



به نام خدا

دانشگاه تهران



دانشگاه مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

پرسش ۱ و ۲	نام و نام خانوادگی رايانame	علی خرم فر
پرسش ۳ و ۴	نام و نام خانوادگی رايانame	علی رمضانی
پرسش ۳ و ۴	نام و نام خانوادگی رايانame	Ali.ramezani.۹۶@ut.ac.ir
	تاریخ ارسال پاسخ	۱۴۰۳/۰۴/۱۰

- هر دو نویسنده در هر چهار پرسش همکاری داشته اند

• فهرست

فهرست تصاویر	ت
پرسش ۱ - تولید برچسب به کمک خوشه بندی	۱
۱. دادگان	۱
۲. شبکه مورد استفاده	۲
معماری شبکه	۲
۳. آموزش شبکه	۵
۴. آموزش شبکه MNIST	۵
۵. آموزش شبکه Fashion MNIST	۶
۶. ارزیابی مدل‌ها و مشاهده خروجی آن‌ها	۶
۷. خوشبندی	۸
۸. ارزیابی خوشبندی‌های مختلف دیتاست MNIST	۸
۹. ارزیابی خوشبندی‌های مختلف دیتاست Fashion MNIST	۹
۱۰. خوشبندی دیتاست MNIST	۱۰
۱۱. خوشبندی دیتاست Fashion MNIST	۱۱
پرسش ۲ - افزایش داده در مدل FaBert	۱۴
۱۲. NLP در Data Augmentation	۱۴
۱۳. تکنیک‌های رایج افزایش داده در NLP	۱۴
۱۴. پیش‌پردازش دادگان	۱۶
۱۵. نگاشت برچسب‌ها	۱۷
۱۶. توزیع آماری داده‌ها	۱۷
۱۷. توزیع تعداد توکن‌ها	۱۹
۱۸. پیش‌پردازش‌های حوزه متن	۲۰

۲۱	اعمال پیش‌پردازش‌های پیاده‌سازی شده
۲۱	۱-۳. افزایش دادگان به روش Back Translation
۲۴	۱-۴. تنظیم دقیق (FineTune) مدل FaBert
۲۵	کتابخانه‌های استفاده شده برای پیش‌پردازش
۲۵	استفاده از توکنایزر
۲۶	آموزش مدل با داده‌های اصلی
۲۶	آموزش مدل با داده‌های افزوده شده
۲۷	۱-۵. ارزیابی و تحلیل نتایج
۳۳	نتیجه‌گیری نهایی
۳۴	پرسش ۳: کلمه بیدار باش
۳۴	۱-۳: جمع آوری داده
۳۵	۲-۳: پیش‌پردازش و استخراج ویژگی
۵۲	۳-۳: طراحی شبکه عصبی
۵۹	پرسش ۴: شبکه بخش بندی تصاویر
۵۹	۱-۴. دادگان
۶۶	۲-۴: شبکه مورد استفاده
۶۸	۳-۴: آموزش شبکه ها
۸۸	۴-۴: ارزیابی و تحلیل نتایج

فهرست تصاویر

- ۱ شکل ۱ تصاویر نمونه از دیتاست Fashion MNIST و MNIST
- ۲ شکل ۲ نرمال‌سازی داده‌ها
- ۳ شکل ۳ معماری شبکه Autoencoder
- ۴ شکل ۴ معماری شبکه Encoder
- ۵ شکل ۵ نمودار مقدار MSE و MAE برای دیتاست MNIST
- ۶ شکل ۶ نمودار مقدار MSE و MAE برای دیتاست Fashion MNIST
- ۷ شکل ۷ تصاویر بازسازی شده از دیتاست MNIST
- ۸ شکل ۸ تصاویر بازسازی شده از دیتاست Fashion MNIST
- ۹ شکل ۹ نمودار میزان Silhouette score بر روی دیتاست MNIST
- ۱۰ شکل ۱۰ نمودار میزان Silhouette score بر روی دیتاست Fashion MNIST
- ۱۱ شکل ۱۱ تعداد برچسب‌های هر خوشة دیتاست MNIST
- ۱۲ شکل ۱۲ درصد برچسب‌های هر خوشة دیتاست MNIST
- ۱۳ شکل ۱۳ تعداد برچسب‌های هر خوشة دیتاست Fashion MNIST
- ۱۴ شکل ۱۴ درصد برچسب‌های هر خوشة دیتاست Fashion MNIST
- ۱۵ شکل ۱۵ نمونه‌هایی از داده‌های آموزشی دیتاست DeepSentiPers
- ۱۶ شکل ۱۶تابع نگاشت به برچسب‌های جدید
- ۱۷ شکل ۱۷ توزیع دسته‌های مختلف در دیتاست آموزشی
- ۱۸ شکل ۱۸ توزیع دسته‌های مختلف در دیتاست آزمون
- ۱۹ شکل ۱۹ توزیع آماری تعداد توکن‌ها در دیتاست آموزشی
- ۲۰ شکل ۲۰تابع BackTranslate
- ۲۱ شکل ۲۱ فریزکردن تمام مدل بجز لایه Classifier و لایه آخر مدل برتر
- ۲۲ شکل ۲۲ نتیجه آموزش مدل بر روی داده‌های اصلی
- ۲۳ شکل ۲۳ نتیجه آموزش مدل بر روی داده‌های افزوده شده
- ۲۴ شکل ۲۴ دقت و مقدار تابع هزینه روی دادگان ارزیابی
- ۲۵ شکل ۲۵ نمودار ایپاک-دقت مدل‌های آموزش‌دیده روی دیتاست ابتدایی و دیتاست افزوده شده
- ۲۶ شکل ۲۶ نمودار ایپاک-هزینه مدل‌های آموزش‌دیده روی دیتاست ابتدایی و دیتاست افزوده شده
- ۲۷ شکل ۲۷ گزارش طبقه‌بندی مدل آموزش‌دیده بر روی دیتاست ابتدایی بر روی دیتاست آزمون

شکل ۲۸	- گزارش طبقه‌بندی مدل آموزش‌دیده بر روی دیتاست افزوده شده بر روی دیتاست آزمون.	۳۰
شکل ۲۹	- ماتریس آشفتگی مدل آموزش‌دیده بر روی دیتاست ابتدایی بر روی دیتاست آزمون.....	۳۰
شکل ۳۰	- ماتریس آشفتگی مدل آموزش‌دیده بر روی دیتاست افزوده شده بر روی دیتاست آزمون....	۳۱
شکل ۳۱	- تابع ضبط wake_word	۳۴
شکل ۳۲	- تابع record_background_noise	۳۴
شکل ۳۳	- ضبط وویس با دو تابع مورد نظر.....	۳۵
شکل ۳۴	- آدرس فایل های wake_up	۳۵
شکل ۳۵	- نمودار و فایل صوتی به صورت رندوم background	۳۶
شکل ۳۶	- نمونه برداری برای ساخت دیتافریم background	۳۶
شکل ۳۷	- دیتافریم داده های background	۳۷
شکل ۳۸	- بررسی وویس های داده های background	۳۷
شکل ۳۹	- ترکیب دو دیتاست	۳۸
شکل ۴۰	- ایجاد مجموعه آموزش و تست	۳۸
شکل ۴۱	- پیش پردازش فایل های صوتی	۴۰
شکل ۴۲	- استخراج Mel Spectrogram	۴۵
شکل ۴۳	- نمایش یک Spectrogram	۴۵
شکل ۴۴	- بردار ویژگی لازم برای مدل CNN بدون داده افزایی	۴۶
شکل ۴۵	- دیتاست مدل CNN	۴۶
شکل ۴۶	- بردار آموزش مدل CNN با داده افزایی	۴۶
شکل ۴۷	- دریافت MFCC	۴۷
شکل ۴۸	- ایجاد بردارهای ویژگی و برچسب و نرمال سازی آن ها	۴۷
شکل ۴۹	- دیتاست مدل MLP	۴۷
شکل ۵۰	- پیاده سازی time_stretch	۴۸
شکل ۵۱	- پیاده سازی pitch_shift	۴۹
شکل ۵۲	- پیاده سازی افروden نویز	۴۹
شکل ۵۳	- پیاده سازی shift Audio	۵۰
شکل ۵۴	- پیاده سازی تغییر صدا	۵۱
شکل ۵۵	- نحوه اعمال داده افزایی به هر فایل صوتی	۵۱
شکل ۵۶	- شبکه CNN مورد استفاده	۵۳

..... ۵۴ شکل ۵۷ - هایپرپارامتر های شبکه CNN
..... ۵۴ شکل ۵۸ - نمودار های مدل CNN بدون داده افزایی
..... ۵۴ شکل ۵۹ - گزارش classification مدل CNN بدون داده افزایی
..... ۵۵ شکل ۶۰ - ماتریس درهم ریختگی مدل CNN بدون داده افزایی
..... ۵۵ شکل ۶۱ - نمودار های مدل CNN با داده افزایی
..... ۵۶ شکل ۶۲ - گزارش classification مدل CNN با داده افزایی
..... ۵۶ شکل ۶۳ - ماتریس درهم ریختگی مدل CNN با داده افزایی
..... ۵۷ شکل ۶۴ - شبکه MLP مورد استفاده
..... ۵۷ شکل ۶۵ - هایپرپارامتر های شبکه MLP
..... ۵۷ شکل ۶۶ - نمودار های مدل MLP با داده افزایی
..... ۵۸ شکل ۶۷ - گزارش classification مدل MLP با داده افزایی
..... ۵۸ شکل ۶۸ - ماتریس درهم ریختگی مدل MLP با داده افزایی
..... ۵۹ شکل ۶۹ - تابع ساخت دیتا فریم آدرس تصاویر و ماسک های نظیر آن ها
..... ۵۹ شکل ۷۰ - تابع جداسازی ۱۰ درصد دیتا برای ولیدیشن
..... ۶۰ شکل ۷۱ - دیتاست آموزش شامل ۱۳۷۳ تصویر
..... ۶۰ شکل ۷۲ - دیتاست ارزیابی شامل ۱۵۲ تصویر
..... ۶۰ شکل ۷۳ - دیتاست تست شامل ۱۱۰ تصویر
..... ۶۱ شکل ۷۴ - تعدادی تصویر و ماسک مرتبط به آن
..... ۶۱ شکل ۷۵ - ماسک های مجزا به ازای هر کلاس در داده تست
..... ۶۲ شکل ۷۶ - بخش اول داده افزایی طبق مقاله Suim
..... ۶۲ شکل ۷۷ - مقاله اصلی Suim
..... ۶۳ شکل ۷۸ - ایمیل TA محترم در باب صحیح بودن هر دو نوع خروجی بر اساس مدل
..... ۶۳ شکل ۷۹ - نوع اول Augmentation و نرمال سازی ماسک و تصاویر
..... ۶۴ شکل ۸۰ - دیتاست های ساخته شده
..... ۶۴ شکل ۸۱ - تصاویر آموزش و ماسک های باینری آن ها
..... ۶۶ شکل ۸۲ - دیتاست نهایی استفاده شده بر اساس هایپرپارامتر های آن
..... ۶۶ شکل ۸۳ - تصاویر داده افزایی دارای ماسک رنگی
..... ۶۷ شکل ۸۴ - شبکه UNet
..... ۶۸ شکل ۸۵ - مکانیزم Triplet Attention

۶۸ شکل ۸۶ - شبکه TA-Unet
۶۹ شکل ۸۷ - ساختار توابع شبکه UNet
۷۰ شکل ۸۸ - شبکه Unet برای ماسک های سیاه و سفید
۷۰ شکل ۸۹ - شبکه Unet برای ماسک های رنگی
۷۱ شکل ۹۰ - لایه Attention
۷۲ شکل ۹۱ - اعمال Attention و برگشت به حالت اولیه
۷۲ شکل ۹۲ - خروجی لایه Attention
۷۳ شکل ۹۳ - لایه Attention
۷۳ شکل ۹۴ - لایه های ZPool و BaseConv
۷۴ شکل ۹۵ - شبکه TA-Unet برای ماسک سیاه و سفید
۷۴ شکل ۹۶ - شبکه TA-UNet برای ماسک های رنگی
۷۵ شکل ۹۷ - تعریف معیار های cross entropy و lovsaz loss و ترکیب آن با iou
۷۶ شکل ۹۸ -تابع آموزش شبکه UNet
۷۷ شکل ۹۹ - نتایج شبکه UNet با ماسک باینری
۷۷ شکل ۱۰۰ - نتایج Unet روی داده های تست
۷۸ شکل ۷۱ - تصویر ماسک های پیش بینی شده توسط مدل
۷۹ شکل ۱۰۲ - تابع آموزش شبکه TA-UNet
۸۰ شکل ۱۰۳ - نتایج شبکه TA-UNet با ماسک باینری
۸۰ شکل ۱۰۴ - نتایج TA-Unet روی داده های تست
۸۱ شکل ۷۵ - تصویر ماسک های پیش بینی شده توسط مدل
۸۲ شکل ۱۰۶ - تابع آموزش شبکه UNet
۸۳ شکل ۱۰۷ - نتایج شبکه UNet با ماسک باینری
۸۳ شکل ۱۰۸ - نتایج Unet روی داده های تست
۸۴ شکل ۷۹ - تصویر ماسک های پیش بینی شده توسط مدل
۸۵ شکل ۱۱۰ - تابع آموزش شبکه TA-UNet
۸۶ شکل ۱۱۱ - نتایج شبکه TA UNet با ماسک باینری
۸۶ شکل ۱۱۲ - نتایج TA Unet روی داده های تست
۸۷ شکل ۸۳ - تصویر ماسک های پیش بینی شده توسط مدل
۸۸ شکل ۱۱۴ - معیار IOU

۸۸	شکل ۱۱۵ - معیار IOU در segmetation
۸۹	شکل ۱۱۶ - فرمول IOU در هر کلاس c
۸۹	شکل ۱۱۷ - فرمول mean iou در هر همه کلاس ها
۸۹	شکل ۸۸ - شبکه TA-Unet
۸۹	شکل ۸۹ - شبکه UNet
۹۰	شکل ۱۲۰ - مقایسه دو شبکه روی دیتاست تست
۹۱	شکل ۹۱ - شبکه UNet
۹۱	شکل ۹۲ - شبکه TA-UNet
۹۲	شکل ۱۲۳ - مقایسه دو شبکه روی دیتاست تست

فهرست جداول:

جدول ۱ نمونه اصلی و BackTranslate جملات دیتاست آموزشی.....	۲۳
جدول ۲ نتایج ارزیابی مدل‌های آموزش داده شده	۲۸
جدول ۳ جدول مقایسه F1-Score مدل‌های آموزش دیده با داده‌های اصلی و افزوده شده	۳۰
جدول ۴ جدول نمونه‌های اشتباه دسته‌بندی شده توسط مدل‌ها	۳۲
جدول ۵ - مقایسه سه مدل	۵۸
جدول ۶ - هایپرپارامتر های UNet با ماسک باینری	۷۶
جدول ۷ - نتایج مدل	۷۸
جدول ۸ - هایپرپارامتر های TA- UNet با ماسک باینری	۷۹
جدول ۹ - نتایج مدل	۸۱
جدول ۱۰ - هایپرپارامتر های UNet با ماسک باینری	۸۲
جدول ۱۱ - نتایج مدل	۸۴
جدول ۱۲ - هایپرپارامتر های TA- UNet با ماسک باینری	۸۵
جدول ۱۳ - نتایج مدل	۸۷

پرسش ۱ - تولید برچسب به کمک خوشه بندی

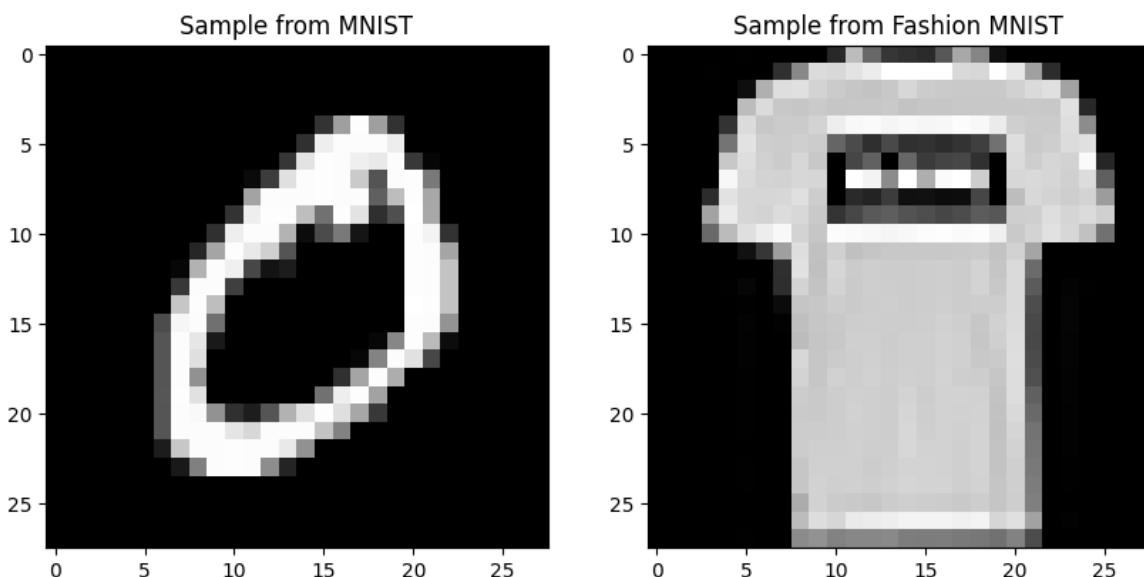
برای حل این سوال از محیط Kaggle استفاده شد. در ادامه به بررسی تمام مراحل انجام شده خواهیم پرداخت.

۱-۱. دادگان

مجموعه داده‌ی MNIST شامل تصاویر دستنویس اعداد از ۰ تا ۹ بوده و تصاویر مختلف از پوشاک است. این دیتاست‌ها از کتابخانه TensorFlow دانلود شد و آن‌ها خروجی گرفته شد که در پایین مشاهده می‌کنیم. پس از بارگیری این دیتاست‌ها، مشخصات کامل آن‌ها مورد بررسی قرار گرفت که هر دو دیتاست دارای ۶۰۰۰۰ تصویر برای آموزش و ۱۰۰۰۰ تصویر برای تست هستند.

```
MNIST training data shape: (60000, 28, 28)  
MNIST test data shape: (10000, 28, 28)  
Fashion MNIST training data shape: (60000, 28, 28)  
Fashion MNIST test data shape: (10000, 28, 28)
```

هر دو مجموعه داده بارگذاری شده دارای ابعاد ۲۸ در ۲۸ بوده و به طور کلی Shape یکسانی دارند.



شکل ۱ تصاویر نمونه از دیتاست **Fashion MNIST** و **MNIST**

تصاویر نمونه از دو دیتاست در شکل بالا که در دیتاست MNIST، تصویری از عدد ۴ و در دیتاست Fashion MNIST، تصویری از یک تی‌شرت وجود دارد.

پس از بارگیری و بررسی اولیه دیتاستها قدم بعدی نرمال‌سازی داده‌ها و تخصیص بخشی از داده‌های آموزشی به عنوان مجموعه‌ی validation است. نرمال‌سازی داده‌ها به این معناست که مقادیر پیکسل‌ها را به محدوده‌ی ۰ و ۱ تبدیل کنیم که این کار با تقسیم هر پیکسل بر ۲۵۵ انجام می‌شود.

Normalize the datasets

+ Code + Markdown

```
mnist_train_images = mnist_train_images / 255.0
mnist_test_images = mnist_test_images / 255.0
fashion_train_images = fashion_train_images / 255.0
fashion_test_images = fashion_test_images / 255.0
```

شکل ۲ نرمال‌سازی داده‌ها

برای هر دیتاست، ۲۵ درصد اول داده‌های آموزشی به مجموعه‌ی اعتبارسنجی اختصاص یافت و بقیه داده‌ها به عنوان مجموعه‌ی آموزشی باقی مانندند.

```
MNIST training set shape after split: (45000, 28, 28)
MNIST validation set shape: (15000, 28, 28)
Fashion MNIST training set shape after split: (45000, 28, 28)
Fashion MNIST validation set shape: (15000, 28, 28)
```

۲-۱. شبکه مورد استفاده

در این بخش، هدف آموزش یک شبکه عصبی Convolutional Autoencoder است که ورودی آن تصویر و خروجی نیز همان تصویر باشد. در این مدل، بخش Encoder شبکه وظیفه تبدیل تصویر ورودی به یک بردار ویژگی با ابعاد کمتر را بر عهده دارد که می‌تواند برای کاربردهای بخش بعدی تمرین مثل خوشبندی مورد استفاده قرار گیرد.

معماری شبکه

معماری شبکه Auto Encoder به صورت زیر است که با کمک متدهای summary() خروجی گرفته شد. در ادامه به بررسی لایه‌های مختلف آن می‌پردازیم:

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73,856
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 6)	37,638
dense_1 (Dense)	(None, 6272)	43,904
reshape (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 64)	73,792
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 32)	18,464
conv2d_3 (Conv2D)	(None, 28, 28, 1)	289

Total params: 266,759 (1.02 MB)
Trainable params: 266,759 (1.02 MB)
Non-trainable params: 0 (0.00 B)

شکل ۳ معماری شبکه Autoencoder

: لایه کانولوشنال با ۳۲ فیلتر و اندازه کرنل 3×3 که با تابع فعال‌سازی ReLU و Conv^۲D از

نوع Same

Same از نوع Padding با اندازه 2×2 و MaxPooling از نوع MaxPooling^۲D

: لایه کانولوشنال با ۶۴ فیلتر و اندازه کرنل 3×3 که با تابع فعال‌سازی ReLU و Conv^۲D از

نوع Same

Same از نوع Padding با اندازه 2×2 و MaxPooling از نوع MaxPooling^۲D

: لایه کانولوشنال با اندازه کرنل 3×3 که با تابع فعال‌سازی ReLU و Conv^۲D از نوع

لایه‌ای برای تبدیل خروجی سه‌بعدی به یک بردار یک‌بعدی.

: Dense Latent space را تولید می‌کند. Dense با ۶ نورون و تابع فعال‌سازی ReLU که بردار ویژگی

بخش دکدر نیز شامل لایه‌های زیر است:

:لایه Dense با ۶۲۷۲ نورون و تابع فعال‌سازی ReLU که خروجی آن دوباره شکل دهی می‌شود.

:لایه‌ای برای بازسازی خروجی به شکل سه‌بعدی ۱۲۸x۷x۷ Reshape

:لایه کانولوشنال Transpose با Conv^۲DTranspose لایه کانولوشنال ۶۴ فیلتر و اندازه کرنل ۳x۳ که با strides برابر ۲ و Same از نوع Padding

:لایه کانولوشنال Transpose با Conv^۲DTranspose لایه کانولوشنال ۳۲ فیلتر و اندازه کرنل ۳x۳ که با strides برابر ۲ و Same از نوع Padding

:لایه کانولوشنال با ۱ فیلتر و اندازه کرنل ۳x۳ که با تابع فعال‌سازی Sigmoid و و از Conv^۲D تعريف شد تا تصویر بازسازی شده نهایی تولید شود. Same از نوع Padding

معماری بخش Encoder به تنها‌یی به صورت زیر است:

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73,856
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 6)	37,638

Total params: 130,310 (509.02 KB)
Trainable params: 130,310 (509.02 KB)
Non-trainable params: 0 (0.00 B)

شکل ۴ معماری شبکه Encoder

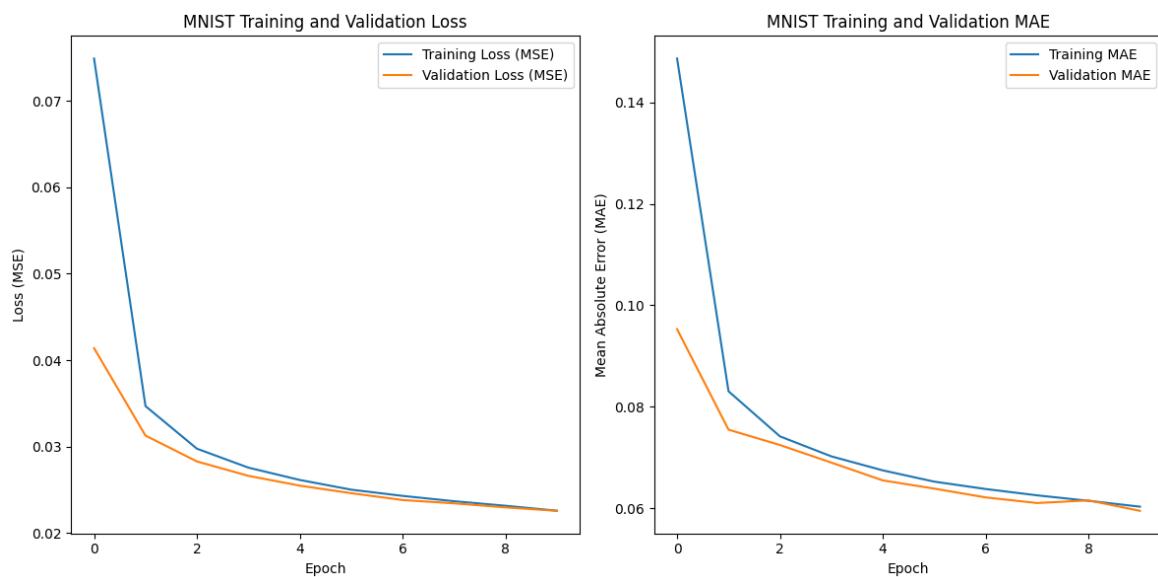
لایه‌ی latent نیز با ابعاد ۶ انتخاب شد تا تعادلی بین فشرده‌سازی اطلاعات و حفظ ویژگی‌های مهم برقرار شود.

۳-۱. آموزش شبکه

مدل Autoencoder پیاده‌سازی شده با استفاده از بهینه‌ساز Adam و نرخ یادگیری ۰.۰۰۱ کامپایل شده و از تابع هزینه MAE و معیار MSE برای ارزیابی استفاده می‌کنیم.

آموزش شبکه MNIST

مدل‌ها با استفاده از مجموعه‌های آموزشی و اعتبارسنجی به مدت ۱۰ ایپاک با اندازه بج ۲۵۶ آموزش داده شدند. در هر ایپاک، خطاهای آموزشی و اعتبارسنجی محاسبه و ثبت شد تا بتوانیم روند یادگیری مدل‌ها را تحلیل کنیم. نتایج آموزش مدل بر روی دیتابست MNIST به صورت نمودارهای زیر نمایش داده شده است:

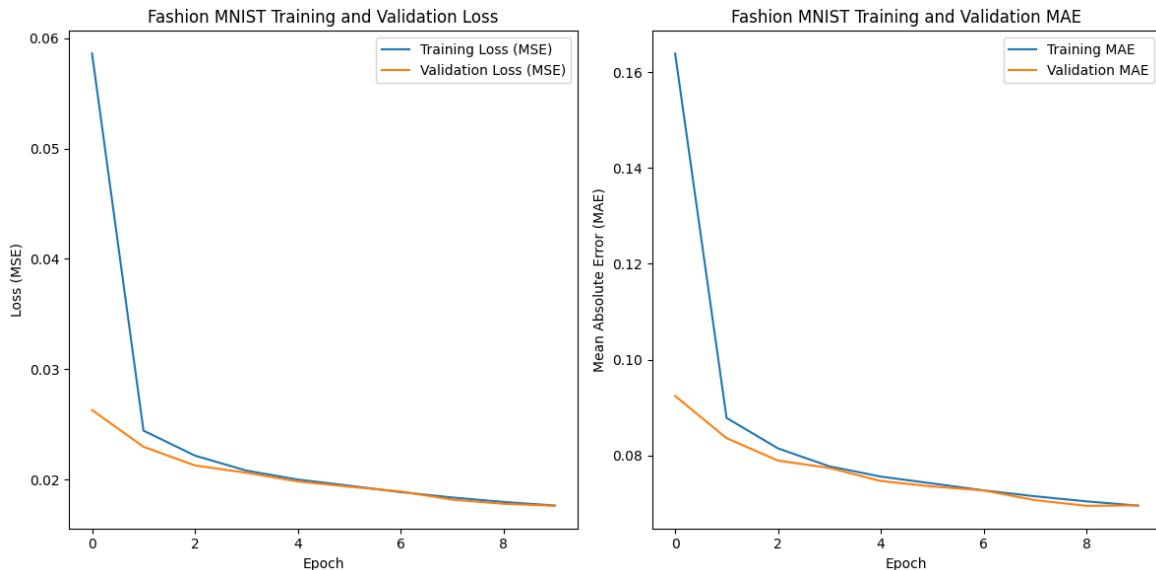


شکل ۵ نمودار مقدار MSE و MAE برای دیتابست MNIST

نمودار سمت چپ، روند تغییرات خطای MSE یا میانگین مربعات خطا را برای داده‌های آموزشی و اعتبارسنجی در طی ۱۰ ایپاک نمایش می‌دهد. مشاهده می‌شود که با گذشت زمان خطای مدل کاهش یافته و به مقدار ثابتی نزدیک می‌شود. نمودار سمت راست نیز روند تغییرات خطای MAE یا میانگین خطای مطلق را برای داده‌های آموزشی و اعتبارسنجی نمایش می‌دهد. این نمودار نیز نشان می‌دهد که مدل به مرور زمان دقیق‌تری در بازسازی تصاویر پیدا کرده است.

آموزش شبکه Fashion MNIST

تمام مراحل گفته شده برای دیتاست MNIST برای Fashion MNIST نیز انجام شد که نتایج به صورت زیر است:



شکل ۶ نمودار مقدار MAE و MSE برای دیتاست Fashion MNIST

مدل Fashion MNIST در نهایت خطاهای کمتری نسبت به مدل MNIST دارد. این تفاوت می‌تواند به دلیل تفاوت‌های ذاتی در پیچیدگی داده‌های دو مجموعه باشد.

هر دو مدل نشان‌دهنده عملکرد قابل قبولی در بازسازی تصاویر بودند، اما مدل Fashion MNIST بهبود بیشتری را نشان داد که ممکن است به دلیل ساختار داده‌های آن باشد که برای شبکه کانولوشنال بهینه‌تر هستند.

