دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین امتیازی

متین بذرافشان - 810100093
شهریار عطار - 810100186

فهرست

پرسش 1. تولید برچسب با خوشه‌بندی

1-1. دیتاست

برای این سوال از دو مجموعه داده MNIST و Fashion MNIST استفاده شده که برای استفاده از این مجموعه ها و نرمال‌سازی هر یک به شکل زیر عمل کردیم:

```
(x_mnist_train, y_mnist_train), (x_mnist_test, y_mnist_test) =
mnist.load_data()
assert x_mnist_train.shape == (60000, 28, 28)
assert x_mnist_test.shape == (10000, 28, 28)
assert y_mnist_train.shape == (60000,)
assert y_mnist_test.shape == (10000,)

x_mnist_train, x_mnist_test = x_mnist_train / 255.0,
x_mnist_test / 255.0
x_mnist_train = x_mnist_train.reshape(x_mnist_train.shape[0],
28, 28, 1)
x_mnist_test = x_mnist_test.reshape(x_mnist_test.shape[0], 28,
28, 1)

x_mnist_train, x_mnist_val, y_mnist_train, y_mnist_val =
train_test_split(x_mnist_train, y_mnist_train, test_size=0.25)
```

شکل 1-1. تصویر نمونه‌ای از کلاس‌های موجود در دیتاست MNIST

```
(x_fashion_train, y_fashion_train), (x_fashion_test,
y_fashion_test) = fashion_mnist.load_data()
assert x_fashion_train.shape == (60000, 28, 28)
assert x_fashion_test.shape == (10000, 28, 28)
assert y_fashion_train.shape == (60000,)
assert y_fashion_test.shape == (10000,)

x_fashion_train, x_fashion_test = x_fashion_train / 255.0,
x_fashion_test / 255.0
```

```
x_fashion_train =
x_fashion_train.reshape(x_fashion_train.shape[0], 28, 28, 1)
x_fashion_test = x_fashion_test.reshape(x_fashion_test.shape[0],
28, 28, 1)

x_fashion_train, x_fashion_val, y_fashion_train, y_fashion_val =
train_test_split(x_fashion_train, y_fashion_train,
test_size=0.25)
```

شکل 1-2. تصویر نمونه‌ای از کلاس‌های موجود در دیتاست Fashion MNIST

2-1. شبکه مورد استفاده

در اینجا ما معماری مدل convolution autoencoder خودمان را معرفی می‌کنیم. این معماری از دو بخش encoder و decoder تشکیل شده که بخش اول برای feature extraction و بخش دوم برای reconstruct استفاده می‌شود. توجه کنید که در این مثال ما از latent space با اندازه 6 استفاده کردیم که بتوانیم برای بخش‌های جلوتر به راحتی ویژگی‌های مورد نظر را بدست بیاوریم و به خوشه‌بندی بپردازیم:

```
def create_model(input_shape, latent_dim):

      # --- Encoder ---
      inputs = Input(shape=input_shape)

      # Block 1
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(inputs)
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      x = MaxPooling2D((2, 2))(x)
      x = BatchNormalization()(x)

      # Block 2
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
```

```
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      x = Conv2D(32, (2, 2), activation="relu",
padding="same")(x)
      x = MaxPooling2D((2, 2))(x)
      x = BatchNormalization()(x)

      # Block 3
      x = Conv2D(16, (2, 2), activation="relu",
padding="same")(x)
      x = Conv2D(4, (2, 2), activation="relu", padding="same")(x)
      x = Conv2D(1, (2, 2), activation="relu", padding="same")(x)

      # Latent space
      x = Flatten()(x)
      encoded = Dense(latent_dim, activation="relu")(x)
      encoder = Model(inputs=inputs, outputs=encoded)
      encoded_inputs = Input(shape=(latent_dim,))

      # --- Decoder ---
      x = Dense(4, activation="relu")(encoded_inputs)
      x = Reshape((2, 2, 1))(x)

      # Block 1
      x = Conv2D(4, (2, 2), activation="relu", padding="same")(x)
      x = Conv2D(16, (2, 2), activation="relu",
padding="same")(x)
      x = BatchNormalization()(x)
      x = UpSampling2D((7, 7))(x)

      # Block 2
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      x = BatchNormalization()(x)
      x = UpSampling2D((2, 2))(x)

      # Block 3
      x = Conv2D(32, (3, 3), activation="relu",
```

```
padding="same")(x)
      x = Conv2D(32, (3, 3), activation="relu",
padding="same")(x)
      decoded = Conv2D(1, (3, 3), activation="sigmoid",
padding="same")(x)
      decoder = Model(inputs=encoded_inputs, outputs=decoded)


      # --- Autoencoder ---
      x = encoder(inputs)
      x = decoder(x)
      model = Model(inputs=inputs, outputs=x)
      model.compile(
      optimizer=Adam(1e-3),
      loss="binary_crossentropy",
      metrics=["accuracy", "mse", "mae"]
      )

      return model, encoder, decoder
```

حال برای اطمینان از تقارن و بررسی معماری summary هر دو بخش مدل را بررسی می‌کنیم:

ابتدا معماری بخش encoder:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 batch_normalization (BatchN  (None, 14, 14, 32)       128
 ormalization)
```

```
 conv2d_2 (Conv2D)           (None, 14, 14, 32)        9248

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        9248

 conv2d_4 (Conv2D)           (None, 14, 14, 32)        4128

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 batch_normalization_1 (Batc  (None, 7, 7, 32)         128
 hNormalization)

 conv2d_5 (Conv2D)           (None, 7, 7, 16)          2064

 conv2d_6 (Conv2D)           (None, 7, 7, 4)           260

 conv2d_7 (Conv2D)           (None, 7, 7, 1)           17

 flatten (Flatten)           (None, 49)                0

 dense (Dense)               (None, 6)                 300

=================================================================
Total params: 35,089
Trainable params: 34,961
Non-trainable params: 128
_________________________________________________________________
```

که همانگونه که مشاهده می‌شود از لایه‌های convolution متوالی برای ساخت feature map ها استفاده شده و در نهایت از یک لایه تمام متصل برای اتصال دو بخش encoder و decoder استفاده شده

حال معماری بخش decoder

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 6)]               0

 dense_1 (Dense)             (None, 4)                 28
```

```
reshape (Reshape)            (None, 2, 2, 1)           0

conv2d_8 (Conv2D)            (None, 2, 2, 4)           20

conv2d_9 (Conv2D)            (None, 2, 2, 16)          272

batch_normalization_2 (Batc  (None, 2, 2, 16)          64
hNormalization)

up_sampling2d (UpSampling2D  (None, 14, 14, 16)        0
)

conv2d_10 (Conv2D)           (None, 14, 14, 32)        4640

conv2d_11 (Conv2D)           (None, 14, 14, 32)        9248

conv2d_12 (Conv2D)           (None, 14, 14, 32)        9248

batch_normalization_3 (Batc  (None, 14, 14, 32)        128
hNormalization)

up_sampling2d_1 (UpSampling  (None, 28, 28, 32)        0
2D)

conv2d_13 (Conv2D)           (None, 28, 28, 32)        9248

conv2d_14 (Conv2D)           (None, 28, 28, 32)        9248

conv2d_15 (Conv2D)           (None, 28, 28, 1)         289

=================================================================
Total params: 42,433
Trainable params: 42,337
Non-trainable params: 96
_________________________________________________________________
```

که همانگونه که دیده می‌شود تا حد خوبی تقارن بخش encoder است و تفاوت‌های موجود به خاطر ابعاد ورودی خروجی لایه‌های قبل می‌باشد (در این معماری ترجیح داده شده از UpSampling2D به جای Conv2DTranspose و ... استفاده شود).

معماری مدل به طور کلی نیز به شکل زیر می‌باشد:

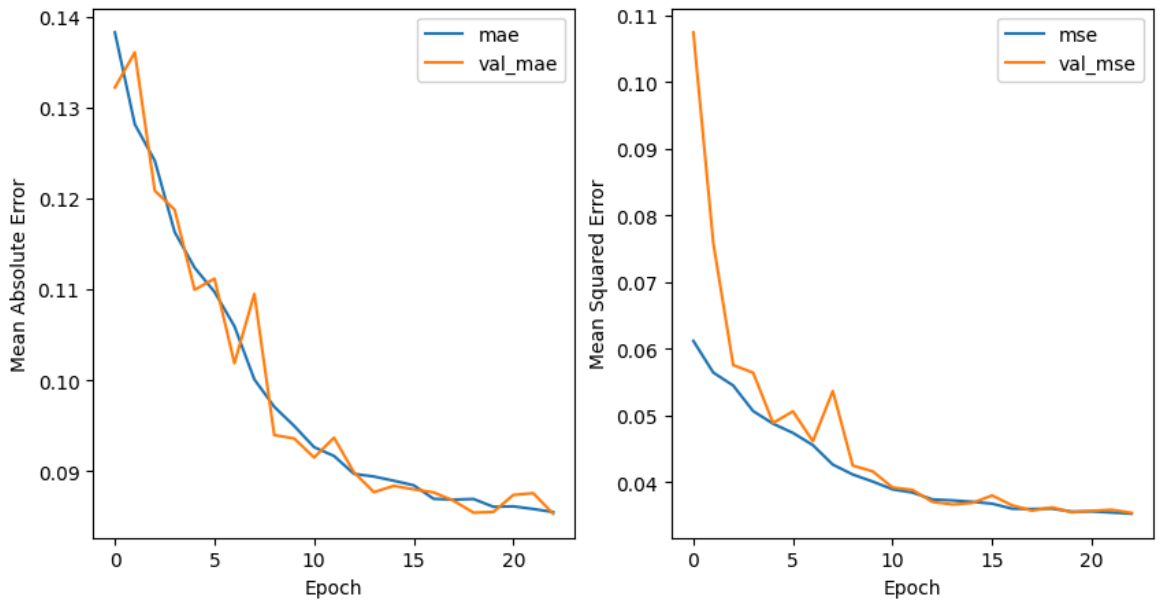
```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 model (Functional)          (None, 6)                 35089

 model_1 (Functional)        (None, 28, 28, 1)         42433

=================================================================
Total params: 77,522
Trainable params: 77,298
Non-trainable params: 224
_________________________________________________________________
```

3-1. آموزش شبکه

برای آموزش مدل از پارامترهای زیر استفاده کردیم (مدل برای نتیجه‌گیری بهتر بیش از ده ایپاک آموزش داده شده)

برای MNIST:

```
mnist_history = mnist_model.fit(
      x_mnist_train, x_mnist_train,
      validation_data=(x_mnist_val, x_mnist_val),
      epochs=50,
      batch_size=256,
      callbacks=[clr, es],
)
```

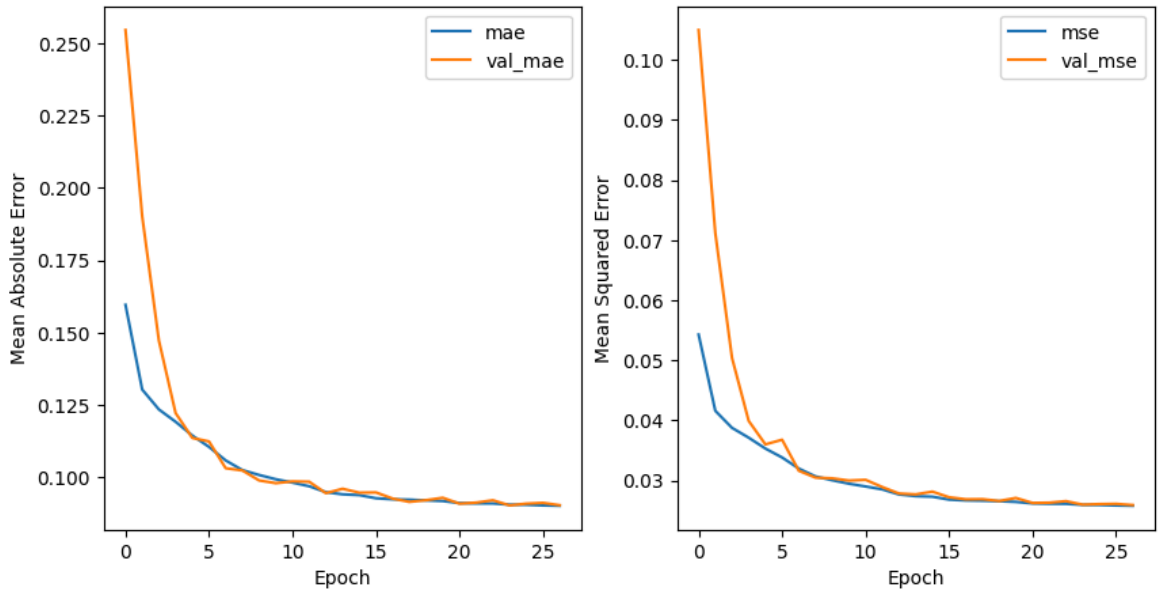


شکل 3-1. نمودار روند آموزش برای دیتاستMNIST

```
Test loss: 0.1671
Test accuracy: 0.7985
Test mse: 0.0354
Test mae: 0.0854
```

برای Fashion MNIST نیز به همین شکل مراحل را تکرار کردیم:

```
fashion_mnist_history = fashion_model.fit(
      x_fashion_train, x_fashion_train,
      validation_data=(x_fashion_val, x_fashion_val),
      epochs=50,
      batch_size=256,
      callbacks=[clr, es],
)
```



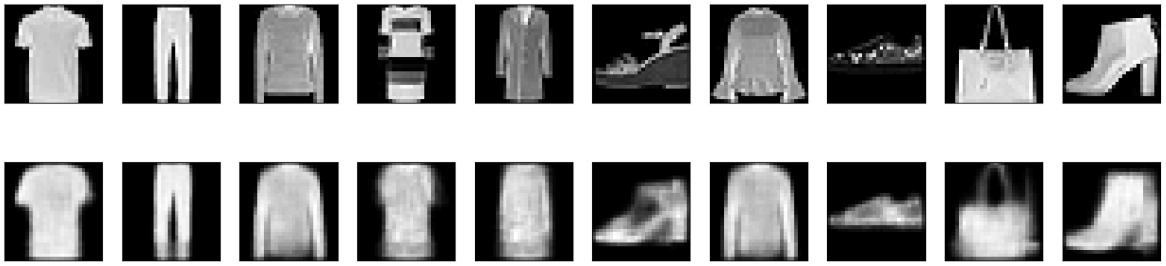
شکل 1-4. نمودار روند آموزش برای دیتاست Fashion MNIST

```
Test loss: 0.3186
Test accuracy: 0.4994
Test mse: 0.0258
Test mae: 0.0905
```

1-4. ارزیابی مدل‌ها و مشاهده خروجی

حال مدل‌ها را ارزیابی می‌کنیم. دقت mse و mae برای هر دو مدل در بخش بالا نشان داده شد. حال خروجی‌های مدل را بررسی می‌کنیم.

