به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

پرسش ۱ و ۲	نام و نام خانوادگی	علی خرم فر
	رایانامه	khoramfar@ut.ac.ir
پرسش ۱ و ۲	نام و نام خانوادگی	علی رمضانی
	رایانامه	Ali.ramezani.۹۶@ut.ac.ir
	تاریخ ارسال پاسخ	۱۴۰۳/۰۳/۲۳

- هر دو نویسنده در هر دو پرسش همکاری داشته اند

- **فهرست**

فهرست تصاویر

فهرست جداول:

پرسش۱ – تشخیص اخبار جعلی مبتنی بر مدل های ترنسفورمر

۱-۱.آشنایی با BERT و CT-BERT

۱- استفاده از رویکرد یادگیری انتقالی در تسک های پردازش زبان طبیعی باعث ایجاد انقلاب مهمی شده است، ابتدا بهتر است به بررسی این تسک پرداخته و سپس مزیت های آن را توضیح دهیم:

یادگیری انتقالی: در این روش ابتدا یک مدل زبانی پایه(foundation model) با مقدار زیادی هزینه محاسباتی و دیتا در domain های مختلفی آموزش میبند، بدیهی است که این مدل در هیچ domain خاصی متخصص نمیشود، بلکه به سبب دیدن دیتای مختلف در زمینه های مختلف، ساختار زبان را به خوبی یادگرفته، همچنین اطلاعات کلی در زمینه های مختلفی کسب میکند، سپس این دانش مبنای آموزش مدل روی domain های تخصصی تری قرار میگیرد به طوری که از این مدل به عنوان مدل پایه استفاده شده و با افزودن لایه های بالایی به مدل برای classification و یا هر نوع لایه ای که مناسب downstream task تسک ها باشد، مدل با وزن های اولیه یادگرفته شده مدل پایه مقداردهی شده و سپس به آموزش در آن زمینه خاص روی دیتاست مورد نظر اصطلاحا فاین تیون میشود، این آموزش خود به دو نوع است که میتواند وزن های مدل پایه را تغییر دهد(فاین تیون) و یا وزن های مدل پایه را فریز کرده و تنها لایه های downstream آموزش ببیند(feature-base)

استفاده از یادگیری انتقالی زمانی توصیه میشود که اولا دیتای مورد نظر در آن زمینه خاص کم باشد، همچنین محدودیت منابع محاسباتی نیز وجود داشته باشد، البته از آنجایی که تسک یادگیری زبان طبیعی اساسا تسک بسیار سختی است، پس ما ناگزیریم در این زمینه در عمده موارد از مدل های پایه(foundation model) که ساختار زبان انسانی را آموخته اند استفاده کنیم و بر پایه ی آن ها کار را گسترش دهیم.

۲ – در مورد تفاوت این دو رویکرد به صورت مختصر در قسمت قبل توضیح داده شد، اما اگر بخواهیم بیشتر و دقیق تر آن ها را مقایسه کنیم:

- رویکرد Fine-tunning: در این روش ما وزن های مدل پایه را نیز آپدیت میکنیم، با این کار مدل، دامنه دیتاست مورد نظر ما را به خوبی یادمیگیرد اما ریسک این کار در overfitting مدل به دلیل تعداد پارامتر های زیاد مدل است، همچنین در این روش لازم است از دیتای نسبتا زیادی برخوردار باشیم و همچنین منابع محاسباتی بیشتری نیز صرف کنیم.
- رویکرد feature-base: در این روش، وزن های مدل پایه فریز شده و آن ها آپدیت نمیشوند، این روش معمولا در برابر overfitting مقاوم است، اما در عین حال توانایی مدل برای یادگیری نیز محدود است چرا که تعداد پارامترهای آموزش مدل محدود است، اما نیاز به دیتای زیادی ندارد و

در عین حال از نظر محاسباتی نیز به صرفه است. این روش زمانی که توزیع دیتاست به توزیع داده اولیه ای مدل روی آن آموزش دیده است نزدیک باشد، میتواند به دقت خوبی برسد.

۱-۲. دادگان:

ابتدا به آماده سازی دادگان مورد نیاز پرداختیم، همانطور که مقاله اشاره میکند، در هنگام استفاده از مدل های زبانی، به دلیل داشتن توکنایزر تخصصی در کنار هر مدل، تغییرات زیادی در دیتا مورد نیاز نیست بلکه طبق مقاله ابتدا به جایگیزنی اموجی ها به متن آن ها پرداختیم و سپس با استفاده از توکنایزر مدل، متن را توکنایز کرده و طول آن را به ۱۲۸ محدود کردیم، مراحل آماده سازی دیتا را به ترتیب شرح خواهیم داد:

- خواندن دیتاست:

```
train_data = '/kaggle/input/dl5-dataset/Train.csv'
valid_data = '/kaggle/input/dl5-dataset/Val.csv'
test_data = '/kaggle/input/dl5-dataset/Test.csv'

train_df = pd.read_csv(train_data)
val_df = pd.read_csv(valid_data)
test_df = pd.read_csv(test_data)
```