۱-۴. ارزیابی مدل‌ها و مشاهده خروجی آن‌ها

پس از آموزش مدل Convolutional Autoencoder بر روی هر دو دیتاست، مدل بر روی داده‌های تست ارزیابی شد تا عملکرد آن بر روی داده‌های دیده‌نشده بررسی شود. نتایج ارزیابی مدل به صورت زیر است:

MNIST Test MSE: 0.0225, Test MAE: 0.0595
Fashion MNIST Test MSE: 0.0177, Test MAE: 0.0701

: MNIST مدل

خطای MSE (میانگین مربع خطأ): ۰.۰۲۲۵

خطای MAE (میانگین خطای مطلق): ۰.۰۵۹۵

: Fashion MNIST مدل

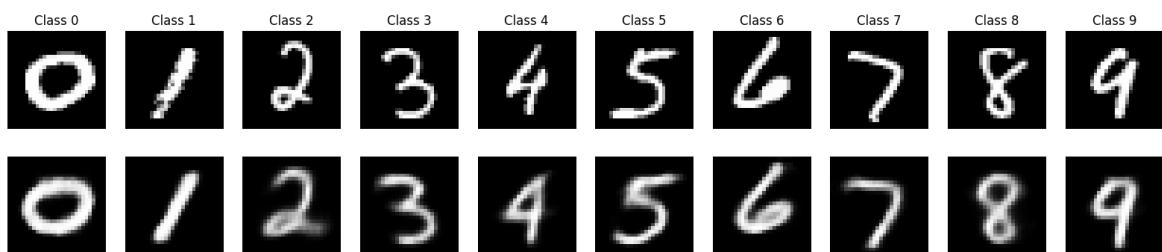
خطای MSE (میانگین مربع خطأ): ۰.۰۱۷۷

خطای MAE (میانگین خطای مطلق): ۰.۰۷۰۱

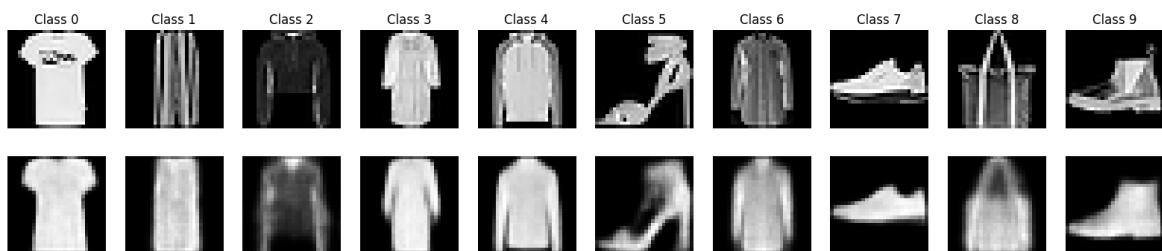
مدل Fashion MNIST با خطای MSE برابر ۰.۰۱۷۷ عملکرد بهتری نسبت به مدل MNIST با خطای ۰.۰۲۲۵ داشته است. این نشان می‌دهد که مدل Fashion MNIST توانسته است تفاوت‌های کوچکتری بین تصاویر اصلی و بازسازی شده ایجاد کند. به عبارت دیگر خطای متوسط پیکسل به پیکسل در مدل کمتر از مدل MNIST بوده است، که نشان‌دهنده دقیق‌تر در بازسازی جزئیات تصویر است.

از طرف دیگر مدل MAE در معیار MNIST عملکرد بهتری داشته است که نشان می‌دهد این مدل در بازسازی کل تصویر، در مجموع دقیق‌تری داشته است. دلیل این تفاوت‌ها این است که تصاویر اعداد دستنویس ساختار بسیار متفاوت و ساده‌تری نسبت به تصاویر Fashion MNIST دارند.

برای ارزیابی عملکرد مدل‌ها تصاویری از هر کلاس در دیتاست‌های MNIST و Fashion MNIST بازسازی شدند. در هر تصویر، ردیف بالا شامل تصاویر اصلی و ردیف پایین شامل تصاویر بازسازی شده توسط مدل‌ها است.



شکل ۷ تصاویر بازسازی شده از دیتاست MNIST



شکل ۸ تصاویر بازسازی شده از دیتاست Fashion MNIST

این تصایر نشان می‌دهند که مدل‌ها به خوبی توانسته‌اند ویژگی‌های اصلی تصاویر را استخراج و بازسازی کنند.

۱-۵. خوشبندی

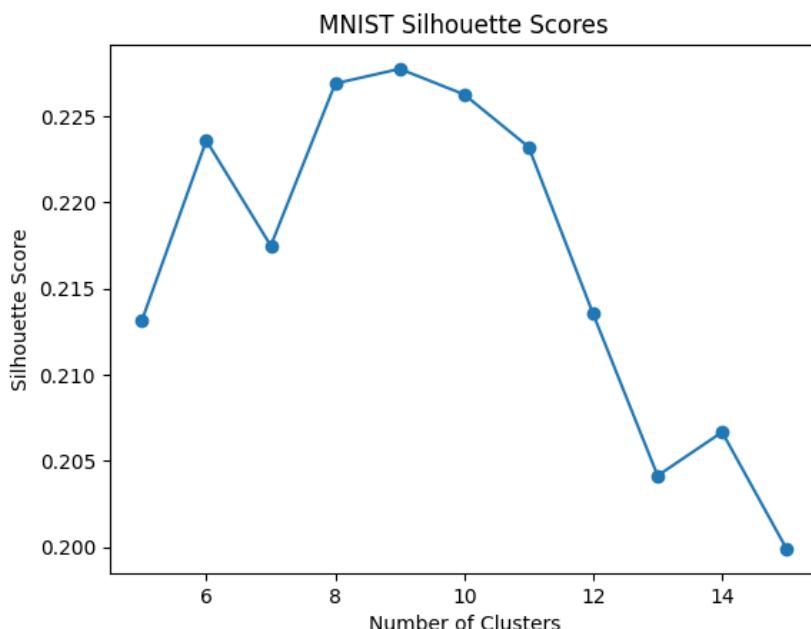
ابتدا با استفاده از بخش انکدر مدل Autoencoder، ویژگی‌های کلیه داده‌های آموزشی، اعتبارسنجی و تست برای هر دو مجموعه داده MNIST و Fashion MNIST استخراج شدند. سپس، این ویژگی‌ها با هم ترکیب شده و بردار نهایی ویژگی‌ها برای هر مجموعه داده ایجاد شد. تمام برچسب‌های مربوط به داده‌های آموزشی، اعتبارسنجی و تست نیز با هم ترکیب شده و بردار نهایی برچسب‌ها آماده شد.

سپس در یک حلقه با تعداد خوشبندی‌های مختلف الگوریتم Kmeans را اجرا کردیم و هر بار مقدار silhouette را محاسبه کردیم. در ادامه به بررسی نتایج هر کدام از دیتاست‌ها می‌پردازیم. برای اینکه نتیجه دسته‌بندی قابل قبول باشد حدود ۱۰ بار مدل‌ها با پارامترهای مختلف و حتی لایه‌های مختلف آموزش دیده شد و نتیجه نهایی که در گزارش آمده بهترین خروجی است.

ارزیابی خوشبندی‌های مختلف دیتاست MNIST

الگوریتم KMeans بر روی ویژگی‌های استخراج شده با استفاده از بخش انکدر اجرا شد. تعداد خوشبندی‌ها بین ۵ تا ۱۵ متغیر بوده و برای هر تکرار حلقه، معیار silhouette score محاسبه شد. این معیار به ارزیابی کیفیت خوشبندی ما کمک کرده به طوری که مقادیر بالاتر نشان‌دهنده خوشبندی بهتر است.

نمودار میزان Silhouette score برای تعداد خوشبندی‌های متفاوت دیتاست MNIST به صورت زیر است:



شکل ۹ نمودار میزان Silhouette score بر روی دیتاست MNIST

یک معیار برای ارزیابی کیفیت خوشبندی است که مقدار آن بین -1 و 1 قرار دارد. مقادیر نزدیک به 1 نشان‌دهنده خوشبندی مناسب و مقادیر نزدیک به -1 نشان‌دهنده خوشبندی ضعیف است.

محور افقی: تعداد خوشبندی (Clusters)

محور عمودی: Silhouette Score

با توجه به نمودار و مقادیر Silhouette Score، تعداد خوشبندی 9 به عنوان بهترین تعداد خوشبندی داده‌های MNIST انتخاب می‌شود. داده‌های MNIST شامل تصاویر دست‌نویس اعداد از 0 تا 9 هستند. هر عدد ویژگی‌های خاص خود را دارد که باعث می‌شود داده‌ها به طور طبیعی در 10 دسته مختلف قرار بگیرند. تعداد خوشبندی 9 نشان‌دهنده نزدیک‌ترین تعداد به تعداد واقعی دسته‌ها است که باعث می‌شود خوشبندی بهینه باشد. دلیل اینکه چرا 10 دسته انتخاب نشده را در قسمت بعدی بررسی خواهیم کرد. همچنین مشاهده می‌شود که با افزایش تعداد خوشبندی از 10 به بالا، مقدار Silhouette Score کاهش می‌یابد. این نشان می‌دهد که افزایش بیش از حد تعداد خوشبندی منجر به پراکندگی و کاهش کیفیت خوشبندی می‌شود.

ارزیابی خوشبندی‌های مختلف دیتاست Fashion MNIST

تمام مراحل قسمت قبل را حال بر روی دیتاست Fashion MNIST تکرار می‌کنیم. نمودار نمودار میزان برای تعداد خوشبندی‌های متفاوت در این دیتاست به صورت زیر است:



شکل ۱۰ نمودار میزان Silhouette score بر روی دیتاست Fashion MNIST

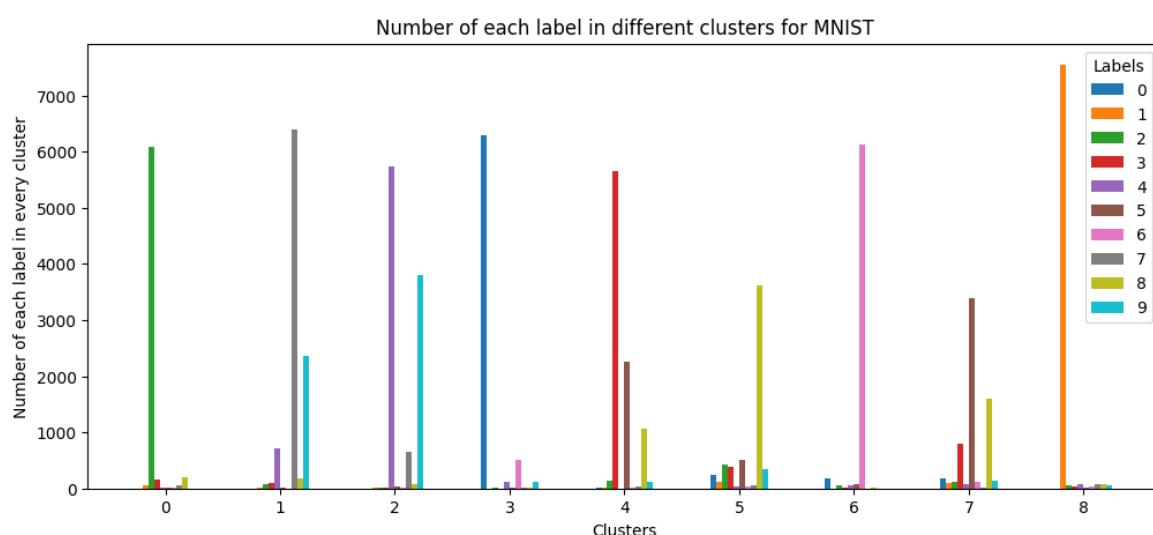
این نمودار نشان می‌دهد که با افزایش تعداد خوشها از ۵ تا ۷، مقدار Silhouette Score به تدریج افزایش می‌یابد که نشان‌دهنده بهبود کیفیت خوشبندی است. بالاترین مقدار Silhouette Score در تعداد خوش ۷ به دست آمده است. پس از تعداد خوش ۷، مقدار Silhouette Score کاهش می‌یابد که نشان‌دهنده کاهش کیفیت خوشبندی با افزایش تعداد خوشهاست.

دلیل این مورد را باتوجه به خروجی‌های قسمت بعد به طور دقیق‌تر بررسی می‌کنیم ولی به طور کلی دلیل آن است که چند دسته از این دیتاست ساختارهای مشابهی دارند و ویژگی‌های نزدیک به هم توسط انکدر تولید می‌شود. همین مورد برای دیتاست MNIST نیز برقرار است و مثلاً عدد ۹ و ۴ انگلیسی در حالت نوشتاری خیلی شبیه به هم هستند.

خوشبندی دیتاست MNIST

باتوجه به قسمت قبل خوشبندی بر روی ۹ دسته انجام شد. برای تحلیل بهتر علاوه بر نمودار گفته شده در صورت تمرين یک نمودار دیگر نیز برای مشاهده بهتر درصد هر خوش نیز خروجی گرفته شد. حال به بررسی آن‌ها می‌پردازیم.

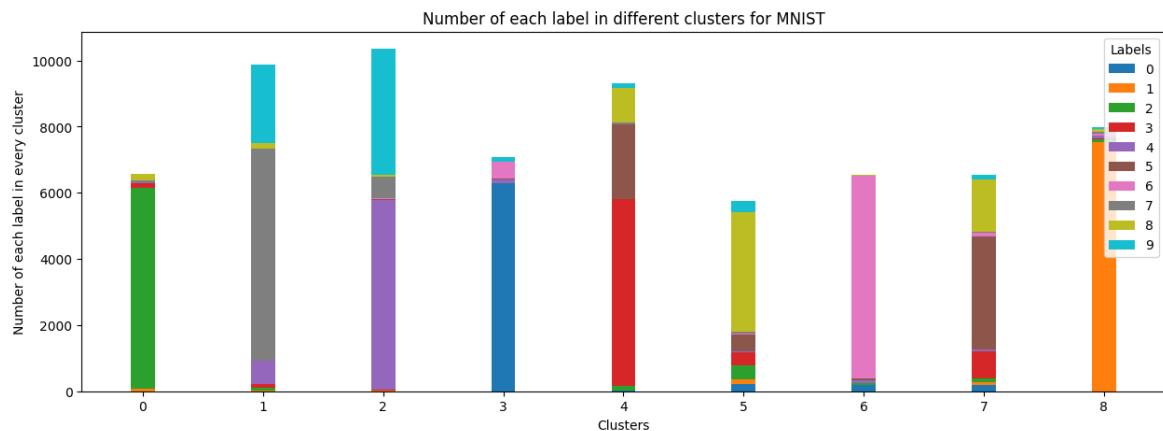
نمودار اول: تعداد برچسب‌های هر خوش



شکل ۱۱ تعداد برچسب‌های هر خوش دیتاست MNIST

این نمودار تعداد برچسب‌های مختلف در هر یک از خوشها را نشان می‌دهد. در محور افقی تعداد خوشها از ۰ تا ۹ یعنی ۹ خوش و در محور عمودی تعداد برچسب‌های موجود در هر خوش نشان داده شده است. هر رنگ در نمودار نمایانگر یکی از برچسب‌های ۰ تا ۹ است.

نمودار دوم: درصد برچسب‌های هر خوشه



شکل ۱۲ درصد برچسب‌های هر خوشه دیتاست MNIST

این نمودار درصد هر برچسب در هر خوشه را نشان می‌دهد. محور افقی نشان‌دهنده شماره خوشه‌ها و محور عمودی تعداد تجمعی شده برچسب‌های موجود در هر خوشه را نشان می‌دهد.

هر خوشه دارای یک برچسب غالب است. این نشان می‌دهد که ویژگی‌های استخراج شده توسط انکدر توانسته‌اند به خوبی داده‌ها را تفکیک کنند و الگوریتم KMeans نیز توانسته است این تفکیک‌پذیری را به درستی پیاده‌سازی کند.

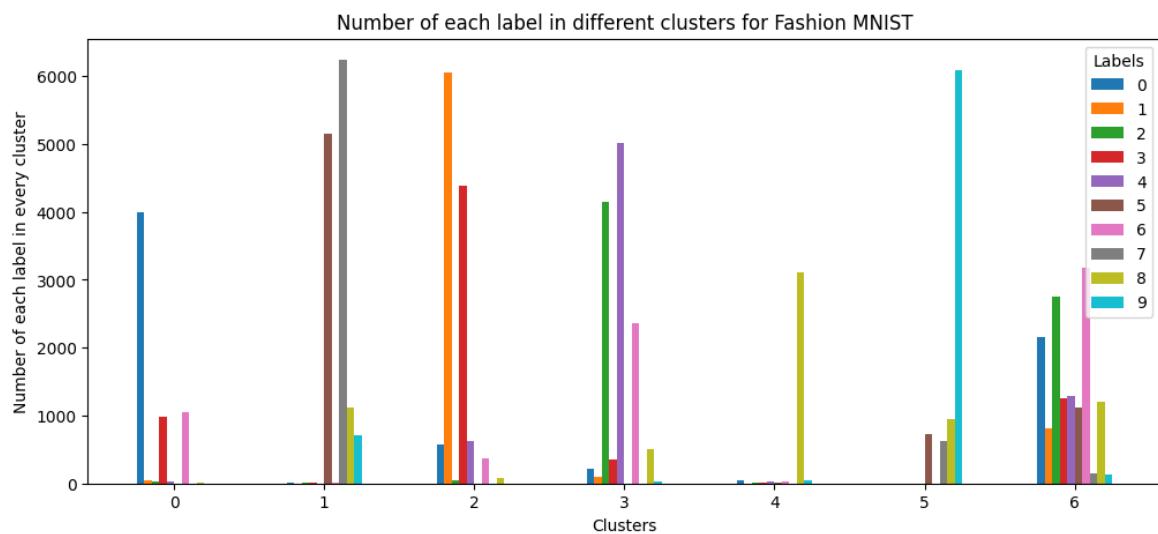
برچسب‌های مانند ۰، ۱ و ۲ به خوبی و دقیق بالا در خوشه‌های مجزا دسته‌بندی شده‌اند. این برچسب‌ها به دلیل ویژگی‌های متفاوت و مشخص خود در تصاویر به خوبی توسط الگوریتم شناسایی و خوشه‌بندی شده‌اند. ولی از طرفی در برخی خوشه‌ها، ترکیبی از برچسب‌ها مشاهده می‌شود. به عنوان مثال، برچسب ۹ و ۴ به دلیل شباهت‌های تصویری در یک خوشه ترکیب شده‌اند. این مورد باعث شده که دسته‌بندی بهینه ۱۰ تا نباشد. هرچند صورت سوال مشخص کرده که فقط ۱۰ ایپاک جلو برویم. احتمالاً اگر مدل‌ها با پارامترهای بهتری آموزش دهیم نیز نتیجه بهتر شود.

خوشه‌بندی دیتاست Fashion MNIST

مراحل انجام شده برای دیتاست MNIST را برای این دیتاست نیز انجام می‌دهیم و هردو نمودار خروجی گرفته شد.

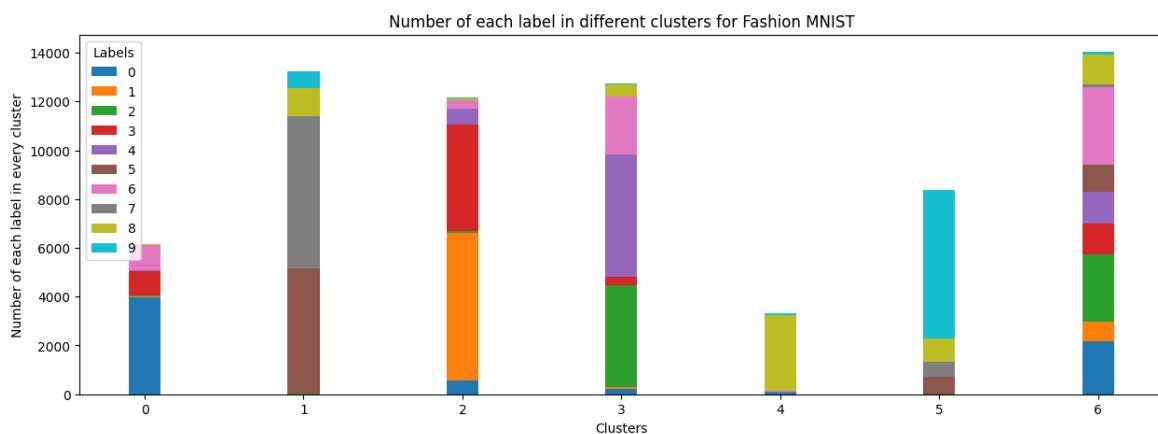
باتوجه به قسمت قبل خوشه‌بندی بر روی ۷ دسته انجام شد.

نمودار اول: تعداد برچسب‌های هر خوشه



شکل ۱۳ تعداد برچسب‌های هر خوشه دیتاست Fashion MNIST

نمودار دوم: درصد برچسب‌های هر خوشه



شکل ۱۴ درصد برچسب‌های هر خوشه دیتاست Fashion MNIST

مثل دیتاست قبلی با نگاهی به این نمودارها می‌توانیم چالش‌ها و موفقیت‌های موجود در خوشبندی داده‌ها را شناسایی کنیم و نتیجه‌گیری کنیم که کدام برچسب‌ها به خوبی تشخیص داده شده‌اند و کدام برچسب‌ها با مشکلاتی مواجه شده‌اند.

هر خوشه به طور واضح یک برچسب غالب دارد، برچسب‌های مانند ۰، ۱، و ۹ به خوبی در خوشه‌های مجزا دسته‌بندی شده‌اند. برای درک بهتر ابتدا برچسب‌های مختلف این دیتاست را بررسی می‌کنیم:

داده‌های Fashion MNIST به ۱۰ دسته زیر تقسیم می‌شوند:

۱. تی‌شرت/تاب (T-shirt/top)
۲. شلوار (Trouser)
۳. پلیور (Pullover)
۴. لباس (Dress)
۵. کت (Coat)
۶. سندل (Sandal)
۷. پیراهن (Shirt)
۸. کتانی (Sneaker)
۹. کیف (Bag)
۱۰. چکمه (Ankle boot)

حال به بررسی خوشبندی می‌پردازیم توجه شود که لیبل‌ها در نمودار از ۰ شروع می‌شوند:

برچسب‌های تی‌شرت/تاب ۰ و پیراهن ۶ در برخی خوشبندی‌ها با هم ترکیب شده‌اند. این به دلیل شباهت‌های ظاهری بین تی‌شرت و پیراهن است. این مورد برای برچسب‌های پولیور ۲ و کت ۴ نیز اتفاق افتاده است. همچنین برچسب‌های سندل ۶ و کتانی ۷ به دلیل شباهت‌های ظاهری در پوشش پا ترکیب شده‌اند. از طرفی برچسب کیف ۹ و شلوار ۱ به دلیل ساختار خاص به نسبت خوبی در خوشبندی مجزا دسته‌بندی شده است.

به طور کلی مدل ما توانسته است داده‌های Fashion MNIST را با دقت نسبتاً خوبی خوشبندی کند. با این حال، شباهت‌های بصری بین برخی دسته‌ها مانند تی‌شرت/تاب و پیراهن یا پولیور و کت، منجر به ترکیب آن‌ها در یک خوشبندی شده است.

پرسش ۲ - افزایش داده در مدل FaBert

برای حل این سوال از محیط Kaggle استفاده شد. در ادامه به بررسی تمام مراحل انجام شده خواهیم پرداخت. برای پاسخ به بخش اول تمرین از منابع زیر استفاده شده است:

Feng, S. Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., & Hovy, E. (۲۰۲۱). A survey of data augmentation approaches for NLP. *arXiv preprint arXiv:2105.03075*.

Li, Bohan, Yutai Hou, and Wanxiang Che. "Data augmentation approaches in natural language processing: A survey." *Ai Open* ۳ (۲۰۲۲): ۷۱-۹۰.

Shorten, Connor, Taghi M. Khoshgoftaar, and Borko Furht. "Text data augmentation for deep learning." *Journal of big Data* ۸, ۱ (۲۰۲۱): ۱۰۱.

NLP Data Augmentation .۱-۱

افزایش داده (Data Augmentation) یک تکنیک مهم در حوزه یادگیری ماشین و یادگیری عمیق است که با تولید نمونه‌های مصنوعی از داده‌های موجود، به افزایش تنوع و حجم داده‌های آموزشی کمک می‌کند. این روش‌ها به بهبود عملکرد مدل‌ها، کاهش بیشبرازش و افزایش Generalization مدل در برخورد با داده‌های جدید و دیدهنشده کمک می‌کنند.

تکنیک‌های رایج افزایش داده در NLP

در زمینه پردازش زبان‌های طبیعی، تکنیک‌های مختلفی برای افزایش داده وجود دارد که در ادامه به بررسی برخی از آن‌ها می‌پردازیم.

جایگزینی مترا遁فها : (Synonym Replacement)

در این روش، کلمات موجود در یک جمله با مترا遁ف‌های آن‌ها جایگزین می‌شوند. این کار به تولید جملات جدید با حفظ معنای اصلی کمک می‌کند و باعث افزایش تنوع در داده‌های آموزشی می‌شود. به عنوان مثال، در جمله The quick brown fox jumps over the lazy dog کلمه quick می‌تواند با مترا遁ف خود مانند fast جایگزین شود.

حذف تصادفی : (Random Deletion)

این تکنیک شامل حذف تصادفی برخی از کلمات در یک جمله است. این کار باعث می‌شود مدل با جملاتی که برخی از اطلاعات در آن‌ها ناقص است، روبرو شود و بهبود یابد.

: (Random Swap) ترتیب‌دهی مجدد کلمات

در این روش، دو کلمه در یک جمله به‌طور تصادفی جایجا می‌شوند. این کار به مدل کمک می‌کند تا با ترکیب مختلف کلمات آشنا شود.

: (Random Grammar Transformations) تغییرات تصادفی در قواعد دستوری

این تکنیک شامل اعمال تغییرات تصادفی در ساختارهای دستوری جمله‌های است، مانند تغییر زمان فعل‌ها یا استفاده از حالت‌های مختلف جمله‌ها.

: Back Translation

یکی از قدرتمندترین روش‌ها برای افزایش داده، بازترجمه است. در این روش، متن از زبان مبدا به یک زبان دیگر ترجمه شده و سپس دوباره به زبان اصلی بازگردانده می‌شود. این فرآیند باعث تولید جملات متنوع با حفظ معنای اصلی می‌شود. در ادامه به بررسی بیشتر این مورد می‌پردازیم.

این روش شامل سه مرحله اصلی است:

: (Intermediate Translation) ترجمه به زبان واسط

در این مرحله، متن اصلی از زبان مبدا به یک زبان دیگر (زبان واسط) ترجمه می‌شود. این زبان می‌تواند هر زبانی باشد که سیستم ترجمه به خوبی پشتیبانی می‌کند. به عنوان مثال، متن انگلیسی به فرانسوی ترجمه می‌شود.

: (Reverse Translation) بازگشت به زبان اصلی

در مرحله بعد، متن ترجمه شده از زبان واسط دوباره به زبان اصلی ترجمه می‌شود. به عنوان مثال، متن فرانسوی دوباره به انگلیسی برگردانده می‌شود.

: Pruning

در این مرحله، جملاتی که تفاوت قابل توجهی با جمله اصلی ندارند حذف می‌شوند تا تنها جملات متنوع باقی بمانند. این مرحله به کاهش جملات تکراری و افزایش تنوع کمک می‌کند.

به طور کلی تکنیک Back Translation چندین مزیت و کاربرد مهم در زمینه پردازش زبان‌های طبیعی دارد. یکی از اصلی‌ترین مزایا، افزایش تنوع داده‌ها است که با تولید جملات متنوع با حفظ معنای اصلی،

مدل‌ها را قادر می‌سازد تا بهتر با داده‌های جدید و نادیده گرفته شده کنار بیایند. این امر بهویژه در مواردی که داده‌های آموزشی محدود هستند، بسیار مؤثر است و باعث بهبود دقت و عملکرد مدل‌ها می‌شود. علاوه بر این، این تکنیک در سیستم‌های ترجمه ماشینی به طور گسترده‌ای مورد استفاده قرار می‌گیرد و می‌تواند به تولید جملات جدیدی کمک کند که کیفیت ترجمه و دقت سیستم‌های ترجمه را افزایش دهد.

با این حال، استفاده از Back Translation با چالش‌ها و محدودیت‌هایی نیز همراه است. یکی از مهم‌ترین چالش‌ها، کیفیت ترجمه است. اگر سیستم ترجمه به خوبی عمل نکند، جملات ترجمه شده ممکن است دقیق نباشند و معنای اصلی را به درستی منتقل نکنند. علاوه بر این، Back Translation می‌تواند زمان بر و هزینه‌بر باشد، بهویژه اگر نیاز به ترجمه حجم زیادی از داده‌ها داشته باشیم.

۱-۲. پیش‌پردازش دادگان

ابتدا کتابخانه‌های مورد نیاز را نصب و Import می‌کنیم. با توجه به اینکه این تمرین در محیط Kaggle پیاده‌سازی شد، فایل‌های مربوط به دیتاست آموزش و آزمون دانلود و سپس به صورت ورودی آپلود شدند. پس از آن داده‌های آموزشی و آزمون را از طریق مسیرهای مربوطه بارگذاری می‌کنیم و به بررسی چند سطر اول از داده‌ها می‌پردازیم تا با ساختار و محتوای آن‌ها آشنا شویم.

در کل این تمرین ۲ کلیدواژه اصلی داریم. Original منظور داده‌های اصلی و Augmented منظور دیتاست جدید است که علاوه بر داده‌های اصلی، داده‌های Augment شده نیز به آن‌ها اضافه شده است.

	comment	label
0	...گوشی خوبیه(قوی و شکل و زیبا و بی رقیب)البته	1
1	سلام خیلی خوبه بخرین	2
2	...می‌تو HTC Desire SV از جمله قابلیت‌های ارتباطی	0
3	... نیز برای انجام مکالمات VGA نهایتا، یک دوربین	0
4	...من حدوداً ۱ ماهی که می‌شه این گوشی رو دارم، ر	1

شکل ۱۵ نمونه‌هایی از داده‌های آموزشی دیتاست DeepSentiPers

برای اطمینان از بارگذاری صحیح داده‌ها تعداد کل نمونه‌های موجود در داده‌های آموزشی و آزمون خروجی گرفته شد:

Number of data in the dataset: ۷۴۱۵

نگاشت برچسب‌ها

برچسب‌های Unique در دیتاست به صورت زیر هستند:

[1 2 0 -1 -2]

در مرحله بعد، برچسب‌های موجود در مجموعه داده که شامل پنج کلاس مختلف بودند، به سه کلاس ادغام شدند تا یک دیتاست ساده‌تر برای Sentiment Analysis ایجاد شود. به منظور این ادغام، برچسب‌های ۲ و +۱ و ۰ به عنوان نظرات مثبت و برچسب‌های -۲ و -۱ به عنوان نظرات منفی در نظر گرفته شد و برچسب ۰ به عنوان نظرات خنثی باقی ماند. برای این کار تابع map_labels پیاده‌سازی شد که هر برچسب را به یکی از سه کلاس ۱ و ۰ و -۱ تبدیل می‌کند.

```
Map Labels

def map_labels(label):
    if label in [-2, -1]:
        return -1
    elif label in [1, 2]:
        return 1
    else:
        return 0
```

شکل ۱۶ تابع نگاشت به برچسب‌های جدید

پس از اعمال تابع بالا به دیتاست مربوطه بار دیگر برچسب‌های Unique را خروجی گرفتیم تا از اعمال درست تابع بالا اطمینان حاصل شود که خروجی به صورت زیر است:

[1 0 -1]

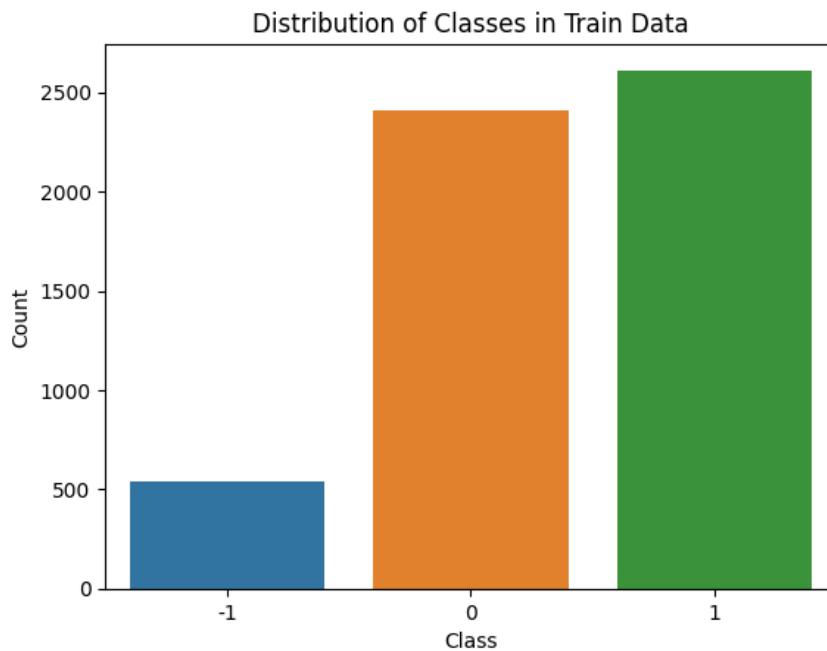
پس اولین مرحله پیش‌پردازش به خوبی انجام شد.