شکل 1-5. نمونه خروجی مدل autoencoder برای دیتاست MNIST



شکل 1-6. نمونه خروجی مدل autoencoder برای دیتاست Fashion MNIST

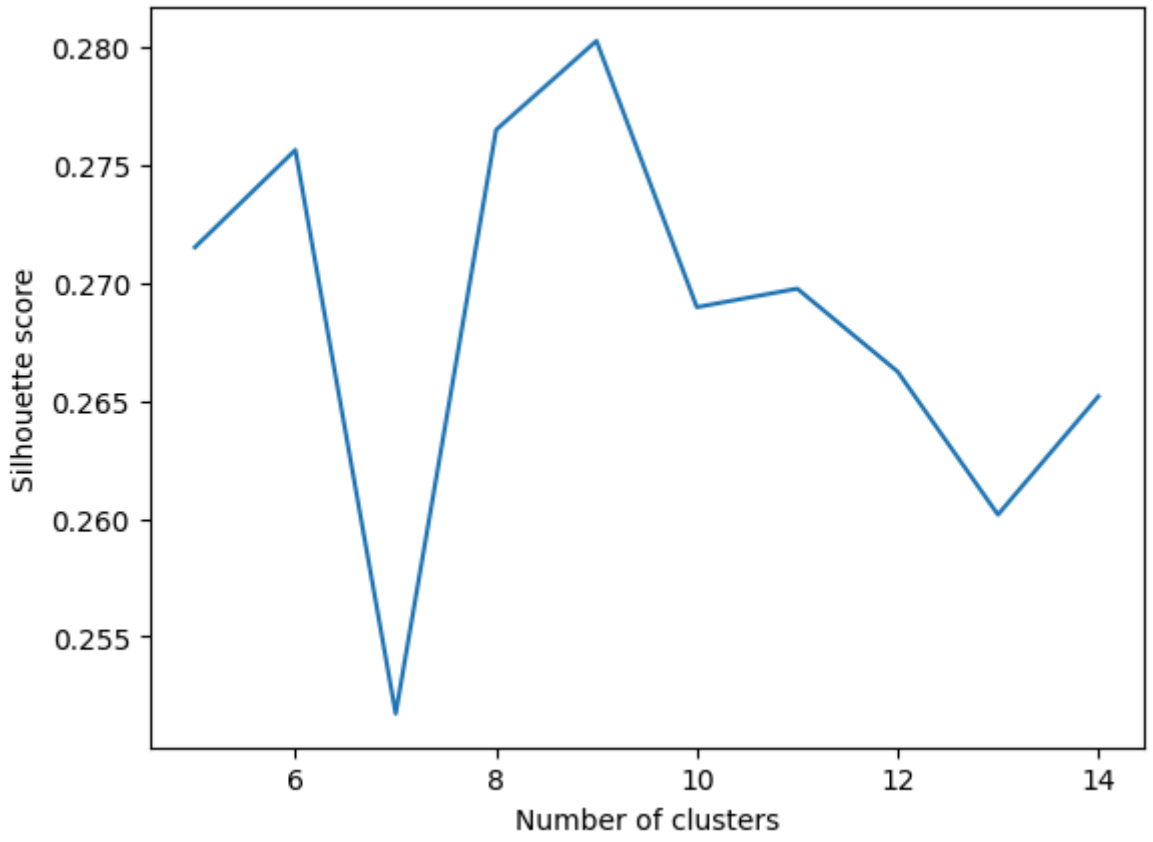
1-5. خوشه‌بندی

حال با کمک خروجی بخش encoder به حل مسئله clustering می‌پردازیم. می‌دانیم که خروجی بخش encoder یک بردار کاهش یافته از ورودی می‌باشد که در اینجا دارای latent space برابر 6 است.

ابتدا داده‌های MNIST را cluster می‌کنیم. برای این کار ابتدا همه داده‌ها را ابتدا در کنار هم می‌گذاریم.

```
x_mnist = np.concatenate([x_mnist_train, x_mnist_val,
x_mnist_test], axis=0)
y_mnist = np.concatenate([y_mnist_train, y_mnist_val,
y_mnist_test], axis=0)
```

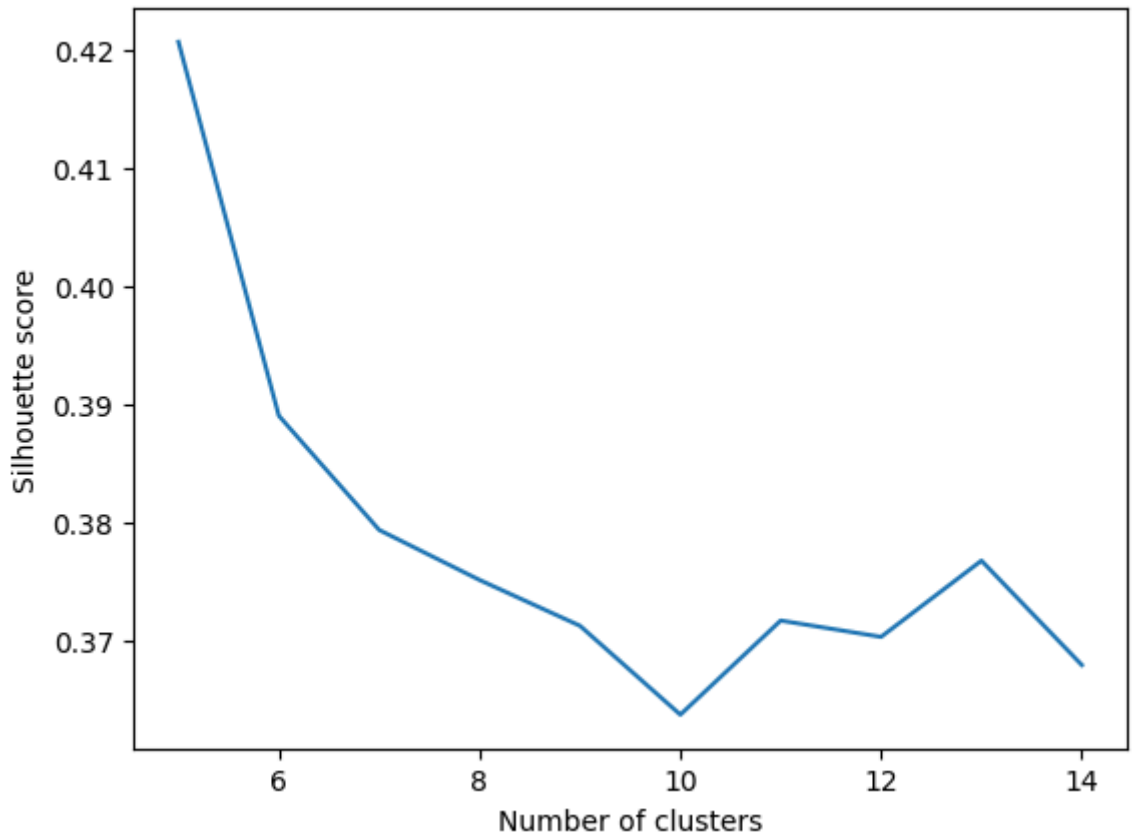
حال با بررسی silhouette score بهترین تعداد cluster برای الگوریتم k-means را بدست می‌آوریم.



شکل 1-7. نتیجه clustering برای دیتاست MNIST

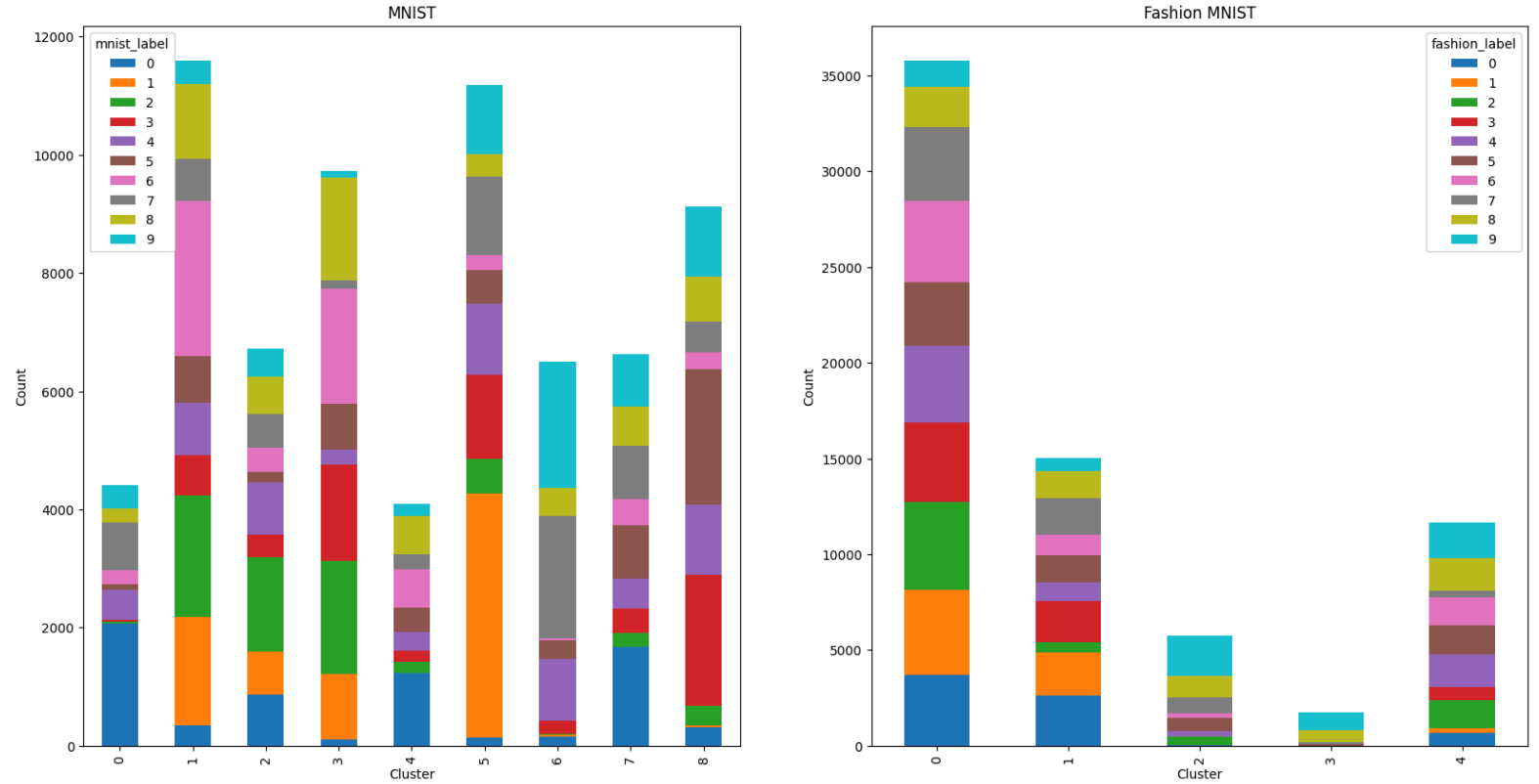
سپس برای Fashion MNIST نیز این مراحل را رفتیم که به شرح زیر بود:

```
x_fashion = np.concatenate([x_fashion_train, x_fashion_val,
x_fashion_test], axis=0)
y_fashion = np.concatenate([y_fashion_train, y_fashion_val,
y_fashion_test], axis=0)
```

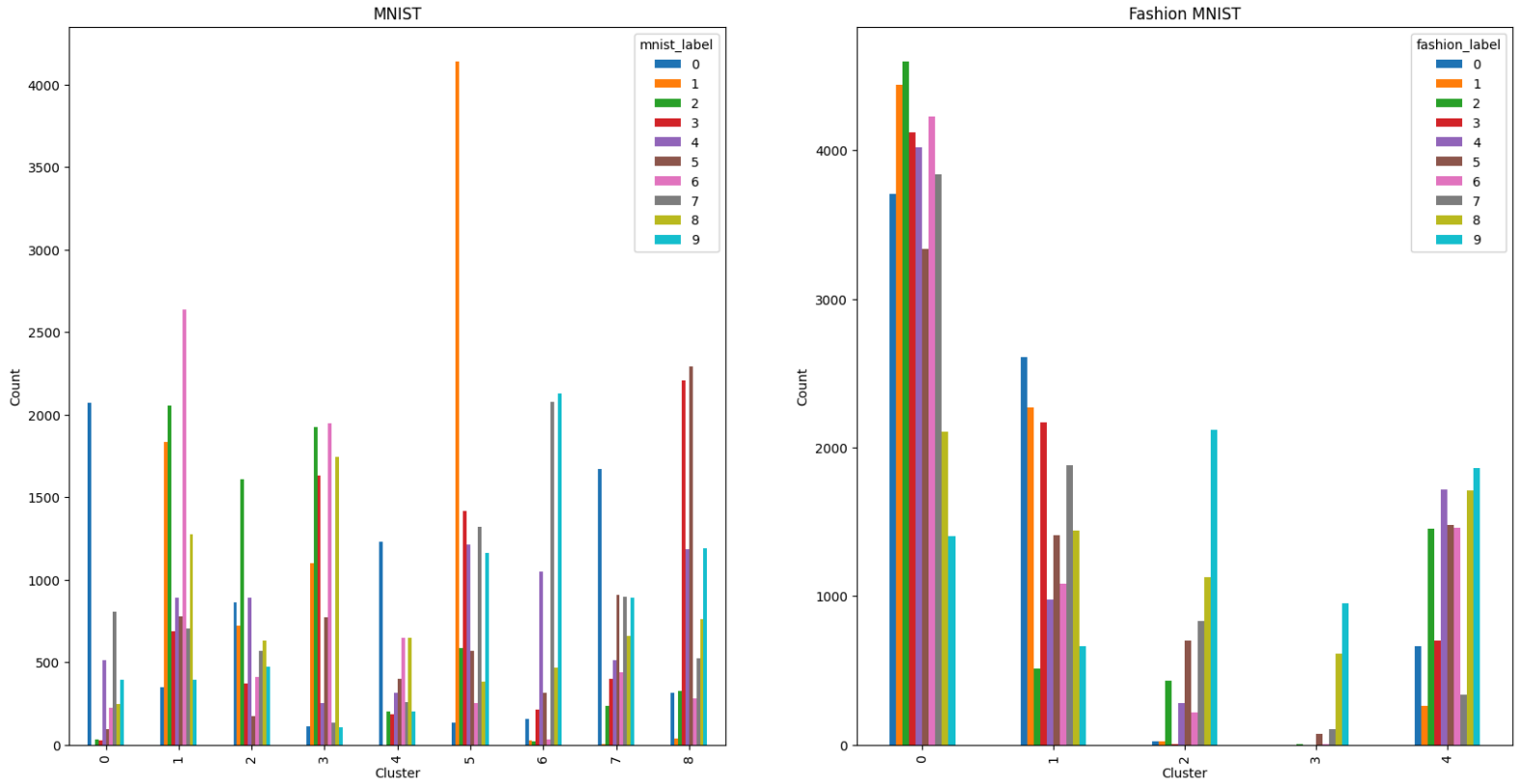


شکل 1-8. نتیجه clustering برای دیتاست Fashion MNIST

همانگونه که از نمودارها نیز واضح است برای MNIST بهترین تعداد cluster برابر 9 و برای Fashion MNIST مقدار 5 بود. حال به بررسی دلیل اینکه چرا به این اعداد به جای مقدار اصلی 10 رسیدیم می‌پردازیم. همانگونه که از دو نمودار زیر مشخص است هر cluster دارای توزیعی از برچسب‌های اصلی می‌باشد که به این معنی است که الگوریتم k-means نتوانسته از روی latent space ساخته شده برای آن‌ها، آن‌ها را به خوبی از هم تشخیص دهد حال سعی می‌کنیم بررسی کنیم مدل در چه شرایطی خوب عمل نکرده.



