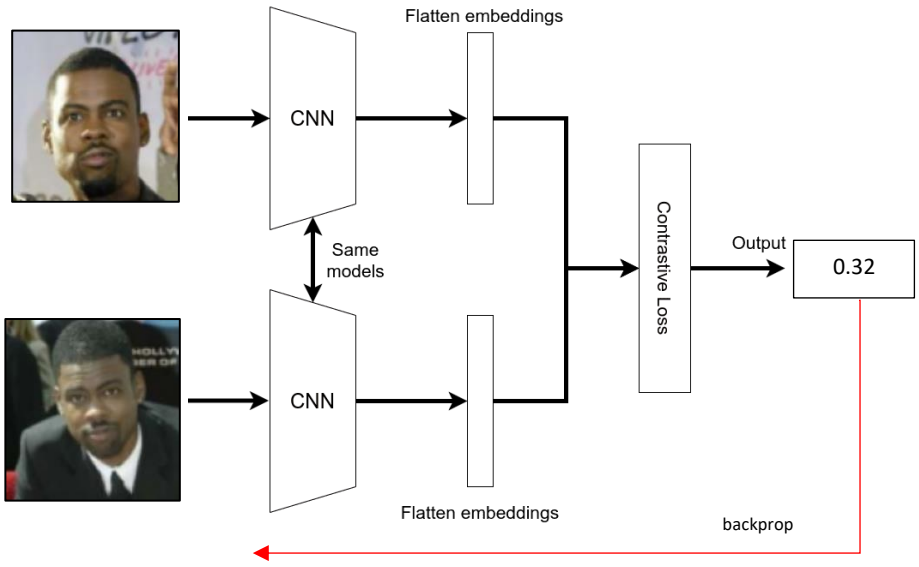
گزارش پروژه مبانی بینایی کامپیوتر

مهرشاد سعادتی نیا

97243039

در این هدف آنست که یک سیستم تایید صورت ایجاد کنیم. برای این کار روش های مختلفی را می توان استفاده کرد، روشی که من استفاده کردم استفاده از شبکه های عصبی از نوع Siamese است. این معماری از دو جفت شبکه عصبی با وزن های مشترک تشکیل شده. از شبکه های عصبی siamese در کاربرد های بسیاری از قبیل تایید امضا، تایید هویت و هر کاربردی که نیاز به مقایسه دو تصویر داشته باشد استفاده می شود. برای تصویر های بزرگ شبکه های عصبی استفاده شده در هر کدام از جفت ها از نوع CNN خواهند بود.



در شکل بالا فرآیند آموزش شبکه را می توان مشاهده کرد که از طریق بهینه سازی تابع خطا آموزش می بیند.

معماری که من در هرکدام از جفت ها استفاده کردم بصورت زیر است و به کمک keras پیاده سازی شده است:

```
def initialize_base_branch():
    input = keras.layers.Input(shape=(DIM, DIM, 3), name="base_input")
    x = keras.layers.Conv2D(filters=32,kernel_size=(3,3),padding="same", activation="relu")(input)
    x = keras.layers.MaxPool2D(pool_size=(2,2),strides=(2,2))(x)
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(2048,activation="relu")(x)
    x = keras.layers.Dropout(0.4)(x)
    x = keras.layers.Dense(2048,activation="relu")(x)
    return Model(inputs=input, outputs=x)
```

خروجی هرکدام از این جفت ها یک embedding یا بازنمایی میانی از تصویر است که در اینجا اندازه ی آن 2048 خواهد بود.

هر کدام از این جفت تصاویر بردار embedding به اندازه 2048 تولید میکنند و با کمک فاصله اقلیدسی اختلاف آنها محاسبه میشود:

```
def euclidean_distance(vects):
    x, y = vects
    sum_square = K.sum(K.square(x - y), axis=1, keepdims=True)
    return K.sqrt(K.maximum(sum_square, K.epsilon()))


def eucl_dist_output_shape(shapes):
    shape1, shape2 = shapes
    return (shape1[0], 1)
```

در نهایت از این فاصله ی اقلیدسی برای محاسبه ی تابع خطا استفاده میشود که ما از تابع خطای contrastive استفاده کردیم:

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

در این فرمول D فاصله اقلیدسی است و Y لیبل هدف

تصاویری که استفاده کرده ایم تصاویر صورت دیتاست lfw هستند که آنها را به صورت رندوم با هم دیگر جفت کرده ایم

اندازه مجموعه داده ی آموزشی 4000 است و دو مجموعه ی دیگر به ترتیب 100 و 50

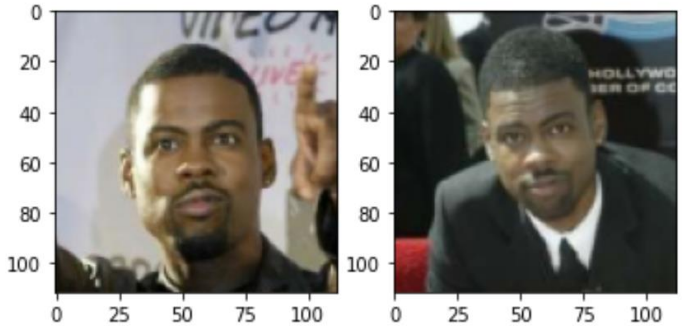
اندازه ی اصلی تصاویر 250 در 250 بود اما بدلیل محدودیت رم در کولب آنها را به اندازه ی 112 در 112 تبدیل کردیم.

پس از 50 اپوک آموزش با نرخ یادگیری 0.00008 دقت حدود 70٪ روی مجموعه ی تست به دست آمد و دقت آموزش به 99.5 درصد رسید که نشان دهنده ی بیش برازش است اما با توجه به آنکه تولید داده را بصورت رندوم از مجموعه داده با الگوریتمی که خودمان نوشتیم انجام میدهیم به سادگی می توان اندازه ی داده را بیشتر و در نتیجه بیش برازش را کاهش داد.

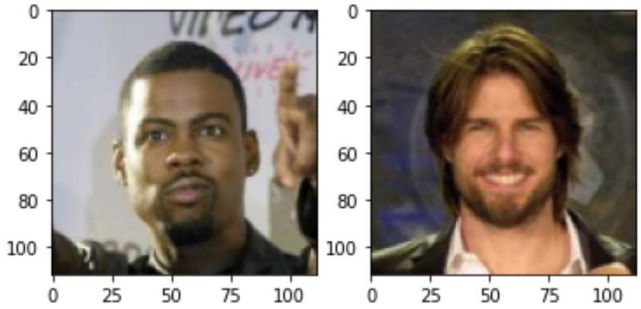
```
rms = RMSprop(learning_rate=0.00008)
model.compile(loss=contrastive_loss_with_margin(margin=1), optimizer=rms)
history = model.fit([X_train[:, 0], X_train[:, 1]], y_train, epochs=50, validation_data=[[X_val[:, 0], X_val[:, 1]], y_val], batch_size= 16)
```

یک نمونه تصویر تست:

```
1/1 [==============================] - 0s 19ms/step
Distance:  0.029081543907523155
Prediction: Images are Same
```



```
1/1 [==============================] - 0s 32ms/step
Distance:  0.9464706778526306
Prediction: Images are Not Same
```



تهیه دیتاست:

برای تهیه دیتاست هم از این تابع استفاده شده که به تعداد مشخص شده هر بار دو نمونه متفاوت از یک کلاس و دو نمونه از دو کلاس متفاوت را جفت می کند و با لیبل مناسب (0 یا 1) به لیستی اضافه میکند.

```python
def create_pairs(size):
  pairs = []
  labels = []

  for i in range(size):
    print(f">> {i}")
    gen_count = 1
    while gen_count == 1:
      c = random.randrange(0, no_classes)
      dir_name = class_names[c]
      full_path = os.path.join(datapath, dir_name)
      gen_count = len(os.listdir(full_path))
    #### create a pair of images from the same class
    selected = random.sample(range(gen_count), 2)
    selected_pair = add_identical_pairs(full_path, selected)
    pairs.append(selected_pair)
    labels.append(1)
    #### create a pair of images from different classes
    c1, c2 = random.sample(range(no_classes), 2)
    dir1, dir2 = class_names[c1],  class_names[c2]
    path1, path2 = os.path.join(datapath, dir1), os.path.join(datapath, dir2)
    count1, count2 = len(os.listdir(path1)), len(os.listdir(path2))
    i, j = random.randrange(0, count1), random.randrange(0, count2)
    different_pairs = add_different_pairs(path1, path2, i, j)
    pairs.append(different_pairs)
    labels.append(0)
  print("============================================================")
  return np.array(pairs), np.array(labels).astype('float32')
```