توزیع آماری داده‌ها

پس از ادغام برچسب‌های اولیه به سه کلاس توزیع نهایی برچسب‌ها به صورت زیر است:

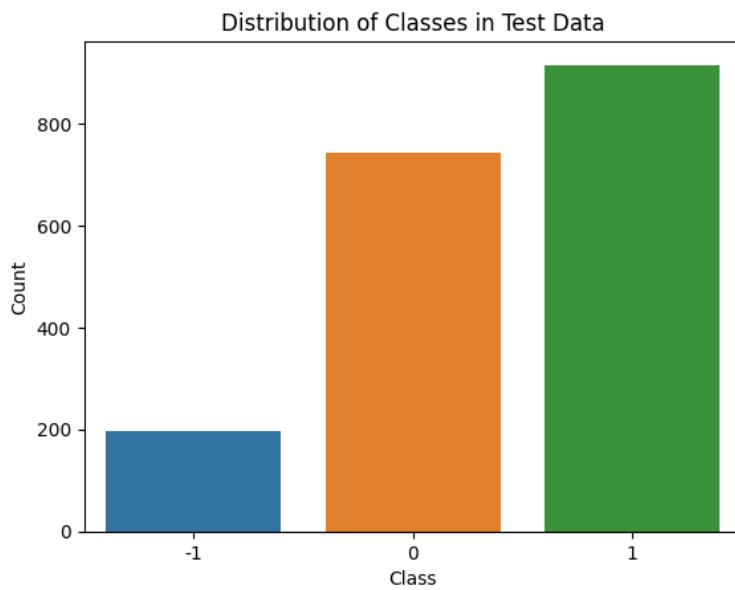
۲۶۱۱ نمونه مثبت، ۲۴۰۹ نمونه خنثی و ۵۴۱ نمونه منفی.

این توزیع نشان‌دهنده یک عدم تعادل در داده‌ها است که می‌تواند بر عملکرد مدل‌های تحلیل احساسات تاثیر بگذارد. برای نمایش توزیع، نموداری از تعداد نمونه‌های هر کلاس خروجی گرفته شد که در شکل زیر آن را مشاهده می‌کنیم.



شکل ۱۷ توزیع دسته‌های مختلف در دیتاست آموزشی

همانطور که در نمودار مشاهده می‌شود، تعداد نمونه‌های مثبت و خنثی به مراتب بیشتر از نمونه‌های منفی است. این عدم تعادل ممکن است نیاز به تکنیک‌های متعادلسازی مانند oversampling یا undersampling داشته باشد. ولی با توجه به اینکه هدف این تمرین چیز دیگری است از اعمال آن در این مرحله صرف نظر کردیم. این مورد فقط مربوط به داده‌های آموزشی نیست و داده‌های آزمون نیز نامتوازن هستند.



شکل ۱۸ توزیع دسته‌های مختلف در دیتاست آزمون

توزیع تعداد توکن‌ها

اطلاعات آماری تعداد توکن‌ها و توزیع آن‌ها می‌تواند دید خوبی در مورد ساختار و پیچیدگی داده‌ها به ما ارائه دهد. نتایج به دست‌آمده از داده‌های آموزشی به صورت زیر هستند:

تعداد کل نمونه‌ها: ۵۵۶۱

میانگین تعداد توکن‌ها: ۲۱.۹۲

انحراف معیار: ۲۰.۴۳

حداکثر تعداد توکن‌ها: ۱

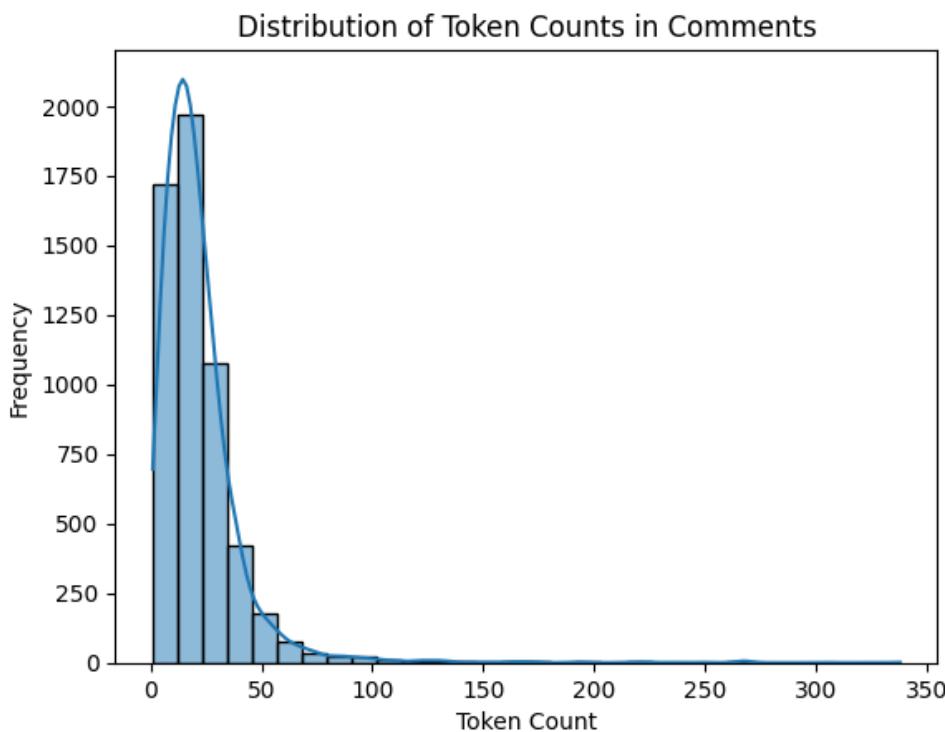
چارک اول (۲۵ درصد): ۱۱

میانه (۵۰ درصد): ۱۸

چارک سوم (۷۵ درصد): ۲۷

حداکثر تعداد توکن‌ها: ۳۳۸

توزیع آن نیز به صورت زیر است :



شکل ۱۹ توزیع آماری تعداد توکن‌ها در دیتاست آموزشی

این نمودار نشان می‌دهد که بیشترین فراوانی تعداد توکن‌ها در بازه ۱۰ تا ۳۰ قرار دارد و تعداد کمی از نظرات دارای تعداد توکن‌های بیش از ۱۰۰ هستند.

پیش‌پردازش‌های حوزه متن

برای انجام این پیش‌پردازش‌ها توابع مختلفی نوشته شد. ولی همه آن‌ها را در این مرحله اعمال نکردیم. درست است که برای تحلیل احساسات مفید هستند ولی با توجه به اینکه باید متن را BackTranslate کنیم برخی از پیش‌پردازش‌ها را در مرحله آخر و پیش از آموزش مدل تحلیل احساسات انجام دادیم. حال به توضیح برخی از این پیش‌پردازش‌ها می‌پردازیم:

حذف کاراکترهای تکراری:

کاراکترهای تکراری مانند خوووووب می‌توانند نویز ایجاد کنند پس با حذف آن‌ها متن یکپارچه‌تری خواهیم داشت:

Before: امرووووز هوا خیلیبییی خوبهههههه

After: امروز هوا خیلی خوبه

ریشه‌یابی (stemming)

ریشه‌یابی کلمات و تبدیل آن‌ها به شکل پایه‌ی کلمه کمک می‌کند تا تنوع کلمات کاهش یابد و تحلیل احساسات ساده‌تر شود.

کتاب‌ها: Before

کتاب: After

البته ریشه‌یابی با توجه به اینکه ممکن است باعث اختلال در BackTranslation شود آن را اعمال نکردیم.

اصلاح املای نادرست کلمات (Normalizer)

منبع تحلیل انجام شده در این تمرین و دیتاست ارائه شده نظراتی هستند که ممکن است کاربر سریع نوشته و یا دقت کافی نداشته باشد و شامل غلط املایی باشند.

اعمال پیش‌پردازش‌های پیاده‌سازی شده

باتوجه به اینکه ممکن است برخی از پیش‌پردازش‌هایی که بررسی شدند، تاثیر مثبتی نداشته باشند، نیازی نیست که تمامی آن‌ها را اعمال کنیم. ابتدا یکتابع نوشته که بتوانیم کنترل بهتری بر روی مواردی که نیاز است اعمال شوند و ترتیب آن‌ها داشته باشیم.

بیشتر پیش‌پردازش‌های حوزه متن در مرحله نهایی و پس از Backtranslation اعمال شد. همچنین برای طول توکن مختلف عمل Padding انجام خواهیم داد.

۱-۳. افزایش دادگان به روش Back Translation

در این مرحله باید بر روی داده‌های آموزشی افزایش داده‌ها را به کمک تکنیک Back Translation مدل کنیم.

برای پیاده‌سازی این روش، از کتابخانه googletrans استفاده شد که یکی از ابزارهای قدرتمند ترجمه مبتنی بر Google Translate API است. در کد زیر،تابع back_translate تعریف شده که یک متن ورودی را از زبان فارسی به انگلیسی ترجمه کرده و سپس آن را به فارسی بازمی‌گرداند:

```

def back_translate(text, src_lang='fa', target_lang='en'):
    try:
        translated = translator.translate(text, src=src_lang, dest=target_lang).text
        back_translated = translator.translate(translated, src=target_lang, dest=src_lang).text
        return back_translated
    except Exception as e:
        print(f"Error: {e}")
        return text

```

شکل ۲۰ تابع BackTranslate

در اجرای این تابع خطایی رخ ندارد پس بر روی تمامی جملات به خوبی اعمال شد. پس از اعمال Back Translation، اندازه مجموعه داده به دلیل تولید کامنت‌های جدید دو برابر می‌شود.

تعداد داده‌های آموزشی در خروجی قبل برابر با ۵۵۶۱ بود که پس از اعمال Backtranslate به ۱۱۱۲۲ رسید. این افزایش داده‌ها به بهبود عملکرد مدل تحلیل احساسات ما کمک خواهد کرد، زیرا مدل‌ها با مجموعه داده‌های متنوع‌تر و گستره‌تری آموزش می‌بینند و توانایی تعمیم‌دهی بهتری پیدا می‌کنند. این تکنیک همچنین به کاهش بیش‌برازش کمک می‌کند زیرا مدل‌ها نمی‌توانند به راحتی الگوهای ثابت و تکراری را یاد بگیرند و به جای آن باید با داده‌های متنوع‌تری کار کنند. در جدول زیر ۱۰ نمونه از خروجی را بررسی می‌کنیم.

Back-Translated	Original
گوشی خوبیه (قوی، شیک، زیبا و بی‌رقیب) البته در رده خودش عالیه!	گوشی خوبیه (قوی و شکیل و زیبا و بی‌رقیب) البته تو تیپ و گروه خودش عالیه!
سلام خیلی خوب بخر	سلام خیلی خوبه بخرین.
از قابلیت‌های ارتباطی HTC Desire SV می‌توان به پشتیبانی از GPRS، Wi-Fi، ۳G اشاره کرد که ارتباط آسان با اینترنت را فراهم می‌کند.	از جمله قابلیت‌های ارتباطی HTC Desire SV می‌توان به پشتیبانی از GPRS، Wi-Fi، ۳G اشاره کرد که برقراری ارتباط با اینترنت را به سهولت فراهم می‌کند.
در نهایت یک دوربین VGA نیز در جلوی گوشی برای مکالمات تصویری تعییه شده است.	نهایتاً، یک دوربین VGA نیز برای انجام مکالمات تصویری در قسمت جلوی گوشی تعییه گردیده است.
من حدوداً ۱ ماهه که این گوشی رو دارم و راضی هستم. در مورد دکمه‌های لمسی پایین گوشی نیز باید گفت که حساسیت این دکمه‌ها کم	من حدوداً ۱ ماهی که می‌شه این گوشی رو دارم، راضی هم هستم، در مورد دکمه‌های لمسی پایین گوشی باید گفت که حساسیت این دکمه‌ها کم

بلکه قسمت حساس این دکمه ها بسیار ظریف طراحی شده است، بنابراین یا باید دستان خود را بکشید یا نقطه Find the حساس است و هنگامی که به آن عادت کردید، می توانید دقیقاً آن نقطه را لمس کنید (باید کمی بالاتر از علامت روی دکمه ها باشد، تقریباً نوک علائم)، بنابراین نیازی به کشیدن نیست.	نیست، بلکه قسمت حساس این دکمه ها خیلی ریز طراحی شده، واسه همین یا باید دستون رو روش بکشید یا اینکه نقطه حساس رو پیدا کرده و طبقه عادت بتونین دقیق اون نقطه رو لمس کنین (دقیقترش میشه یه ذره بالاتر از علامت روی دکمه ها، تقریباً نوک علامت ها)، اینطوری دیگه نیازی به کشیدن هم نداره.
اندازه نسبتا مناسب و وزن خوب ۴.	اندازه نسبتا مناسب و وزن خوب ۴.
گوشی خوب و شیک است / اما خیلی مقرر نیست صرفه نیست	گوشیه خوب و شیکیه / ولی خیلی به صرفه نیست
من نزدیک به ۲ سال است که از این اسکنر استفاده می کنم و کاملا از آن راضی هستم	تقریباً ۲ سال هست که از این اسکنر استفاده می کنم و کاملا از آن راضیم
کلیدها فضای کافی برای فشار دادن دارند و بازخورد بسیار خوبی دارند.	کلیدها فضای کافی برای فشار دادن داشته و بازخورد بسیار خوبی دارند.
سلام من یک هفته پیش دوربین را خریدم اولین نکته این است که این دوربین ساخت کشور چین است دوم اینکه در مود دستی که رنگ های استاندارد عکاسی کرم و سیاه و سفید دارد نمی توان عکس کرم گرفت و فقط سیاه و سفید و استاندارد را می توان انتخاب کرد. ثالثاً به دلیل قیمت نمیشه عکس های خوبی گرفت و فقط یک کار استارتاپی هست	با سلام من یک هفته است که دوربین رو خریداری کردم اولین نکته که این دوربین ساخت چین است دوم اینکه در مود دستی که رنگ های عکاسی استاندارد و کرم و سیاه و سفید دارد شما عکس کرم را نمی توانید بگیرید و فقط سیاه سفید و استاندارد را میتوان انتخاب کرد سوم اینکه با توجه به قیمت آن عکسهای خوبی نمی توان گرفت و فقط کار راهانداز است

جدول ۱ نمونه اصلی و BackTranslate جملات دیتابست آموزشی

در اکثر نمونه های بررسی شده، معنای اصلی جملات به خوبی حفظ شده است.

برای مثال، جملات گوشی خوبیه (قوی و شکیل و زیبا و بی رقیب) البته تو تیپ و گروه خودش عالیه! و گوشی خوبیه (قوی، شیک، زیبا و بی رقیب) البته در رده خودش عالیه! هر دو معنای یکسانی را منتقل می کنند.

در برخی نمونه‌ها، تغییرات جزئی در ساختار جمله مشاهده می‌شود، اما این تغییرات تاثیری بر معنای کلی جمله ندارند. این ویژگی مهمی برای تحلیل احساسات است، زیرا مدل‌ها به دنبال الگوها و احساسات کلی در جملات هستند.

برای مثال جمله سلام خیلی خوب بخرین. به سلام خیلی خوب بخر تبدیل شده است. این تغییرات تاثیر منفی بر تحلیل احساسات ندارند چون که معنی و احساس کلی جمله حفظ شده است.

ترجمه‌های انجام شده توسط Google Translate به طور کلی دقیق‌تر و در مجموع، روش Back Translation موفق به حفظ معنای اصلی جملات شده است، هرچند برخی تغییرات جزئی در ساختار جملات مشاهده می‌شود.

در نهایت داده‌های اصلی و داده‌های افزوده شده به روش Back Translation ترکیب و به عنوان دیتاست‌های جدید ذخیره شدند. داده‌های اصلی، داده‌های آزمون پس از پیش‌پردازش و همچنین داده‌های Augment شده به صورت فایل‌های CSV ذخیره شدند تا در مراحل بعدی پروژه پس از اضافه شدن در محیط Kaggle به صورت دیتاست ورودی مورد استفاده قرار گیرند.

۴-۱. تنظیم دقیق (FineTune) مدل

سه مجموعه داده شامل داده‌های افزوده شده، داده‌های اصلی و داده‌های آزمون را بارگذاری می‌کنیم. همچنین برچسب‌های داده‌ها به سه کلاس جدید ۰ برای منفی، ۱ برای خنثی، و ۲ برای مثبت تبدیل می‌کنیم تا با فرمت مدل FaBert و توابع Loss سازگار باشند.

قبل از انجام پیش‌پردازش‌های مربوط به متن مجموعه داده‌ای برای اعتبارسنجی (Validation Set) ایجاد شد تا در طی فرآیند آموزش مدل FaBert، بتوان عملکرد مدل را ارزیابی کرد. به منظور ایجاد این دیتاست، ۳۰٪ از داده‌های آزمون به عنوان داده‌های اعتبارسنجی جدا کردیم تا مطمئن باشیم پیش‌پردازش‌ها و BackTranslate بر روی آن‌ها اعمال نشده است و دقت خالص را بدست آوریم. پس از آن از تعداد داده‌های مربوط به هر کدام از دیتاست‌ها خروجی گرفتیم:

داده‌های آموزشی افزوده شده: ۱۱۱۲۲

داده‌های آموزشی اصلی: ۵۵۶۱

داده‌های اعتبارسنجی: ۵۵۶

داده‌های آزمون: ۱۲۹۸

پس از این مرحله پیش‌پردازش‌های مختلف پیاده‌سازی شده مربوط به حوزه NLP بر روی داده‌ها اعمال شد. بدون هیچ گونه پیش‌پردازشی دقت حدود ۵ درصد کمتر خروجی فعلی می‌شود. حدود ۱۰ ترکیب مختلف آزمایش شد تا بهترین ترکیب از پیش‌پردازش‌ها انتخاب شود. برای نمونه در ابتدا تمامی پیش‌پردازش‌های مذکور اعمال شد ولی بعداً متوجه شدیم که اگر stemming را انجام ندهیم دقت مدل افزایش پیدا می‌کند. پس از اعمال آن صرف نظر کردیم.

کتابخانه‌های استفاده شده برای پیش‌پردازش

در پیش‌پردازش متن نظرات از چند کتابخانه مهم حوزه NLP استفاده کردیم. در ادامه توضیح مختصری درباره هر یک از این کتابخانه‌ها را بررسی می‌کنیم.

کتابخانه re یکی از کتابخانه‌های استاندارد پایتون برای کار با عبارات منظم (regular expressions) است. این کتابخانه امکان جستجو، تطابق و جایگزینی الگوهای در رشته‌ها را فراهم می‌کند.

کتابخانه Hazm نیز یکی از کتابخانه‌های محبوب برای پردازش زبان فارسی است. این کتابخانه ابزارهای مختلفی برای نرمال‌سازی، Tokenization، ریشه‌یابی و حذف واژگان غیرمفید فراهم می‌کند که در بخش‌های مربوط به پیش‌پردازش از آن استفاده شد.

در این مرحله باید مدل FaBert و توکنایزر مربوطه از کتابخانه transformers بارگذاری کنیم.

استفاده از توکنایزر

ابتدا مدل توکن‌ساز از پیش‌آموزش دیده شده FaBert را فراخوانی کرده و برای تبدیل داده‌های متنی پیش‌پردازش شده به اعداد استفاده می‌کنیم. همچنین برای یکسان‌سازی طول تمام سطرها از Padding استفاده کرده و طول تمام سطرها را برابر با ۱۲۸ تنظیم می‌کنیم.

پس از آماده‌شدن ورودی مدل را بارگذاری کردیم و از مدل خروجی گرفتیم تا لایه‌های مختلف آن را بررسی کنیم. در نهایت تمام لایه‌ها بجز لایه Classifier و لایه آخر مدل برت freeze شدند تا مقادیر پارامترهای آن‌ها بروزرسانی نشود.

```

for param in model_org.bert.parameters():
    param.requires_grad = False

for param in model_org.bert.encoder.layer[-1:].parameters():
    param.requires_grad = True

for param in model_org.classifier.parameters():
    param.requires_grad = True

```

شکل ۲۱ فریزکردن تمام مدل بجز لایه Classifier و لایه آخر مدل برت

آموزش مدل با داده‌های اصلی

در اولین مرحله آموزش، مدل FaBert با استفاده از داده‌های اصلی آموزش داده شد. نتایج آموزش شامل مقادیر مختلفی از معیارهای ارزیابی مانند دقت (Accuracy)، دقت (Precision)، F1-score و فراخوانی (Recall) برای هر ایپاک (Epoch) است. این نتایج به ما کمک می‌کنند تا عملکرد مدل را در طول فرآیند آموزش بررسی و ارزیابی کنیم.

[1740/1740 02:34, Epoch 5/5]						
Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	No log	0.524230	0.767986	0.732557	0.701849	0.767986
2	0.679800	0.472566	0.807554	0.808153	0.808859	0.807554
3	0.500200	0.466837	0.812950	0.812416	0.812949	0.812950
4	0.500200	0.486477	0.811151	0.809333	0.811389	0.811151
5	0.426100	0.470390	0.823741	0.823217	0.823247	0.823741

شکل ۲۲ نتیجه آموزش مدل بر روی داده‌های اصلی

نتایج نشان می‌دهد که مدل FaBert به خوبی توانسته است الگوهای موجود در داده‌ها را یاد بگیرد و عملکرد مناسبی در دسته‌بندی احساسات داشته باشد. در ادامه یک مدل جداگانه را بر روی داده‌های افزوده شده آموزش می‌دهیم.

آموزش مدل با داده‌های افزوده شده

به مانند مدل قبلی، تمامی لایه‌های مدل به جز لایه‌های آخر فریز شده‌اند تا از بیش‌برازش جلوگیری شود. با همان پارامترهای قبلی این مدل هم آموزش داده شد که نتیجه آن به صورت زیر است:

[3480/3480 04:51, Epoch 5/5]							
Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall	
1	0.650700	0.404164	0.843525	0.846584	0.854010	0.843525	
2	0.478000	0.365991	0.863309	0.862979	0.863509	0.863309	
3	0.371300	0.380872	0.857914	0.857218	0.856776	0.857914	
4	0.332300	0.391883	0.852518	0.852811	0.853182	0.852518	
5	0.300900	0.410125	0.857914	0.858961	0.862358	0.857914	

شکل ۲۳ نتیجه آموزش مدل بر روی داده‌های افزوده شده

با مقایسه نتایج آموزش مدل با داده‌های اصلی و افزوده شده، مشاهده می‌شود که مدل آموزش دیده با داده‌های افزوده شده دقت و کارایی بهتری در تحلیل احساسات داشته است.

برای ذخیره دقت و Loss ابتدا لگ‌های آموزش مدل‌ها برای داده‌های اصلی و داده‌های افزوده شده تجزیه و تحلیل می‌شوند تا خروجی‌های مورد نظر استخراج شوند. سپس این خروجی‌ها را در قالب DataFrame Pandas‌های Kaggle CSV در محیط ذخیره می‌کنیم.

نمونه این فایل ذخیره شده به صورت زیر است :

org_df	
	eval_loss eval_accuracy
0	0.524230 0.767986
1	0.472566 0.807554
2	0.466837 0.812950
3	0.486477 0.811151
4	0.470390 0.823741

شکل ۲۴ دقت و مقدار تابع هزینه روی دادگان ارزیابی

۱-۵. ارزیابی و تحلیل نتایج

خروجی Evaluate مدل بر روی داده‌های ارزیابی و آزمون به صورت زیر است:

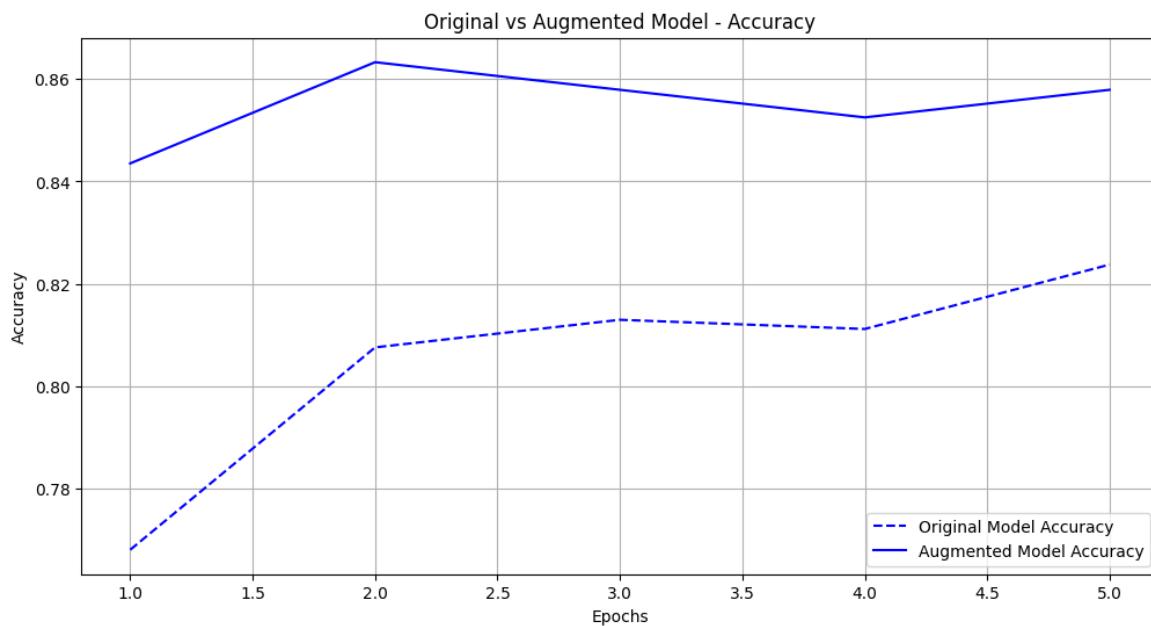
Runtime	Recall	Precision	F1	Accuracy	Loss	Data	Model
۲.۱۲۶۱	۰.۸۰۲۱۵	۰.۸۰۰۳۰۹	۰.۸۰۱۰۶	۰.۸۰۲۱۵۸	۰.۵۰۳۳۸	Validation	Original Model

۴.۸۶۶۵	۰.۷۹۸۹۲	۰.۸۰۲۸۷۲	۰.۷۹۹۹۷	۰.۷۹۸۹۲۱	۰.۵۰۰۷۹	Test	Original Model
۲.۰۷۷۳	۰.۸۶۳۳۰	۰.۸۶۳۵۰۹	۰.۸۶۲۹۷	۰.۸۶۳۳۰۹	۰.۳۶۵۹۹	Validation	Augmented Model
۴.۸۸۸۶	۰.۸۳۱۲۷	۰.۸۳۰۰۷۱	۰.۸۳۰۱۶	۰.۸۳۱۲۷۹	۰.۴۳۵۰۱	Test	Augmented Model

جدول ۲ نتایج ارزیابی مدل‌های آموزش داده شده

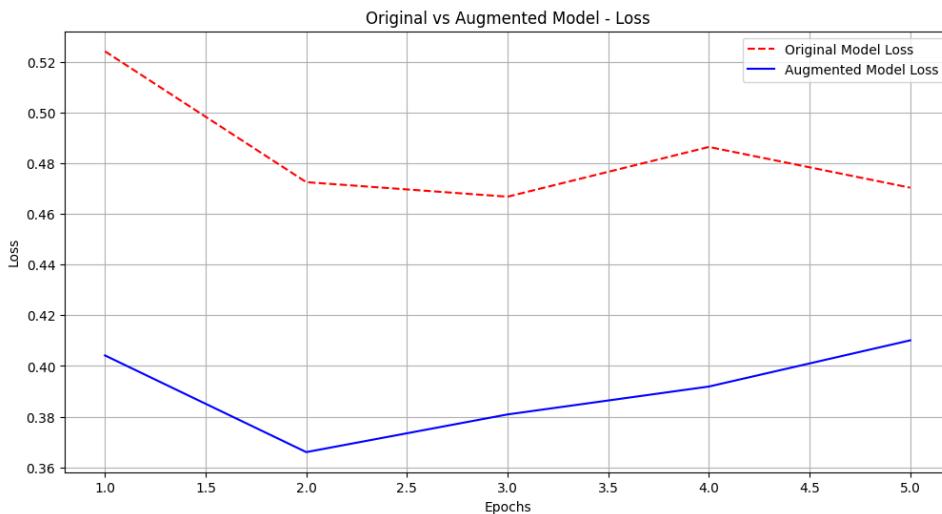
باتوجه به نتایج جدول بالا استفاده از داده‌های افزوده شده (از طریق Back Translation) منجر به بهبود در عملکرد مدل FaBert در تحلیل احساسات متون فارسی شده است. کاهش Eval Loss و افزایش Precision، Recall و F1-Score نشان‌دهنده بهبود عملکرد مدل با داده‌های افزوده شده است.

مقدار افزایش دقت در داده‌های آزمون حدود ۴ درصد است.