شکل 1-9. توزیع cluster به صورت stack برای دیتاست‌ها



شکل 1-10. توزیع cluster به صورت unstack برای دیتاست‌ها

برای دیتاست MNIST اگر بررسی کنیم دلیل اینکه 9 تا cluster داریم به جای 10 تا این است که تعدادی از clusterها (1 و 3 و 6 و 8) به طور کامل نتوانستند یک عدد را capture کنند. اگر به عددهایی که در یک cluster هستند دقت کنیم مشاهده می‌کنیم که مثلا 3 و 5 با هم زیاد اشتباه گرفته شده، و یا 2 و 6 و 8 هم تشخصیشان برای مدل سخت بوده.

برای Fashion MNIST مدل به خاطر کیفیت و شباهت خروجی به پنج دسته تقسیم کرده که 5 دسته را می‌توان تقریبی به شکل زیر نام‌گذاری کرد: لباس آستین کوتاه، شلوار، کفش، کیف، لباس استین بلند. بقیه دسته‌ها به دلیل شباهت در یکی از این 5 دسته قرار گرفتند. توزیع‌های clusterها نیز این موضوع را تایید می‌کنند.

به طور کلی الگوریتم روی دیتاست MNIST به خاطر تفاوت بیشتر و ساده‌تر بودن بهتر عمل کرده اما بر روی Fashion MNIST نیز عملکرد قابل قبولی داشته.

پرسش 2 - افزایش داده در مدل FaBert

2-1. افزایش دادگان

در اینجا تعدادی از تکنیک‌های پرکاربرد افزایش داده در پردازش زبان طبیعی (NLP) آورده شده است:

I. جایگزینی هم‌معنی (Synonym Replacement - SR): این روش شامل انتخاب تصادفی یک کلمه غیر توقف (stopword) و جایگزینی آن با هم‌معنی‌اش است.
II. درج تصادفی (Random Insertion - RI): این تکنیک هم‌معنی یک کلمه انتخاب شده تصادفی را در یک موقعیت تصادفی در متن درج می‌کند.
III. تعویض تصادفی (Random Substitution - RS): این تکنیک دو کلمه را به طور تصادفی در متن جابه‌جا می‌کند.
IV. حذف تصادفی (Random Delete - RD): این تکنیک یک کلمه را به طور تصادفی از متن حذف می‌کند.
V. ترجمه معکوس (Back-Translation): این روش شامل ترجمه یک جمله از زبان اصلی به یک زبان دیگر و سپس ترجمه مجدد آن به زبان اصلی است.

در اینجا ما می‌خواهیم از روش Back-Translation استفاده کنیم. توضیح کامل‌تر این روش به شرح زیر است:

1. انتخاب زبان میانی: در این مرحله، یک زبان میانی (زبان دوم) انتخاب می‌شود که جملات به آن زبان ترجمه شوند. معمولاً زبان‌هایی انتخاب می‌شوند که از نظر ساختاری به زبان اصلی نزدیک باشند تا معنای جمله به خوبی حفظ شود. زبان‌های پرکاربرد در این مرحله شامل زبان‌های پرطرفدار مانند انگلیسی، اسپانیایی، و فرانسوی هستند.
2. ترجمه از زبان اصلی به زبان میانی: جملات یا متون اصلی (زبان منبع) به زبان میانی ترجمه می‌شوند. این مرحله با استفاده از مدل‌های ترجمه ماشینی (مانند Google Translate یا مدل‌های یادگیری عمیق ترجمه ماشینی) انجام می‌شود.
3. ترجمه مجدد به زبان اصلی: جملات ترجمه شده به زبان میانی دوباره به زبان اصلی (زبان مقصد) ترجمه می‌شوند. این مرحله نیز با استفاده از مدل‌های ترجمه ماشینی انجام می‌شود.

مثال

فرض کنید یک جمله ساده در زبان فارسی داریم:

- جمله اصلی: "کتاب جدیدی که خواندم بسیار جالب بود."

مراحل ترجمه معکوس به شکل زیر خواهد بود:

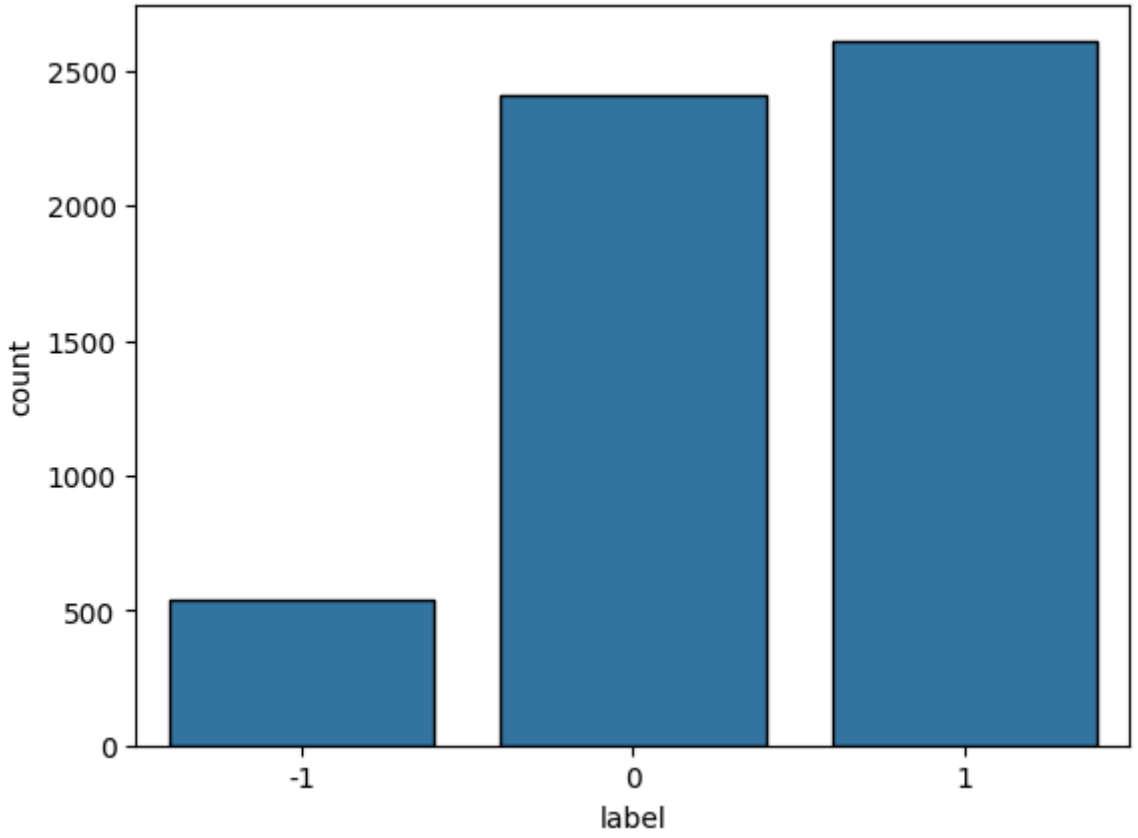
1. ترجمه به زبان میانی (مثلاً انگلیسی):
- "The new book I read was very interesting."
2. ترجمه مجدد به زبان اصلی:
- "کتاب تازه‌ای که خواندم بسیار جذاب بود."

همانطور که می‌بینید، جمله نهایی از نظر معنایی مشابه جمله اصلی است اما با استفاده از کلمات و ساختار زبانی کمی متفاوت.

2-2. پیش‌پردازش

در ابتدا لیبل‌های های 1+ و 2+ را به عنوان 1+ و همچنین لیبل‌های 2- و 1- را به 1- تغییر می‌دهیم. لیبل 0 را نیز بدون تغییر نگه می‌داریم. در نتیجه سه نوع داده داریم:

1. **لیبل 1+**: کامنت مثبت درباره محصول
2. **لیبل 1-**: کامنت منفی درباره محصول
3. **لیبل 0**: نظر خنثی درباره محصول

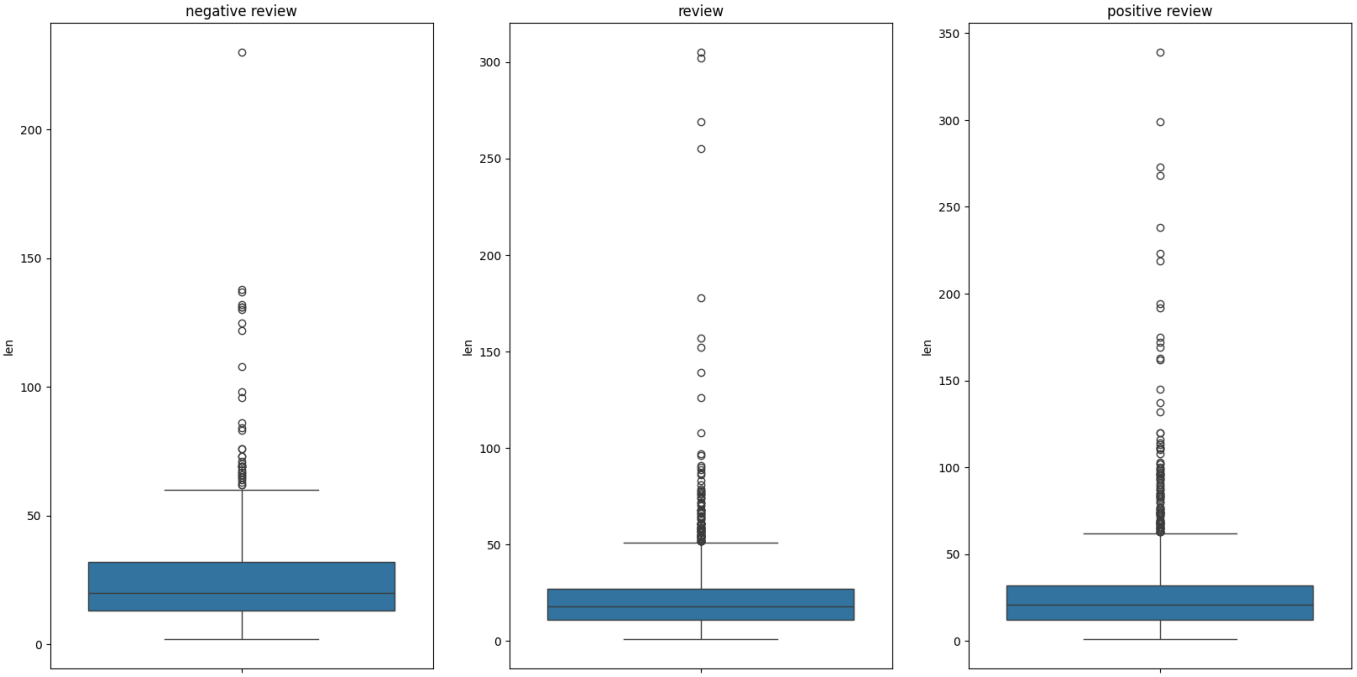


شکل 2-1. توزیع لیبل‌ها در مجموعه داده

سپس پیش‌پردازش را روی متن انجام می‌دهیم[1]، که این کار شامل:

1. حذف تگ‌های html
2. حذف لینک‌ها و url
3. جایگزینی ایموجی با متن معادل آن

[1] این پردازش‌ها در صورت وجود انجام می‌شوند.



شکل 2-2. توزیع تعداد توکن هر کلاس

2-3. افزایش دادگان با Back Translation

در ابتدا ده درصد داده‌های آموزش را برای صحت‌سنجی کنار گذاشتیم. در ادامه بر روی داده‌های آموزش با استفاده از Back Translation و انتخاب زبان فرانسوی، تعداد داده‌های آموزش را دو برابر کردیم.

چند مثال:

```
0
اما حافظه داخلی این گوشی 16 گیگابایت است که با پشتیبانی گوشی از کارت حافظه microSD تا 32 گیگابایت دیگر نیز می توان به آن افزود.
اما حافظه داخلی این گوشی 16 گیگابایت است که با پشتیبانی از کارت حافظه میکرو اس دی تا 32 گیگابایت قابل افزایش است.
---------------------------------------
1
متشکرم دیجي کالا
ممنون دیجی کالا
---------------------------------------
2
```

در ضمن میتونین به عنوان هارد دیسک اینترنال هم ازش استفاده کنین!
به علاوه، می توانید از آن به عنوان هارد دیسک داخلی استفاده کنید!

3
اصلا قصد تبلیغات ندارم.
من اصلا قصد تبلیغ ندارم

4
از نظر ضبط ویدیویی، فیلم های ضبط شده بسیار شارپ و با کیفیت هستند.
از نظر فیلمبرداری فیلم های ضبط شده بسیار واضح و با کیفیت هستند.

5
از اینکه راحت هر برنامه اي را بخواهید میتوانید دانلود کنید و استفاده کنید لذت میبرید.
شما عاشق این واقعیت خواهید بود که می توانید به راحتی هر برنامه ای را که می خواهید دانلود و استفاده کنید.