+ Code + Markdown

```
train_df.head()
```

	id	tweet	label
0	1	The CDC currently reports 99031 deaths. In gen...	real
1	2	States reported 1121 deaths a small rise from ...	real
2	3	Politically Correct Woman (Almost) Uses Pandem...	fake
3	4	#IndiaFightsCorona: We have 1524 #COVID testin...	real
4	5	Populous states can generate large case counts...	real

شکل ۱- خواندن دیتاست

- جایگزینی اموجی ها و نگاشت برچسب های متنی به عدد:

```
train_tweets = train_df['tweet'].apply(demojize).values
train_labels = train_df['label'].apply(lambda x: 1 if x == 'real' else 0).values

val_tweets = val_df['tweet'].apply(demojize).values
val_labels = val_df['label'].apply(lambda x: 1 if x == 'real' else 0).values

test_tweets = test_df['tweet'].apply(demojize).values
test_labels = test_df['label'].apply(lambda x: 1 if x == 'real' else 0).values
```

شکل ۲ - جایگزینی اموجی و لیبل ها

- سپس به توکنایز کردن متن حین ساختن دیتاست پرداختیم:

```
class TweetDataset(Dataset):
    def __init__(self, tweets, labels, tokenizer, max_len):
        self.tweets = tweets
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.tweets)

    def __getitem__(self, item):
        tweet = str(self.tweets[item])
        label = self.labels[item]

        encoding = self.tokenizer.encode_plus(
            tweet,
            add_special_tokens=True,
            max_length=self.max_len,
            truncation=True,
            padding='max_length',
            return_token_type_ids=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'tweet_text': tweet,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'token_type_ids':encoding['token_type_ids'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }
```

در تصویر روبه رو در کادر قرمز رنگ میبینم که با استفاده از متد tokenizer.encode_plus به توکنایز کردن دیتا پرداختیم، ماکسیمم طول را ۱۲۸ محدود کردیم و توییت های بزرگ تر را truncate کرده و توییت های کوچک تر را نیز pad کردیم.

شکل ۳ - توکنایز کردن دیتاست

سپس در نهایت به ساختن دیتالودر مناسب مدل با سایز بچ ۴ که در مقاله مناسب تشخیص داده شده بود اقدام کردیم.

```
MAX_LEN = 128
BATCH_SIZE = 4

train_dataset = TweetDataset(
    tweets=train_tweets,
    labels=train_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

val_dataset = TweetDataset(
    tweets=val_tweets,
    labels=val_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

test_dataset = TweetDataset(
    tweets=test_tweets,
    labels=test_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

شکل ۴ - دیتالودر مناسب برای مدل

- در نهایت ساختار دیتای مورد نظر که آماده دریافت توسط مدل باشد آماده شد:

```
data = next(iter(train_loader))
print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['token_type_ids'].shape)
print(data['labels'].shape)
```

```
torch.Size([4, 128])
torch.Size([4, 128])
torch.Size([4, 128])
torch.Size([4])
```

شکل ۵ - دیتای ورودی به مدل

۳-۱. پیاده سازی با رویکرد fine tunning:

در این بخش به پیاده سازی سه مدل به رویکرد فاین تیونینگ میپردازیم:

- مدل BERT که روی توکن CLS، تسک کلاسیفیکشن انجام میدهیم:
  - ساختار مدل: طبق ساختار پیشنهادی مقاله از مدل bert-base استفاده کردیم، همچنین لایه خروجی را نیز به تابع sigmoid دادیم که در مقاله بر آن تاکید شده بود.

```
class BertBaseClassifier(nn.Module):
    def __init__(self, n_classes=1):
        super(BertBaseClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        pooled_output = outputs.pooler_output
        logits = self.out(pooled_output)
        return self.sigmoid(logits)
```

شکل ۶ - ساختار مدل **Bert**

  - سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: در مقاله اشاره شد که

```
EPOCHS = 3
bert_finetune_model = BertBaseClassifier(n_classes=1).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(bert_finetune_model.parameters(), lr=2e-5)