شکل ۲۵ نمودار ایپاک-دقت مدل‌های آموزش دیده روی دیتاست ابتدایی و دیتاست افزوده شده در نمودار بالا که مربوط به دقت در هر ایپاک است، خط قرمز نقطه‌چین مربوط به مدل آموزش دیده با داده‌های اصلی و خط آبی مربوط به مدل آموزش دیده با داده‌های افزوده شده است. مدل با داده‌های افزوده

شده به طور کلی دقت بالاتری نسبت به مدل با داده‌های اصلی دارد و این دقت در طول ایپاک‌های مختلف نسبتاً ثابت و پایدار است.



شکل ۲۶ نمودار ایپاک-هزینه مدل‌های آموزش‌دیده روی دیتاست ابتدایی و دیتاست افزوده شده

شکل بالا نیز نمودار Loss نشان‌دهنده کاهش خطای مدل‌ها در طول ایپاک‌های مختلف است. مدل با داده‌های افزوده شده (خط آبی) در طول ایپاک‌ها کاهش قابل توجهی در Loss نشان می‌دهد و به طور کلی مقدار Loss کمتری نسبت به مدل با داده‌های اصلی دارد. در ایپاک‌های بعدی مقداری بیش‌برازش رخ می‌دهد که تاثیر آن را در کاهش دقت از ۸۶ به ۸۵ در نمودار قبلی مشاهده کردیم.

حال به بررسی گزارش طبقه‌بندی هر کدام از مدل‌ها می‌پردازیم:

Original Dataset Classification Report:				
	precision	recall	f1-score	support
negative	0.71	0.65	0.68	146
neutral	0.74	0.81	0.78	513
positive	0.86	0.81	0.84	639
accuracy			0.80	1298
macro avg	0.77	0.76	0.76	1298
weighted avg	0.80	0.80	0.80	1298

شکل ۲۷ گزارش طبقه‌بندی مدل آموزش‌دیده بر روی دیتاست ابتدایی بر روی دیتاست آزمون

Augmented Dataset Classification Report:				
	precision	recall	f1-score	support
negative	0.78	0.71	0.74	146
neutral	0.82	0.79	0.80	513
positive	0.85	0.89	0.87	639
accuracy			0.83	1298
macro avg	0.82	0.80	0.81	1298
weighted avg	0.83	0.83	0.83	1298

شکل ۲۸ گزارش طبقه‌بندی مدل آموزش‌دیده بر روی دیتاست افزوده شده بر روی دیتاست آزمون

همانطور که در ابتدای تمرین مشاهده کردیم دیتاست نامتوازن است و برای اینکه ارزیابی مناسبی داشته باشیم باید از معیار F_1 وزن دار استفاده کنیم. خود F_1 یک نوع میانگین وزن دار از دقت و فراخوانی است.

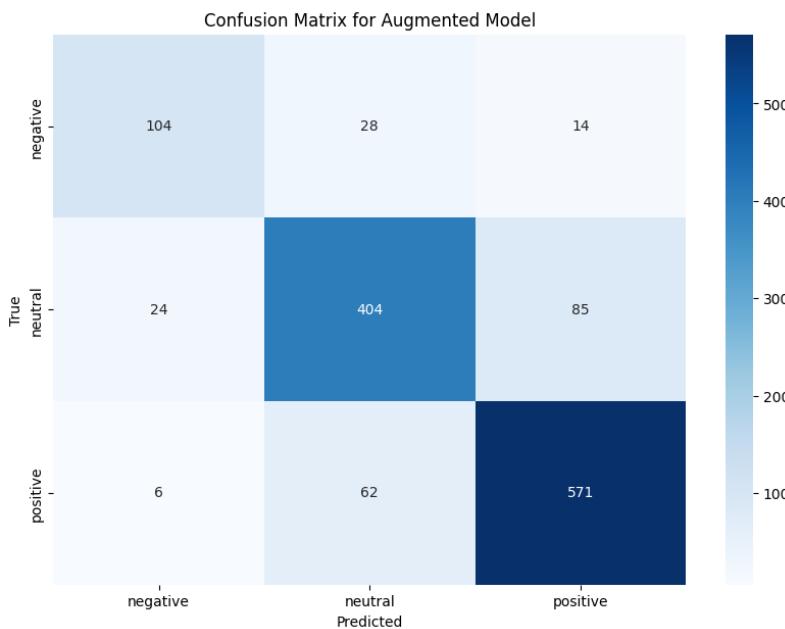
$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Weighted Avg	Positive	Neutral	Negative	
• .۸۰	• .۸۴	• .۷۸	• .۶۸	Original Model
• .۸۳	• .۸۷	• .۸۰	• .۷۴	Augmented Model

جدول ۳ جدول مقایسه F_1 -Score مدل‌های آموزش دیده با داده‌های اصلی و افزوده شده



شکل ۲۹ ماتریس آشتفتگی مدل آموزش‌دیده بر روی دیتاست ابتدایی بر روی دیتاست آزمون



شکل ۳۰ ماتریس آشتفتگی مدل آموزش دیده بر روی دیتاست افزوده شده بر روی دیتاست آزمون

ماتریس آشتفتگی مدل آموزش دیده با داده های اصلی نشان دهنده چالش هایی در تشخیص کلاس های مختلف، به ویژه کلاس با تعداد داده کمتر مثلا negative است. مدل در تشخیص این کلاس دقیق کمتری دارد و اغلب نمونه های این کلاس ها را به اشتباه به کلاس های دیگر اختصاص می دهد. این عملکرد ضعیف ناشی از نامتوازن بودن داده ها است که باعث می شود مدل نتواند به درستی الگوهای مربوط به کلاس های با داده کمتر را یاد بگیرد. در مقابل، مدل آموزش دیده با داده های افزوده شده بهبود قابل توجهی در تشخیص همه کلاس ها نشان می دهد. این بهبود نشان دهنده تاثیر مثبت استفاده از داده های افزوده شده در کاهش اثرات نامتوازن بودن داده ها و افزایش دقیق مدل در تشخیص صحیح نمونه ها است.

در نهایت برخی از نمونه ها که به اشتباه دسته بندی شده اند را بررسی می کنیم. برای مدل ابتدایی ۵ نمونه اول دیتاست و برای مدل افزوده شده ۵ نمونه آخر که به اشتباه دسته بندی شده را در جدول زیر مشاهده می کنیم:

Predicted	True	Text	Dataset
neutral	positive	اگه دیجی کالا بیاره حتما نفر اول خودم می خرم!	Original
neutral	positive	مثلا به راحتی می توانید بوسیله ای تجهیزات استقرار مختلف؛ دوربین را به دسته ای دوچرخه خود، به سطح جلوی یک اسکیت برد، به یک تخته ای موج سواری، میله های کایت و یا حتی مج و یا سینه ای خود متصل کنید.	Original

neutral	positive	که در حال حاضر این مشکل در ورزنهای جدید برطرف شده است!	Original
neutral	positive	در کل اگر می‌خواید یک دستگاه به روز رو داشته باشید تا بتونید بازی‌های روز جهان رو با اون بازی کنید من در صد این کنسول رو پیشنهاد می‌کنم.	Original
positive	negative	کیفیت صدا: کیفیت صدای هر دو عالیه ولی t_f به نظر من مقداری صدایش بلندتر است ولی اصلاً محسوس نه و اختلاف خیلی کمeh ولی کیفیت صدا در ایپد کمی بهتره.	Original
neutral	positive	اگر بزرگنمای اپتیکال یه کم بالاتر بود گزینه بسیار مناسبی بود	Augmented
neutral	positive	r hd می‌تواند با کیفیت p ful avchd در فرمت hd و با صدای استریو dolby digital فیلمبرداری کند.	Augmented
neutral	negative	باتری این گوشی یک باتری میلی‌متری آمپر ساعتیست که از باتری مدل‌های و بسیار ضعیفتر است، با این حال بیش از ساعت مکالمه متده است را پشتیبانی می‌نماید.	Augmented
neutral	negative	گوشی motorola razr m با وجود ابعاد بسیار کوچک‌تر یعنی \times میلی‌متر، صفحه نمایشی با اندازه یکسان با این گوشی دارد، و این نکته ایست که باعث می‌گردد به این نتیجه برسیم که htc در طراحی این گوشی، یک عقب‌گرد داشته است، حتی به نسبت گوشی سه سال پیش خود یعنی hd .	Augmented
positive	neutral	(یعنی گرم سبک‌تر از) ipad صفحه نمایش و بلند‌گوهای گوشی padfone دارای یک صفحه نمایش عریض / اینچی از نوع super ips+ lcd بوده و از تکنولوژی igzo کمپانی شارپ نیز بهره می‌برد که طبق ادعای این کمپانی، انرژی کمتری مصرف کرده و طول عمر بالاتری نسبت به صفحات lcd معمولی دارد.	Augmented

جدول ۴ جدول نمونه‌های اشتباه دسته‌بندی شده توسط مدل‌ها

مدل آموزش دیده با داده های اصلی تمایل دارد نمونه های مثبت را به عنوان خنثی دسته بندی کند. این نشان می دهد که مدل در تشخیص دقیق احساسات مثبت در برخی متن ها دچار مشکل است، احتمالاً به دلیل پیچیدگی و تنوع کمتر داده های مثبت در مجموعه آموزشی اصلی. همچنین، مدل نمونه های منفی را به اشتباه به عنوان مثبت دسته بندی کرده که نشان دهنده عدم تعادل در تشخیص دقیق احساسات منفی است.

در مقابل، مدل آموزش دیده با داده های افزوده شده عملکرد بهتری در تشخیص نمونه های منفی و خنثی دارد. اگرچه همچنان برخی نمونه های مثبت به عنوان خنثی و برخی نمونه های خنثی به عنوان مثبت دسته بندی می شوند. این نشان می دهد که با وجود بهبود های حاصل از افزایش داده ها، مدل هنوز در تشخیص دقیق تمامی احساسات با توجه به پیچیدگی متن ها چالش هایی دارد.

نتیجه گیری نهایی

مدل آموزش دیده با داده های اصلی در تشخیص نمونه های مثبت و خنثی عملکرد خوبی دارد، اما در تشخیص نمونه های منفی با چالش هایی مواجه است. این عملکرد ضعیفتر در کلاس های کمتر نماینده، ناشی از نامتوازن بودن داده ها در مجموعه آموزشی است. از طرفی دل آموزش دیده با داده های افزوده شده به وضوح عملکرد بهتری در تمامی کلاس ها دارد. استفاده از تکنیک Back Translation برای افزایش داده ها منجر به بهبود F1-Score در کلاس های منفی، خنثی و مثبت شده است. نتایج به دست آمده نشان می دهد که استفاده از تکنیک های افزایش داده ها می تواند به طور قابل توجهی عملکرد مدل های NLP را در تحلیل احساسات بهبود بخشد. مدل آموزش دیده با داده های افزوده شده توانسته بهبود قابل ملاحظه ای در دقت، فراخوانی و F1-Score داشته باشد و در تشخیص صحیح نمونه ها عملکرد بهتری نشان دهد.

این نتایج اهمیت تعادل و تنوع در داده های آموزشی را برجسته می کنند و نشان می دهند که تکنیک های افزایش داده ها می توانند به بهبود عملکرد مدل در مواجهه با داده های نامتوازن کمک کنند. برای بهبود بیشتر، استفاده از داده های بیشتر و متنوع تر، همراه با تکنیک های پیش پردازش مناسب، می تواند مدل را بهبود داده و دقت آن را بیشتر کند.

پرسش ۳: کلمه بیدار باش

۱-۳: جمع آوری داده

در بخش اول به پیاده سازی توابع لازم برای جمع آوری داده پرداختیم، البته به دلیل زمان محدود و سورس جمع آوری داده که فقط دو نفر میتوانستند آن را انجام دهند، تنها ۶ صدای صوتی با استفاده از این توابع به عنوان شاهدی بر کارآمدی این دو تابع ضبط کردیم که در پوشه **Voice** آن ها را ذخیره کرده ایم و در ادامه از دیتابست آمده ای که در لینک مورد نظر قرار داده بودید استفاده کردیم.

ما بر این دو سوال از دو کتابخانه **Scipy.io.wavfile** و **sounddevice** استفاده کردیم، کتابخانه اول برای ضبط صدای صوتی و کتابخانه دوم برای **write** کردن فایل به صورت یک فایل **wav** در محلی در حافظه استفاده شد.

در ادامه این دو تابع را بررسی خواهیم کرد:

```
def record_wake_word(filename: str, duration: int = 2, fs: int = 44100):

    print("Recording wake-up word...")
    myrecording = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')
    sd.wait()
    write(filename, fs, myrecording)
    print(f"Wake-up word recorded and saved as {filename}")
```

شکل ۳۱ - تابع ضبط **wake_word**

در ورودی این تابع، نام فایل، مدت آن و نرخ نمونه برداری **sample rate** را تعریف میکنیم، سپس با استفاده از متده **rec** به ضبط فایل بر اساس مدت مشخص شد و نرخ نمونه برداری میپردازیم، و در نهایت فایل را در محلی از حافظه که مشخص کرده ایم **write** میکنیم

همین تابع به عینه برای ضبط صدای پس زمینه هم استفاده میشود:

```
def record_background_noise(filename: str, duration: int = 2, fs: int = 44100):

    print("Recording background noise...")
    myrecording = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')
    sd.wait()
    write(filename, fs, myrecording)
    print(f"Background noise recorded and saved as {filename}")
```

شکل ۳۲ - تابع **record_background_noise**

در نهایت برای اطمینان از کارکرد این تابع سه وویس با هر کدام ضبط کردیم:

```
record_wake_word(filename='wake_3', duration= 2, fs= 44100)
```

Recording wake-up word...
Wake-up word recorded and saved as wake_3

```
record_background_noise(filename='background_3', duration=2, fs=44100)
```

Recording background noise...
Background noise recorded and saved as background_3

شکل ۳۳ - ضبط وویس با دو تابع مورد نظر

اما در ادامه سوال از دیتاست قرار داده شده در صورت سوال استفاده کردیم که در ادامه توضیحات آن را ارائه خواهیم کرد:

۳-۲: پیش پردازش و استخراج ویژگی

در ادامه ابتدا به دانلود فایل های مورد نظر پرداختیم، ما در این تمرین از وویس های Alexa برای wake_word، در این فایل جمیا ۳۲۹ فایل صوتی از کلمه Alexa موجود بود، که آدرس تمامی این فایل ها را در قالب یک دیتافریم pandas ذخیره کردیم، نکته مهم این است که تمامی این فایل ها لبیل ۱ دارند زیرا آن ها کلاس صحیح ما هستند، در ادامه نتیجه این بخش که یک دیتافریم است دیده میشود، تعداد فایل نیز در یک کادر قرمز رنگ نمایش داده شده است.

329

Audio label		
0	alexa/0.flac	1
1	alexa/1.flac	1
2	alexa/10.flac	1
3	alexa/100.flac	1
4	alexa/101.flac	1

شکل ۳۴ - آدرس فایل های wake_up

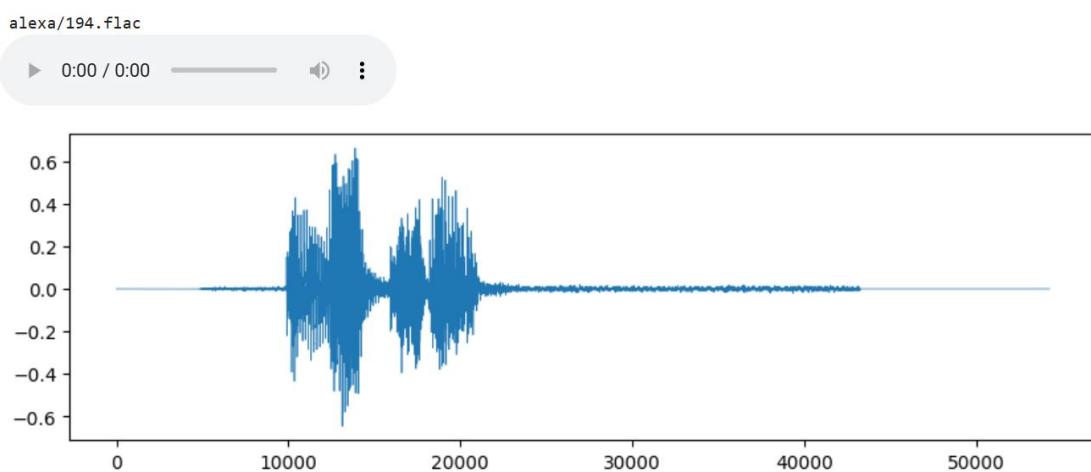
سپس از دیتافریم موجود، ۳ نمونه تصادفی را نمونه گرفته و با استفاده از lobrosa فرکانس و نرخ نمونه برداری آن را دریافت کردیم، در ادامه این فایل را به صورت صدا خروجی گرفتیم و از نمودار فرکانس آن را نیز رسم کردیم.

این کار به دو منظور اصلی است:

اولاً: به صورت تصادفی چندین نمونه تصادفی فایل ها را گوش دهیم تا از کیفیت آن ها درکی داشته باشیم تا بتوانیم در ادامه روش های مناسب برای پیش پردازش فایل صوتی را لحاظ کنیم.

دوماً: با رسیم نمودار فرکانس، دید بهتری نسبت به فایل صوتی داشته باشیم.

یک نمونه این فایل ها در ادامه خواهیم دید:



شکل ۳۵ - نمودار و فایل صوتی به صورت رندوم

نکته مهم اینجاست، که دیتاست مورد نظر، فایل های لازم برای background noise را در خودش ندارد، البته لینکی در آن سایت موجود بود که از مجموعه دیگری برای استفاده از background استفاده شده است، ما نیز از مجموعه "test-clean.tar.gz" معرفی شده در سایت استفاده کردیم، این فایل را نیز به همین شکل دانلود کردیم و همه دیتای موجود در آن را در قالب آدرس هایی در یک دیتابریم پانداش ذخیره کردیم، نکته مهم اینجاست که تعداد بسیار زیادی فایل صوتی در این مجموعه موجود بود، به همین دلیل هم از میان این مجموعه برای رعایت شدن توازن، تنها به تعداد وویس های مجموعه Alexa یعنی ۳۲۹ عدد، به صورت تصادفی سمپل گرفتیم و آن ها را مبنای صدای background noise قرار دادیم، در ادامه این موضوع را مشاهده میکنید، لیبل تمام این وویس ها برابر با صفر در نظر گرفته شده است.

```
background_df = pd.DataFrame(all_background)
background_df = background_df.rename(columns={0: "Audio"})
background_df['label'] = 0
background_df = background_df.sample(len(alexa_df), random_state=158, ignore_index=True)
background_df.head()
```

شکل ۳۶ - نمونه برداری برای ساخت دیتابریم

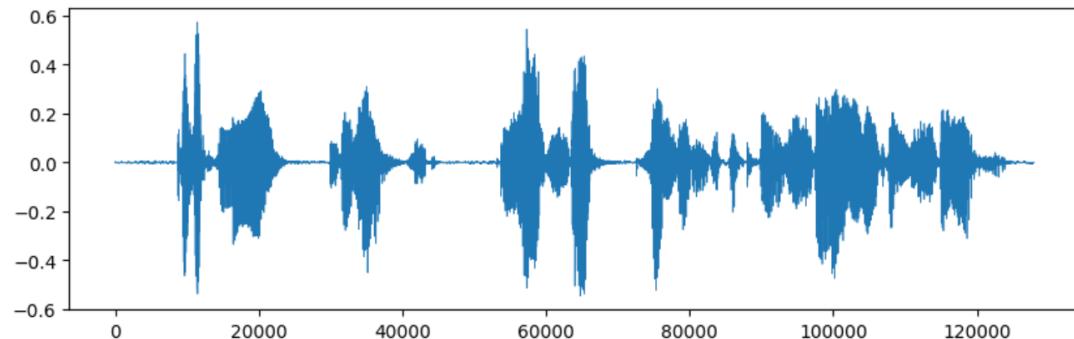
همچنین در ادامه این دیتافریم ساخته شده را مشاهده میکنید:

	Audio	label
0	nonalexa/121/121726/121-121726-0000.flac	0
1	nonalexa/260/123288/260-123288-0023.flac	0
2	nonalexa/4507/16021/4507-16021-0027.flac	0
3	nonalexa/8230/279154/8230-279154-0021.flac	0
4	nonalexa/908/157963/908-157963-0025.flac	0

شکل ۳۷ - دیتافریم داده های background

همچنین در این قسمت نیز تعدادی از این صداها را بررسی کردیم و نمودار آن را نیز رسم کردیم:

nonalexa/908/31957/908-31957-0007.flac



شکل ۳۸ - بررسی وویس های داده های background

نکته مهم اینجاست که این دیتاست برخلاف Alexa شامل وویس های با طول مختلف و عموماً بیشتر از ۲ ثانیه است و لازم که پیش پردازش شود، دیتاست Alexa از نظر طول زمانی تنها دارای وویس های دارای با طول ۲ ثانیه است، نکته مهم دیگر این بود که هر دو دیتاست ها شامل نویز و صداهای مزاحم و ... نبودند و اساساً دیتاست هایی تمیزی به حساب می‌آمدند.

در ادامه دیتاست های بدست آمده را در یک فایل نهایی دیتاست قرار دادیم و آن را به دو مجموعه آموزشی و تست تقسیم کردیم، در این بخش دقت کردیم که حتماً در مجموعه آموزش و تست، تعداد برابر از لیبل ها قرار گرفته باشد.

```
dataset = pd.concat([alexa_df, background_df])
dataset.head()
```

شکل ۳۹ - ترکیب دو دیتاست

در کادر های قرمز نحوه تقسیم بندی دیتا در دو مجموعه دیده میشود:

```
train_df, test_df = train_test_split(dataset, test_size=0.1, random_state=158, shuffle=True, stratify=dataset['label'])

train_df = train_df.reset_index()
train_df['label'].value_counts()

label
0    296
1    296
Name: count, dtype: int64

test_df = test_df.reset_index()
test_df['label'].value_counts()

label
0    33
1    33
Name: count, dtype: int64
```

شکل ۴۰ - ایجاد مجموعه آموزش و تست

پیش پردازش فایل های صوتی:

پیش پردازش داده‌های صوتی یکی از مراحل اساسی در تحلیل و پردازش سیگنال‌های صوتی است. این مرحله شامل تکنیک‌ها و روش‌های مختلفی است که به بهبود کیفیت داده‌ها و استخراج ویژگی‌های مهم از سیگنال صوتی کمک می‌کنند. در ادامه، انواع روش‌های پیش پردازش داده‌های صوتی توضیح داده شده است:

۱. حذف نویز (Noise Reduction)

حذف نویز یکی از مراحل مهم پیش پردازش است که برای کاهش نویزهای غیرمطلوب از سیگنال صوتی به کار می‌رود. روش‌های مختلفی برای حذف نویز وجود دارد:

- **فیلترهای پایین‌گذر و بالاگذر (Low-Pass and High-Pass Filters):** این فیلترها برای حذف فرکانس‌های بالا یا پایین غیرمطلوب استفاده می‌شوند.
- **فیلترهای میان‌گذر (Band-Pass Filters):** این فیلترها تنها فرکانس‌های خاصی را عبور می‌دهند و سایر فرکانس‌ها را حذف می‌کنند.
- **فیلترهای Adaptive:** این فیلترها خود را با شرایط سیگنال و نویز تطبیق می‌دهند تا بهینه‌ترین حذف نویز را انجام دهند.

۲. تقسیم‌بندی سیگنال (Segmentation)

تقسیم‌بندی سیگنال صوتی به بخش‌های کوچکتر یکی دیگر از مراحل پیش پردازش است. این کار معمولاً برای تجزیه و تحلیل دقیق‌تر سیگنال و استخراج ویژگی‌های خاص از هر بخش انجام می‌شود.

- **تقسیم‌بندی مبتنی بر زمان (Time-Based Segmentation):** سیگنال به بخش‌های زمانی ثابت تقسیم می‌شود.
- **تقسیم‌بندی مبتنی بر رویداد (Event-Based Segmentation):** سیگنال بر اساس رویدادهای خاصی مانند شروع و پایان کلمات یا جملات تقسیم می‌شود.

۳. نرمال سازی (Normalization)

نرمال سازی سیگنال صوتی برای هم تراز کردن سطوح بلندی صدا و مقایسه بهتر سیگنال ها به کار می رود.
این مرحله شامل موارد زیر است:

- نرمال سازی دامنه (Amplitude Normalization): تنظیم دامنه سیگنال به یک بازه مشخص.
- نرمال سازی توان (Power Normalization): تنظیم توان سیگنال به یک مقدار استاندارد.

پس از بررسی دقیق فایل های صوتی بر اساس برچسب های Alexa و background مشخص شده که این فایل های صوتی مشکلی از بابت نویز ندارند، و همه آن ها با کیفیت بسیار بالایی بدون نویز ضبط شده اند، اما دو مورد مهم در این قسمت لحاظ شد:

۱ - با توجه به احتمال وجود نویز، ابتدا فایل صوتی مورد نظر را خواندیم و فرکانس های کمتر از ۲۰ دسی بل را trim کردیم، با اینکار صدای های سکوت و پس زمینه موجود حذف میشود و خیالمن را بابت نویز راحت میکند.

۲ - از آنجایی که تمامی فایل های صوتی باید ۲ ثانیه باشند، در این قسمت، تمام فایل های صوتی بیشتر از ۲ ثانیه را تنها ۲ ثانیه ابتدایی را در نظر گرفتیم و فایل های کوتاه تر نیز تا دو ثانیه pad شدند.

```
def process_audio(file_path, target_length=2.0, sample_rate=16000):

    audio, sr = librosa.load(file_path, sr=sample_rate)
    audio, _ = librosa.effects.trim(audio, top_db=20)

    target_samples = int(target_length * sample_rate)

    if len(audio) > target_samples:
        audio = audio[:target_samples]
    elif len(audio) < target_samples:
        audio = np.pad(audio, (0, target_samples - len(audio)), mode='constant')
    return audio

train_df['processed_audio'] = train_df['Audio'].apply(process_audio)
test_df['processed_audio'] = test_df['Audio'].apply(process_audio)
```

شکل ۴۱ - پیش پردازش فایل های صوتی

استخراج ویژگی (Feature Extraction)

استخراج ویژگی شامل شناسایی و استخراج ویژگی‌های مهم از سیگنال صوتی است که می‌تواند به طبقه‌بندی و تحلیل سیگنال کمک کند.

- ضرایب کپسترال (MFCC) (Mel-Frequency Cepstral Coefficients): ویژگی‌هایی مانند Cepstral Coefficients که نمایانگر ویژگی‌های فرکانسی سیگنال هستند.
- ویژگی‌های زمانی (Temporal Features): (Pitch) مانند انرژی، توان، و زیروبمی.
- ویژگی‌های آماری (Statistical Features): (Skewness) مانند میانگین، واریانس، و چولگی.

۲. تبدیل‌های حوزه فرکانس (Frequency Domain Transformations)

تبدیل‌های حوزه فرکانس برای تحلیل سیگنال در حوزه فرکانس به جای زمان استفاده می‌شوند. این تبدیل‌ها شامل موارد زیر هستند:

- تبدیل فوریه (Fourier Transform): تبدیل سیگنال از حوزه زمان به حوزه فرکانس.
- تبدیل فوریه کوتاه‌مدت (Short-Time Fourier Transform - STFT): تحلیل فرکانسی سیگنال در بازه‌های زمانی کوتاه.
- تبدیل (Wavelet Transform): تحلیل چندمقیاسی سیگنال که می‌تواند ویژگی‌های محلی و جهانی سیگنال را بررسی کند.

۳. اسپکتروگرام (Spectrogram)

اسپکتروگرام یک نمایش بصری از طیف فرکانسی سیگنال صوتی در طول زمان است. این نمایش اطلاعات فرکانسی را به صورت گرافیکی نشان می‌دهد که در تحلیل و پردازش سیگنال‌های صوتی بسیار مفید است.

مراحل تولید اسپکتروگرام:

۱. تقسیم‌بندی سیگنال به پنجره‌های زمانی (**Windowing**): سیگنال صوتی به بخش‌های کوچکتری به نام "پنجره" تقسیم می‌شود. این پنجره‌ها معمولاً همپوشانی دارند تا از دست رفتن اطلاعات جلوگیری شود.
۲. تبدیل فوریه کوتاه‌مدت (**Short-Time Fourier Transform - STFT**): برای هر پنجره، تبدیل فوریه کوتاه‌مدت انجام می‌شود تا فرکانس‌های موجود در آن پنجره مشخص شوند. به ما این امکان را می‌دهد که تحلیل فرکانسی سیگنال را در بازه‌های زمانی مختلف، انجام دهیم.
۳. نمایش بصری (**Visualization**): نتایج STFT به صورت یک ماتریس دو بعدی نمایش داده می‌شوند که محور افقی نشان‌دهنده زمان، محور عمودی نشان‌دهنده فرکانس و شدت رنگ‌ها یا شدت روشنایی نشان‌دهنده دامنه (Amplitude) سیگنال در هر فرکانس است.

مل اسپکتروگرام (**Mel Spectrogram**)

مل اسپکتروگرام یک نوع خاص از اسپکتروگرام است که فرکانس‌های سیگنال صوتی را به مقیاس مل (Mel Scale) تبدیل می‌کند. مقیاس مل یک مقیاس فرکانسی لگاریتمی است که بر اساس نحوه درک انسان از صدا طراحی شده است. این مقیاس برای تحلیل گفتار و شناسایی گفتار به کار می‌رود، زیرا با حساسیت‌های شنوایی انسان هماهنگی بیشتری دارد.

مراحل تولید مل اسپکتروگرام:

۱. تبدیل اسپکتروگرام (**Spectrogram Transformation**): ابتدا اسپکتروگرام معمولی از سیگنال تولید می‌شود، مشابه آنچه در بالا توضیح داده شد.
۲. فیلتر (**Mel Filter Bank**): برای تبدیل اسپکتروگرام به مل اسپکتروگرام، از یک مجموعه‌ای از فیلترهای مثلثی استفاده می‌شود که به طور یکنواخت بر روی مقیاس مل توزیع شده‌اند. این فیلترها فرکانس‌های خطی اسپکتروگرام را به فرکانس‌های مل تبدیل می‌کنند.
۳. محاسبه توان (**Power Calculation**): برای هر فیلتر مل، توان سیگنال در آن محدوده فرکانسی محاسبه می‌شود.

۴. محاسبه لگاریتم (Logarithmic Calculation): توان محاسبه شده برای هر فیلتر به صورت لگاریتمی تبدیل می شود تا با نحوه درک انسان از بلندی صدا تطابق داشته باشد.