6
اما در مجموع، صفحه نمایش Optimus VU نیز عملکرد مناسبی را داراست.
اما به طور کلی، صفحه نمایش Optimus VU نیز به خوبی کار می کند.

7
پرينتر خوبي است و كيفيت خوبي دارد و عالي چاپ ميكند اگر پرينتري كارا و عالي مي خواهيد شک نکنید من از ديجي كالا خريدم
چاپگر خوبی است، کیفیت خوبی دارد و به خوبی چاپ می کند. اگر یک چاپگر کارآمد و عالی می خواهید شک نکنید که من آن را از دیجی خریدم.

8
اين گوشي شايد يک ماه در صدر باشد .
این گوشی می تواند یک ماه در بالا باشد.

9
البته اپيد4G دارد که بدرد ما نمي خورد با اپديتي که داره واسه سيستم عامل اپل ارائه ميشه شما دوباره لذت واقعي را خواهيد برد.
البته آیپد 4G دارد که هیچ فایده ای برای ما ندارد. با به روز رسانی پیشنهادی برای سیستم عامل اپل، لذت واقعی خواهید داشت.

10
بخش اول مدل های ارزان تر هستند که از پردازشگرهای آنبرد Intel HD سری های 3000 و 4000 استفاده می کنند.
بخش اول مربوط به مدل‌های ارزان‌تری است که از پردازنده‌های یکپارچه Intel HD 3000 و 4000 استفاده می‌کنند.

11
در کیفیت FastRes 600 این کار 10 ثانیه طول کشید.
با کیفیت FastRes 600 10 ثانیه طول کشید.

12
من از این تبلت خریدم مهمترین نکته اینکه از نظر من باطریش ضعیف است بعد یک ماه استفاده از این تبلت و مقایسه آن با تبلت رفقا پیشنهاد میکنم این تبلت نخرید cpu آن فقط اسمش چهار هسته اي و به نسبت ايسوس و سامسونگ واقعا ضعيف است عرايض بنده را فقط وقتي متوجه ميشويد كه در عمل كارايي اين تبلت با يكي دو برند معتبر مقايسه كنيد به هر حال انتخاب با خودتونه شايد بهتر باشه كمي صبر كنيد و يه تبلت بهتر بخريد.
من این تبلت رو خریدم، مهمتر از همه اینه که به نظر من باتریش کمه، بعد از یک ماه استفاده از این تبلت و مقایسه با تبلت، پیشنهاد میکنم این تبلت رو نخرید، سی پی یوش فقط چهار هسته ای هستش و در مقایسه با ایسوس و سامسونگ واقعا ضعیف است. تنها با مقایسه عملکرد این تبلت با یکی دو برند معتبر متوجه خواهید شد. در هر صورت، انتخاب با شماست، شاید بهتر باشد کمی صبر کنید و بهتر باشد. تبلت.

13
قابلیت پخش ویدئو در پس زمینه .
امکان پخش ویدیو در پس زمینه.

14
كيفيت تصوير همه بازيها هم تاحالا عالي بوده.
کیفیت تصویر همه بازی ها تا الان عالی بوده است.

15
باریکیش حرف نداره.
باریک بودنش مهم نیست

16
بله براي من هم صدا كمه كه براي من مهم نيست.
بله صدا برای من خیلی آرام است که برای من مهم نیست.

17
باتری باتری موجود بر روی Optimus VU یک باتری 2100 میلی آمپر ساعتی لیتیوم یونی می باشد.
باتری باتری Optimus VU یک باتری لیتیوم یونی 2100 میلی آمپری است.

18
هرچند که رزولوشن آن پایین تر از نمایشگر Retinaی iPhone 4 است ولی با وجود بزرگ تر بودن، کیفیتی تقریبا برابر را ارایه می دهد.
با اینکه رزولوشن آن کمتر از نمایشگر رتینا آیفون 4 است، اما با اینکه بزرگتر است، کیفیت تقریبا برابری را ارائه می دهد.

```
-----------------------------------
```

همانطور که مشاهده می‌کنید، عبارات ایجاد کیفیت قابل قبولی دارند و غالبا با حفظ محتوای اصلی، برخی کلمه‌ها و جمله‌بندی‌ها تغییر کرده است.

Fine-Tuning FaBert .4-2

ابتدا مدل FaBert را لود می‌کنیم. سپس با استفاده از توکنایزر آن، دیتاست خود را می‌سازیم. برای این کار هر جمله را به این توکنایز می‌دهیم و سپس شناسه[2]ها و ماسک توجه[3] آن را ذخیره می‌کنیم.

در ادامه شبکه عصبی خود را تعریف می‌کنیم. در ابتدای آن مدل FaBert قرار می‌دهیم و در ادامه یک لایه Dense با سه نورون و فعال‌ساز Softmax قرار می‌دهیم تا عمل دسته‌بندی[4] را انجام دهد.

```
def create_model(base_model):
	input_ids = Layers.Input(shape=(MAXLEN,), dtype=tf.int32,
name="input_ids")
	attention_mask = Layers.Input(shape=(MAXLEN,),
dtype=tf.int32, name="attention_mask")
	bert_output = base_model(input_ids,
attention_mask=attention_mask)[1]
	dense_output = Layers.Dense(NUMCLASS,
activation='softmax')(bert_output)
	model = Model(inputs=[input_ids, attention_mask],
outputs=dense_output)


	bert_layers = bert_model.layers[0].encoder.layer
	bert_layers[-1].trainable = True
	for i in range(len(bert_layers) - 1):
		bert_layers[i].trainable = False
	return model
```

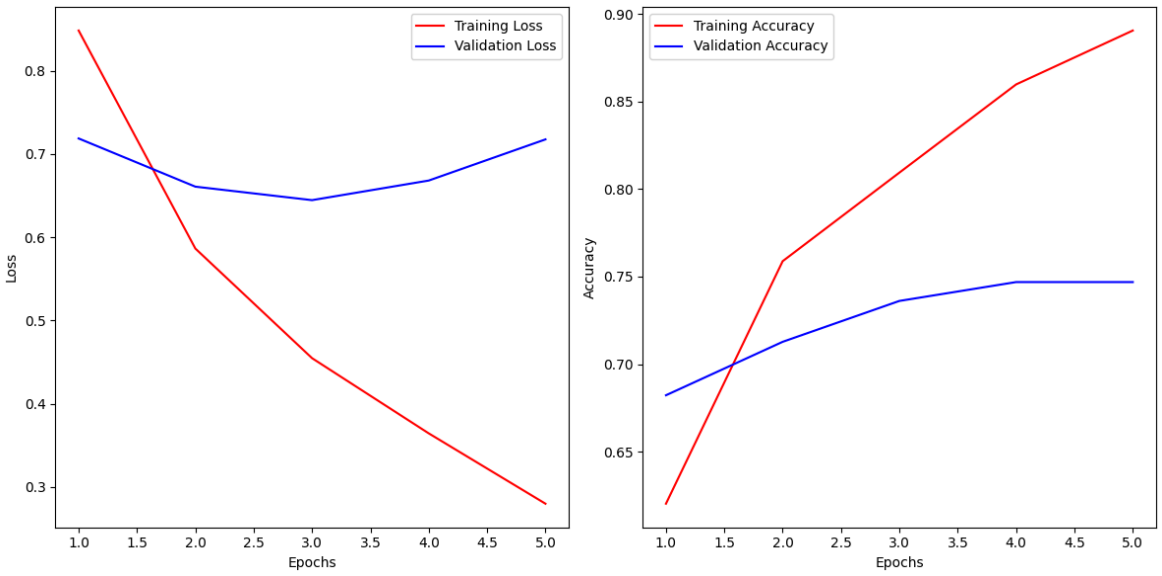
[2] Input ID

[3] Attention Mask

[4] Classification

در انتها لایه های مدل FaBert را فریز می‌کنیم و تنها لایه آخر آن را قابل آموزش نگه می‌داریم تا به درستی آن را Fine-Tune کنیم.

2-5. ارزیابی و تحلیل نتایج



شکل 2-3. نمودار هزینه و دقت مدل بدون افزایش دادگان

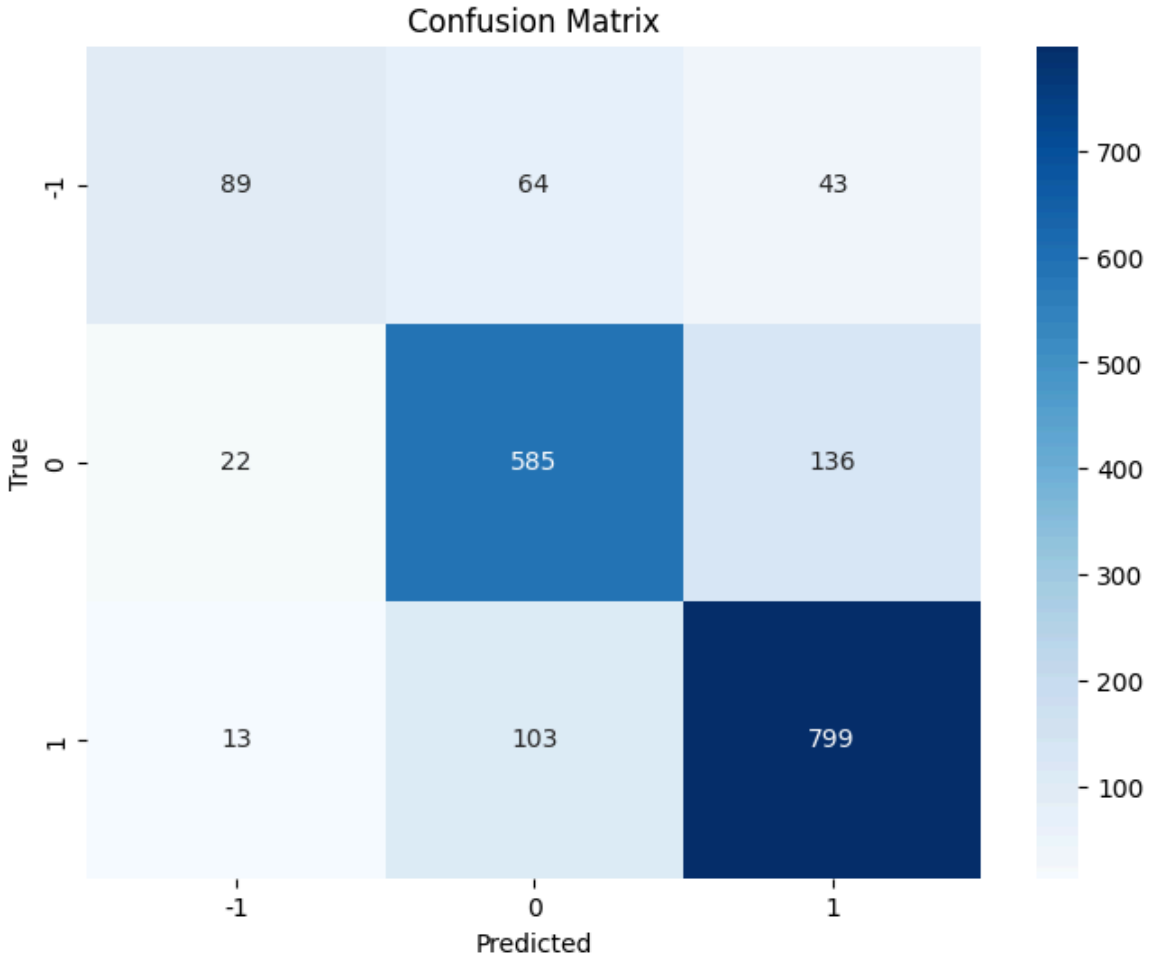
Metric	Value
Accuracy	0.7945

Class	Precision	Recall	F1-score
Class -1	0.817	0.8732	0.8442
Class 0	0.7177	0.4541	0.5563
Class 1	0.7779	0.7873	0.7826

Average Type	Precision	Recall	F1-score

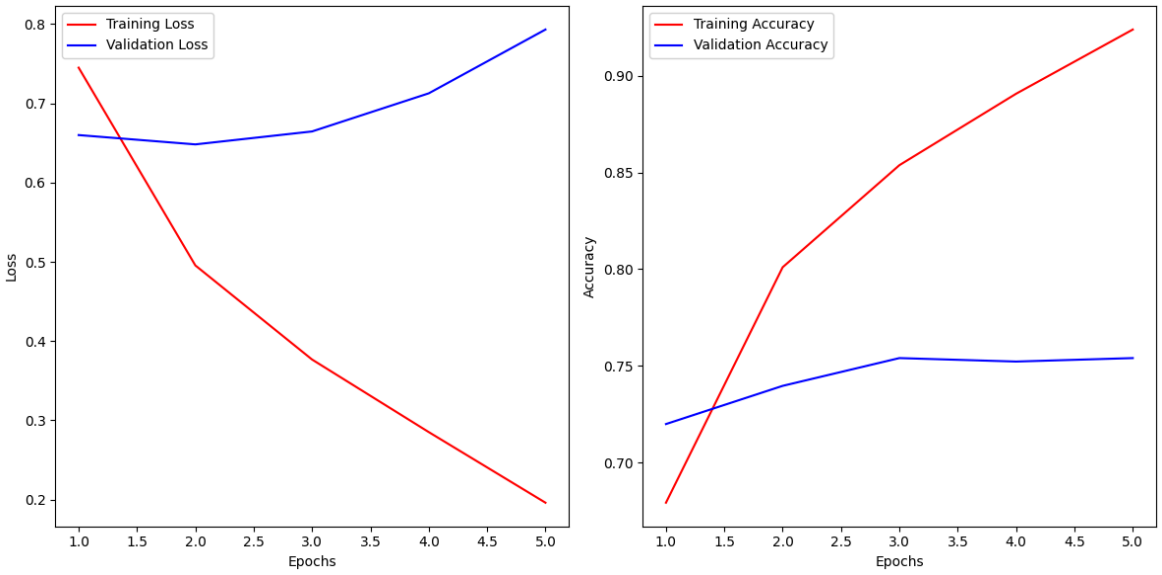
Macro average	0.7709	0.7049	0.7277
Micro average	0.7945	0.7945	0.7945
Weighted average	0.7908	0.7945	0.7891

جدول 2-1. سنجه‌های داده‌های تست در مدل بدون افزایش دادگان



شکل 2-4. ماتریس آشفتگی داده‌های آزمون در مدل بدون افزایش دادگان

در انتها داده‌ها افزوده شده را با داده‌های آموزش ادغام می‌کنیم و همچنین داده‌های تکراری را حذف می‌کنیم.