bert_finetune_history = train(bert_finetune_model, loss_fn, optimizer, EPOCHS)
```

شکل ۷ - پارامتر های آموزش مدل **BERT**

تعداد epoch باید ۳ باشد، و همچنین از تابع هزینه binary cross entroy با استفاده از آپتیمایزر Adam با lr = ۲*۱۰^(-۵) در مدل bert استفاده شود که سعی کردیم دقیقا طبق مقاله باشد.

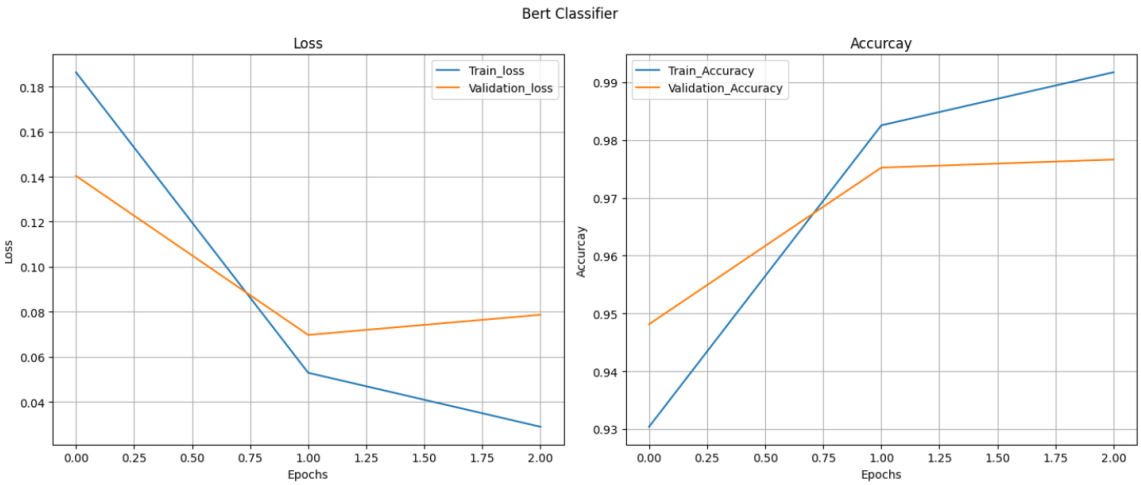
  - تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۱۰۹ میلیون پارامتر

```
count_parameters(bert_finetune_model)
```

```
109483009
```

شکل ۸ - تعداد پارامتر های آموزش مدل **BERT**

- نمودار های آموزش مدل:

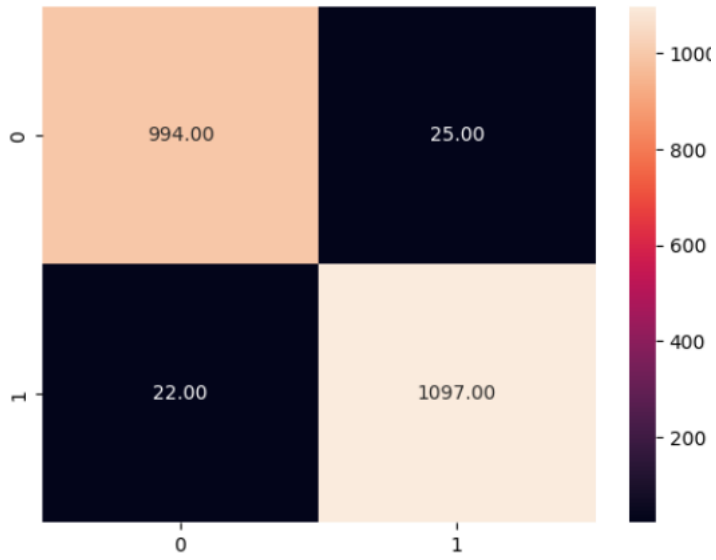


شکل ۹ - نمودار آموزش مدل **BERT** در فاین تیون

- نتایج داده های تست:

Classification Report:	precision	recall	f1-score	support
fake	0.98	0.98	0.98	1019
real	0.98	0.98	0.98	1119
accuracy			0.98	2138
macro avg	0.98	0.98	0.98	2138
weighted avg	0.98	0.98	0.98	2138

شکل ۱۰ – گزارش طبقه بندی آموزش مدل **BERT-Finetune**



شکل ۱۱- ماتریس درهم ریختگی آموزش مدل **BERT-Finetune**

- مدل BERT + BiGRU که آخرین لایه مدل را به یک لایه خطی داده و خروجی را به صورت احتمال sigmoid محاسبه میکنیم:
  - ساختار مدل: از تکرار مکررات و بخش هایی که در قبل آمده میپرهیزیم اما در این ساختار، اولا از GRU دو طرفه استفاده کردیم و به همین دلیل در لایه ی خطی از ابعاد دو برابر hidden-dim استفاده کردیم، همچنین خروجی مدل GRU روی آخرین بعد تنسور concat شده و سپس به مدل خطی داده میشود.

```python
class BertBiGRUClassifier(nn.Module):
    def __init__(self, n_classes=1, hidden_dim=128, num_layers=1, bidirectional=True):
        super(BertBiGRUClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.gru = nn.GRU(self.bert.config.hidden_size, hidden_dim, num_layers=num_layers, bidirectional=bidirectional, batch_first=True)
        self.out = nn.Linear(hidden_dim * 2, n_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask, token_type_ids):

        bert_outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        sequence_output = bert_outputs.last_hidden_state  # (batch_size, sequence_length, hidden_size)
        gru_output, hidden = self.gru(sequence_output)  # hidden: (num_layers * num_directions, batch_size, hidden_dim)
        final_hidden_state = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)  # (batch_size, hidden_dim * 2)
        output = self.out(final_hidden_state)

        return self.sigmoid(output)
```

شکل ۱۲ - ساختار مدل **BERT + BiGRU** در **Fine-Tunning**

  - سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: به تکرار پارامتر هایی که با بخش قبل یکی باشند نمیپردازیم، اما در این بخش پارامتر های BI-GRU را داریم که با کادر قرمز مشخص شده اند.