مزایای مل اسپکتروگرام

- هماهنگی با شنواهی انسان: مقیاس مل به گونه ای طراحی شده است که با نحوه درک انسان از فرکانس ها همخوانی دارد.
- کاهش پیچیدگی: تبدیل فرکانس ها به مقیاس مل، پیچیدگی محاسبات را کاهش می دهد و ویژگی های مهم تر سیگنال را برجسته تر می کند.
- استفاده در تحلیل گفتار: مل اسپکتروگرام به طور گسترده ای در پردازش گفتار و شناسایی گفتار استفاده می شود زیرا ویژگی های گفتاری را بهتر نمایش می دهد.

در ادامه به مقایسه انواع ویژگی های مختلف که در موضوعی مفید و در موضوعی دیگر مضر هستند می پردازیم:

۱. ویژگی :MFCC (Mel-Frequency Cepstral Coefficients)

مفید در تشخیص گفتار:

- تشخیص گفتار (Speech Recognition): MFCC یکی از ویژگی های اصلی مورد استفاده در تشخیص گفتار است. این ویژگی ها فرکانس های سیگنال را به مقیاس مل تبدیل کرده و به مدل های یادگیری ماشین کمک می کنند تا ویژگی های گفتاری را به خوبی شناسایی کنند.

مضر در شناسایی گفتار گوینده:

- شناسایی گفتار گوینده (Speaker Identification) : MFCC ممکن است اطلاعات خاصی از فرد گوینده مانند زیروبمی و تمبر صدا را حذف کند، که برای شناسایی گوینده ضروری هستند. بنابراین، استفاده از MFCC ممکن است دقت سیستم شناسایی گوینده را کاهش دهد.

۲. زیروبمی (Pitch)

مفید در شناسایی گفتار گوینده:

- شناسایی گوینده (Speaker Identification): زیروبمی یکی از ویژگی‌های متمایز هر فرد است. استفاده از زیروبمی به سیستم شناسایی گوینده کمک می‌کند تا گویندگان مختلف را بر اساس ویژگی‌های صوتی منحصر به فردشان تشخیص دهد.

مضر در تشخیص گفتار:

- تشخیص گفتار (Speech Recognition): زیروبمی می‌تواند بین گویندگان مختلف تغییر کند و باعث ایجاد تنوع غیرضروری در سیگنال شود. این تنوع ممکن است باعث کاهش دقت سیستم‌های تشخیص گفتار شود که به دنبال شناسایی کلمات و جملات هستند، نه ویژگی‌های فردی گوینده.

۳. ویژگی‌های زمانی (Temporal Features)

مفید در تحلیل موسیقی:

- تحلیل موسیقی (Music Analysis): ویژگی‌های زمانی مانند مدت زمان نت‌ها و فاصله‌های زمانی بین نت‌ها برای تحلیل ساختار و ریتم موسیقی بسیار مفید هستند.

مضر در تشخیص گفتار:

- تشخیص گفتار (Speech Recognition): برخی ویژگی‌های زمانی ممکن است برای تشخیص کلمات و جملات خاص بی‌اهمیت یا حتی مزاحم باشند. برای مثال، تغییرات کوچک در مدت زمان تلفظ کلمات می‌تواند مدل‌های تشخیص گفتار را گیج کند.

ما در این تمرین دو نوع شبکه آموزش دادیم، یک شبکه CNN که با استفاده از mel spectrogram کار میکند، چراکه تمام وویس های ما دو ثانیه هستند، شبکه های CNN برای اینکار بسیار عالی هستند، عملاً یک تصویر از صوت میسازد که بر اساس آن میتوان، با صوت دقیقاً مانند تصویر برخورد کرد، در ادامه به استخراج Mel Spectrogram از داده های صوتی میپردازیم:

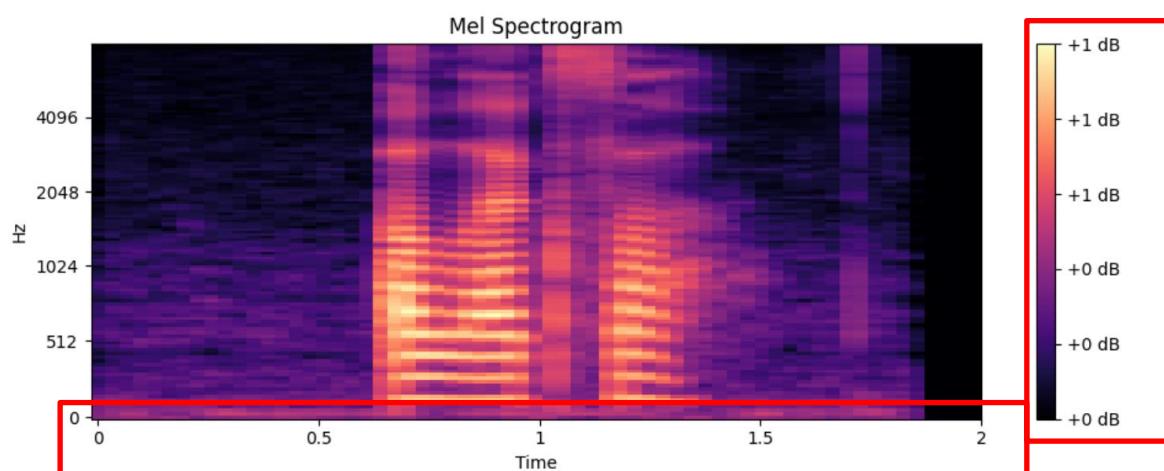
۱- استخراج Mel spectrogram

با تابع زیر در ابتدا Mel spectrogram را از هر یک از فایل های صوتی استخراج کرده، سپس آن را به دسی بل تبدیل میکنیم و در نهایت آن را بین ۰ و ۱ نرمال میکنیم، این نرمال سازی به منظور آموزش بهتر و سریعتر مدل صورت میگیرد.

```
def generate_spectrogram(audio, sample_rate=16000, n_fft=2048, hop_length=512):
    S = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_fft=n_fft, hop_length=hop_length)
    S_dB = librosa.power_to_db(S, ref=np.max)
    S_dB_normalized = (S_dB - S_dB.min()) / (S_dB.max() - S_dB.min())
    return S_dB_normalized
```

شکل ۴۲ - استخراج Mel Spectrogram

به عنوان نمونه یکی از Mel های خروجی را نمایش داده ایم:



شکل ۴۳ - نمایش یک Spectrogram

در کادرهای قرمز میبینیم که هم فرکانس در حوزه دسی بل بین صفر و یک نرمال شده است و هم طول فایل صوتی حداقل دو ثانیه است.

سپس با اعمال این تابع بر فایل های صوتی pre process شده، توانستیم بردارهای ویژگی لازم برای آموزش مدل را بسازیم

```
train_features = train_features[..., np.newaxis]
test_features = test_features[..., np.newaxis]
print(train_features.shape)

(592, 128, 63, 1)
```

شکل ۴۴ - بردار ویژگی لازم برای مدل CNN بدون داده افزایی

و در نهایت با کنار هم قرار دادن بردار ویژگی و لیبل ها، دیتابست های لازم ساخته شد

```
train_dataset = tf.data.Dataset.from_tensor_slices((train_features, train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices((test_features, test_labels))

batch_size = 16
train_dataset = train_dataset.shuffle(buffer_size=200).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

شکل ۴۵ - دیتابست مدل CNN

لازم به ذکر است که من یکبار مدل CNN را بدون Augmentation و یکبار با آموزش دادم، که در حالت که داده افزایی شده باشد تعداد داده های آموزشی عملاً دو برابر مقدار بالاست.

```
combined_features.shape, combined_labels.shape

((1184, 128, 63, 1), (1184,))
```

شکل ۴۶ - بردار آموزش مدل CNN با داده افزایی

۲- استخراج MFCC

علاوه بر مدل CNN، ما یک مدل MLP بر اساس میانگین ویژگی های MFCC نیز آموزش دادیم، ویژگی های آن نیز به شکل زیر استخراج شد:

با تابع زیر ابتدا ۱۳ ویژگی مختلف MFCC را دریافت کرده و از هر یک از آن میانگین میگیریم:

```
def extract_mfcc(audio, sr=16000, n_mfcc=13):
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
    mfcc_avg = np.mean(mfcc, axis=1)
    return mfcc_avg
```

```
train_df['mfcc_avg'] = train_df['augmented_audio'].apply(lambda x: extract_mfcc(x, sr=16000))
test_df['mfcc_avg'] = test_df['processed_audio'].apply(lambda x: extract_mfcc(x, sr=16000))
```

شکل ۴۷ - دریافت MFCC

در ادامه از این ویژگی های به عنوان ورودی X و لیبل های مدل به عنوان y استفاده کردیم، ضمنا مقادیر آن ها را نیز بین ۱ و ۱ نرمال کردیم:

```
x_train = np.array(train_df['mfcc_avg'].tolist())
y_train = np.array(train_df['label'].tolist())
```

```
x_test = np.array(test_df['mfcc_avg'].tolist())
y_test = np.array(test_df['label'].tolist())
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

شکل ۴۸ - ایجاد بردارهای ویژگی و برچسب و نرمال سازی آن ها

و در نهایت دیتابست مورد نظر برای آموزش مدل بر اساس ویژگی های MFCC را ساختیم، قابل ذکر است که از ویژگی که یک بعدی است تنها در مدل MLP استفاده کردیم.

```
batch_size = 16
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(200).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

شکل ۴۹ - دیتابست مدل MLP

افزایش داده‌ها (Data Augmentation)

در ادامه روش‌های داده افزایی میپردازیم و پیاده سازی آن‌ها را نیز در همین قسمت ذکر میکنیم:

۱. تغییر زمان (Time Stretch)

توضیحات:

- تغییر زمان به تغییر سرعت پخش سیگنال صوتی بدون تغییر در زیروبمی (pitch) گفته می‌شود.
- این تکنیک به مدل اجازه می‌دهد تا با تغییرات مختلف سرعت گفتار یا موسیقی آشنا شود و عملکرد بهتری داشته باشد.

مزایا:

- به مدل کمک می‌کند تا با سیگنال‌های صوتی با سرعت‌های مختلف سازگار شود.
- می‌تواند به بهبود تعمیم‌دهی مدل کمک کند.

پیاده سازی:

```
def time_stretch(audio, rate=1.0):
    return librosa.effects.time_stretch(audio, rate=rate)
```

شکل ۵۰ - پیاده سازی **time_stretch**

۲. تغییر زیروبمی (Pitch Shift)

توضیحات:

- تغییر زیروبمی به تغییر زیروبمی سیگنال صوتی بدون تغییر در طول مدت زمان آن اشاره دارد.
- این تکنیک به مدل کمک می‌کند تا با تن‌های مختلف صدا آشنا شود.

مزایا:

- بهبود تعمیمدهی مدل در مواجهه با صدای زیروبمی‌های مختلف.
- می‌تواند برای شبیه‌سازی تفاوت‌های صوتی بین گویندگان مختلف مفید باشد.

پیاده سازی:

```
def pitch_shift(audio, sr=16000, n_steps=0):
    return librosa.effects.pitch_shift(audio, sr=sr, n_steps=n_steps)
```

شکل ۵۱ - پیاده سازی **pitch_shift**

۳. اضافه کردن نویز (Add Noise)

توضیحات:

- اضافه کردن نویز شامل افزودن نویزهای پس زمینه به سیگنال صوتی اصلی است.
- این تکنیک به مدل کمک می‌کند تا مقاومت بیشتری در برابر نویزهای محیطی داشته باشد.

مزایا:

- بهبود مقاومت مدل در برابر نویز و افزایش دقت در شرایط نویزی.
- کمک به تعمیمدهی بهتر مدل در محیط‌های مختلف.

پیاده سازی:

```
def add_noise(audio, noise_factor=0.005):
    noise = np.random.randn(len(audio))
    augmented_audio = audio + noise_factor * noise
    return augmented_audio
```

شکل ۵۲ - پیاده سازی افزودن نویز

۴. جابجایی صوت (Shift Audio)

توضیحات:

- جابجایی صوت شامل جابجا کردن سیگنال صوتی در طول زمان است.
- این تکنیک به مدل کمک می کند تا با تغییرات زمانی در سیگنال صوتی سازگار شود.

مزایا:

- بهبود تعمیمدهی مدل در مواجهه با تغییرات زمانی.
- کمک به مدل برای شناسایی بهتر الگوهای صوتی.

پیاده سازی:

```
def shift_audio(audio, shift_max, shift_direction='both'):
    shift = np.random.randint(shift_max)
    if shift_direction == 'right':
        shift = -shift
    elif shift_direction == 'both':
        direction = np.random.choice(['left', 'right'])
        if direction == 'right':
            shift = -shift
    augmented_audio = np.roll(audio, shift)
    return augmented_audio
```

شکل ۵۳ - پیاده سازی shift Audio

۵. تغییر حجم (Change Volume)

توضیحات:

- تغییر حجم شامل تنظیم سطح بلندی صدای سیگنال صوتی است.
- این تکنیک به مدل کمک می کند تا با تغییرات مختلف در سطح بلندی صدا آشنا شود.

مزایا:

- بهبود تعمیمدهی مدل در مواجهه با تغییرات سطح بلندی صدا.
- کمک به مدل برای شناسایی بهتر سیگنالهای صوتی با حجم‌های مختلف.

پیاده سازی:

```
def change_volume(audio, volume_factor=1.0):
    return audio * volume_factor
```

شکل ۵۴ - پیاده سازی تغییر صدا

در نهایت از طریق یک تابع به نام `apply_augmentation` این موارد را به هر فایل صوتی اعمال کردیم، به طوری که بعد از مشخص شدن فایل صوتی، به طور رندوم تنها یکی از موارد انتخاب شده و سپس آن تابع خاص `Augmentation` به فایل صوتی مورد نظر اعمال می‌شود، همچنین با توجه به اینکه ممکن است، طول صوت بعد از `Augmentation` از نظر زمانی تغییر کند، در نتیجه مجدداً طول صوت با اندازه ۲ ثانیه سنجیده شده و اگر کمتر شده باشد `pad` شده و اگر بیشتر باشد، تنها ۲ ثانیه اول آن مورد استفاده قرار می‌گیرد.

```
def apply_augmentation(audio, sr, duration=2.0):
    augmentations = [
        (time_stretch, {'rate': random.uniform(0.8, 1.2)}),
        (pitch_shift, {'sr': sr, 'n_steps': random.randint(-2, 2)}),
        (add_noise, {'noise_factor': random.uniform(0.001, 0.005)}),
        (shift_audio, {'shift_max': sr // 10, 'shift_direction': 'both'}),
        (change_volume, {'volume_factor': random.uniform(0.7, 1.3)})
    ]

    augmentation, params = random.choice(augmentations)
    augmented_audio = augmentation(audio, **params)

    if len(augmented_audio) < sr * duration:
        augmented_audio = np.pad(augmented_audio, (0, max(0, int(sr * duration) - len(augmented_audio))), 'constant')
    else:
        augmented_audio = augmented_audio[:int(sr * duration)]

    return augmented_audio
```

شکل ۵۵ - نحوه اعمال داده افزایی به هر فایل صوتی

۳-۳: طراحی شبکه عصبی:

سوال اول: با توجه به اینکه تمامی صوت ها طول ثابتی دارند، بهترین شبکه برای اینکار شبکه‌ی CNN است، با دریافت ویژگی Mel Spectrogram و ایجاد یک بازنمایی تصویری از بردar ویژگی، این شبکه به خوبی قادر به آموزش دیدن و طبقه‌بندی صوت های نویز نسبت به کلمه بیدار باش! هستند، ویژگی CNN ها عبارتند از:

پردازش ویژگی‌های محلی:

- CNN‌ها برای تشخیص الگوهای محلی در داده‌ها بسیار مناسب هستند. این ویژگی باعث می‌شود که CNN‌ها بتوانند الگوهای زمانی-فرکانسی موجود در اسپکتروگرام‌های صوتی را به خوبی شناسایی کنند.

کاهش پیچیدگی محاسباتی:

- شبکه‌های CNN با استفاده از فیلترهای کانولوشنی و لایه‌های پائین‌بر (Pooling) می‌توانند حجم داده‌های ورودی را کاهش دهند و به این ترتیب پردازش داده‌ها را سریع‌تر و کارآمدتر کنند.

ثبتات در برابر تغییرات کوچک:

- CNN‌ها به دلیل ساختار خود نسبت به تغییرات کوچک در داده‌ها (مانند نویز یا تغییرات جزئی در گفتار) مقاوم هستند. این ویژگی به مدل کمک می‌کند تا کلمات بیدارباش را با دقت بالاتری شناسایی کند.

سوال دوم: در صورتی که طول صوت‌ها متفاوت باشد و بخواهیم فریم به فریم جدا کنیم و به مدل بدھیم، بهترین انتخاب استفاده از شبکه‌های بازگشتی (Recurrent Neural Networks - RNNs) است. در میان انواع RNN‌ها، شبکه‌های GRU (Gated Recurrent Unit) و LSTM (Long Short-Term Memory) به دلیل توانایی‌شان در مدیریت وابستگی‌های طولانی مدت و به یاد سپردن اطلاعات در توالی‌های طولانی، گزینه‌های مناسبی هستند.

ویژگی RNN ها و شبکه های وابسته (LSTM, GRU) عبارتند از:

۱. مدیریت توالی های زمانی با طول متغیر:

- RNN ها قادر به پردازش توالی هایی با طول های مختلف هستند، زیرا هر ورودی به صورت گام به گام به مدل تغذیه می شود. این ویژگی به مدل اجازه می دهد تا طول های مختلف سیگنال های صوتی را بدون نیاز به برش یا پد کردن داده ها پردازش کند.

۲. حفظ وابستگی های زمانی:

- LSTM و GRU ها می توانند اطلاعات مهم را در طول توالی های زمانی بلندمدت حفظ کنند و وابستگی های زمانی را بهتر از RNN های ساده مدیریت کنند. این ویژگی برای کاربردهایی مانند تشخیص کلمات بیدار باش که نیازمند درک وابستگی های زمانی هستند، بسیار مفید است.

علاوه بر توضیحات بالا من قبل از شبکه های fully connected برای تشخیص جنسیت گوینده بر اساس ویژگی MFCC استفاده کرده بودم، به همین علت علاوه بر شبکه CNN بر اساس اسپکتروگرام، از یک شبکه MFCC هم استفاده کردیم، این موضوع و مقایسه آن با CNN به عنوان یک Baseline صرفا برای خودم جالب بود که نتایج را در ادامه می آورم.

۱ - شبکه CNN

من شبکه CNN را یکبار بدون داده افزایی و یکبار با داده افزایی آموزش دادم:

- شبکه CNN مورد استفاده:

شبکه CNN بسیار ساده برای اینکار انتخاب شد که ساختار آن را در ادامه می بینیم:

```
def CNN_spectrogram(input_shape, num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
```

شکل ۵۶ - شبکه CNN مورد استفاده

سپس به آموزش این شبکه بر اساس هایپرپارامتر های زیر پرداختیم:

```
input_shape = train_features[0].shape
num_classes = 2

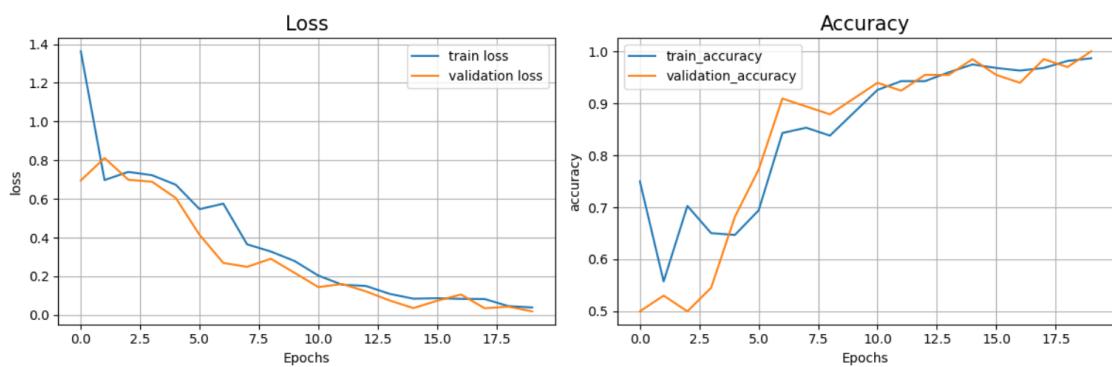
cnn_model = CNN_spectrogram(input_shape, num_classes)

cnn_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = cnn_model.fit(train_dataset, epochs=20,
                        validation_data=test_dataset)
```

شکل ۵۷ - هایپرپارامتر های شبکه CNN

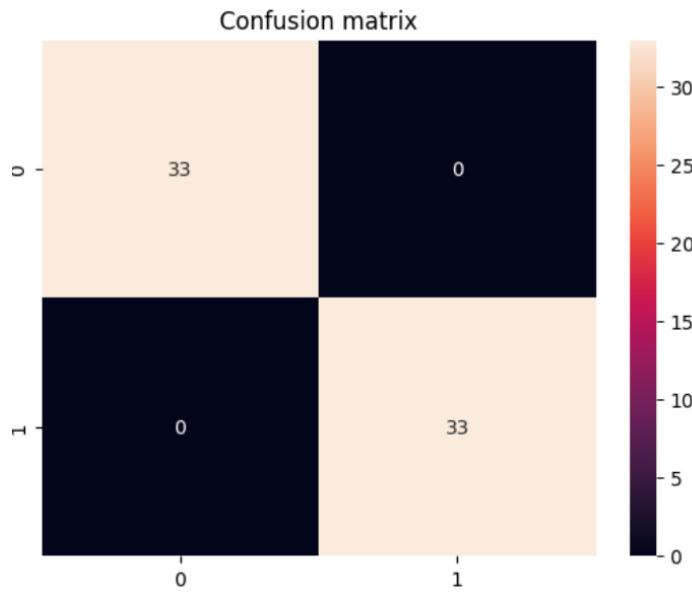
- نتایج شبکه CNN بدون داده افزایی:



شکل ۵۸ - نمودار های مدل CNN بدون داده افزایی

5/5		0s 46ms/step			
		precision	recall	f1-score	support
0		1.00	1.00	1.00	33
1		1.00	1.00	1.00	33
	accuracy			1.00	66
	macro avg	1.00	1.00	1.00	66
	weighted avg	1.00	1.00	1.00	66

شکل ۵۹ - گزارش CNN مدل classification بدون داده افزایی

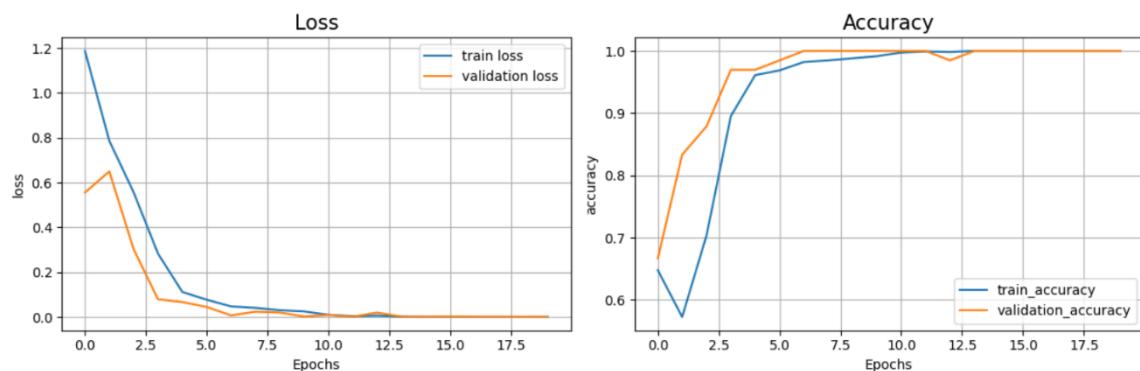


شکل ۶۰ - ماتریس درهم ریختگی مدل **CNN** بدون داده افزایی

آنچه بسیار جالب است این است که مدل بدون هیچ داده افزایی موفق شده در زمانی بسیار سریع، این طبقه بنده را انجام دهد، حتی جالب است من نیازی به استفاده از GPU هم برای آموزش مدل نداشتم.

- نتایج شبکه **CNN** با داده افزایی:

لازم به ذکر است که من در این بخش شبکه و هایپرپارامتر ها را برای قابل مقایسه بودن نتایج تغییری ندادم.



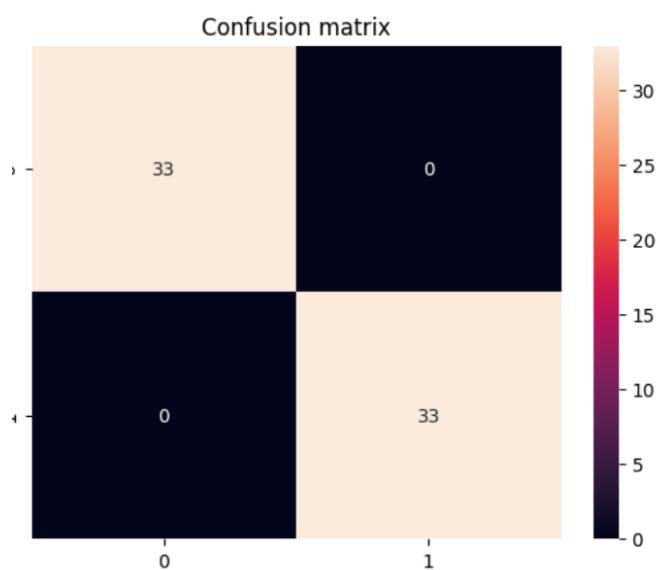
شکل ۶۱ - نمودار های مدل **CNN** با داده افزایی

در این بخش میبینیم که نمودارهای یادگیری بسیار نرم تر و سریعتر همگرا شده اند به طوری که ۱۰ ایپاک برای رسیدن به درصد ۱۰۰ کافی است، این بسیار جالب است زیرا عملاً داده ها بیشتر شده اند اما مدل بسیار سریع تر همگرا شده است.

5/5 ————— 0s 52ms/step

	precision	recall	f1-score	support
0	1.00	1.00	1.00	33
1	1.00	1.00	1.00	33
accuracy			1.00	66
macro avg	1.00	1.00	1.00	66
weighted avg	1.00	1.00	1.00	66

شکل ۶۲ - گزارش CNN مدل classification با داده افزایی



شکل ۶۳ - ماتریس درهم ریختنی مدل CNN با داده افزایی

آنچه بسیار جالب است این است که مدل در زمانی بسیار سریعتر نسبت به حالت قبل، این طبقه بندی را انجام دهد، حتی جالب است من نیازی به استفاده از GPU هم برای آموزش مدل نداشتم.

- شبکه و نتایج شبکه MLP با داده افزایی:

شبکه مورد استفاده:

من در این بخش از یک شبکه MLP بسیار ساده و سبک استفاده کردم:

```
mlp_model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2, activation='softmax')
])
```

شکل ۶۴ - شبکه MLP مورد استفاده

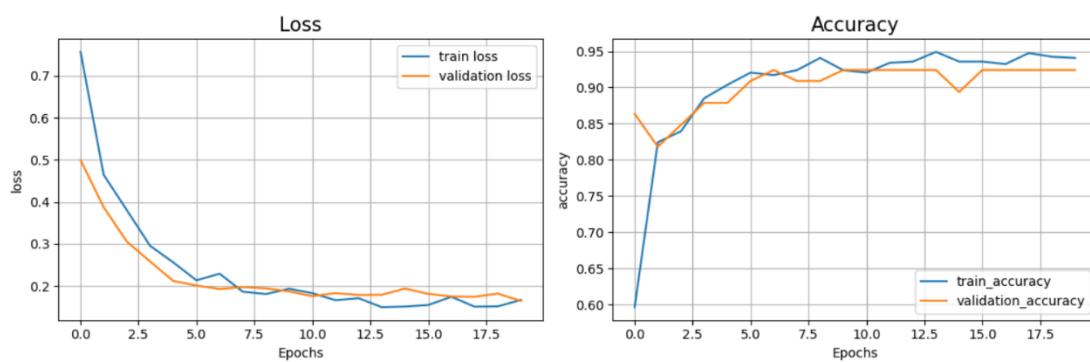
لازم به ذکر است که من در این بخش هایپرپارامتر ها را برای قابل مقایسه بودن نتایج دو نوع شبکه تغییری ندادم.

```
epochs = 20
mlp_model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

mlp_history = mlp_model.fit(train_dataset, epochs=epochs, validation_data=test_dataset)
```

شکل ۶۵ - هایپرپارامتر های شبکه MLP

نتایج شبکه MLP

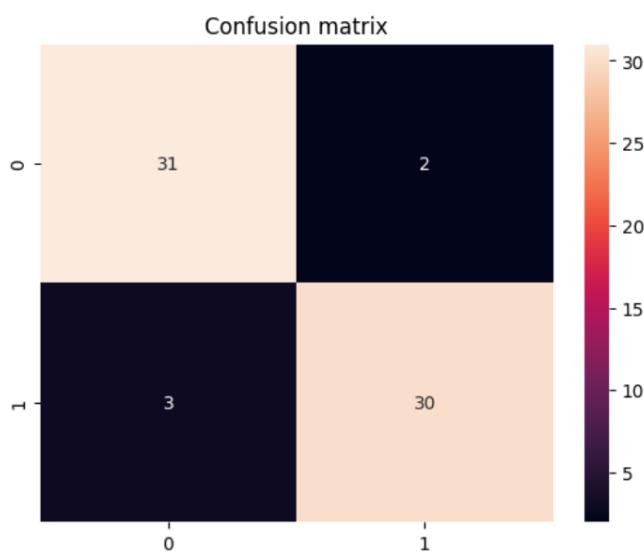


شکل ۶۶ - نمودار های مدل MLP با داده افزایی

اگر چه مدل به خوبی یادگرفته است اما میبینیم که بر خلاف شبکه CNN به دقت صد درصد دست نیافته است، از طرفی آموزش را انقدری ادامه داده ایم که مطمئن شویم، شبکه نیازی به زمان بیشتر برای یادگیری نداشته باشد، از سوی دیگر مشخص است که شبکه دچار overfitting نشده زیرا مجموعه ارزیابی و آموزش به هزینه و دقت قابل مقایسه ای در نهایت دست یافته اند.

	precision	recall	f1-score	support
0	0.91	0.94	0.93	33
1	0.94	0.91	0.92	33
accuracy			0.92	66
macro avg	0.92	0.92	0.92	66
weighted avg	0.92	0.92	0.92	66

شکل ۶۷ - گزارش MLP مدل classification با داده افزایی



شکل ۶۸ - ماتریس درهم ریختنی مدل MLP با داده افزایی

با این مقایسه برتری مدل های CNN در این تسك بر مدل های fully connected به خوبی مشخص میشود، زیرا میبینیم که دقیق‌ترین شبکه حدود ۹۲٪ درصد است که تفاوت نسبتاً جدی با ۱۰۰ درصد در مدل های CNN دارد، به همین علت مشخص میشود که شبکه CNN بهترین انتخاب برای این موضوع هستند.

مقایسه نهایی سه شبکه:

	CNN	CNN Data Augmentation	MLP Data Augmentation
Accuracy	۱۰۰٪	۱۰۰٪	۹۲٪
F1_Score	۱۰۰٪	۱۰۰٪	۹۲٪

جدول ۵ - مقایسه سه مدل

پرسش ۴ : شبکه بخش بندی تصاویر

۴-۱. دادگان:

در ابتدا دادگان مورد نظر دانلود شد، برای دسترسی بهتر به دادگان و پردازش های بعدی که به صورت on the fly انجام شوند، در ابتدا از داده های آموزش و تست، دیتا فریم هایی که آدرس تصویر و ماسک نظیر آن ها را دقیقاً نگه داری کند، ساخته شد، تابع مورد نظر برای این موضوع را در ادامه مشاهده میکنید. قابل مشاهده است که با sort کرده لیست ماسک و تصاویر، اطمینان حاصل شده که ماسک هر تصویر دقیقاً مطابق با آن تصویر باشد.