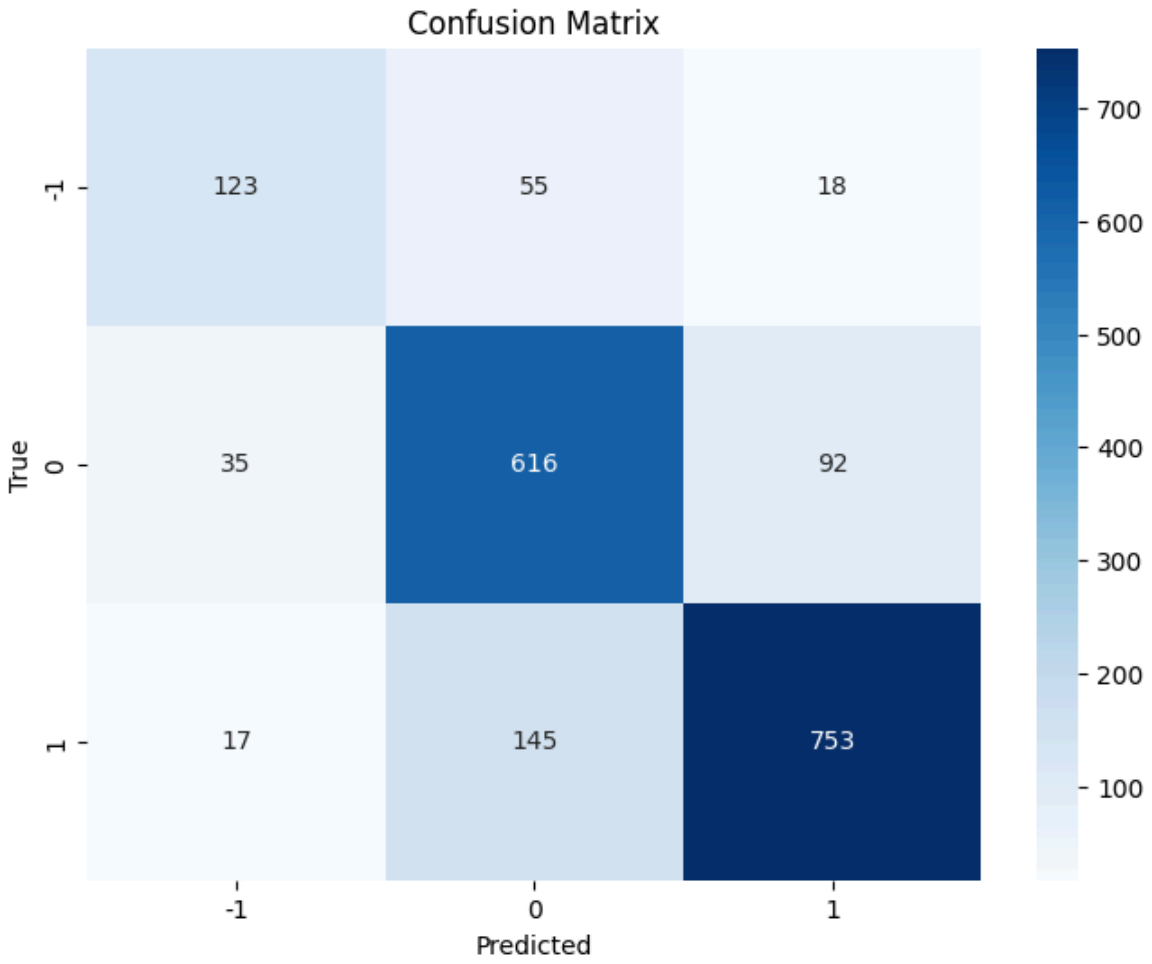
شکل 2-5. نمودار هزینه و دقت مدل با افزایش دادگان

Metric	Value
Accuracy	0.8047

Class	Precision	Recall	F1-score
Class -1	0.8725	0.823	0.847
Class 0	0.7029	0.6276	0.6631
Class 1	0.7549	0.8291	0.7903

Average Type	Precision	Recall	F1-score
Macro average	0.7768	0.7599	0.7668
Micro average	0.8047	0.8047	0.8047
Weighted average	0.8075	0.8047	0.8048

جدول 2-2. سنجه‌های داده‌های تست در مدل با افزایش دادگان



شکل 2-6. ماتریس آشفتگی داده‌های آزمون در مدل بدون افزایش دادگان

همانطور که می‌توانید مشاهده کنید دقت مدل با افزایش دادگان 1 درصد افزایش یافته است. همچنین امتیاز F-1 کلاس صفر از 55 درصد به 66 درصد افزایش یافته است که مقدار قابل توجهی است. در حالت میانگین هم هر سه معیار F-1 و Recall و Precision نیز افزایش یافته است. در نهایت می‌توانیم ببینیم که افزایش دادگان باعث بهبود کلی عملکرد شبکه شده است.

پرسش 3. کلمه بیدار باش

3-1. جمع‌آوری داده

برای ضبط صداها به مدت دو ثانیه از تابع زیر استفاده کردیم:

```
def record_2_sec_voice(folder_path):
      fs = 44100
      seconds = 2

      myrecording = sd.rec(int(seconds * fs), samplerate=fs,
channels=2, dtype="int16")
      print("Recording Audio")
      sd.wait()
      print("Audio recording complete")

      num_files = len([f for f in os.listdir(folder_path)if
os.path.isfile(os.path.join(folder_path, f))])
      write(folder_path + "/output" + str(num_files) + ".wav",
fs, myrecording)
```

سپس برای اینکه برچسب داده‌ها به درستی زده شود از دو تابع زیر کمک گرفتیم

```
def record_wake_word(num_files):
      path = "../data/wake_word"

      ipd.clear_output(wait=True)
      for i in tqdm(range(num_files)):
      record_2_sec_voice(path)
      print("Recorded Wake Word")
      ipd.clear_output(wait=True)
      time.sleep(1)
```

```
def record_not_wake_word(num_files):
      path = "../data/not_wake_word"
```

```
    ipd.clear_output(wait=True)
    for i in tqdm(range(num_files)):
    record_2_sec_voice(path)
    print("Recorded Not Wake Word")
    ipd.clear_output(wait=True)
    time.sleep(1)
```

سپس به کمک این دو تابع برای هر دو بخش (کلمه wake up و بقیه کلمات) 100 وویس ضبط کردیم.

3-2. پیش‌پردازش و استخراج ویژگی

پیش‌پردازش داده‌های صوتی به طور کلی شامل موارد زیر می‌تواند باشد:

1. Feature Extraction

- MFCC - Mel-Frequency Cepstral Coefficients: این روش یکی از پرکاربردترین روش‌های استخراج ویژگی‌ها در تحلیل صوت است که ویژگی‌های فرکانسی صدا را بر مبنای مقیاس مل تحلیل می‌کند. این ویژگی برای تشخیص گفتار مفید است اما برای تشخیص موسیقی کمک چندانی نمی‌کند.
- Spectrogram: یک نمایش بصری از طیف فرکانسی سیگنال صوتی که تغییرات فرکانس را در طول زمان نشان می‌دهد.
- Chroma Features: این ویژگی‌ها نشان‌دهنده توزیع انرژی طیف فرکانسی در 12 گروه (هر گروه مربوط به یک نت موسیقی) هستند و برای تحلیل‌های موسیقی مفید هستند، اما برای تشخیص گفتار مناسب نیست.
- Zero-Crossing Rate: این ویژگی نشان‌دهنده تعداد دفعاتی است که سیگنال صوتی از محور صفر عبور می‌کند و به تشخیص صداهای مختلف کمک می‌کند.

2. Normalization

- Amplitude Normalization: تنظیم دامنه سیگنال صوتی به یک محدوده مشخص برای کاهش تأثیر نویز و اختلافات شدت صدا.

- Mean and Variance Normalization: تنظیم میانگین و واریانس سیگنال برای کاهش تأثیر تفاوت‌های دینامیکی بین سیگنال‌ها.

3. Noise Reduction

- Spectral Subtraction: کاهش نویز با تخمین طیف نویز و کسر آن از طیف سیگنال.

- Wiener Filtering: یک فیلتر تطبیقی برای کاهش نویز که براساس مدل‌های آماری سیگنال و نویز عمل می‌کند.

4. Data Augmentation

- Time Stretching: کشیدن یا فشرده کردن سیگنال صوتی در زمان بدون تغییر فرکانس.
- Pitch Shifting: تغییر ارتفاع صدا (pitch) بدون تغییر سرعت پخش.
- Adding Noise: افزودن نویز سفید یا سایر نویزها به سیگنال برای افزایش تنوع داده‌ها.

5. Transforms

- Fourier Transform: تحلیل فرکانسی سیگنال با استفاده از تبدیل فوریه.

- Short-Time Fourier Transform - STFT: تحلیل فرکانسی در بازه‌های زمانی کوتاه برای مشاهده تغییرات فرکانس در طول زمان.

- Wavelet Transform: تحلیل چند رزولوشنی سیگنال که می‌تواند جزئیات بیشتری از سیگنال را نشان دهد.

6. Filtering

- Band-pass Filter: اجازه عبور به یک باند فرکانسی خاص و حذف فرکانس‌های خارج از این باند.

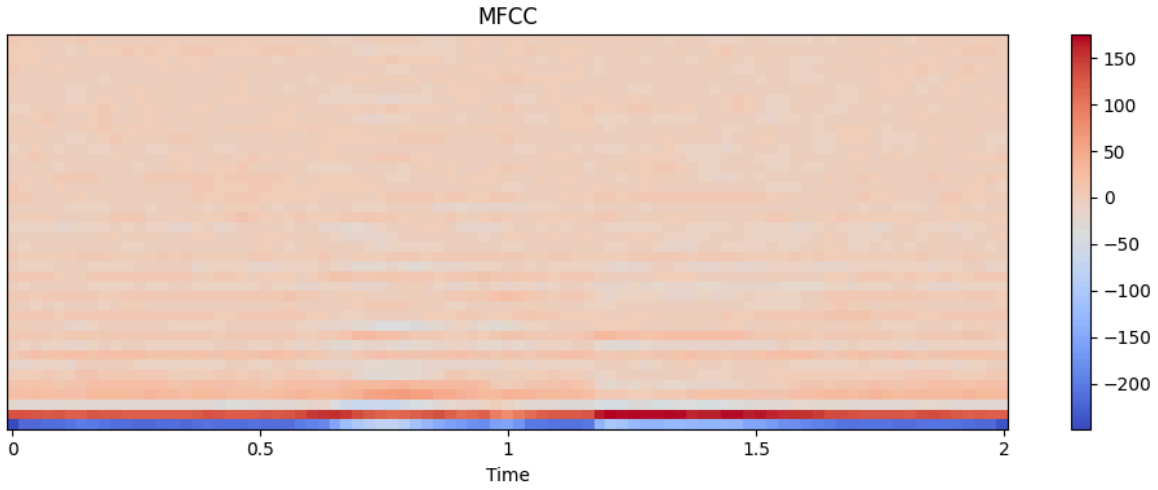
- High-pass and Low-pass Filters: حذف فرکانس‌های پایین‌تر یا بالاتر از یک مقدار مشخص.

در این جا ما ابتدا به شکل زیر feature extraction را انجام دادیم:

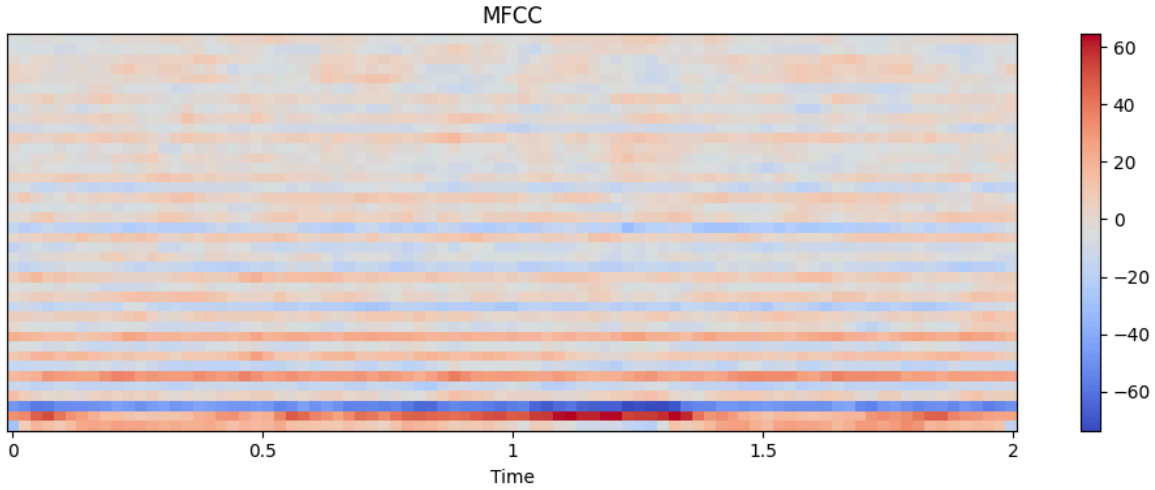
```
def extract_features(file_name):
      try:
      audio, sample_rate = librosa.load(file_name,
res_type="kaiser_fast")
      mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=40)
      mfccsscaled = np.mean(mfccs.T, axis=0)
      except Exception as e:
      print("Error encountered while parsing file: ", file_name)
      print(e)
      return None
```

```
    return mfccsscaled, mfccs

def get_features(data):
    extracted_features = []
    for idx in tqdm(range(data.shape[0])):
    file_name = data["file"].iloc[idx]
    final_class_labels = data["label"].iloc[idx]
    features, mfccs = extract_features(file_name)
    extracted_features.append([features, final_class_labels])

    return extracted_features, mfccs
```

که خروجی آن به شکل زیر بود:



شکل 3-1. نمونه mfcc استخراج شده برای یک وویس در دیتاست wake word



شکل 3-2. نمونه mfcc استخراج شده برای یک وویس در دیتاست not wake word

برای نرمال‌سازی از به شکل زیر عمل کردیم:

```
wake_word_features_df["feature"] =
wake_word_features_df["feature"].apply(lambda x: x /
np.max(np.abs(x)))
not_wake_word_features_df["feature"] =
not_wake_word_features_df["feature"].apply(lambda x: x /
np.max(np.abs(x)))
```