تعداد های نمونه برداری شده از هر کلاس:

```
train_size = 2000
val_size = 100
test_size = 50

X_train, y_train = create_pairs(train_size)
X_val, y_val = create_pairs(val_size)
X_test, y_test = create_pairs(test_size)
```

می توان این داده های را برای آموزش بهینه تر در قالب دیتاست های تنسورفلو قرار داد، اما برای کار ما خیلی تفاوتی ایجاد نکرد پس کامنت شده در کد:

```
#batch_size = 8
#
#train_dataset = tf.data.Dataset.from_tensor_slices(({"left_input": X_train[:, 0], "right_input": X_train[:, 1]}, y_train))
#val_dataset = tf.data.Dataset.from_tensor_slices(({"left_input": X_val[:, 0], "right_input": X_val[:, 1]}, y_val))
#
#AUTOTUNE = tf.data.AUTOTUNE
#train_ds = train_dataset.batch(batch_size, drop_remainder=False).prefetch(buffer_size=AUTOTUNE)
#val_ds = val_dataset.batch(batch_size, drop_remainder=False).prefetch(buffer_size=AUTOTUNE)
```

نتایج:

```
7/7 [==============================] - 19s 2s/step - loss: 0.2252
125/125 [==============================] - 273s 2s/step
7/7 [==============================] - 13s 2s/step
Loss = 0.22523827850818634, Train Accuracy = 99.45% Test Accuracy = 69.5%
```

مشخصاً با اورفیت سروکار داریم، اما با تولید داده ی بیشتر قطعاً نتایج بهتر هم خواهد شد.

نسخه ی دوم:

یک نسخه ی دیگر هم ا این پروژه پیاده سازی شده است که به جای تهیه دیتاست توسط خودمان، از دیتاست آماده جفت های lfw آماده شده توسط scikit learn استفاده کردیم. در این دیتاست فقط قسمت صورت از عکس ها استخراج شده و عکس ها به ابعاد 47 در 62

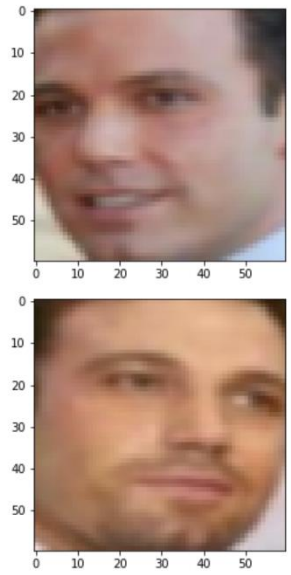
هستند. من آنها را برای آموزش روی شبکه عصبی به ابعاد 60 در 60 تغیر دادم. برای آموزش هم از VGG-16 استفاده کردم با این تفاوت که لایه ورودی به جای 224 در 224 دارای ابعاد 60 در 60 است.

```
input = keras.layers.Input(shape=(DIM, DIM, 3), name="base_input")
x = keras.layers.Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu")(input)
x = keras.layers.MaxPool2D(pool_size=(2,2),strides=(2,2))(x)
x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=(2,2),strides=(2,2))(x)
x = keras.layers.Flatten()(x)
x = keras.layers.Dense(4096,activation="relu")(x)
x = keras.layers.Dense(4096,activation="relu")(x)
return Model(inputs=input, outputs=x)
```

نتایج این مدل:

```
94/94 [==============================] - 3s 30ms/step - loss: 0.1523
163/163 [==============================] - 5s 27ms/step
94/94 [==============================] - 2s 27ms/step
Loss = 0.15225492417812347, Train Accuracy = 99.92% Validation Accuracy = 78.47%
```

```
Not same
1/1 [==============================] - 0s 495ms/step
Prediction: 0.0
Distance: 1.217295527458191
```



```
Same
1/1 [==============================] - 0s 28ms/step
Prediction: 1.0
Distance: 0.04967010021209717
```