```
def dir_list(images_dir, masks_dir):  
    images_names = sorted(os.listdir(images_dir))  
    masks_names = sorted(os.listdir(masks_dir))  
  
    image_list = []  
    for name in images_names:  
        image_list.append(os.path.join(images_dir, name))  
  
    mask_list = []  
    for name in masks_names:  
        if name.endswith('.bmp'):  
            mask_list.append(os.path.join(masks_dir, name))  
  
    df = pd.DataFrame({'images':image_list,  
                      'masks':mask_list})  
  
    return df
```

شکل ۶۹- تابع ساخت دیتا فریم آدرس تصاویر و ماسک های نظیر آن ها

همچنین طبق خواسته سوال به جدا سازی ۱۰ درصد از داده برای ولیدیشن پرداختیم، همچنین لازم است که داده های آموزش و ولیدیشن به صورت متوازن انتخاب شوند که در تابع زیر ما با sample گرفتن از دیتا فریم ساخته شده سعی کرده ایم، این کار به صورت رندوم انجام شود، دلیل این توازن این است که شبکه در حین آموزش تنها پرن خاصی را یادنگیرد که قابلیت generalization نداشته باشد، بلکه با دیدن انواع حالت های مختلف بتواند، یادگیری اش را تعمیم بدهد، تابع زیر در ادامه مشخص است:

```
train_mask = '/kaggle/working/data/train_val/masks'  
train_image = '/kaggle/working/data/train_val/images'  
  
train_val_df = dir_list(train_image, train_mask)  
  
val_df = train_val_df.sample(frac=0.1, random_state=158)  
train_df = train_val_df.drop(val_df.index)  
val_df = val_df.reset_index(drop=True)  
train_df = train_df.reset_index(drop=True)
```

شکل ۷۰- تابع جدا سازی ۱۰ درصد دیتا برای ولیدیشن

در نهایت دیتا فریم ساخته شده قابل مشاهده هستند، همچنین تعداد دیتا در هر یک از مجموعه های آموزش و ولیدیشن و تست نیز در کادر قرمز مشخص شده است، همچنین اسمی یکسان تصاویر و ماسک ها نیز مشخص شده اند.

```
print(len(train_df))
train_df.head()
```

1373

	images	masks
0	/kaggle/working/data/train_val/images/d_r_103...	/kaggle/working/data/train_val/masks/d_r_103_.bmp
1	/kaggle/working/data/train_val/images/d_r_105...	/kaggle/working/data/train_val/masks/d_r_105_.bmp
2	/kaggle/working/data/train_val/images/d_r_112...	/kaggle/working/data/train_val/masks/d_r_112_.bmp
3	/kaggle/working/data/train_val/images/d_r_113...	/kaggle/working/data/train_val/masks/d_r_113_.bmp
4	/kaggle/working/data/train_val/images/d_r_125...	/kaggle/working/data/train_val/masks/d_r_125_.bmp

شکل ۷۱ - دیتابس آموزش شامل ۱۳۷۳ تصویر

```
print(len(val_df))
val_df.head()
```

152

	images	masks
0	/kaggle/working/data/train_val/images/f_r_1875...	/kaggle/working/data/train_val/masks/f_r_1875...
1	/kaggle/working/data/train_val/images/w_r_64_.jpg	/kaggle/working/data/train_val/masks/w_r_64_.bmp
2	/kaggle/working/data/train_val/images/f_r_2011...	/kaggle/working/data/train_val/masks/f_r_2011...
3	/kaggle/working/data/train_val/images/f_r_319...	/kaggle/working/data/train_val/masks/f_r_319_.bmp
4	/kaggle/working/data/train_val/images/f_r_2041...	/kaggle/working/data/train_val/masks/f_r_2041...

شکل ۷۲ - دیتابس ارزیابی شامل ۱۵۲ تصویر

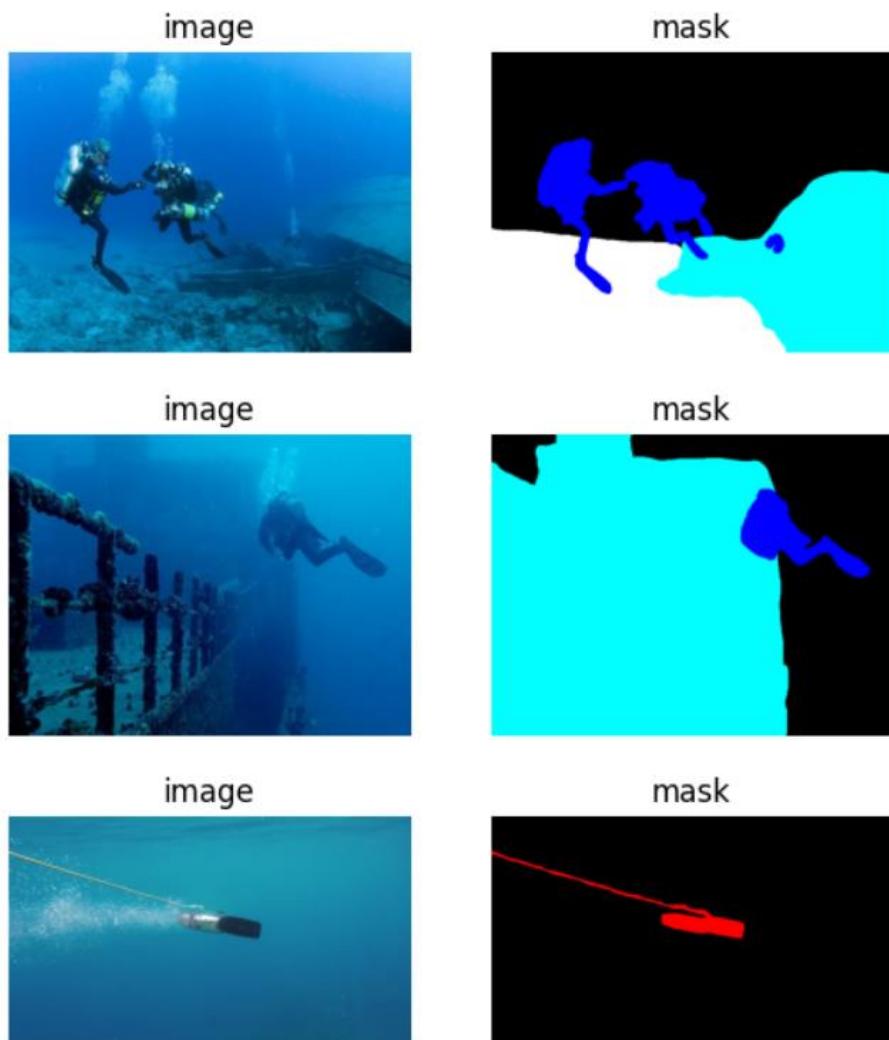
```
print(len(test_df))
test_df.head()
```

110

	images	masks
0	/kaggle/working/data/TEST/images/d_r_122.jpg	/kaggle/working/data/TEST/masks/d_r_122_.bmp
1	/kaggle/working/data/TEST/images/d_r_129.jpg	/kaggle/working/data/TEST/masks/d_r_129_.bmp
2	/kaggle/working/data/TEST/images/d_r_144.jpg	/kaggle/working/data/TEST/masks/d_r_144_.bmp
3	/kaggle/working/data/TEST/images/d_r_166.jpg	/kaggle/working/data/TEST/masks/d_r_166_.bmp
4	/kaggle/working/data/TEST/images/d_r_182.jpg	/kaggle/working/data/TEST/masks/d_r_182_.bmp

شکل ۷۳ - دیتابس تست شامل ۱۱۰ تصویر

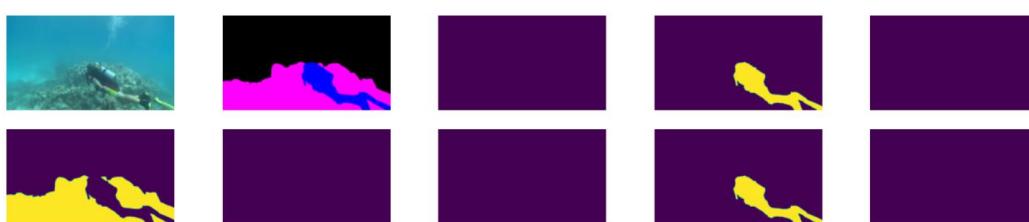
در نهایت نیز چند تصویر از میان داده ها را نمایش دادیم:



شکل ۷۴ - تعدادی تصویر و ماسک مرتبط به آن

قابل توجه است که داده‌ی تست، علاوه بر ماسک ترکیبی، به ازای هر کلاس نیز ماسک مجزایی ارائه میدهد که آن را نیز در یک دیتا فریم مجزا ذخیره کرده و بررسی کردیم:

	images	masks	RO	Saliency
0	/kaggle/working/data/TEST/images/d_r_122.jpg	/kaggle/working/data/TEST/masks/d_r_122.bmp	/kaggle/working/data/TEST/masks/RO/d_r_122.bmp	/kaggle/working/data/TEST/masks/Saliency/d_r_1...
1	/kaggle/working/data/TEST/images/d_r_129.jpg	/kaggle/working/data/TEST/masks/d_r_129.bmp	/kaggle/working/data/TEST/masks/RO/d_r_129.bmp	/kaggle/working/data/TEST/masks/Saliency/d_r_1...
2	/kaggle/working/data/TEST/images/d_r_144.jpg	/kaggle/working/data/TEST/masks/d_r_144.bmp	/kaggle/working/data/TEST/masks/RO/d_r_144.bmp	/kaggle/working/data/TEST/masks/Saliency/d_r_1...
3	/kaggle/working/data/TEST/images/d_r_166.jpg	/kaggle/working/data/TEST/masks/d_r_166.bmp	/kaggle/working/data/TEST/masks/RO/d_r_166.bmp	/kaggle/working/data/TEST/masks/Saliency/d_r_1...
4	/kaggle/working/data/TEST/images/d_r_182.jpg	/kaggle/working/data/TEST/masks/d_r_182.bmp	/kaggle/working/data/TEST/masks/RO/d_r_182.bmp	/kaggle/working/data/TEST/masks/Saliency/d_r_1...



شکل ۷۵ - ماسک های مجزا به ازای هر کلاس در داده تست

در ادامه به نرمالسازی و augmentation داده های مورد نظر پرداختیم، اینکار از طریق تابع زیر انجام شد که هر قسمت آن را توضیح میدهیم، نکته مهم اینجاست که ما شبکه را به دو شکل پیاده سازی کردیم که هر کدام Augmentation های مخصوص خود را داشت:

۱ - پیاده سازی شبکه ها با ماسک های سیاه و سفید(دقیقا طبق مقاله ارسال شده) که در Augmentation و نرمال سازی آن را توضیح میدهیم:

```
def train_generator(data_frame, batch_size, target_size=(128,128), train = True, seed=1):
    if train:
        image_datagen = ImageDataGenerator(rotation_range=0.2, width_shift_range=0.05, height_shift_range=0.05, shear_range=0.05,
                                            zoom_range=0.05, horizontal_flip=True, fill_mode='nearest')

        mask_datagen = ImageDataGenerator(rotation_range=0.2, width_shift_range=0.05, height_shift_range=0.05, shear_range=0.05,
                                            zoom_range=0.05, horizontal_flip=True, fill_mode='nearest')

    else:
        image_datagen = ImageDataGenerator()
        mask_datagen = ImageDataGenerator()

    image_generator = image_datagen.flow_from_dataframe(
        data_frame,
        x_col = "images",
        color_mode = 'rgb',
        class_mode = None,
        target_size = target_size,
        batch_size = batch_size,
        seed = seed)

    mask_generator = mask_datagen.flow_from_dataframe(
        data_frame,
        x_col = "mask",
        color_mode = 'grayscale',
        class_mode = None,
        target_size = target_size,
        batch_size = batch_size,
        seed = seed)
```

شکل ۷۶ - بخش اول داده افزایی طبق مقاله **Suim**

در کادر اول قرمز رنگ مشاهده میشود که ما فقط Augmentation ها را برای داده های آموزش لحاظ کرده ایم، همچنین از Augmentation های مقاله اصلی دیتابست Suim که در ایمیلی توسط TA محترم ارسال شد استفاده کردیم.

The specific parameters are as follows: **rotation range of 0.2; width shift, height shift, shear, and zoom range of 0.05; horizontal flip is enabled; and the rest of the parameters are left as default.**

شکل ۷۷ - مقاله اصلی **Suim**

ما نیز همین پارامتر ها را در نظر گرفته ایم، دلیل اصلی نیز مشخص است، در اصلی این تصاویر از زیر دریا جمع آوری شده اند، در این موارد هدف ما افزایش داده هاست و ترجیح میدهیم داده افزایی را به

نوعی لحاظ کنیم که عواملی مانند تغییرات زاویه و زوم و rotation را که بیشتر به عامل انسانی عکاس بستگی دارد در بربگیرد و توزیع اصلی داده ها را به چیزی در واقعیت اتفاق نمی افتد تغییر ندهد.

در کادر های دوم و سوم مشاهده میشود که ما تصویر اصلی را به صورت rgb در نظر گرفته ایم و در عین حال ماسک را به صورت grayscale به مدل داده ایم، اینکار به این دلیل است که مقاله اصلی این سوال که مدل های UNet و TA-UNet را آموزش داده خروجی به این شکل دارد، البته برای این مورد تایید محترم نیز دریافت شد.

:Tir 1403 8:31 PM, "Fateme Mirzadeh" <fatemehmirzadeh99@gmail.com> wrote 5

فرقی نداره من کد هردو مدل رو زدم فقط برهمنبا میخواستین پیاده کنید
لطفا در گزارشتون حتما ذکر کنید لطفا.
موفق باشید

On Tue, 25 Jun 2024, 19:52 , <ali.ramezani.96@ut.ac.ir> wrote:

سپاس گرام ازتون.
فقط موردي وجود دارد، در مقاله و مدل TA-UNet اشتراک گذاشته شده در سوال، در نهایت ماسک ها به صورت سیاه و سفید(دو کلاسه و یک چنل در خروجی) بدست آمده اند، اما دیتابست و لینکی که ارسال کردید شامل 8 کلاس و با 3 چنل RGB از مدل خروجی گرفته است.
میخواستم ببینم مینا رو بر مقاله و مدل TA-UNet بگذاریم یا بر اساس دیتابست آن را تغییر دهیم؟

شكل ۷۸ - ایمیل TA محترم در باب صحیح بودن هر دو نوع خروجی بر اساس مدل

در نهایت به نرمالسازی پرداختیم و ماسک ها را بر اساس این نوع از پیاده سازی تغییر دادیم:

```
train_gen = zip(image_generator, mask_generator)

for (img, mask) in train_gen:
    img = img/255.
    mask = mask/255.
    mask[mask>=0.1]= 1,
    mask[mask<0.1]= 0,

    yield (img, mask)
```

شكل ۷۹ - نوع اول Augmentation و نرمال سازی ماسک و تصاویر

در این بخش ماسک و تصاویر بر ۲۵۵ تقسیم شده و نرمال شده اند، از طرفی با در نظر گرفتن threshold برابر ۰.۱ توانستیم تمامی object های موجود در تصویر را به صورت ماسک سفید و پس زمینه آب دریا و سطوح زیر دریا را سیاه در نظر بگیریم، این مقدار threshold بسیار مهم است و با چندین تغییر و بررسی

تصاویر به آن رسیدیم چرا که مثلا اگر threshold برابر ۰.۵ در نظر گرفته شود، بخشی از object ها مثلما ماهی ها نیز سیاه در نظر گرفته شده و پس زمینه فرض میشوند.

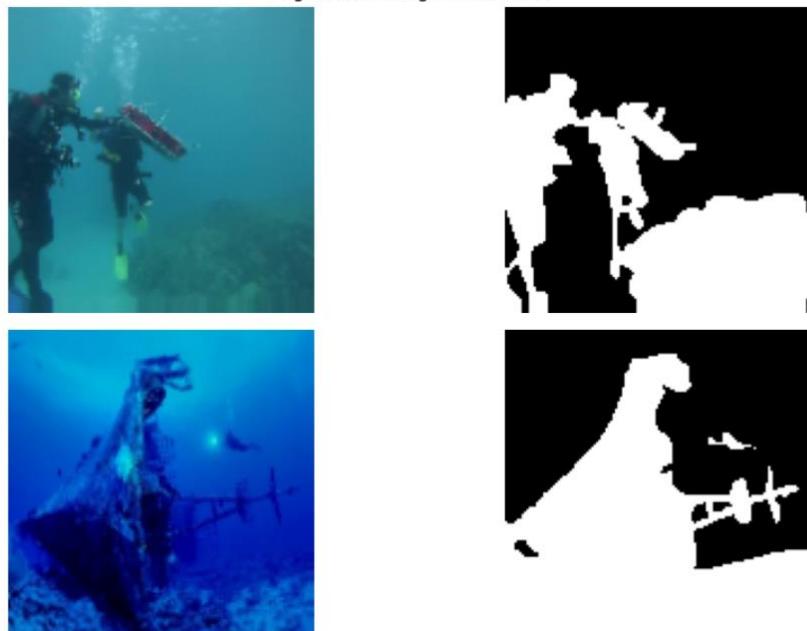
در نهایت، دیتاست های مورد نظر را ساختیم و تعدادی تصویر از دیتاست آموزش و ماسک آن ها را نیز رسم کردیم:

```
BATCH_SIZE = 4
im_height = 320
im_width = 240

train_gen = train_generator(train_df, BATCH_SIZE, target_size=(im_height,im_width), train = True, seed=1)
valid_gen = train_generator(val_df, BATCH_SIZE, target_size=(im_height,im_width), train = False, seed=1)
test_gen = train_generator(test_df, BATCH_SIZE, target_size=(im_height,im_width), train = False, seed=1)
```

شکل ۸۰ - دیتاست های ساخته شده

Augmented image with Masks



شکل ۸۱ - تصاویر آموزش و ماسک های باینری آن ها

۲ - من در این تمرین شبکه را با ماسک های رنگی آن ها نیز پیاده کردم، به همین دلیل نیاز بود که این نوع Augmentation و نرمال سازی آن در نظر گرفته شود، در این بخش داده افزایی را تغییر ندادم بلکه تنها نرمال سازی را تغییر دادم، که در تصویر زیر دیده میشود:

```
image_generator = image_datagen.flow_from_dataframe(  
    data_frame,  
    x_col = "images",  
    color_mode = 'rgb',  
    class_mode = None,  
    target_size = target_size,  
    batch_size = batch_size,  
    seed = seed)  
  
mask_generator = mask_datagen.flow_from_dataframe(  
    data_frame,  
    x_col = "masks",  
    color_mode = 'rgb',  
    class_mode = None,  
    target_size = target_size,  
    batch_size = batch_size,  
    seed = seed)  
  
train_gen = zip(image_generator, mask_generator)  
  
for (img, mask) in train_gen:  
    img = img/255.  
    mask = mask/255.  
    # mask[mask>=0.1] = 1,  
    # mask[mask<0.1] = 0,  
  
    yield (img, mask)
```

در این بخش میبینیم که ماسک ها نیز به صورت rgb در نظر گرفته شده اند و تنها با تقسیم به ۲۵۵ نرمال شده اند و نیاز به تعیین threshold نبوده است.

همچنین پرسیده شده که دلیل این نرمال سازی چیست: با تقسیم به ۲۵۵ تمامی اعداد پیکسل ها بین ۰ و ۱ قرار گرفته و این آموزش شبکه را سریعتر و robust تر میکند، از طرفی distribution shift های شدید نیز در دیتا اتفاق نمی افتد که البته برای آن از batch normalization استفاده کرده ایم.

نکته آخر اما در زمینه اندازه تصویر ورودی به شبکه است، مقاله TA-UNet اندازه تصاویر را ۶۴۰ در ۳۶۸ نمایم که دلیل این است که در نظر گرفته است و مقاله SUIM آن را ۳۶۰ در ۲۴۰، اما هر دو این مقادیر تنها با اندازه batch بسیار کوچک قابل استفاده بود (به دلیل محدودیت های حاصلی از GPU)، من اندازه هر دو مورد را با تغییر سایر های پرپارامتر ها امتحان کردم اما شبکه قادر نبود به دلیل محدودیت های حاصلی از GPU، در نتیجه مجبور به تغییر اندازه به ۱۲۸ در ۱۲۸ نمایم که این دلیل بود که بتوانم اندازه batch size را افزایش دهم تا از طریق آن شبکه را نسبت به واریانس داده های ورودی invariant کنم. با توجه به اینکه این دیتاست روی این شبکه آموزش ندیده بود، من با این تغییر توانستم آن را تا دقیقت نسبتاً مناسبی آموزش دهم.

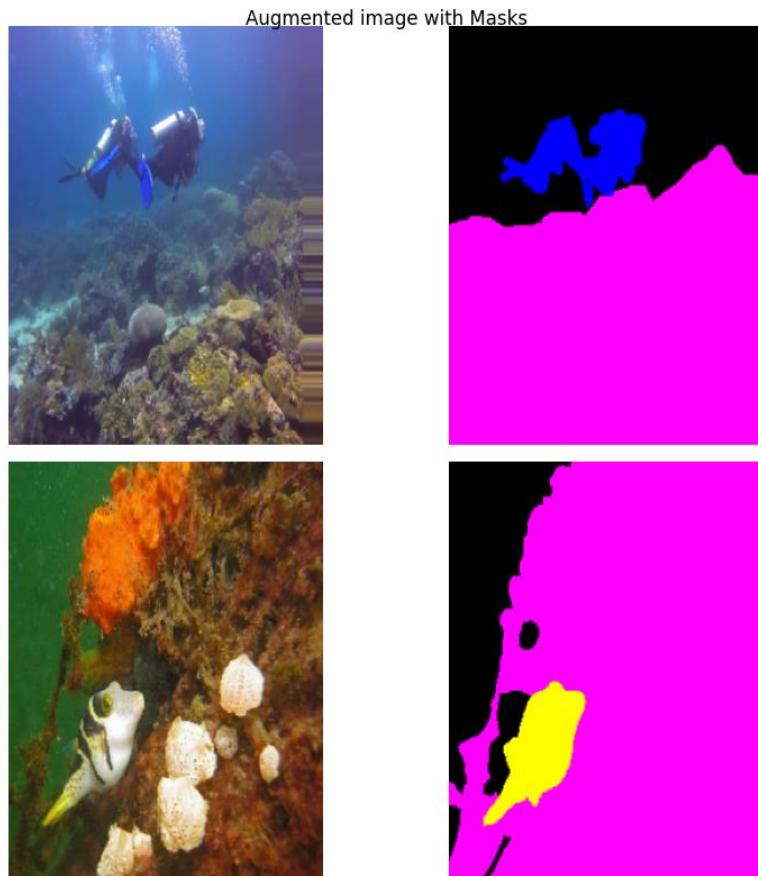
```

BATCH_SIZE = 32
im_height = 128
im_width = 128

train_gen = train_generator(train_df, BATCH_SIZE, target_size=(im_height,im_width), train = True, seed=1)
valid_gen = train_generator(val_df, BATCH_SIZE, target_size=(im_height,im_width), train = False, seed=1)
test_gen = train_generator(test_df, BATCH_SIZE, target_size=(im_height,im_width), train = False, seed=1)

```

شکل ۸۲ - دیتاست نهایی استفاده شده بر اساس هایپرپارامتر های آن



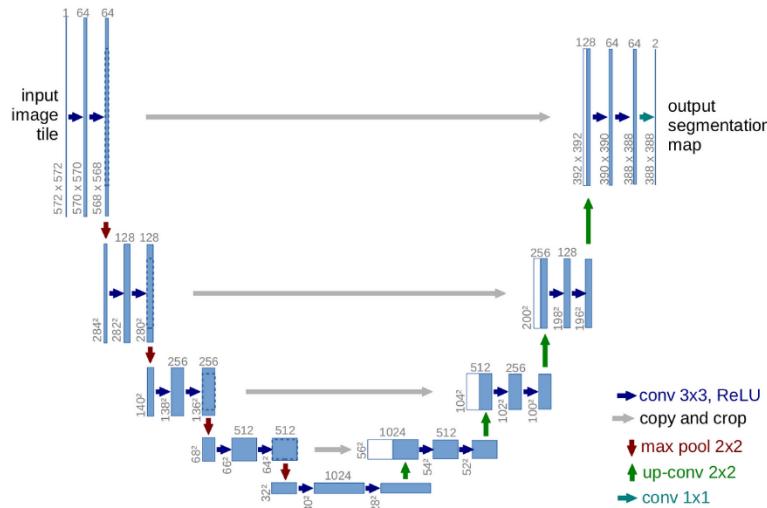
شکل ۸۳ - تصاویر داده افزایی دارای ماسک رنگی

۲-۴: شبکه مورد استفاده

در این بخش به توضیح دو شبکه UNet و TA-UNet میپردازیم:

- **شبکه UNet:** این شبکه اولین بار برای segmentation در زمینه پزشکی معرفی شد اما به دلیل موفقیت بسیار خوب در زمینه های دیگر نیز از آن استفاده شد، این شبکه در حقیقت شامل یک encoder و یک decoder است، که ابتدا با کاهش سایز سایز تصویر ورودی با استفاده از لایه های کانوورلوشن و max-pooling یک فضای latent میسازد، سپس ویژگی های low-level بدست آمده را اول up sample میکند و از طرفی به اتصال های residual آن ها را با ویژگی های سطح بالا ترکیب میکند، به همین دلیل شبکه به خوبی قادر است، که تصاویر اصلی را بازسازی کند، در عین

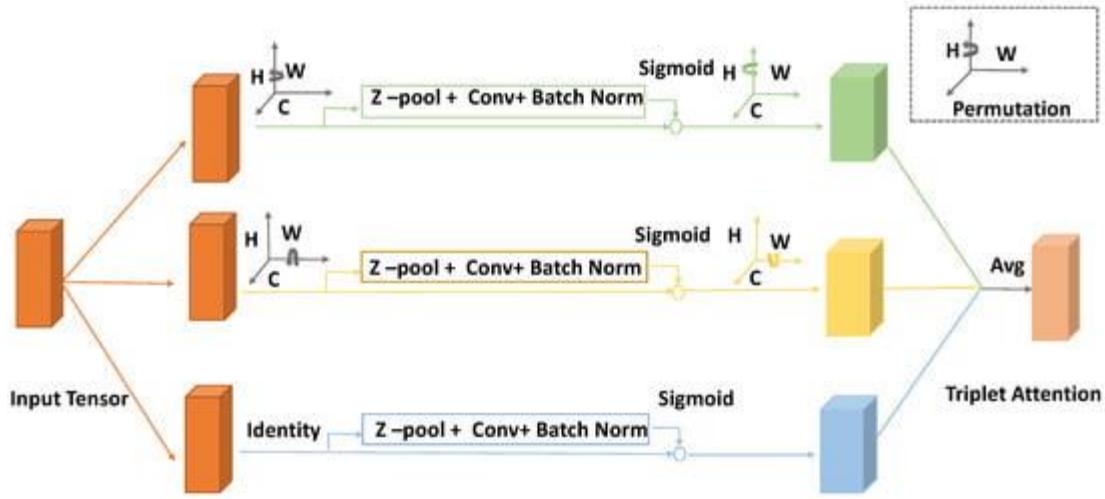
حال ماسک ها را دقیقاً منطبق بر تصویر اصلی تشخیص دهد، از آن مهمتر اینکه این شبکه بسیار سبک است، بنابر چند نوع پیاده سازی مختلف بین ۲۸ تا ۳۱ میلیون پارامتر دارد در حالی که شبکه های دیگر بیش از ۱۰۰ میلیون پارامتر دارند، این شبکه به یک ساختار U شکل دارد و به همین دلیل به این نام خوانده میشود.



شکل ۸۴ - شبکه UNet

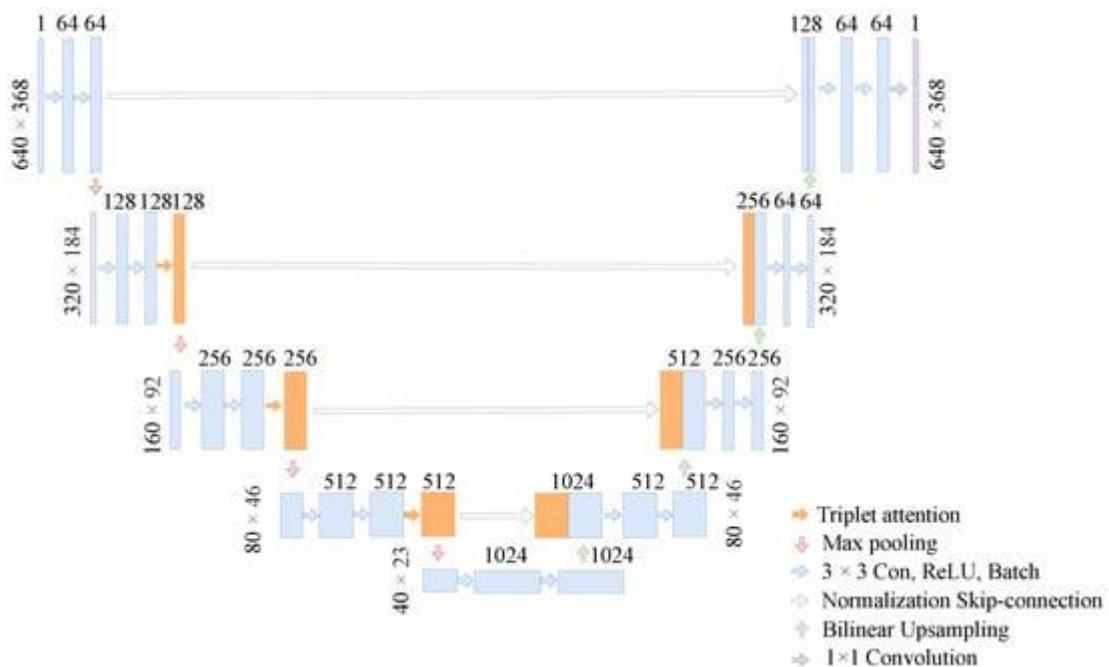
شبکه TA-UNet: بعد از موفقیت UNet انواع مختلفی از شبکه های مبتنی بر آن به وجود آمدند که سعی در اینجا بهبود در این شبکه داشتند، از طرفی مکانیزم Triplet Attention در شبکه دیگری اولین بار معرفی شد، اما این مقاله این دو موضوع را با هم ترکیب کرده است. اساساً Attention موضوعی است که ابتدا در زمینه NLP به وجود آمد، اما بعداً به حوزه تصویر نیز وارد شد، هدف اصلی Attention توجه به قسمت های دارای اطلاعات اصلی و نادیده گرفتن سایر قسمت هاست، اولین بار channel Attention ها تنها روی channel تصاویر اعمال میشدند، که ارتباط میان channel ها در هر feature map مورد بررسی قرار بگیرد، اما در مکانیزم triplet attention، هر سه کanal طول و عرض و تعداد کanal در مکانیزم Attention لحاظ میشود، به طوری که هر تصویر ورودی ابتدا حول یکی از محور هایش میچرخد تا ابعاد دو به دو به ترتیب در در کنار هم قرار بگیرند، سپس با محاسبه score در آن دو بعد خاص آن را به ابعاد اصلی اضافه کرده و در نهایت تصویر را به شکل اصلی در می آوریم، سپس این Attention را که در سه بعد اصلی لحاظ شده است که در هر سه بعد میانگین میگیریم و به عنوان ورودی به بخش بعدی

میدهیم



شکل ۸۵ - مکانیزم Triplet Attention

در مقاله TA-UNet این لایه Attention در انتهای هر بخش encoder لحاظ شده تا خروجی آن اولاً maxpool شده و همچنین همین تنسور خروجی در decoder به صورت residual Attention استفاده شود، تصویر این مدل را در ادامه میبینید:



TA-Unet - شبکه ۸۶

۳-۴: آموزش شبکه ها:

ابتدا به پیاده سازی شبکه ها و سپس به هایپرپارامتر ها و نتایج هر یک میپردازیم:

۱ - شبکه UNet: همانطور که گفتیم این شبکه شامل یک بخش encoder و decoder است، ابتدا ما ساختار conv های این شبکه را در قالب conv_down یعنی کانولوشن هایی که شبکه در encoder در است و دیگری با تابع conv_up یعنی کانولوشن هایی که شبکه در حالت decoder است طراحی کردیم، ساختار دو تابع را در ادامه میبینید:

```
def conv_down(inputs, num_filter, attention=False):
    conv1 = Conv2D(num_filter, (3, 3), padding='same', activation='relu')(inputs)
    bn1 = BatchNormalization()(conv1)
    conv2 = Conv2D(num_filter, (3, 3), padding='same', activation='relu')(bn1)
    bn2 = BatchNormalization()(conv2)
    drp = Dropout(0.1)(bn2)

    if attention:
        drp = TripletAttention()(drp)

    pool = MaxPooling2D(pool_size=(2, 2))(drp)
    return conv2, pool, drp

def conv_up(input1, input2, num_filter1, num_filter2):
    up1 = UpSampling2D()(input1)
    up1 = Conv2D(num_filter1, (1, 1), padding='same', activation='relu')(up1)
    up2 = concatenate([up1, input2], axis=3)
    conv1 = Conv2D(num_filter2, (3, 3), padding='same', activation='relu')(up2)
    bn1 = BatchNormalization()(conv1)
    conv2 = Conv2D(num_filter2, (3, 3), padding='same', activation='relu')(bn1)
    bn2 = BatchNormalization()(conv2)
    drp = Dropout(0.1)(bn2)
    return drp
```

شکل ۸۷- ساختار توابع شبکه UNet

این بخش ها ساده هستند اما دو نکته مهم وجود دارد:

- در تابع اول، میبینیم که مکانیزم Attention در انتهای لایه های encoder قرار دارد و همچنین استفاده از مشروط است، و به همین دلیل در ساختار UNet استفاده نشده است، البته خود لایه Attention را نیز بررسی خواهیم کرد.
- در تابع دوم، میبینیم که skip connection انجام شده است.

در این بخش به پیاده سازی تابع اصلی Unet پرداختیم، همانطور که میبینیم در این بخش تنها از توابع قبلی استفاده شده است و تنها یک لایه Conv با یک فیلتر برای خروجی ایجاد شده است، این تابع Unet برای پیش بینی ماسک های سیاه و سفید است.