```python
hidden_dim = 128
num_layers = 1
bidirectional=True
n_classes = 1
bert_gru_finetune_model = BertBiGRUClassifier(n_classes, hidden_dim, num_layers, bidirectional).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(bert_gru_finetune_model.parameters(), lr=2e-5)

bert_gru_finetune_history = train(bert_gru_finetune_model, loss_fn, optimizer, EPOCHS = 3)
```

شکل ۱۳ - پارامتر های آموزش مدل **BERT + BiGRU**

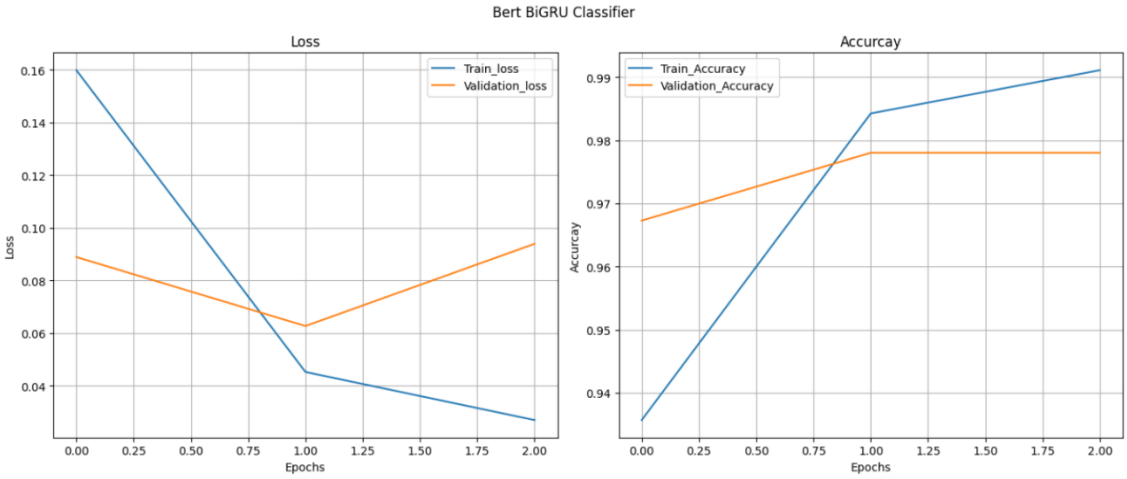
  - تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۱۱۰ میلیون پارامتر

```python
count_parameters(bert_gru_finetune_model)
```

```
110172161
```

شکل ۱۴ - تعداد پارامتر های آموزش مدل **BERT + BiGRU**

- نمودار های آموزش مدل:

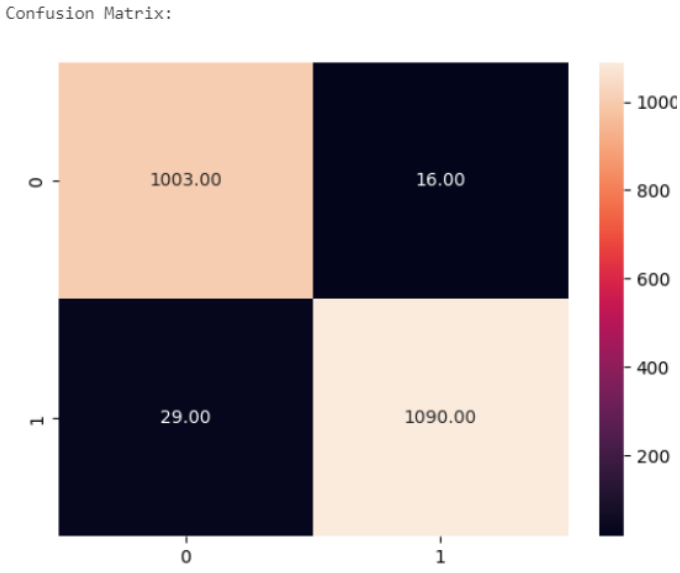


شکل ۱۵ - نمودار آموزش مدل **BERT+BiGRU** در فاین تیون

- نتایج داده های تست:

	precision	recall	f1-score	support
fake	0.97	0.98	0.98	1019
real	0.99	0.97	0.98	1119
accuracy			0.98	2138
macro avg	0.98	0.98	0.98	2138
weighted avg	0.98	0.98	0.98	2138

شکل ۱۶ – گزارش طبقه بندی آموزش مدل **BERT+BiGRU-Finetune**



شکل ۱۷- ماتریس درهم ریختگی آموزش مدل **BERT+BiGRU-Finetune**

- مدل CT-BERT + BiGRU که آخرین لایه مدل را به یک لایه خطی داده و خروجی را به صورت احتمال sigmoid محاسبه میکنیم:
  - در این ساختار مجددا توکنایزر خود مدل را فراخوانی کردیم و داده ها را بر اساس توکنایزر همین مدل خاص توکنایز کردیم.