سپس برای بخش augmentation از روش‌های زیر استفاده کردیم و برای همه دیتاست نمونه‌های جدید ساختیم:

```
def add_noise(data):
      noise = 0.001 * np.random.uniform(size=len(data))
      return data + noise

def shift(data):
      return np.roll(data, 1600)

def stretch(data, rate=1):
      input_length = 1024
      data = librosa.effects.time_stretch(data, rate=rate)
      if len(data) > input_length:
      data = data[:input_length]
      else:
      data = np.pad(data, (0, max(0, input_length - len(data))),
"constant")

      return data

def pitch(data, sampling_rate, pitch_factor):
      return librosa.effects.pitch_shift(data, sr=sampling_rate,
n_steps=pitch_factor)

def augment_data(data):
      augmented_data = []
      for idx in range(data.shape[0]):
      feature = data["feature"].iloc[idx]
      label = data["label"].iloc[idx]
      augmented_data.append([feature, label])
      augmented_data.append([add_noise(feature), label])
      augmented_data.append([shift(feature), label])
      augmented_data.append([stretch(feature), label])
      augmented_data.append([pitch(feature, 22050, 4), label])
      return augmented_data
```

3-3.طراحی شبکه عصبی

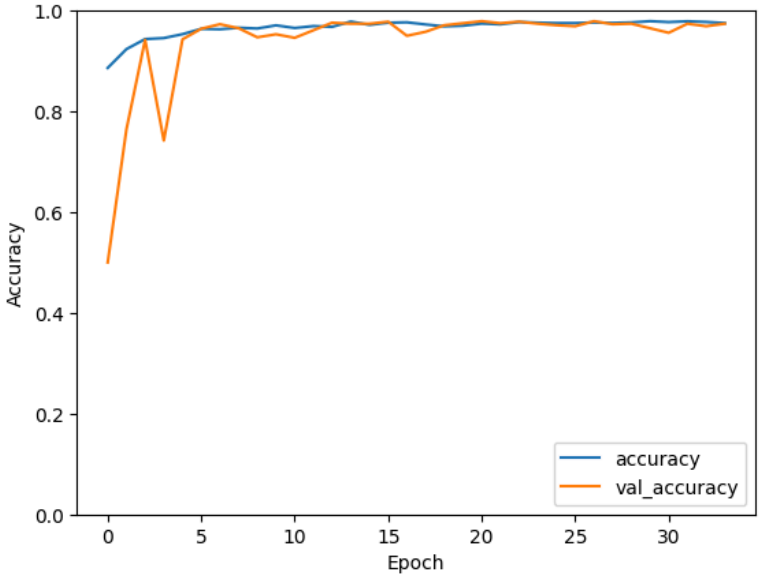
با توجه به اینکه وویس‌ها دارای طول ثابت و ویژگی‌های استخراج شده نیز دارای طول ثابت 40 بودند از شبکه‌های مختلفی می‌توانستیم استفاده کنیم مانند fully connected، و یا convolution و یا RNNها و ... که در اینجا ما از شبکه convolution استفاده کردیم. در صورتی که طول یکسانی نداشت می‌شد از روش‌هایی مانند padding, windowing, dynamic time warping و ... استفاده کرد یا حتی از مدل‌های recurrent مانند RNN نیز بهره برد.

```
def create_model():
      model = Sequential()
      model.add(Conv1D(32, 5, activation="relu", input_shape=(40,
1)))
      model.add(BatchNormalization())
      model.add(MaxPooling1D(2))
      model.add(Dropout(0.2))
      model.add(Conv1D(64, 5, activation="relu"))
      model.add(BatchNormalization())
      model.add(MaxPooling1D(2))
      model.add(Dropout(0.2))
      model.add(Flatten())
      model.add(Dense(128, activation="relu"))
      model.add(Dropout(0.2))
      model.add(Dense(2, activation="softmax"))
      model.compile(loss="categorical_crossentropy",
optimizer="adam", metrics=["accuracy"])
      return model
```

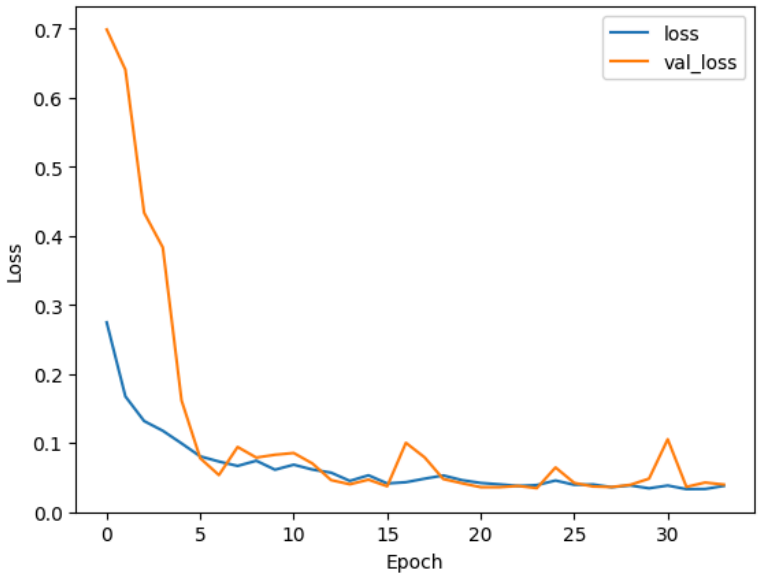
```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)            (None, 36, 32)            192

 batch_normalization_2 (Batc  (None, 36, 32)           128
 hNormalization)

 max_pooling1d_2 (MaxPooling  (None, 18, 32)           0
 1D)

 dropout_3 (Dropout)          (None, 18, 32)            0

 conv1d_3 (Conv1D)            (None, 14, 64)            10304

 batch_normalization_3 (Batc  (None, 14, 64)           256
 hNormalization)

 max_pooling1d_3 (MaxPooling  (None, 7, 64)            0
 1D)

 dropout_4 (Dropout)          (None, 7, 64)             0
```

```
 flatten_1 (Flatten)       (None, 448)               0

 dense_2 (Dense)           (None, 128)               57472

 dropout_5 (Dropout)       (None, 128)               0

 dense_3 (Dense)           (None, 2)                 258

=================================================================
Total params: 68,610
Trainable params: 68,418
Non-trainable params: 192
_________________________________________________________________
```

که پس از آموزش نتایج زیر را مشاهده کردیم:



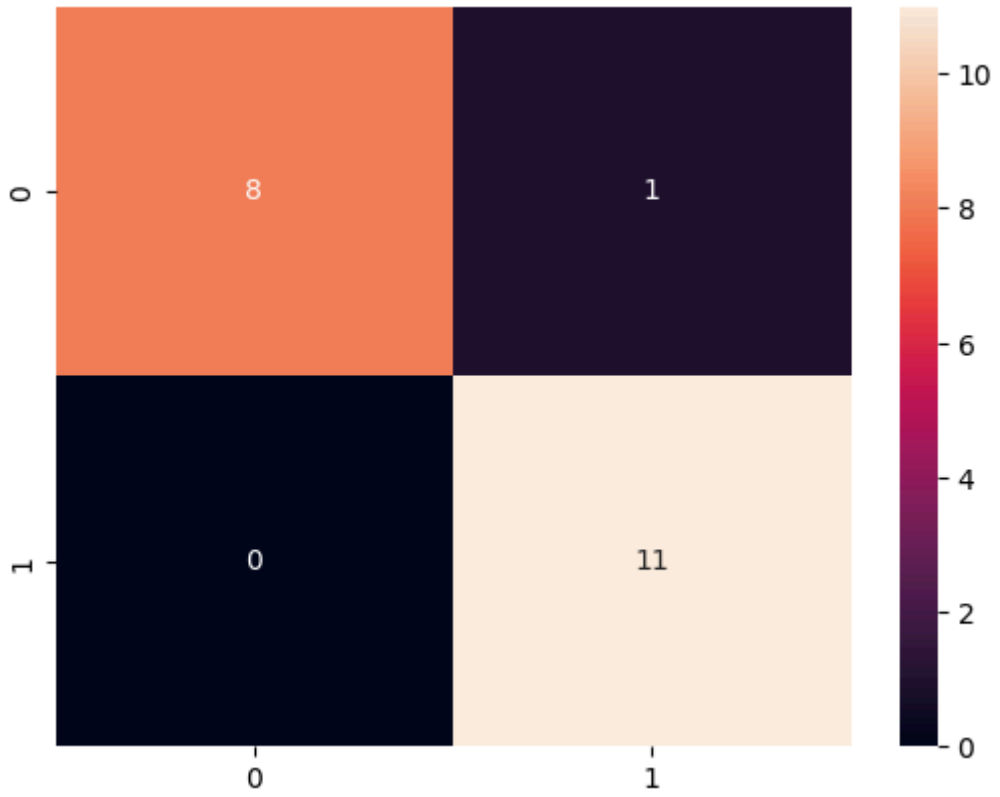
شکل 3-3. نمونه accuracy مدل در زمان آموزش



شکل 3-4. نمونه loss مدل در زمان آموزش

در نهایت نیز دقت مدل را بررسی کردیم:

```
Test Loss:  0.03955698758363724
Test Accuracy:  0.9739999771118164
```



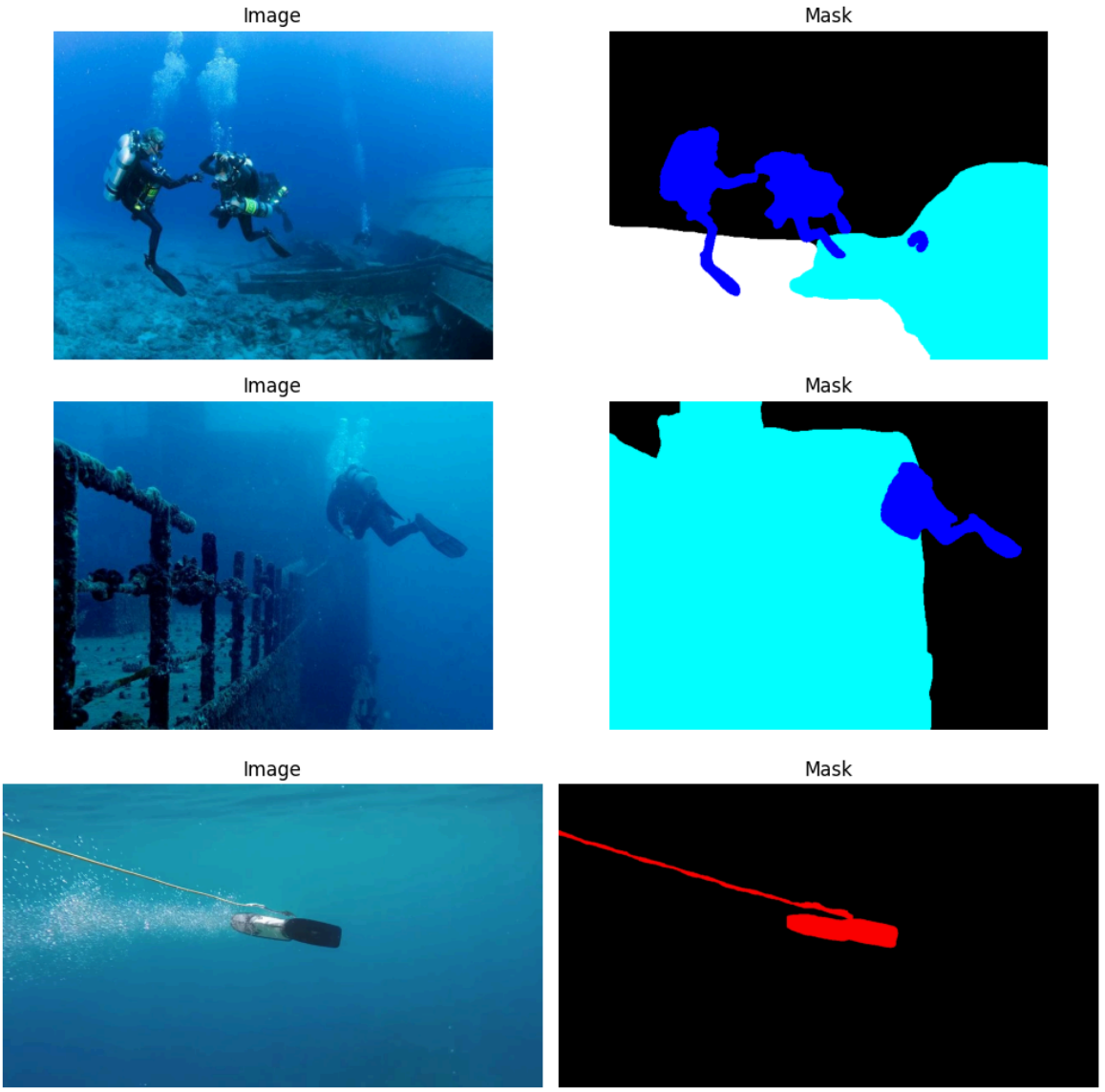
شکل 3-5. confusion matrix برای 20 نمونه تست

نتایج نشان می‌دهد که مدل CNN ما با دقت بسیار خوبی قابلیت تشخیص و طبقه‌بندی صدای wake up از بقیه صداها را دارد که در نتیجه می‌توان از این مدل یا مدل‌های شبیه این برای voice assistant-ها استفاده کرد.

پرسش 4. شبکه بخش‌بندی[5] تصاویر

4-1. دیتاست

دیتاست مود استفاده حاوی تعدادی تصاویر زیر دریا به همراه ماسک بخش‌بندی آن‌ها است. در کل 8 نوع کلاس مختلف وجود دارد که هر کدام با یک رنگ منحصربفرد مشخص شده‌اند.



شکل 4-1. نمایش چند نمونه عکس به همراه ماسک نظیر آن.

[5] Segmentation

افزایش دادگان

برای این کار تابع افزایش داده را را در پایپ-لاین خواندن عکس ها قرار دادیم و از تکنیک‌های زیر استفاده کردیم

1. تغییر روشنایی
2. چرخش افقی
3. چرخش عمودی

لازم به ذکر است که حالت های دیگر با توجه به ماهیت عکس‌ها باعث افت کیفیت می‌شد.

نرمال‌سازی داده‌ها

به علت قرارگیری مقادیر RGB بین 0 تا 255، برای نرمال سازی کافی است تا اعداد را به 255 تقسیم کنیم تا همگی بین 0 تا 1 قرار بگیرند. همچنین به علت استفاده تابع فعال‌ساز ReLU در شبکه از نرمال‌سازی گوسی استفاده نکردیم تا مقادیر منفی نداشته باشیم.