```

def UNet(input_size=(im_height,im_width,3)):

    inputs = Input(input_size)

    conv1, pool1, drp1 = conv_down(inputs, 64)
    conv2, pool2, drp2 = conv_down(pool1, 128)
    conv3, pool3, drp3 = conv_down(pool2, 256)
    conv4, pool4, drp4 = conv_down(pool3, 512)
    _, _, drp5 = conv_down(pool4, 1024)

    drp6 = conv_up(drp5, drp4, 512, 512)
    drp7 = conv_up(drp6, drp3, 256, 256)
    drp8 = conv_up(drp7, drp2, 128, 64)
    drp9 = conv_up(drp8, drp1, 64, 64)

    out = Conv2D(1, (1, 1), activation='sigmoid')(drp9)

    return Model(inputs=[inputs], outputs=[out])

```

شکل ۸۸ - شبکه **Unet** برای ماسک های سیاه و سفید

همچنین همین تابع با تغییر زیر برای ماسک های رنگی استفاده شد، مشاهده میکنید که این تابع دقیقاً مثل تابع قبلی است با این تفاوت که در لایه خروجی، تعداد چنل خروجی از یک به ۳ رسیده است، نکته مهم این است که لایه خروجی حتماً باید sigmoid باشد و نه softmax. درست است ما در حال حدس زدن ۳ عدد در طول ۳ چنل هستیم اما نباید از این ۳ عدد تنها یکی از آن ها یک باشد، چرا که در این صورت تنها سه رنگ قابل پیش بینی میشوند، بلکه ممکن است، هر سه عدد ۱ باشند یعنی عدد (۱۱۱) و همینطور (۰۰۱) هم در نهایت رنگ های درستی هستند که ماسک را پیش بینی میکنند، در نتیجه ما روی هر عدد یک لحاظ sigmoid میکنیم که مقدار آن را بین صفر و یک پیش بینی کند.

```

def UNet(input_size=(im_height,im_width,3)):

    inputs = Input(input_size)

    conv1, pool1, drp1 = conv_down(inputs, 64)
    conv2, pool2, drp2 = conv_down(pool1, 128)
    conv3, pool3, drp3 = conv_down(pool2, 256)
    conv4, pool4, drp4 = conv_down(pool3, 512)
    _, _, drp5 = conv_down(pool4, 1024)

    drp6 = conv_up(drp5, drp4, 512, 512)
    drp7 = conv_up(drp6, drp3, 256, 256)
    drp8 = conv_up(drp7, drp2, 128, 64)
    drp9 = conv_up(drp8, drp1, 64, 64)

    out = Conv2D(3, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(drp9)
    out = Conv2D(3, (1, 1), activation='sigmoid')(out)

    return Model(inputs=[inputs], outputs=[out])

```

شکل ۸۹ - شبکه **Unet** برای ماسک های رنگی

ابتدا لایه Attention را پیاده سازی کردیم که به آن میپردازیم:

```
class TripletAttention(Layer):
    def __init__(self, no_spatial=False):
        super(TripletAttention, self).__init__()
        self.cw = AttentionGate()
        self.hc = AttentionGate()
        self.no_spatial = no_spatial
        if not no_spatial:
            self.hw = AttentionGate()

    def call(self, x):
        x_perm1 = tf.transpose(x, perm=[0, 3, 2, 1])
        x_out1 = self.cw(x_perm1)
        x_out11 = tf.transpose(x_out1, perm=[0, 3, 2, 1])

        x_perm2 = tf.transpose(x, perm=[0, 1, 3, 2])
        x_out2 = self.hc(x_perm2)
        x_out21 = tf.transpose(x_out2, perm=[0, 1, 3, 2])

        if not self.no_spatial:
            x_out = self.hw(x)
            x_out = (x_out + x_out11 + x_out21) / 3
        else:
            x_out = (x_out11 + x_out21) / 2

        return x_out
```

شکل ۹۰ - لایه Attention

در این تابع همانطور که از تابع AttentionGate استفاده میشود، این تابع در بخش call ابتدا تنسور ورودی را حول محور های میچرخاند

و سپس Attention مورد نظر را به تنسور چرخیده شده اعمال میکند، چراکه لایه Attention تنها به دو مقدار میانی در تنسور اعمال میشود، بیایید فرض کنیم بردار ورودی دارای shape ورودی زیر است:

$$(Batch, Height, Width, Channel) = (B, H, W, C)$$

ما به ترتیب تنسور های زیر را ایجاد میکنیم:

$$(B, \boxed{C, W}, H) - ۱$$

$$(B, \boxed{H, C}, W) - ۲$$

$$(B, \boxed{H, W}, C) - ۳$$

این کار با دستور transpose انجام شده و سپس لایه Attention رو آن لحاظ شده و در نهایت به شکل اولیه برمیگیرد.

```
x_perm1 = tf.transpose(x, perm=[0, 3, 2, 1])
x_out1 = self.cw(x_perm1)
x_out11 = tf.transpose(x_out1, perm=[0, 3, 2, 1])
```

شکل ۹۱ - اعمال Attention و برگشت به حالت اولیه

و در نهایت هر سه این بردارها به صورت میانگین با هم جمع میشوند:

```
if not self.no_spatial:
    x_out = self.hw(x)
    x_out = (x_out + x_out11 + x_out21) / 3
else:
    x_out = (x_out11 + x_out21) / 2
```

شکل ۹۲ - خروجی لایه Attention

اما خود لایه Attention چیست:

```
class AttentionGate(Layer):
    def __init__(self, kernel_size=7):
        super(AttentionGate, self).__init__()
        self.compress = ZPool()
        self.conv = BasicConv(out_planes=1, kernel_size=kernel_size, padding='same', relu=False)

    def call(self, x):
        x_compress = self.compress(x)
        x_out = self.conv(x_compress)
        scale = tf.sigmoid(x_out)
        return x * scale
```

شکل ۹۳ - لایه Attention

این لایه از لایه Convolution ساده و یک Zpool تشکیل شده است، که ابتدا با استفاده از Zpool ورودی Compress شده و سپس Convolve میشود، و در نهایت با اعمال sigmoid روی آن به صورت ضرایب اهمیت هر پیکسل بدست می آید و در مقدار تنسور ورودی اولیه ضرب میشود.

در زیر ساختار BaseConv و Zpool نیز قابل مشاهده است:

```
class BasicConv(Layer):
    def __init__(self, out_planes, kernel_size, stride=1, padding='valid',
                 dilation=1, groups=1, relu=True, bn=True, bias=False):
        super(BasicConv, self).__init__()

        self.conv = Conv2D(filters=out_planes, kernel_size=kernel_size,
                           strides=stride, padding=padding, dilation=dilation,
                           groups=groups, use_bias=bias)

        self.bn = BatchNormalization(epsilon=1e-5, momentum=0.01) if bn else None
        self.relu = ReLU() if relu else None

    def call(self, x):
        x = self.conv(x)
        if self.bn is not None:
            x = self.bn(x)
        if self.relu is not None:
            x = self.relu(x)
        return x

class ZPool(Layer):
    def call(self, x):
        max_pool = tf.reduce_max(x, axis=3, keepdims=True)
        avg_pool = tf.reduce_mean(x, axis=3, keepdims=True)
        return tf.concat([max_pool, avg_pool], axis=3)
```

شکل ۹۴ - لایه های ZPool و BaseConv

لایه BaseConv که یک لایه کانولوشنی ساده است که تنها وزن های قبل یادگیری به مدل اضافه میکند، از طرفی Zpool بهم چسباندن دو max pool و avg pool را در تنسور ورودی است، به همین دلیل هم لایه Attention تنها به دو بعدی میانی (۱ و ۲) اهمیت میدهد.

سپس به پیاده سازی تابع اصلی TA-UNet پرداختیم، توابع conv_up , conv_down دقیقاً مانند قبل هستند و تنها تابع اصلی مدل تغییر میکند، در تصویر پایین در کادر قرمز، نحوه اعمال توابع Conv را در بخش decoder میبینیم.

```
def TA_UNet(input_size=(im_height,im_width,3)):

    inputs = Input(input_size)

    conv1, pool1, drp1 = conv_down(inputs, 64)
    conv2, pool2, drp2 = conv_down(pool1, 128, attention=True)
    conv3, pool3, drp3 = conv_down(pool2, 256, attention=True)
    conv4, pool4, drp4 = conv_down(pool3, 512, attention=True)
    _, _, drp5 = conv_down(pool4, 1024)

    drp6 = conv_up(drp5, drp4, 512, 512)
    drp7 = conv_up(drp6, drp3, 256, 256)
    drp8 = conv_up(drp7, drp2, 128, 64)
    drp9 = conv_up(drp8, drp1, 64, 64)

    out = Conv2D(1, (1, 1), activation='sigmoid')(drp9)

    return Model(inputs=[inputs], outputs=[out])
```

شکل ۹۵ - شبکه TA-Unet برای ماسک سیاه و سفید

```
def TA_UNet(input_size=(im_height,im_width,3)):

    inputs = Input(input_size)

    conv1, pool1, drp1 = conv_down(inputs, 64)
    conv2, pool2, drp2 = conv_down(pool1, 128, attention=True)
    conv3, pool3, drp3 = conv_down(pool2, 256, attention=True)
    conv4, pool4, drp4 = conv_down(pool3, 512, attention=True)
    _, _, drp5 = conv_down(pool4, 1024)

    drp6 = conv_up(drp5, drp4, 512, 512)
    drp7 = conv_up(drp6, drp3, 256, 256)
    drp8 = conv_up(drp7, drp2, 128, 64)
    drp9 = conv_up(drp8, drp1, 64, 64)

    out = Conv2D(3, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(drp9)
    out = Conv2D(3, (1, 1), activation='sigmoid')(out)

    return Model(inputs=[inputs], outputs=[out])
```

شکل ۹۶ - شبکه TA-UNet برای ماسک های رنگی

در ادامه به ذکر هایپرپارامتر های آموزش و نتایج هر شبکه میپردازیم:
پیش از آن خوب است ذکر کنیم که مقاله استفاده از loss مبتنی به iou یعنی lovasz را به عنوان پیشنهادی برای آموزش شبکه استفاده میکند که آن را در این بخش پیاده سازی کردیم، اما طبق تجربه cross_entropy loss نیز دقیقی به همان خوبی داشت و ما از آن بهره بردیم.

```
from tensorflow.keras.losses import binary_crossentropy

def iou_coef(y_true, y_pred, smooth=100):
    intersection = K.sum(y_true * y_pred)
    union = K.sum(y_true) + K.sum(y_pred)
    iou = (intersection)/(union - intersection + smooth)
    return iou

def lovasz_loss(y_true, y_pred, smooth=100):
    return 1-iou_coef(y_true, y_pred, smooth)

def combined_loss(y_true, y_pred, smooth=100, bce_weight=0.7, lovasz_weight=0.3):
    bce_loss = binary_crossentropy(y_true, y_pred)
    lovasz = lovasz_loss(y_true, y_pred, smooth)
    return bce_weight * bce_loss + lovasz_weight * lovasz
```

شکل ۹۷ - تعریف معیار های **iou** و **lovsaz loss** و ترکیب آن با **cross entropy**

۱ - شبکه UNet با ماسک سیاه و سفید:

هایپرپارامتر ها

۱۲۸*۱۲۸*۳	ابعاد تصویر ورودی
۳۲	اندازه Batch
۵۰	تعداد Epoch
۰.۰۰۰۱	نرخ یادگیری
Binary_crossentropy	تابع هزینه

جدول ۶ - هایپرپارامتر های UNet با ماسک باینری

```
EPOCHS = 50
n_classes = 2
BATCH_SIZE = 32
im_height = 128
im_width = 128

mean_iou = keras.metrics.MeanIoU(num_classes=num_class, name="mean_iou")
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0001)

unet_model = UNet(input_size=(im_height, im_width, 3))
unet_model.compile(optimizer=optimizer, loss= 'binary_crossentropy', metrics=["binary_accuracy", mean_iou, iou_coef])

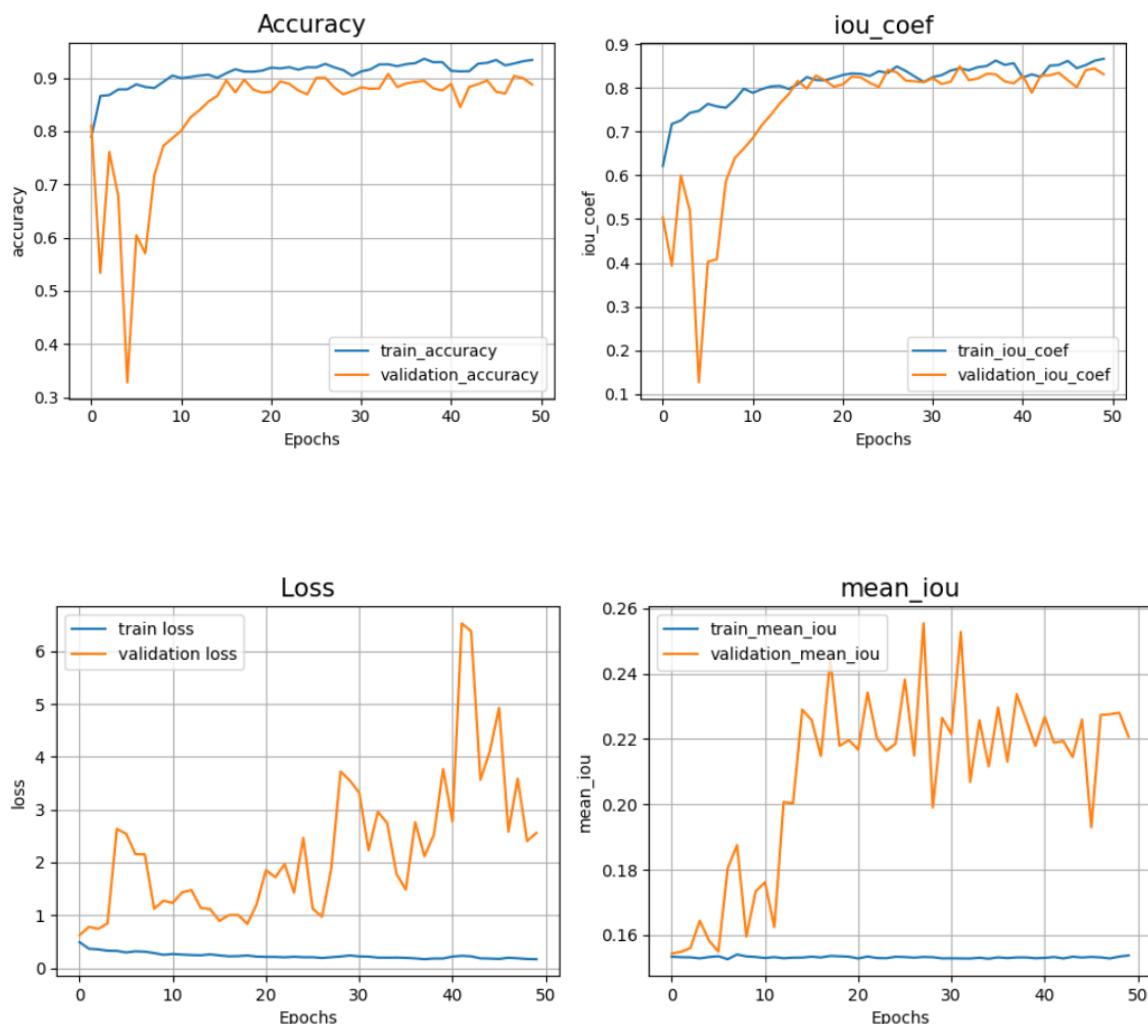
unet_model_history = unet_model.fit(train_gen,
                                     steps_per_epoch = len(train_df) / BATCH_SIZE,
                                     epochs=EPOCHS,
                                     validation_data = valid_gen,
                                     validation_steps = len(val_df) / BATCH_SIZE)
#callbacks = callbacks)
```

شکل ۹۸ - تابع آموزش شبکه UNet

نتایج:

در ادامه تصویر نتایج شبکه را مشاهده میکنید:

در نمودارهای زیر معیار loss detection که در خود کتابخانه tensorflow است رفتار نوسانی دارد، البته با توجه به تصاویر پایانی، معیار IOU که توسط خودمان و با فرمول مقاله پیاده سازی شده است، معیار مناسب تری برای بررسی خروجی ها نسبت به miou است، این معیار miou در کتابخانه tensorflow احتمالا برای ترسیمت نه است به همین دلیل هم خروجی خوبی نشان نمیدهد.



شکل ۹۹ - نتایج شبکه UNet با ماسک باینری

نکته مهمی وجود دارد و این است که معیار IOU Coef که خودمان آن را تعریف کردیم و در تصویر اول مشاهده میکنید، معیار دقیق تری است، منتها یک معیار mean IOU در تسنورفلو موجود است که آن را نیز استفاده کردیم، اما در اصل به دلیل باینری بودن ماسک ها همان معیار IOU coef به معنی mean IOU در این شبکه است.

نتایج روی داده های تست:

```
Found 110 validated image filenames.
Found 110 validated image filenames.
3/3 [=====] - 0s 132ms/step - loss: 0.3133 - binary_accuracy: 0.9104 - mean_iou: 0.2100 - iou_coef: 0.8078
Test loss: 0.31
Test acc: 91.04%
Test mean IOU: 0.21
Test IOU coef: 0.81
```

شکل ۱۰۰ - نتایج Unet روی داده های تست

و در نهایت تعدادی از ماسک های پیش بینی شده:



شکل ۱۰۱ - تصویر ماسک های پیش بینی شده توسط مدل

میبینیم که شبکه به خوبی توانسته ماسک های باینری را پیش بینی کند و ماسک بسیار شبیه به نمونه های واقعی هستند.

و معیار های نهایی نیز به شکل زیر هستند:

IOU_coef	Accuracy
.۸۱	۹۱.۴٪

جدول ۷ - نتایج مدل

۲ - شبکه TA-UNet با ماسک سیاه و سفید:

هایپرپارامتر ها

۱۲۸*۱۲۸*۳	ابعاد تصویر ورودی
۳۲	اندازه Batch
۵۰	تعداد Epoch
۰.۰۰۰۱	نرخ یادگیری
Binary_crossentropy	تابع هزینه

جدول ۸ - هایپرپارامتر های TA-UNet با ماسک باینری

```
EPOCHS = 50
n_classes = 2
BATCH_SIZE = 32
im_height = 128
im_width = 128

mean_iou = keras.metrics.MeanIoU(num_classes=n_classes, name="mean_iou")
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0001)

taunet_model = TA_UNet(input_size=(im_height, im_width, 3))
taunet_model.compile(optimizer=optimizer, loss= 'binary_crossentropy', metrics=["binary_accuracy", mean_iou, iou_coef])

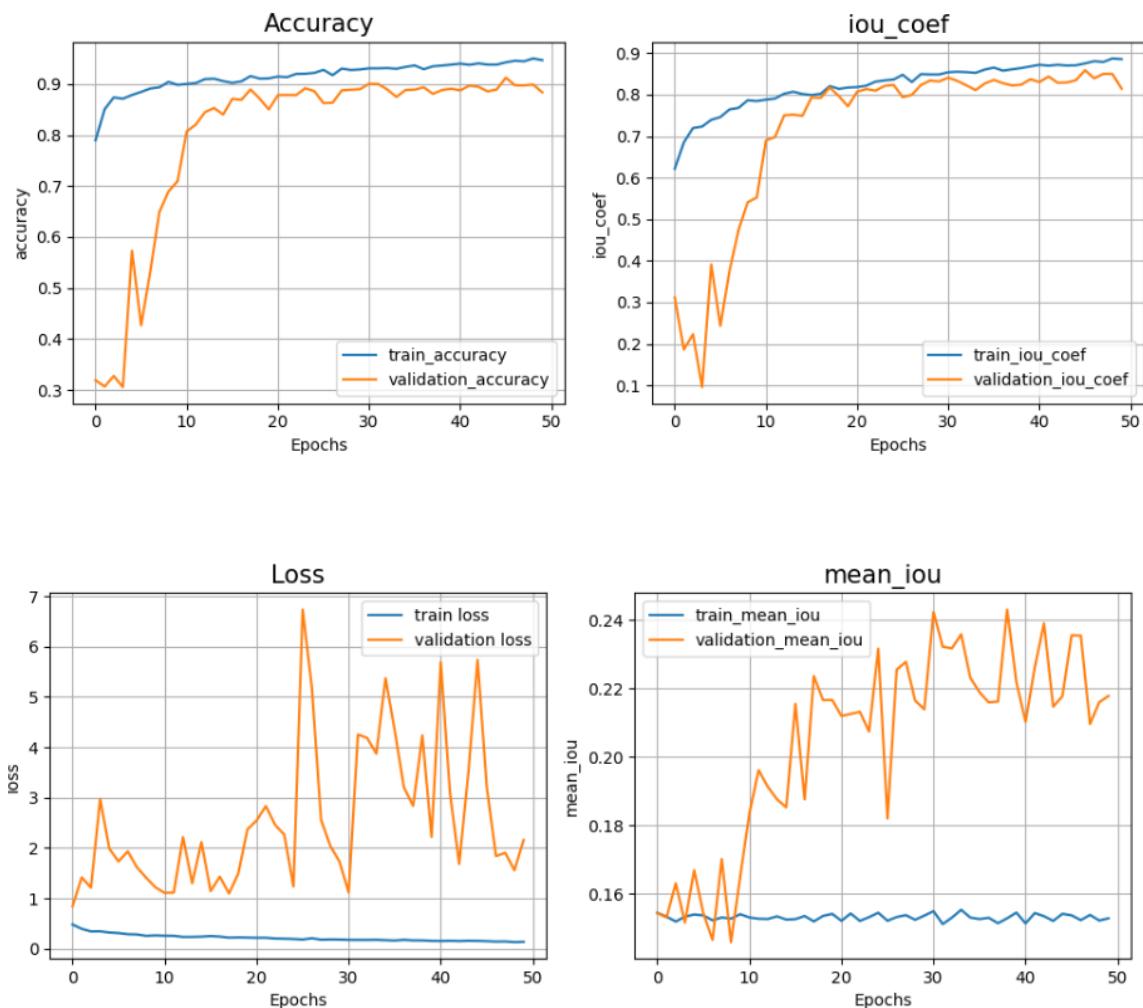
taunet_model_history = taunet_model.fit(train_gen,
                                         steps_per_epoch = len(train_df) / BATCH_SIZE,
                                         epochs=EPOCHS,
                                         validation_data = valid_gen,
                                         validation_steps = len(val_df) / BATCH_SIZE)
#callbacks = callbacks)
```

شکل ۱۰۲ - تابع آموزش شبکه TA-UNet

نتایج:

در ادامه تصویر نتایج شبکه را مشاهده میکنید:

در نمودارهای زیر معیار Accuracy و iou_coef به خوبی اموزش مدل را نشان میدهد، مقدار تابع loss نیز کاهش داشته، البته روی مجموعه ولیدیشن نشان از overfit شدن هم دارد، از طرفی معیار mean IOU که در خود کتابخانه tensorflow است رفتار نوسانی دارد، البته با توجه به تصاویر پایانی، معیار iou_coef که توسط خودمان و با فرمول مقاله پیاده سازی شده است، معیار مناسب تری برای بررسی خروجی ها نسبت به miou است، این معیار miou در کتابخانه tensorflow احتمالا برای تسک detection است نه segmetation به همین دلیل هم خروجی خوبی نشان نمیدهد.



شکل ۱۰۳ - نتایج شبکه TA-UNet با ماسک باینری

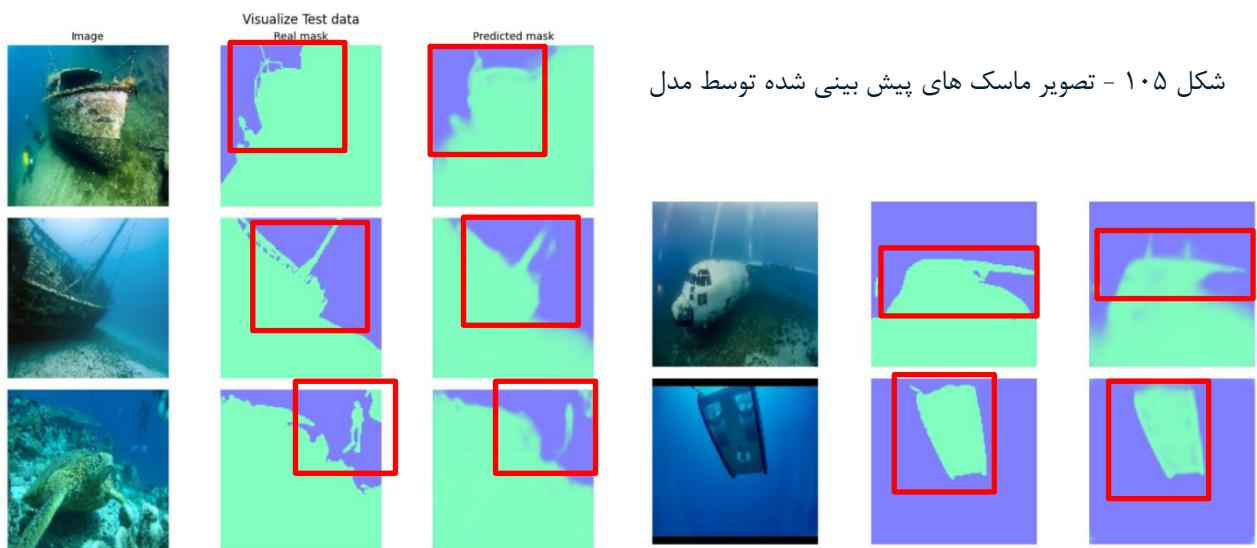
نکته مهمی وجود دارد و این است که معیار IOU Coef که خودمان آن را تعریف کردیم و در تصویر اول مشاهده میکنید، معیار دقیق تری است، منتها یک معیار mean IOU در تسنورفلو موجود است که آن را نیز استفاده کردیم، اما در اصل به دلیل باینری بودن ماسک ها همان معیار IOU coef به معنی mean IOU در این شبکه است.

نتایج روی داده های تست:

```
3/3 [=====] - 0s 143ms/step - loss: 0.6585 - binary_accuracy: 0.8766 - mean_iou: 0.2252 - iou_coef: 0.7686
Test loss: 0.66
Test acc: 87.66%
Test mean IOU: 0.23
Test IOU coef: 0.77
```

شکل ۱۰۴ - نتایج TA-UNet روی داده های تست

و در نهایت تعدادی از ماسک های پیش بینی شده:



شکل ۱۰۵ - تصویر ماسک های پیش بینی شده توسط مدل

میبینیم که شبکه به خوبی توانسته ماسک های باینری را پیش بینی کند و ماسک بسیار شبیه به نمونه های واقعی هستند.

و معیار های نهایی نیز به شکل زیر هستند:

IOU_coef	Accuracy
.۷۷	۸۷.۶۶

جدول ۹ - نتایج مدل

۳ - شبکه UNet با ماسک رنگی:

هایپرپارامتر ها

۱۲۸*۱۲۸*۳	ابعاد تصویر ورودی
۴۰	اندازه Batch
۵۰	تعداد Epoch
۰.۰۰۰۵	نرخ یادگیری
Binary_crossentropy	تابع هزینه

جدول ۱۰ - هایپرپارامتر های UNet با ماسک باینری

```
EPOCHS = 50
num_class = 3
BATCH_SIZE = 40
im_height = 128
im_width = 128

mean_iou = keras.metrics.MeanIoU(num_classes=num_class, name="mean_iou")
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0005)

unet_model = UNet(input_size=(im_height, im_width, 3))
unet_model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=["binary_accuracy", mean_iou, iou_coef])

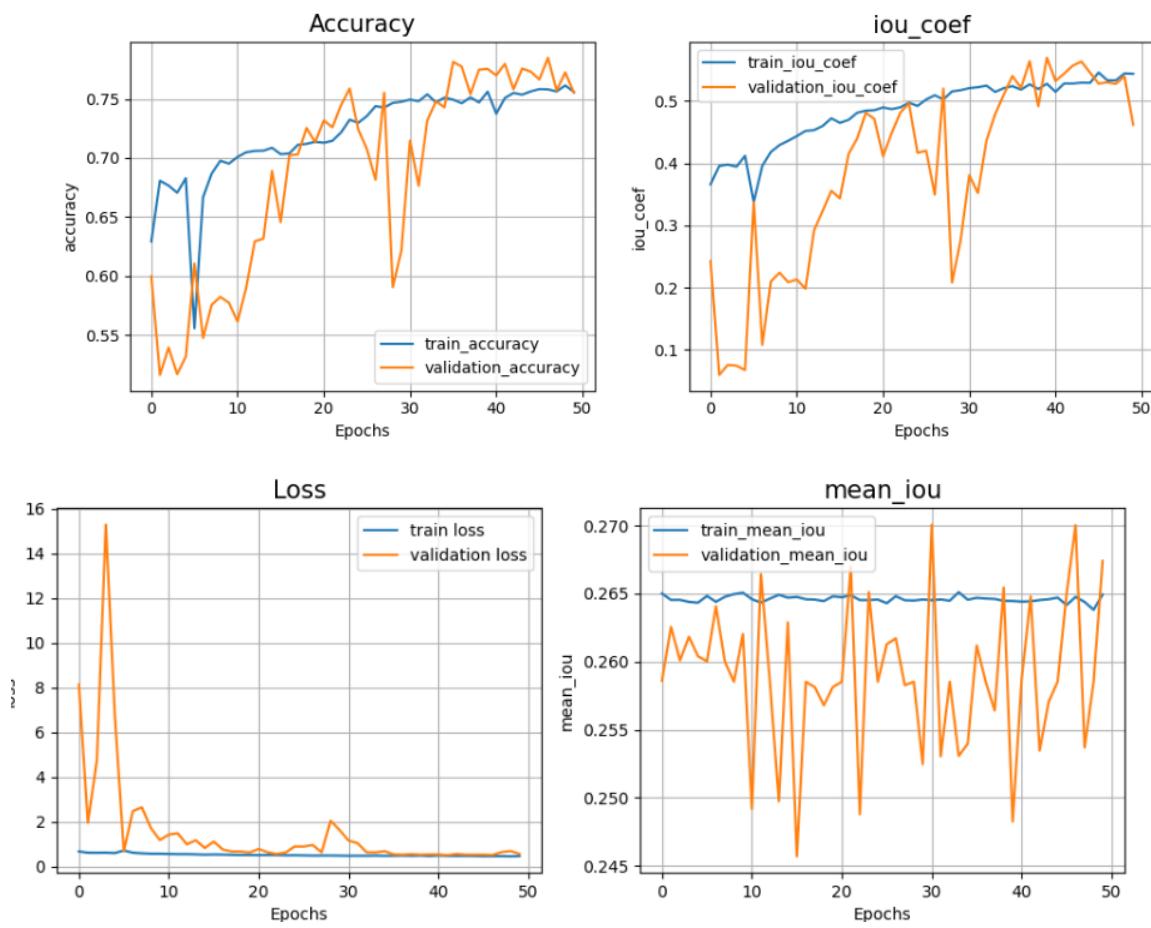
unet_model_history = unet_model.fit(train_gen,
                                     steps_per_epoch = len(train_df) / BATCH_SIZE,
                                     epochs=EPOCHS,
                                     validation_data = valid_gen,
                                     validation_steps = len(val_df) / BATCH_SIZE)
#callbacks = callbacks)
```

شکل ۱۰۶ - تابع آموزش شبکه UNet

نتایج:

در ادامه تصویر نتایج شبکه را مشاهده میکنید:

در نمودارهای زیر معیار Accuracy و iou_coef به خوبی اموزش مدل را نشان میدهد، مقدار تابع loss نیز کاهش بسیار خوبی داشته است، از طرفی معیار mean IOU که در خود کتابخانه tensorflow است رفتار نوسانی دارد، البته با توجه به تصاویر پایانی، معیار iou_coef که توسط خودمان و با فرمول مقاله پیاده سازی شده است، معیار مناسب تری برای بررسی خروجی ها نسبت به miou است، این معیار miou در کتابخانه tensorflow برای ترسیمت segmetation detection است نه به همین دلیل هم خروجی خوبی نشان نمیدهد.



شکل ۱۰۷ - نتایج شبکه UNet با ماسک باینری

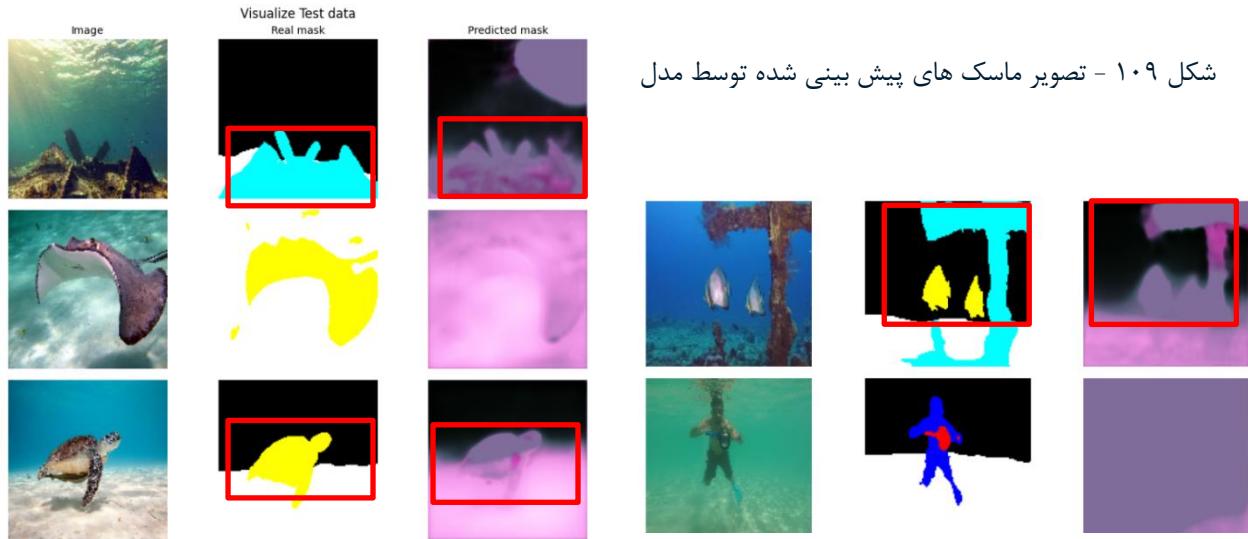
نکته مهمی وجود دارد و این است که معیار IOU Coef که خودمان آن را تعریف کردیم و در تصویر اول مشاهده میکنید، معیار دقیق تری است، منتها یک معیار mean IOU در تسنورفلو موجود است که آن را نیز استفاده کردیم، اما در اصل به دلیل باینری بودن ماسک ها همان معیار IOU coef به معنی mean IOU در این شبکه است.

نتایج روی داده های تست:

```
Found 110 validated image filenames.
Found 110 validated image filenames.
2/2 [=====] - 2s 1s/step - loss: 0.5150 - binary_accuracy: 0.7370 - mean_iou: 0.2827 - iou_coef: 0.4510
Test acc: 73.70%
Test IOU coef: 0.45
```

شکل ۱۰۸ - نتایج Unet روی داده های تست

و در نهایت تعدادی از ماسک های پیش بینی شده:



شکل ۱۰.۹ - تصویر ماسک های پیش بینی شده توسط مدل

میبینیم که شبکه به خوبی نتوانسته ماسک های رنگی را پیش بینی کند، البته این تسلیک بسیار سخت تری نسبت به قبل است، اما میبینیم که شبکه در حال یادگیری است و چه بسا با ادامه یادگیری بتوان به نتایج بهتری رسید.

و معیار های نهایی نیز به شکل زیر هستند:

IOU_coef	Accuracy
.۴۵	۷۳.۷۰٪

جدول ۱۱ - نتایج مدل

۴ - شبکه UNet با ماسک رنگی:

هایپرپارامتر ها

۱۲۸*۱۲۸*۳	ابعاد تصویر ورودی
۴۰	اندازه Batch
۵۰	تعداد Epoch
۰.۰۰۰۵	نرخ یادگیری
Binary_crossentropy	تابع هزینه

جدول ۱۲ - هایپرپارامتر های TA-UNet با ماسک باینری

```
EPOCHS = 50
n_classes = 3
BATCH_SIZE = 40
im_height = 128
im_width = 128

mean_iou = keras.metrics.MeanIoU(num_classes=n_classes, name="mean_iou")
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0005)

taunet_model = TA_UNet(input_size=(im_height, im_width, 3))
taunet_model.compile(optimizer=optimizer, loss= 'binary_crossentropy', metrics=["binary_accuracy", mean_iou, iou_coef])

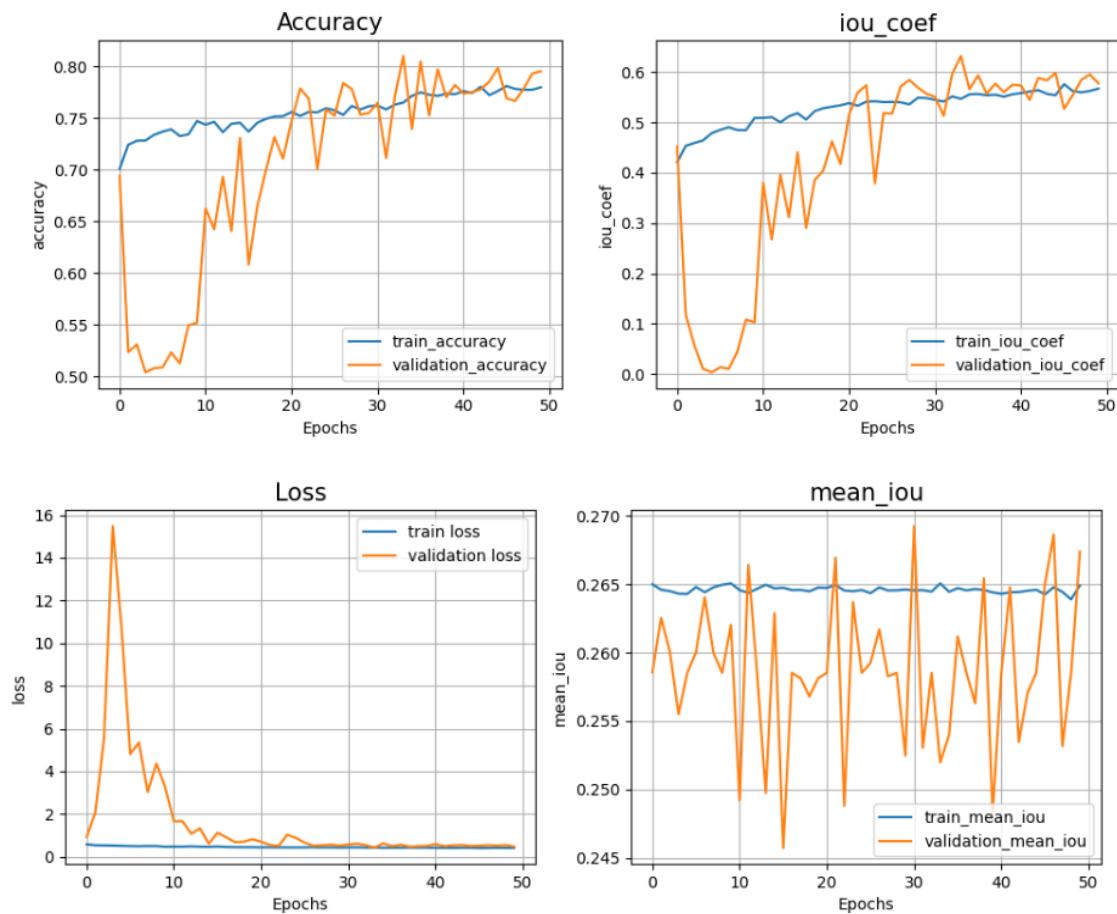
taunet_model_history = taunet_model.fit(train_gen,
                                         steps_per_epoch = len(train_df) / BATCH_SIZE,
                                         epochs=EPOCHS,
                                         validation_data = valid_gen,
                                         validation_steps = len(val_df) / BATCH_SIZE)
#callbacks = [reduce_lr]
```

شکل ۱۱۰ - تابع آموزش شبکه TA-UNet

نتایج:

در ادامه تصویر نتایج شبکه را مشاهده میکنید:

در نمودارهای زیر معیار Accuracy و iou_coef به خوبی اموزش مدل را نشان میدهد، مقدار تابع loss نیز کاهش بسیار محسوسی داشته است، از طرفی معیار mean IOU که در خود کتابخانه tensorflow است رفتار نوسانی دارد، البته با توجه به تصاویر پایانی، معیار iou_coef که توسط خودمان و با فرمول مقاله پیاده سازی شده است، معیار مناسب تری برای بررسی خروجی ها نسبت به miou است، این معیار در کتابخانه tensorflow احتمالا برای تsek segmetation detection است نه به همین دلیل هم خروجی خوبی نشان نمیدهد.



شکل ۱۱۱ - نتایج شبکه **TA UNet** با ماسک باینری

نکته مهمی وجود دارد و این است که معیار IOU Coef که خودمان آن را تعریف کردیم و در تصویر اول مشاهده میکنید، معیار دقیق تری است، منتها یک معیار mean IOU در تسنورفلو موجود است که آن را نیز استفاده کردیم، اما در اصل به دلیل باینری بودن ماسک ها همان معیار IOU coef به معنی mean IOU در این شبکه است.

نتایج روی داده های تست:

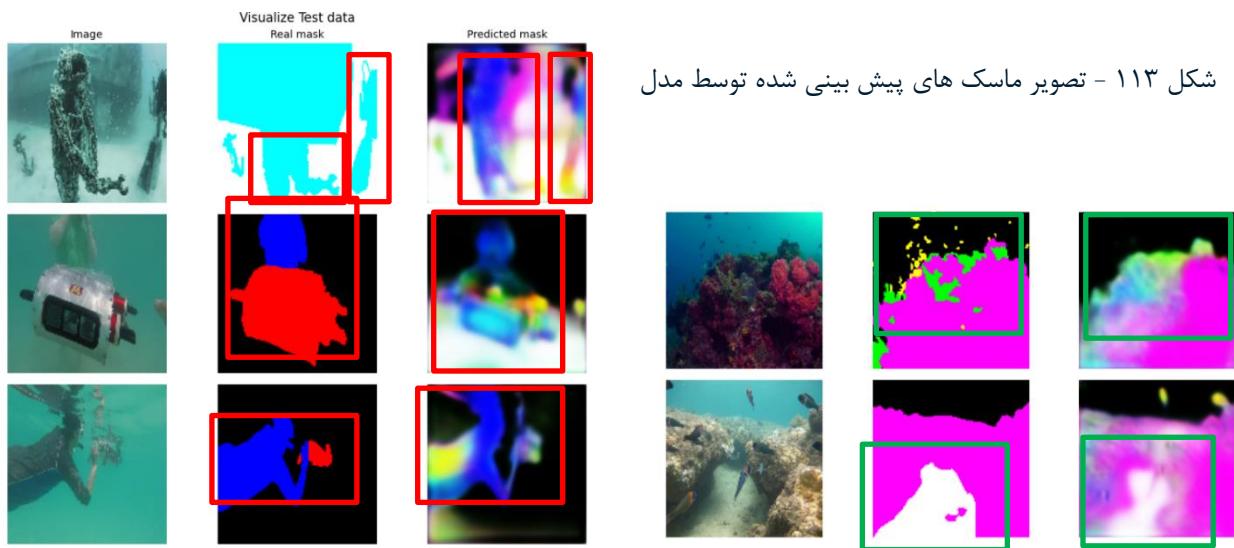
```

Found 110 validated image filenames.
Found 110 validated image filenames.
2/2 [=====] - 3s 1s/step - loss: 0.4874 - binary_accuracy: 0.7874 - mean_iou: 0.2827 - iou_coef: 0.5356
Test loss: 0.49
Test acc: 78.74%
Test mean iou: 0.2827
Test IOU coef: 0.54

```

شکل ۱۱۲ - نتایج **TA Unet** روی داده های تست

و در نهایت تعدادی از ماسک های پیش بینی شده:



شکل ۱۱۳ - تصویر ماسک های پیش بینی شده توسط مدل

میبینیم که شبکه به خوبی نتوانسته ماسک های رنگی را پیش بینی کند، البته این ترسک بسیار سخت تری نسبت به قبل است، اما میبینم که شبکه در حال یادگیری است و چه بسا با ادامه یادگیری بتوان به نتایج بهتری رسید، البته نتایج شبکه TA-Unet بسیار بهتر از شبکه Unet شده است و این دقیقا جایی است که اهمیتی مکانیزم Attention دیده میشود.

و معیار های نهایی نیز به شکل زیر هستند:

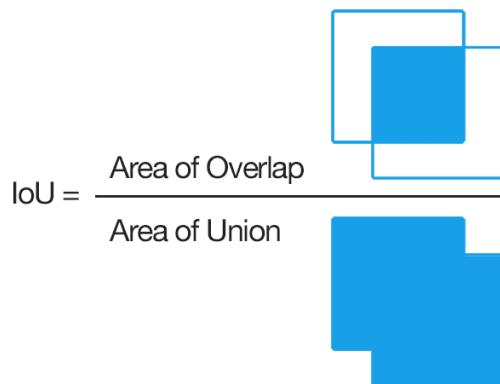
IOU_coef	Accuracy
۰.۵۴	۷۸.۷۴٪

جدول ۱۳ - نتایج مدل

لازم به ذکر است که در این قسمت تنها به نتایج اشاره کردیم و تحلیل نهایی را در بخش بعدی انجام خواهیم داد.

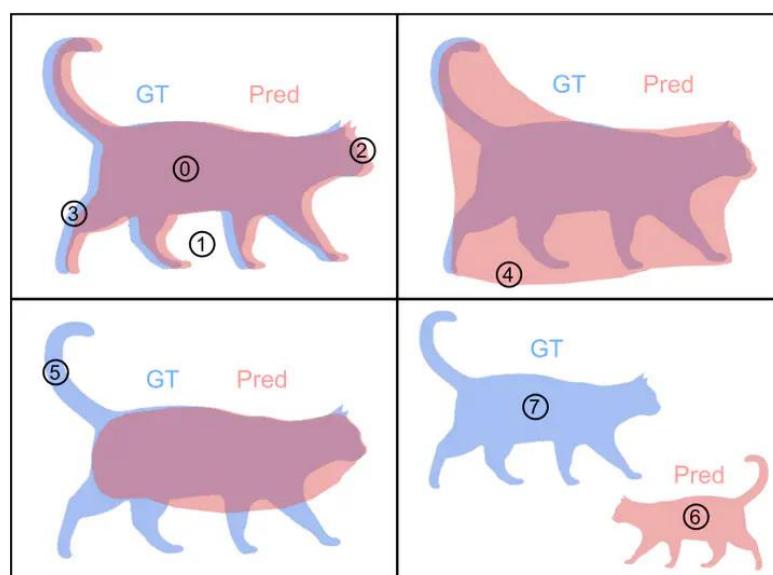
۴-۴: ارزیابی و تحلیل نتایج

معیار IOU در حقیقت به معنی Intersection over Union است، که بین دو کلاس، اشتراک را بر بخش اجتماع تقسیم میکند



شکل ۱۱۴ - معیار IOU

حال وقتی در مورد Segmentation صحبت میکنیم، این معیار را در حقیقت در سطح پیکسل بررسی میکنیم، گویی تعداد پیکسل هایی که به درستی به آن کلاس انتساب داده ایم را بر تعداد کل پیکسل های دو کلاس تقسیم میکنیم، در شکل زیر این موضوع مشخص است:



شکل ۱۱۵ - معیار segmetation IOU در

حال اگر معیار IOU را با ادبیات ماشین لرنینگ بیان کنیم، به فرمول زیر برای هر کلاس میرسیم، یعنی مثلا در کلاس گربه، تعداد پیکسل های درستی که گربه ذکر شده اند، نسبت به همه پیکسل های دیگر iou در آن کلاس را میسازد

$$\text{IoU}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c},$$

شکل ۱۱۶ - فرمول **IOU** در هر کلاس c

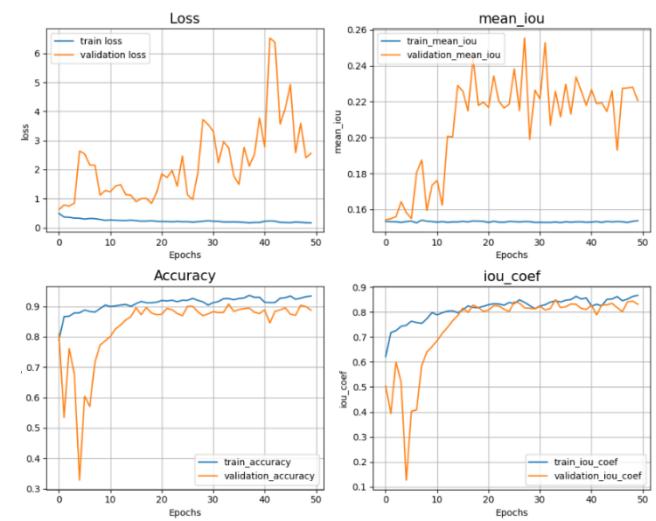
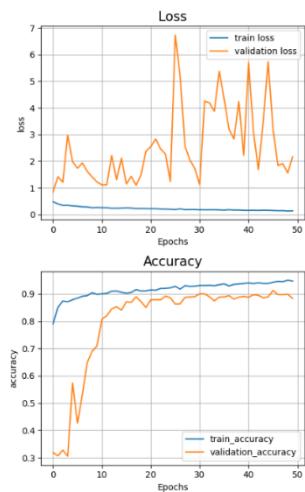
اگر این کار را به ازای همه کلاس‌های تکرار کنیم و نتایج را بر تعداد کلاس‌ها تقسیم کنیم، در نهایت، به فرمول زیر برای mean IOU میرسیم.

$$\text{mean_IoU} = \frac{1}{C} \sum_c \text{IoU}_c.$$

شکل ۱۱۷ - فرمول **mean iou** در هر همه کلاس‌ها

در این ادامه بر اساس ماسک‌های باینری و سپس ماسک‌های رنگی شبکه‌ها را تحلیل می‌کنیم:

۱ - ماسک باینری:



شکل ۱۱۸ - شبکه TA-Unet

شکل ۱۱۹ - شبکه UNet

در این دو تصویر می‌بینیم که در ماسک‌های باینری هر دو شبکه عملکرد بسیار خوبی دارند، بر اساس معیارهای iou و Accuracy عملکرد عالی دارند، اما loss در مجموعه ولیدیشن نوسانی است که البته یکی از دلایل آن میتواند dropout باشد چراکه لاس را نوسانی می‌کند، اما نشانی از Accuracy overfit در دیده نمی‌شود اگرچه در loss چنین چیزی محال نیست.

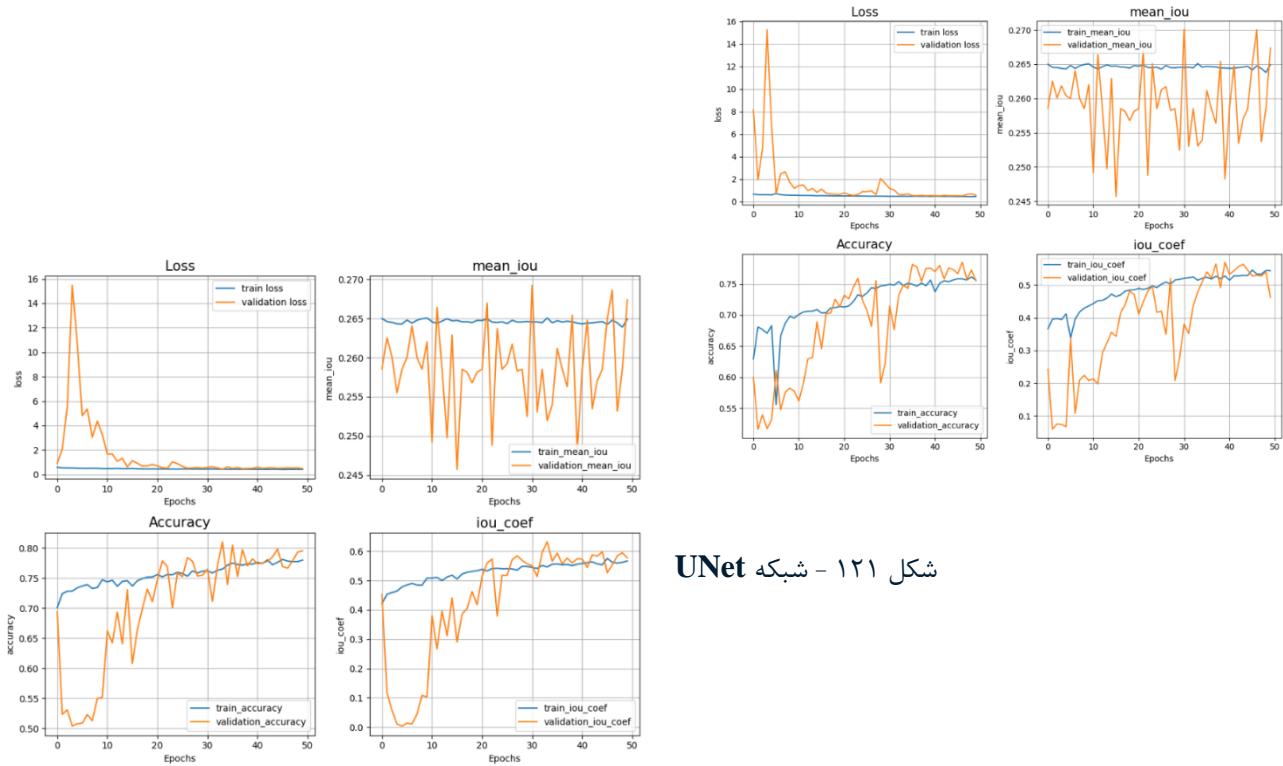
نکته مهم این است که هر دو شبکه در این تسك عالی هستند، و به نظر میرسد که مکانیزم Attention آنچنان بهتر عمل نکرده است چراکه تسك اساسا برای این شبکه UNet سخت نبوده است، مقدار معیار ها روی مجموعه تست را مشاهده میکنید.

	Accuracy	IOU_coef
UNet	۹۱.۰۴٪	۰.۸۱
AT-UNet	۸۷.۶۶٪	۰.۷۷

شکل ۱۲۰ - مقایسه دو شبکه روی دیتاست تست

در این بخش اگر بخواهیم یک شبکه انتخاب کنیم، تقریباً دو شبکه مثل هم عمل کرده اند حتی UNet به سبب ساده بودن بهتر از AT-UNet عمل کرده است، البته بدیهی است که با آموزش بیشتر شبکه، و رسیدن به نهایت توانایی شبکه های چه بسا UNet در همین دقت ۹۱ درصد متوقف شود اما AT-UNet به دقت های بالاتر هم بتواند برسد.

۲ - ماسک RGB



شکل ۱۲۱ - شبکه UNet

۱۲۲ - شبکه TA-UNet

در این تصویر میبینیم که روی هر سه معیار اصلی هر دو شبکه به خوبی در حال یادگیری ماسک های رنگی هستند اما، این آموزش همچنان میتواند ادامه یابد و هنوز به انتهای خودش نرسیده است، میبینیم که دقت و IOU_coef نسبت به بخش قبل به وضوح کمتر هستند که دو دلیل دارد، اول اینکه این شبکه طبق مقاله برای ماسک های باینری بهینه شده است و برای این نوع از تسک ادعایی ندارد، و از طرفی تسک ذاتا بسیار سخت تر است و یادگیری آن نیازمند مقدار بسیار بیشتری از منابع محاسباتی برای ادامه آموزش است، من مبنای IOU_Coeff را نیز بر IOU را خودم طبق فرمول مقاله پیاده سازی کردم قرار داده ام زیرا mean IOU که در کتابخانه تنسورفلو موجود است، در هر شرایطی عملکرد نوسانی داشت و تنها برای مقایسه آن را ذکر کرده ام، تصاویر خروجی مدل نیز نشان میدهد که IOU_coeff نماد بهتری از موفقیت مدل در تسک خودش است.

اما وقتی به مقایسه معیار ها میرسیم، مکانیزم Attention اینجا خودش را نشان میدهد، این تسلیل سخت بود ذاتی که دارد، شبکه AT-UNet توانسته عملکرد بسیار بهتری در تعداد Epoch مساوی نسبت به UNet داشته باشد، و این یعنی این شبکه توانایی یادگیری بسیار بیشتری از خود به نمایش گذاشته است.

	Accuracy	IOU_coef
UNet	۷۳.۷۰	۰.۴۵
AT-UNet	۷۸.۷۴٪	۰.۵۴

شکل ۱۲۳ - مقایسه دو شبکه روی دیتاست تست

در نهایت بدیهی است که شبکه AT-UNet را برای تسلیل Segmentation انتخاب میکنیم ، چرا که به خوبی توانسته از عهده هر دو نوع ماسک رنگی و سیاه و سفید برآید، و مکانیزم Attention به آن کمک میکند، تا پیکسل های همسایه در آن دو کanalی که Attention هر بار لحاظ میشود در نظر گرفته شود، نتایج نیز حاکی توان بیشتر این شبکه برای یادگیری است.