```
tokenizer = AutoTokenizer.from_pretrained("digitalepidemiologylab/covid-twitter-bert")
```

شکل ۱۸- توکنایزر مدل **CT-BERT**

بدیهی است که مراحل ساخت دیتاست و دیتالودر تکرار شد:

```
MAX_LEN = 128
BATCH_SIZE = 4

train_dataset = TweetDataset(
    tweets=train_tweets,
    labels=train_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

val_dataset = TweetDataset(
    tweets=val_tweets,
    labels=val_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

test_dataset = TweetDataset(
    tweets=test_tweets,
    labels=test_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

شکل ۱۹- دیتاست و دیتالودر مدل **CT-BERT**

  - ساختار مدل: از تکرار مکررات و بخش هایی که در قبل آمده میپرهیزیم اما در این ساختار، تنها تفاوت مهم نسبت به بخش قبل جایگزینی CT-BERT به جای BERT به عنوان مدل پایه است.

```
class CT_BertBiGRUClassifier(nn.Module):
    def __init__(self, n_classes=1, hidden_dim=128, num_layers=1, bidirectional=True):
        super(CT_BertBiGRUClassifier, self).__init__()
        self.ctbert = AutoModel.from_pretrained("digitalepidemiologylab/covid-twitter-bert")
        self.gru = nn.GRU(self.ctbert.config.hidden_size, hidden_dim, num_layers=num_layers, bidirectional=bidirectional, batch_first=True)
        self.out = nn.Linear(hidden_dim * 2, n_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask, token_type_ids):

        ctbert_outputs = self.ctbert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        sequence_output = ctbert_outputs.last_hidden_state  # (batch_size, sequence_length, hidden_size)
        gru_output, hidden = self.gru(sequence_output)  # hidden: (num_layers * num_directions, batch_size, hidden_dim)
        final_hidden_state = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)  # (batch_size, hidden_dim * 2)
        output = self.out(final_hidden_state)

        return self.sigmoid(output)
```

شکل ۲۰ - ساختار مدل **CT-BERT + BiGRU** در **Fine-Tunning**

- سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: به تکرار پارامتر هایی که با بخش قبل یکی باشند نمیپردازیم، در این بخش تنها، نرخ یادگیری بر اساس پیشنهاد مقاله برای مدل CT-BERT متفاوت است

```
hidden_dim = 128
num_layers = 1
bidirectional=True
n_classes = 1
ctbert_finetune_model = CT_BertBiGRUClassifier(n_classes, hidden_dim, num_layers, bidirectional).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(ctbert_finetune_model.parameters(), lr=1e-5)

ctbert_finetune_history = train(ctbert_finetune_model, loss_fn, optimizer, EPOCHS = 3)
```

شکل ۲۱ - پارامتر های مدل **CT-BERT + BiGRU** در **Fine tunning**

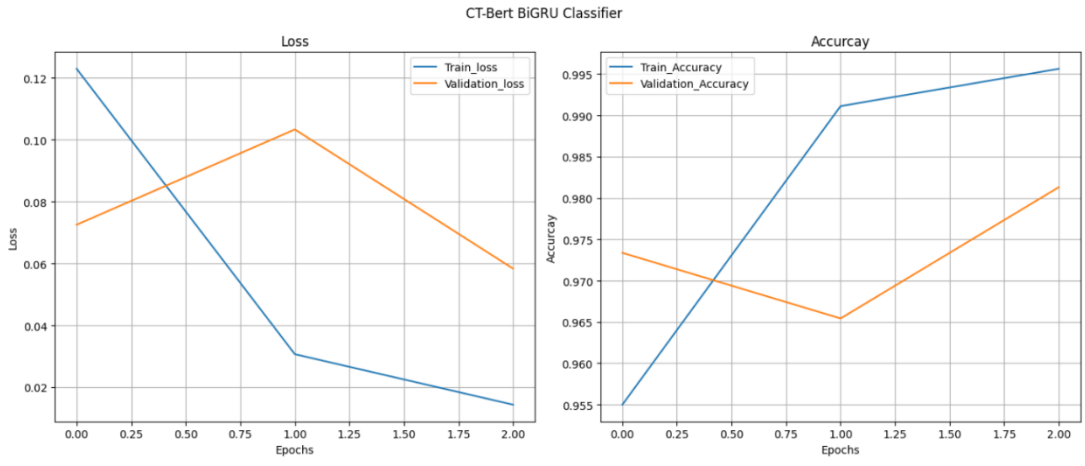
- تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۳۳۶ میلیون پارامتر

```
count_parameters(ctbert_finetune_model)
```

```
336028417
```

شکل ۲۲ - تعداد پارامتر های آموزش مدل**CT-BERT + BiGRU**

- نمودار های آموزش مدل:



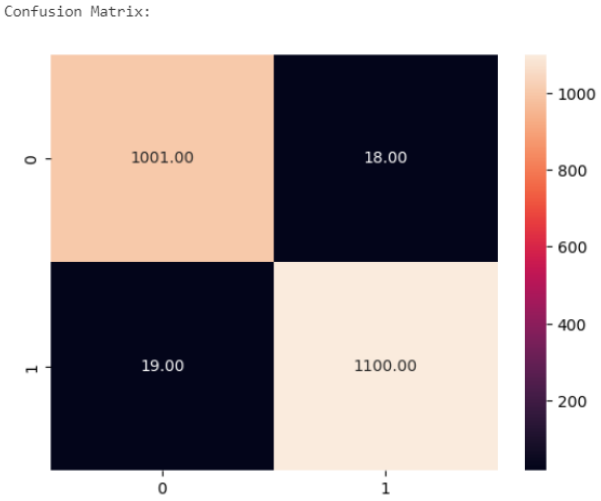
شکل ۲۳ - نمودار آموزش مدل **Ct-BERT+BiGRU** در فاین تیون

- نتایج داده های تست:

Classification Report:

	precision	recall	f1-score	support
fake	0.98	0.98	0.98	1019
real	0.98	0.98	0.98	1119
accuracy			0.98	2138
macro avg	0.98	0.98	0.98	2138
weighted avg	0.98	0.98	0.98	2138

شکل ۲۴ – گزارش طبقه بندی آموزش مدل **CT-BERT+BiGRU-Finetune**



شکل ۲۵- ماتریس درهم ریختگی آموزش مدل **CT-BERT + BiGRU-Finetune**

۱-۴. پیاده سازی با رویکرد Feature-Base:

در این بخش به پیاده سازی سه مدل به رویکرد Feature-Base میپردازیم، در این بخش سعی میکنیم از تکرار توضیحات جلوگیری کرده و تنها تفاوت های هر مدل با بخش قبل را نمایان کنیم.

- مدل BERT که روی توکن CLS، تسک کلاسیفیکشن انجام میدهیم:
  - ساختار مدل: مدل با بخش قبل تفاوتی ندارد، و تنها لایه های بخش BERT فریز شده اند.

```
class BertBaseClassifier(nn.Module):
    def __init__(self, n_classes):
        super(BertBaseClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
        self.sigmoid = nn.Sigmoid()

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

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids= token_type_ids)
        pooled_output = outputs.pooler_output
        output = self.out(pooled_output)
        return self.sigmoid(output)
```

شکل ۲۶ - ساختار مدل **Feature base Bert –**

  - سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: در این بخش نیز تنها تفاوت اصلی، محدود کردن آپتیامیزر به پارامتر هایی است که نیاز به گرفتن گرادیان به آن ها وجود دارد

```
bert_plm_model = BertBaseClassifier(n_classes=1).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, bert_plm_model.parameters()), lr=2e-5)

bert_plm_history = train(bert_plm_model, loss_fn, optimizer, EPOCHS = 3)
```

شکل ۲۷ - پارامتر های آموزش مدل **BERT – Feature Base**

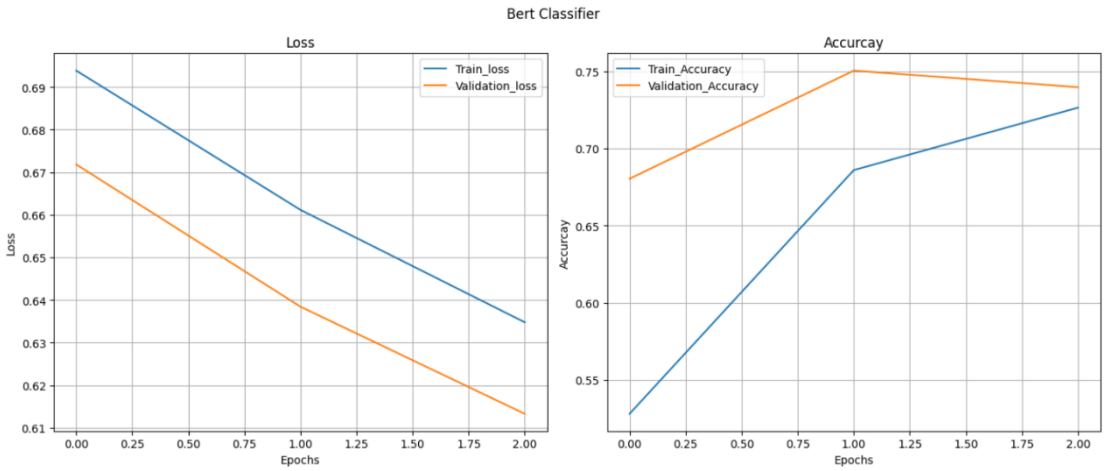
  - تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۷۶۹ عدد که بسیار بسیار محدود است.

```
count_parameters(bert_plm_model)
```

```
769
```

شکل ۲۸ - تعداد پارامتر های آموزش مدل **BERT – Feature base**

- نمودار های آموزش مدل:

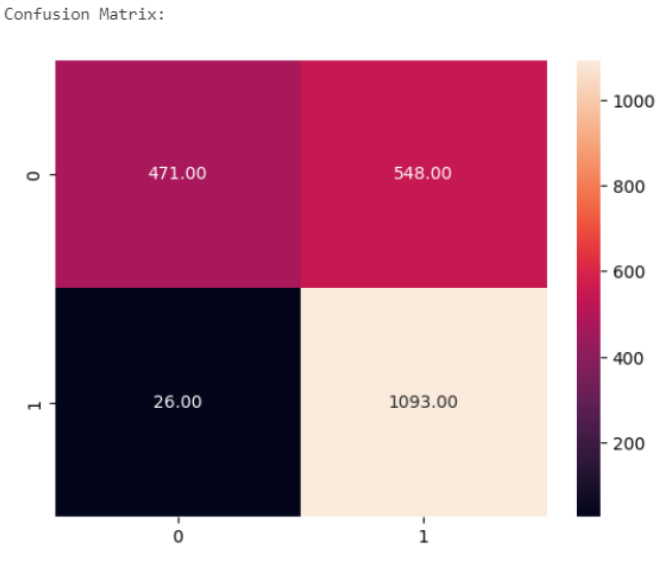


شکل ۲۹ - نمودار آموزش مدل **BERT** در **feature base**

- نتایج داده های تست:

```
...   Classification Report:
                  precision    recall  f1-score   support

              fake       0.95      0.46      0.62      1019
              real       0.67      0.98      0.79      1119

          accuracy                           0.73      2138
         macro avg       0.81      0.72      0.71      2138
      weighted avg       0.80      0.73      0.71      2138
```

شکل ۳۰ – گزارش طبقه بندی آموزش مدل **BERT- Feature Base**



شکل ۳۱- ماتریس درهم ریختگی آموزش مدل **BERT-Feature base**

- مدل BERT + BiGRU که آخرین لایه مدل را به یک لایه خطی داده و خروجی را به صورت احتمال sigmoid محاسبه میکنیم:
  - ساختار مدل: تفاوت اصلی با بخش متناظر قبل، محدود کردن لایه های آموزش از Bert است که آن ها فریز کرده ایم.

```
class BertBiGRUClassifier(nn.Module):
    def __init__(self, n_classes, hidden_dim=128, num_layers=1, bidirectional=True):
        super(BertBiGRUClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.gru = nn.GRU(self.bert.config.hidden_size, hidden_dim, num_layers=num_layers, bidirectional=bidirectional, batch_first=True)
        self.out = nn.Linear(hidden_dim * 2, n_classes)
        self.sigmoid = nn.Sigmoid()

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

    def forward(self, input_ids, attention_mask, token_type_ids):

        bert_outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        sequence_output = bert_outputs.last_hidden_state  # (batch_size, sequence_length, hidden_size)
        gru_output, hidden = self.gru(sequence_output)  # hidden: (num_layers * num_directions, batch_size, hidden_dim)
        final_hidden_state = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)  # (batch_size, hidden_dim * 2)
        output = self.out(final_hidden_state)

        return self.sigmoid(output)
```

شکل ۳۲- ساختار مدل **BERT + BiGRU** در **Feature-base**

  - سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: که آپتیامیزر تنها محدود به پارامتر هایی است که نیاز به گرادیان دارند.

```
hidden_dim = 128
num_layers = 1
bidirectional=True
n_classes = 1
bert_gru_plm_model = BertBiGRUClassifier(n_classes, hidden_dim, num_layers, bidirectional).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, bert_gru_plm_model.parameters()), lr=2e-5)

bert_gru_plm_history = train(bert_gru_plm_model, loss_fn, optimizer, EPOCHS = 3)
```

شکل ۳۳ - پارامتر های آموزش مدل**BERT + BiGRU - Fine tunning**

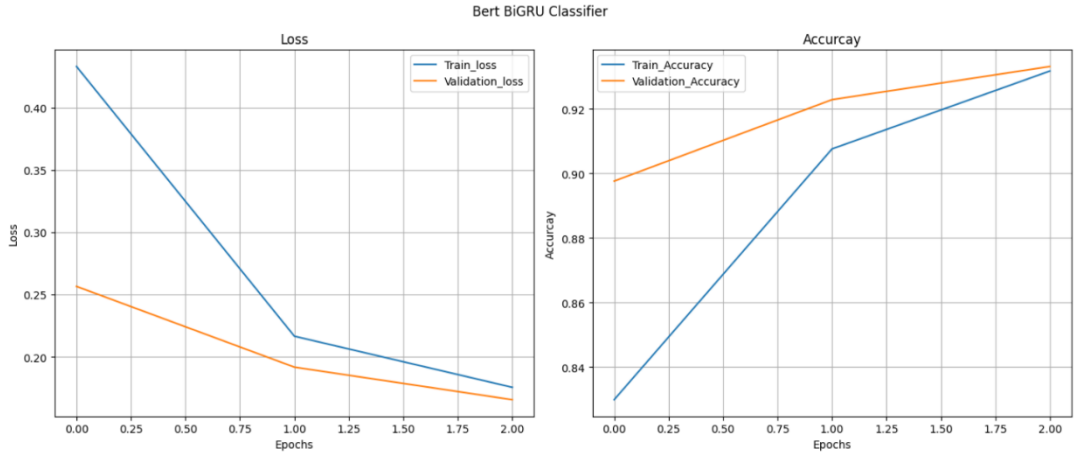
  - تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۶۸۹ هزار پارامتر

```
count_parameters(bert_gru_plm_model)
```

```
689921
```

