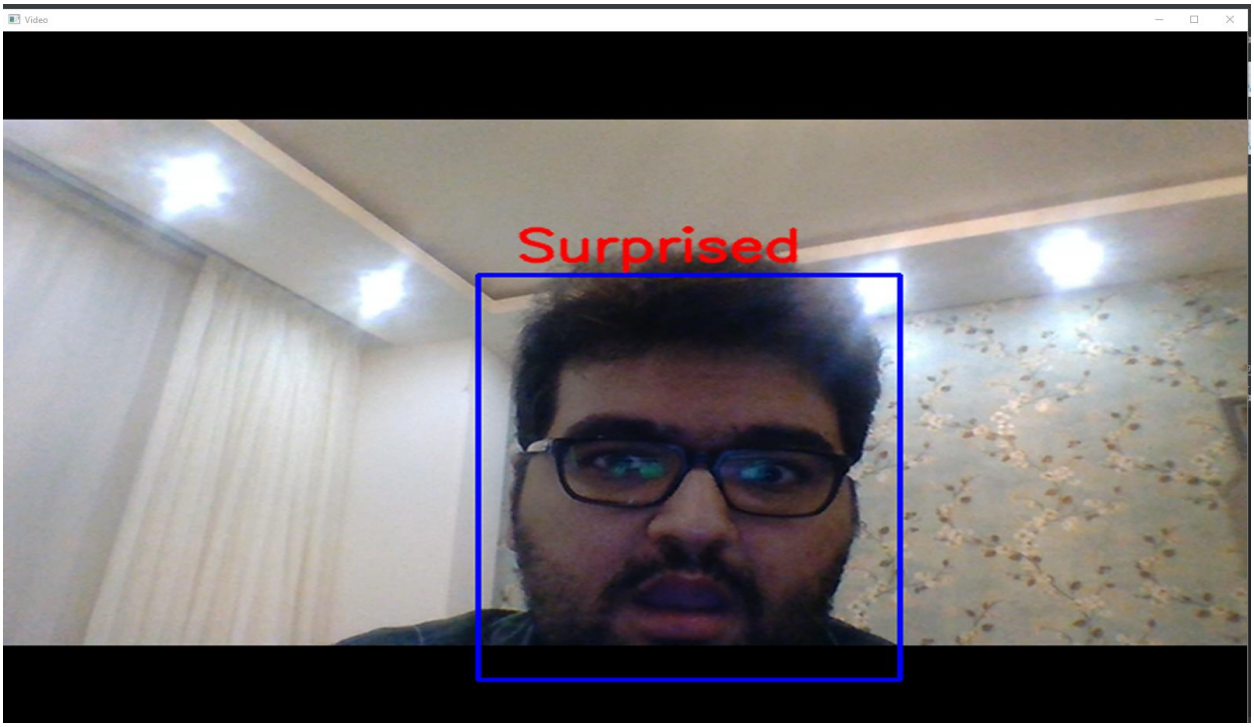
خروجی 1



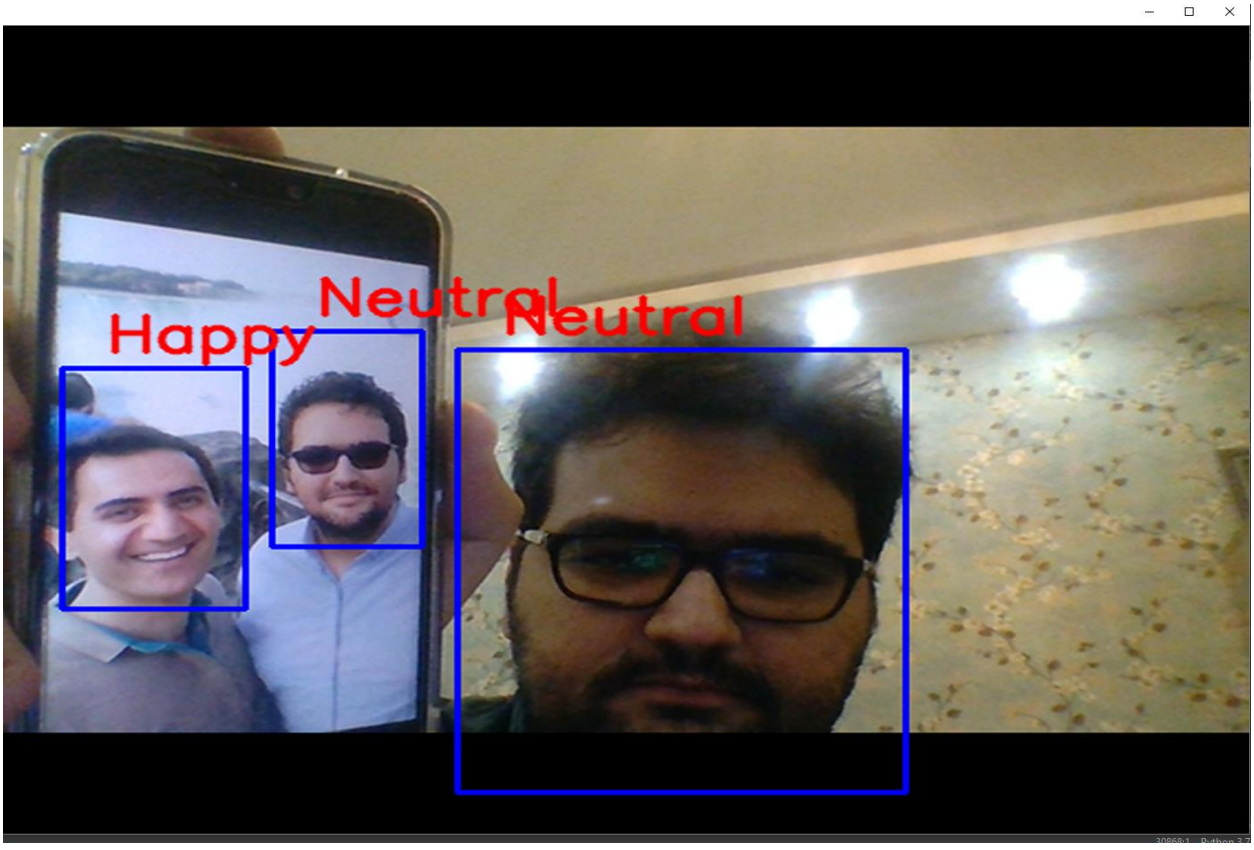
خروجی 2



خروجی 3



خروجی 4



مراجع و پیوست

1. https://datascience.stackexchange.com/questions/72568/getting-confusion-matrix-with-keras-flow-from-directory
2. https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php
3. https://www.udemy.com/course/tensorflow-2/
4. https://www.tensorflow.org/api_docs/python/tf