4-2. شبکه مورد استفاده

شبکه Unet

یک معماری شبکه عصبی کانولوشنی (CNN) است که برای وظایف بخش‌بندی در حوزه بینایی ماشین طراحی شده است. نام "UNet" از ساختار U شکل شبکه گرفته است که شامل یک مسیر رمزگذار و یک مسیر رمزگشا است. مسیر رمزگذار، متناظر با تصویر ورودی، ابعاد فضایی را کاهش داده ویژگی‌ها را از تصویر استخراج می‌کند، در حالی که مسیر رمزگشا، ابعاد فضایی را بازیابی کرده و نقشه بخش‌بندی را تولید می‌کند.

در ادامه، یک نمای کلی از معماری UNet آورده شده است:

۱. **مسیر رمزگذار**[6]: مسیر رمزگذار تصویر ورودی را دریافت می‌کند و با استفاده از لایه‌های کانولوشنی و لایه‌های پولینگ، ابعاد فضایی آن را به تدریج کاهش می‌دهد. این لایه‌ها ویژگی‌های تصویر را با حفظ اطلاعات فضایی استخراج می‌کنند.

[6] Encoder

۲. **پل**[7]: در پایین معماری شکل U، یک اتصال پل وجود دارد که ویژگی‌های با کیفیت بالا یادگرفته شده توسط رمزگذار را نگه می‌دارد. این پل به رمزگشا اجازه می‌دهد در فرایند باز نمودن، به هر دو ویژگی نقشه‌های با کیفیت پایین و با کیفیت بالا دسترسی داشته باشد.

۳. **مسیر رمزگشا**[8]: مسیر رمزگشا ویژگی‌های کاهش یافته از رمزگذار را دریافت می‌کند و آن‌ها را به اندازه اولیه تصویر ورودی بزرگ می‌کند. هر مرحله از باز نمودن شامل ترکیبی از افزایش اندازه (به عنوان مثال با استفاده از تراکم خطی) و لایه‌های کانولوشنی است. هدف از رمزگشا بازیابی اطلاعات فضایی از دست رفته در فرایند رمزگذاری و تولید نقشه نهایی بخش‌بندی است.

۴. **اتصال‌های پرش**[9]: UNet از اتصال‌های پرش برای اتصال لایه‌های متناظر رمزگذار و رمزگشا استفاده می‌کند. این اتصالات ویژگی‌های رمزگذار را با ویژگی‌های رمزگشا در همان رزولوشن ارتباط می‌دهند. اتصالات پرش کمک می‌کنند تا جزئیات ریز را حفظ کنند و به شبکه اطلاعات محلی را ارائه دهند تا دقت بخش‌بندی را بهبود بخشند.

۵. **خروجی**: خروجی نهایی UNet یک نقشه بخش‌بندی است با همان ابعاد فضایی تصویر ورودی. هر پیکسل در نقشه بخش‌بندی یک برچسب کلاس را نشان می‌دهد که نشان‌دهنده دسته‌بندی پیش‌بینی شده برای پیکسل متناظر در تصویر ورودی است.

UNet به دلیل قابلیت مدیریت همزمان اطلاعات محلی و سراسری به محبوبیت رسیده است. اتصالات پرش به شبکه امکان استفاده از ویژگی‌های با کیفیت پایین و با کیفیت بالا را فراهم می‌کنند، که بهبود جزئیات ریز را حفظ کرده و آگاهی از متناظر های سراسری را حفظ می‌کنند. این معماری با موفقیت در برنامه‌های مختلف بخش‌بندی مانند تشخیص تصاویر پزشکی، تشخیص سلول‌ها و تشخیص اشیاء در صحنه‌های طبیعی مورد استفاده قرار گرفته است.

لایه‌های مورد استفاده

1. **لایه ورودی (Input)**: این لایه ورودی تصویر را به عنوان ورودی شبکه دریافت می‌کند. ابعاد ورودی به اندازه (128، 128، 3) است که نشان‌دهنده ابعاد تصویر و عمق رنگ آن است.
2. **لایه‌های کانولوشن (Conv2D)**: این لایه‌ها عملیات کانولوشن را روی ورودی انجام می‌دهند. هر لایه کانولوشن یک تعدادی فیلتر با اندازه ویژگی خاص خود دارد و با استفاده از تابع فعال‌سازی ReLU، نقاط قوی تصویر را برجسته می‌کند.

---

[7] Bridge

[8] Decoder

[9] Skip Connections

3. **لایه نرمال‌سازی بچ (Batch Normalization)**: لایه Batch Normalization با محاسبه میانگین و واریانس مینی‌بچ، ورودی‌ها را نرمال‌سازی کرده و سپس با استفاده از پارامترهای قابل آموزش گاما و بتا آنها را تغییر مقیاس و جابجا می‌کند. این لایه به بهبود پایداری و سرعت آموزش شبکه‌های عصبی کمک می‌کند.
4. **لایه‌های ادغام حداکثرگیر (MaxPooling2D)**: این لایه‌ها با استفاده از عملیات حذف نمونه‌ها حداکثر گیری ابعاد تصویر را کاهش می‌دهند. این کار باعث می‌شود که ویژگی‌های مهم تصویر حفظ شده و تعداد پارامترها و محاسبات در شبکه کاهش یابد.
5. **لایه‌های افزایش اندازه (UpSampling2D)**: این لایه‌ها با استفاده از عملیات ترکیب و افزایش اندازه (UpSampling) ابعاد تصویر را افزایش می‌دهند. این کار باعث می‌شود که اطلاعات دقیق‌تری در مقیاس بزرگتر در دسترس باشد.
6. **لایه‌های الحاق (Concatenate)**: این لایه‌ها از دو ورودی مختلف (لایه‌های قبلی) استفاده کرده و آن‌ها را در یک محور مشخص (محور سوم) ترکیب می‌کنند.
7. **لایه‌ی خروجی (Conv2D و Sigmoid)**: در انتها، با استفاده از لایه Conv2D با یک فیلتر به ابعاد (1, 1) و تابع فعال‌سازی Sigmoid، خروجی نهایی شبکه تولید می‌شود. این خروجی یک تصویر به ابعاد ورودی است که مقادیر آن بین 0 و 1 قرار دارد و مقدار نرمال شده یک پیکسل را نشان می‌دهد.

شرح بلوک‌ها

1. بلوک‌های Encoder:

این بلوک‌ها شامل چهار لایه (شش لایه، اگر تابع فعال‌ساز را جداگانه در نظر بگیریم) می باشند. یک لایه کانولوشنی که ابعاد فیلترهای آن 3×3 می‌باشد. با فعال‌ساز RelU که در ادامه‌ی آن لایه Dropout می‌آید. این لایه‌ها دو بار تکرار می‌شوند. دقت کنید تعداد فیلترهای لایه‌های کانولوشنی در بلوک‌های مختلف، متفاوت است. همچنین ضریب dropout برابر 0.2 درنظر گرفته شده‌است.

علت تفاوت نرخ dropout با مقاله، جلوگیری از بیش‌برازش به علت ساده‌تر بودن وظیفه شبکه مورد استفاده می‌باشد.

```
Conv2D -> RelU -> BatchNorm -> Conv2D -> RelU -> BatchNorm
```

2. اتصال بلوک‌های Encoder با یک `MaxPooling2D` با اندازه 2×2 انجام می‌شود.
3. بلوک‌های Decoder:

این‌بلوک‌ها در ابتدا با فیچر مپ هم‌ابعادشان در بخش encoding الحاق می‌شوند و سپس تعدادی لایه مانند بخش encoder دارند و در انتهای لایه `UpSample2D` مورد استفاده قرار می‌گیرد.

```
Concatenate Conv2D -> RelU -> BatchNorm -> Conv2D -> RelU ->
BatchNorm -> UpSample2D
```

شبکه Ta-Unet

یکی از تفاوت‌های کلیدی TA-Unet نسبت به U-Net استفاده از مکانیزم توجه است. مکانیزم توجه به مدل اجازه می‌دهد که به بخش‌های مهم تصویر بیشتر توجه کند و بخش‌های کم اهمیت‌تر را نادیده بگیرد. این کار به افزایش دقت مدل در تشخیص و بخش‌بندی اجزای مهم تصویر کمک می‌کند.

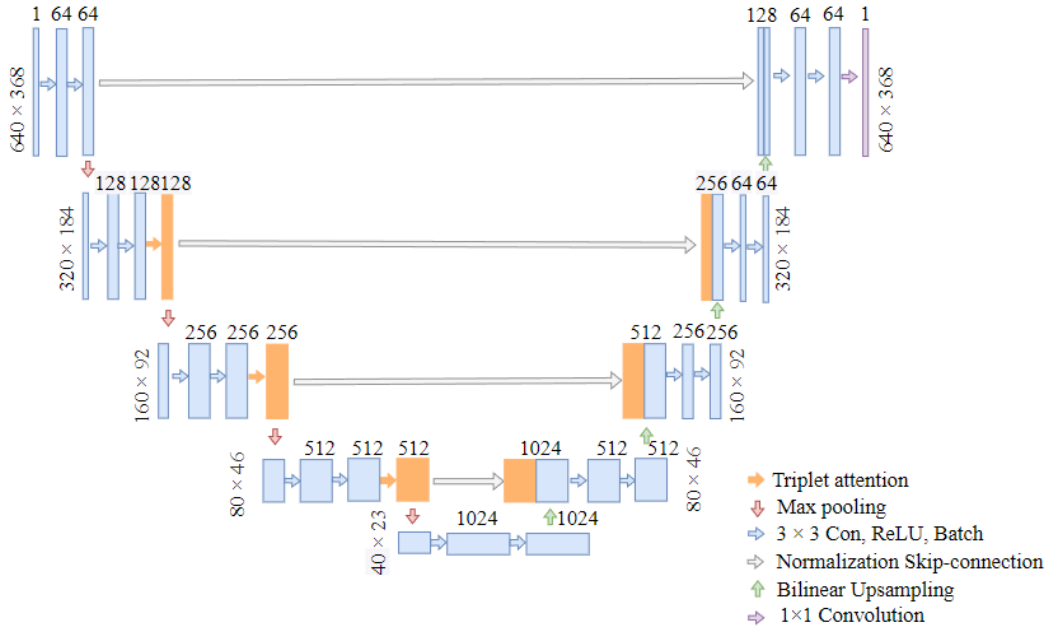
برای این کار یک لایه مکانیزم توجه سه‌گانه[10] تعریف می‌کنیم که به طور موازی برای سه حالت ممکن از انتخاب سه بعد یک تصوری یعنی طول × عرض، طول × کانال[11] و عرض × کانال، توجه را محاسبه می‌کنیم و سپس میانگین آنها را به دست میاوریم. دقت کند که این توجه را انتهای بلوک انکودر قرار می‌دهیم تا ویژگی‌های مهم را فیلتر کند و سپس به طرف مقابل وصل کنیم و یک skip connection را تشکیل دهیم و ای ویژگی‌های فیلتر شده با ویژگی های دیکود شده، الحاق[12] شوند.

قابل ذکر است به علت پیچیدگی بیش از حد این مدل، کمی ساده‌تر از جزئیات مقاله این شبکه را پیاده‌سازی کردیم.

[10] Triplet Attention

[11] همان Channel های یک عکس، در اینجا سه کانال RGB

[12] Concatenate



شکل 4-2. معماری شبکه Ta-Unet

تابع هزینه

Lovász Softmax و Cross-Entropy دو تابع هزینه[13] هستند که برای بهبود عملکرد مدل‌های یادگیری عمیق در تقسیم‌بندی تصویر استفاده می‌شوند. در بسیاری از موارد، ترکیب این دو تابع می‌تواند نتایج بهتری نسبت به استفاده از هر یک به تنهایی ارائه دهد.

Lovász Softmax Loss[14]

Lovász Softmax Loss یک تابع هزینه است که برای بهبود عملکرد مدل‌های تقسیم‌بندی چندکلاسه طراحی شده است. این تابع بر اساس تابع Lovász ساخته شده و مستقیماً mIoU را بهینه‌سازی می‌کند. مزایای اصلی این تابع عبارتند از:

- **مستقیماً بهینه‌سازی mIoU:** برخلاف سایر توابع هزینه Lovász Softmax مستقیماً به معیار mIoU مرتبط است، که این امر منجر به بهبود دقت در تقسیم‌بندی‌های دقیق‌تر می‌شود.
- **مؤثر در داده‌های نامتعادل:** این تابع زیان در مواردی که توزیع کلاس‌ها نامتعادل است، عملکرد بهتری دارد.

[13] Loss Function

[14] برای پیاده سازی این تابع، از این لینک بهره جستیم.

Cross-Entropy Loss

تابع CE یک تابع هزینه رایج است که در مسائل دسته‌بندی چندکلاسه استفاده می‌شود. فرمول آن به صورت زیر است.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

ترکیب Lovász Softmax و Cross-Entropy

ترکیب Lovász Softmax Loss و Cross-Entropy Loss می‌تواند مزایای هر دو تابع هزینه را به دست بیاورد. این ترکیب به مدل اجازه می‌دهد تا هم دقت کلی (با CE) و هم دقت جزئی‌تر در تقسیم‌بندی کلاس‌ها (با Lovász Softmax) را بهبود بخشد.

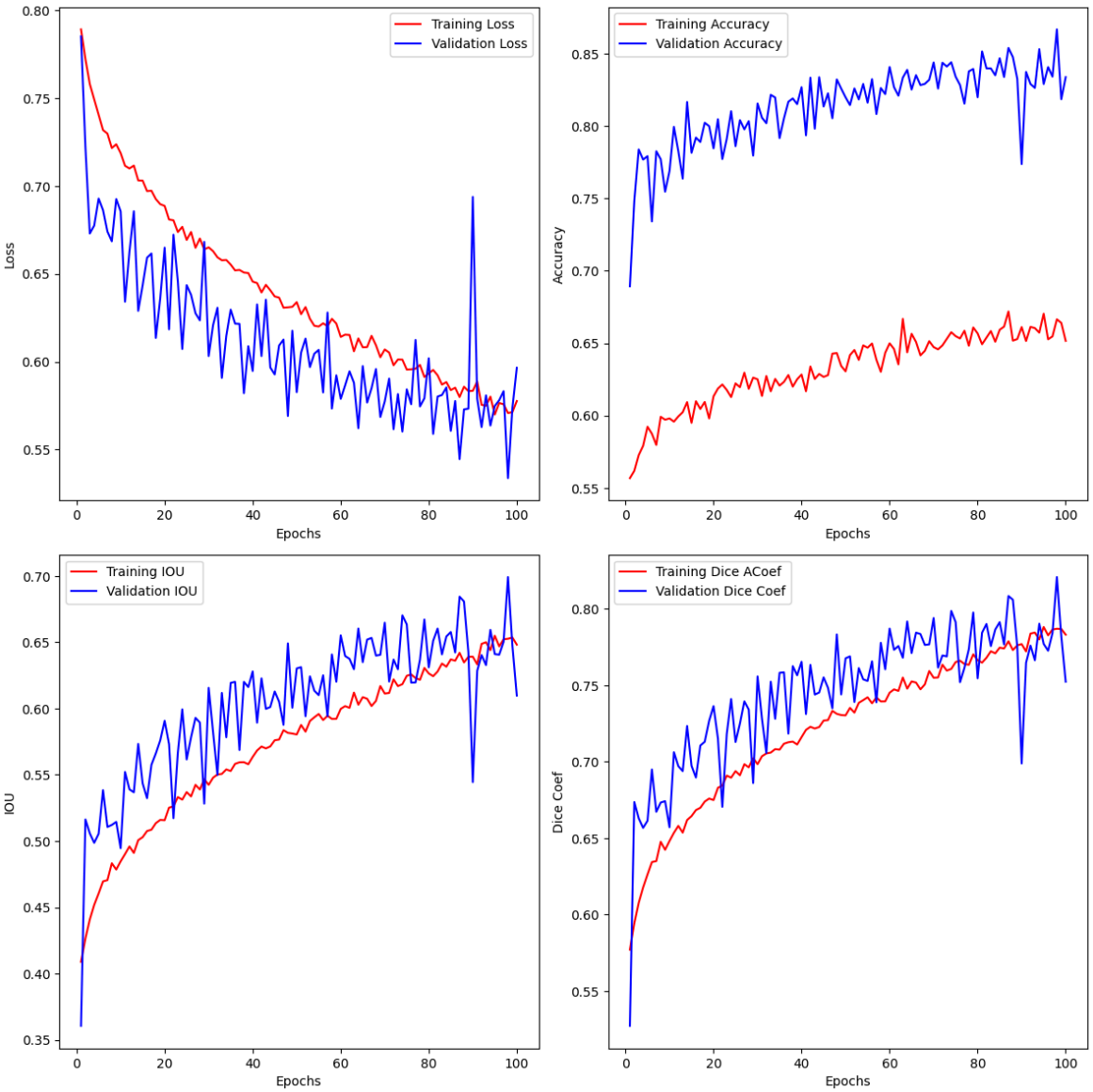
این ترکیب به ویژه در مسائل تقسیم‌بندی تصویر با تعداد زیادی کلاس و داده‌های نامتعادل مفید است، زیرا به بهینه‌سازی مستقیم معیار mIoU کمک می‌کند و در عین حال دقت کلی مدل را حفظ می‌کند.

3-4. آموزش شبکه

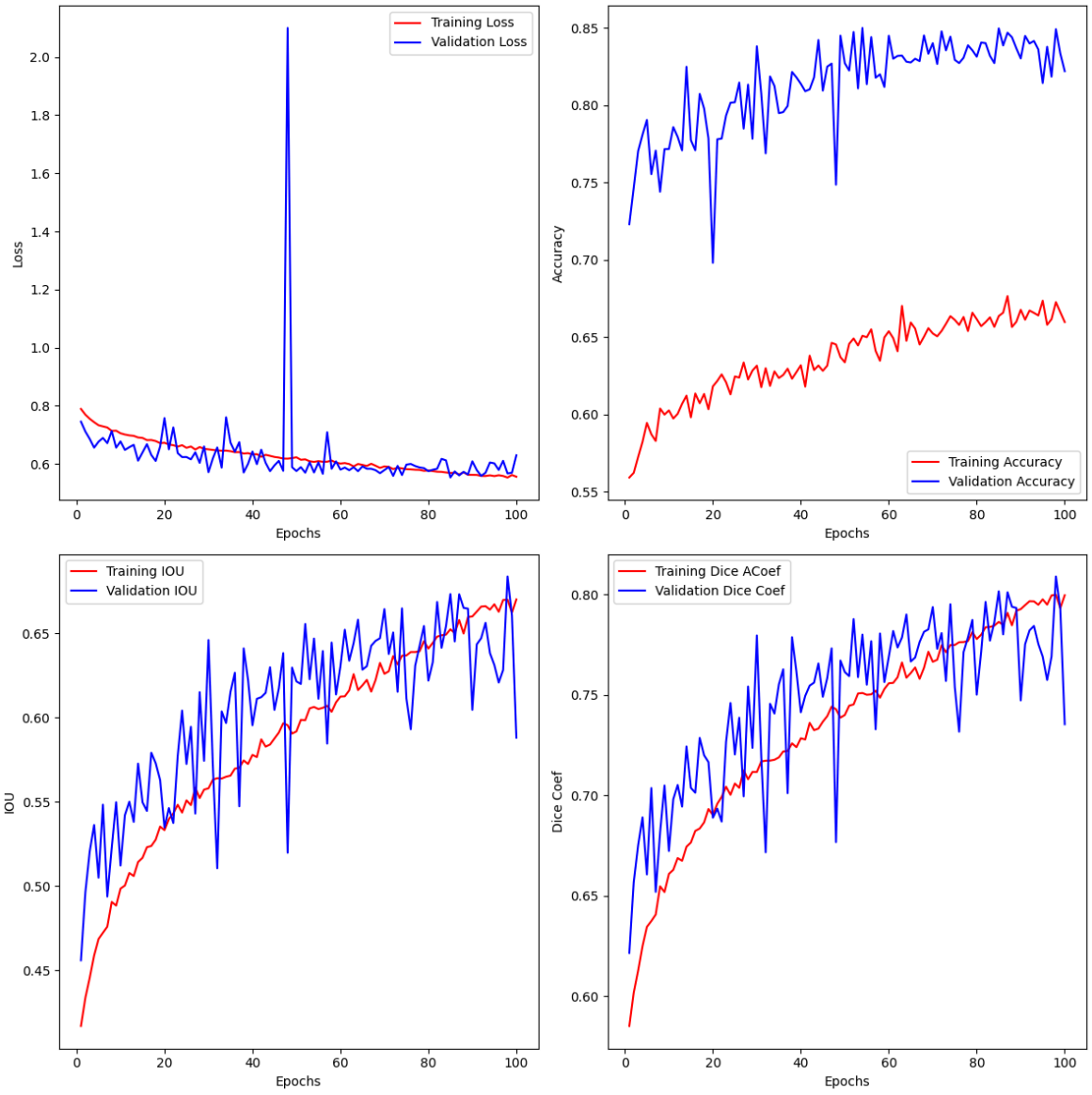
برای آموزش شبکه از پارامترهای زیر استفاده کردیم:

Epochs	100
Loss	Lovasz-Softmax + Cross-Entropy
Metrics	binary accuracy, IoU, meanIoU, Dice Coefficient
Optimizer	Adam
Batch Size	4

جدول 4-1. هایپرپارامترهای مورد استفاده



شکل 4-3. نتایج سنجه‌ها بر روی داده‌های آموزش و صحت‌سنجی در طول آموزش در شبکه Unet



شکل 4-4. نتایج سنجه‌ها بر روی داده‌های آموزش و صحت‌سنجی در طول آموزش در شبکه Ta-Unet

4-4. ارزیابی و تحلیل نتایج

سنجه[15] Mean Intersection over Union

این سنجه میانگین IoU برای همه کلاس های موجود است.

[15] Metric

$$\mathrm{mIoU} = \frac{1}{C} \sum_{i=1}^{C} \mathrm{IoU}_i$$

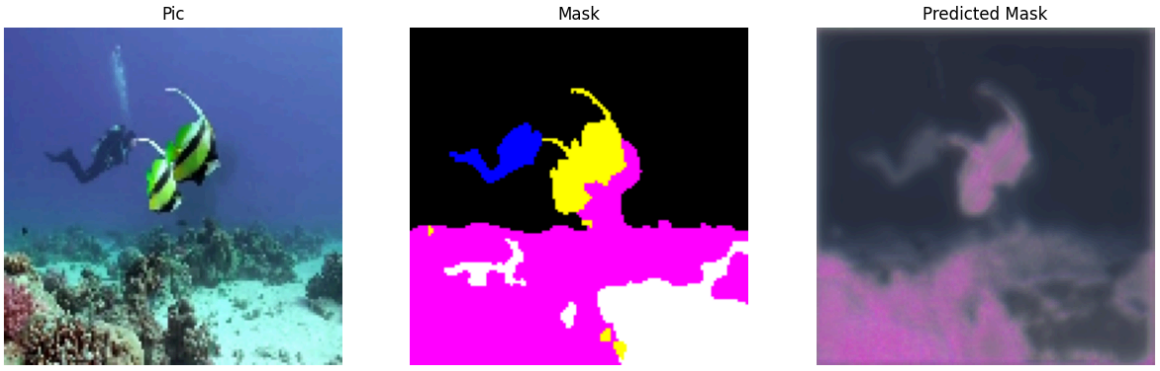
اما سنجه IoU یک سنجه برای فهمیدن میزان درستی تشخیص یک شی است. به این صورت که تعداد پیکسل های ناحیه مشترک پیش‌بینی و واقعیت را تقسیم بر مجموع نواحی این دو می‌کنیم.در حقیقت اگر یک شی واقعی داشته باشیم و شی پیش بینی شده، داریم اشتراک نواحی این رو بر اجتماعشان تقسیم می‌کنیم. هرچه این مقدار به یک نزدیک‌تر شود، پیش‌بینی ما به واقعیت نزدیک‌تر است.

$$\mathrm{IoU} = \frac{|A \cap B|}{|A \cup B|}$$

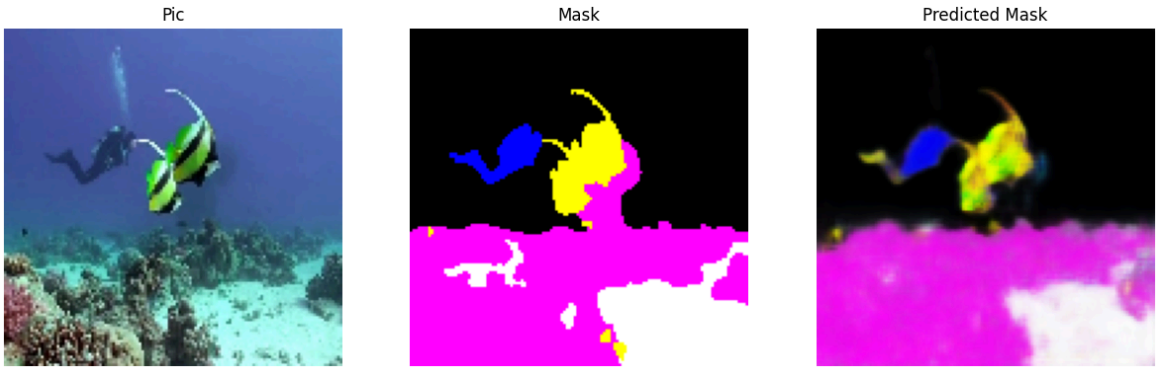
نتایج مدل‌ها

	Unet	**Ta-Unet**
mIoU	0.556	0.574
Dice Coefficient	0.715	0.729
Accuracy	0.811	0.823

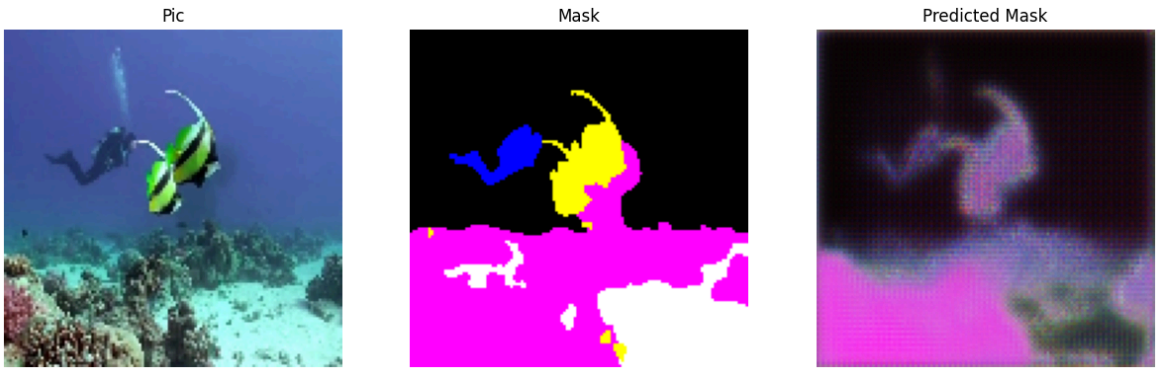
جدول 4-2. نتایج سنجه‌ها در داده‌های آزمون



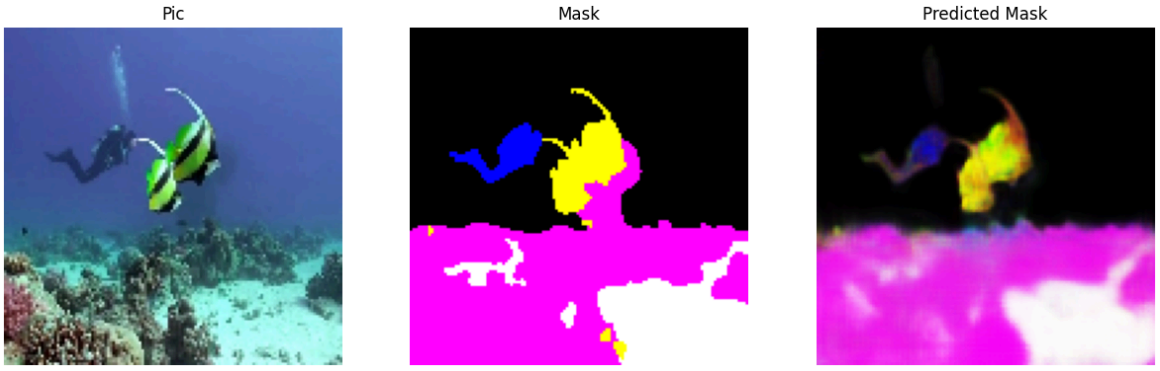
شکل 4-5. نمونه از عملکرد شبکه Unet در ایپاک اول



شکل 4-6. نمونه از عملکرد شبکه Unet در ایپاک 95-ام



شکل 4-7. نمونه از عملکرد شبکه Ta-Unet در ایپاک اول



شکل 4-8. نمونه از عملکرد شبکه Ta-Unet در ایپاک 96-ام

در انتها مشاهده می‌کنیم شبکه Ta-Unet با بهره‌گیری از مکانیزم توجه دقت بهتری را ارائه می‌کند و به mIoU بیشتری دست پیدا می‌کند.