شکل ۳۴ - تعداد پارامتر های آموزش مدل **BERT + BiGRU Feature base**

- نمودار های آموزش مدل:

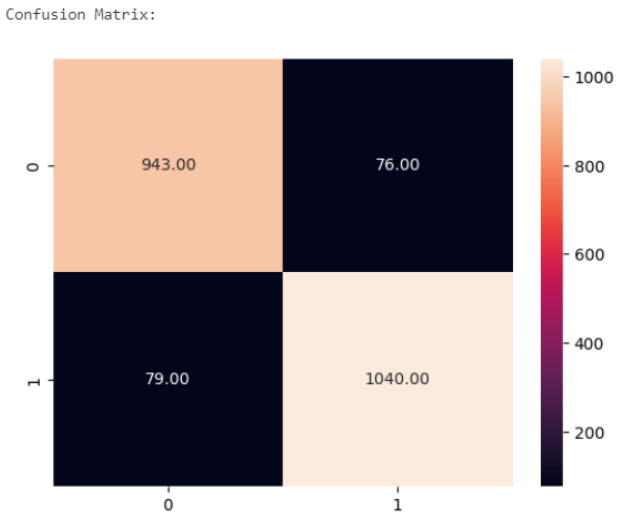


شکل ۳۵ - نمودار آموزش مدل **BERT+BiGRU** در **Feature base**

- نتایج داده های تست:

Classification Report:	precision	recall	f1-score	support
fake	0.92	0.93	0.92	1019
real	0.93	0.93	0.93	1119
accuracy			0.93	2138
macro avg	0.93	0.93	0.93	2138
weighted avg	0.93	0.93	0.93	2138

شکل ۳۶ – گزارش طبقه بندی آموزش مدل **BERT+BiGRU-feature base**



شکل ۳۷- ماتریس درهم ریختگی آموزش مدل **BERT+BiGRU-Feature base**

- مدل CT-BERT + BiGRU که آخرین لایه مدل را به یک لایه خطی داده و خروجی را به صورت احتمال sigmoid محاسبه میکنیم:
  - در این ساختار مجددا توکنایزر خود مدل را فراخوانی کردیم و داده ها را بر اساس توکنایزر همین مدل خاص توکنایز کردیم.

```
tokenizer = AutoTokenizer.from_pretrained("digitalepidemiologylab/covid-twitter-bert")
```

شکل ۳۸توکنایزر مدل **CT-BERT – Fature base**

بدیهی است که مراحل ساخت دیتاست و دیتالودر تکرار شد:

```
MAX_LEN = 128
BATCH_SIZE = 4

train_dataset = TweetDataset(
    tweets=train_tweets,
    labels=train_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

val_dataset = TweetDataset(
    tweets=val_tweets,
    labels=val_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)

test_dataset = TweetDataset(
    tweets=test_tweets,
    labels=test_labels,
    tokenizer=tokenizer,
    max_len=MAX_LEN
)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

شکل ۳۹- دیتاست و دیتالودر مدل **CT-BERT – fature base**

  - ساختار مدل: از تکرار مکررات و بخش هایی که در قبل آمده میپرهیزیم اما در این ساختار، تنها تفاوت مهم نسبت به بخش قبل فریز کردن مدل CT-Bert است.

```
class CT_BertBiGRUClassifier(nn.Module):
    def __init__(self, n_classes, hidden_dim=128, num_layers=1, bidirectional=True):
        super(CT_BertBiGRUClassifier, self).__init__()
        self.ctbert = AutoModel.from_pretrained("digitalepidemiologylab/covid-twitter-bert")
        self.gru = nn.GRU(self.ctbert.config.hidden_size, hidden_dim, num_layers=num_layers, bidirectional=bidirectional, batch_first=True)
        self.out = nn.Linear(hidden_dim * 2, n_classes)
        self.sigmoid = nn.Sigmoid()

        for param in self.ctbert.parameters():
            param.requires_grad = False

    def forward(self, input_ids, attention_mask, token_type_ids):

        ctbert_outputs = self.ctbert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        sequence_output = ctbert_outputs.last_hidden_state  # (batch_size, sequence_length, hidden_size)
        gru_output, hidden = self.gru(sequence_output)  # hidden: (num_layers * num_directions, batch_size, hidden_dim)
        final_hidden_state = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)  # (batch_size, hidden_dim * 2)
        output = self.out(final_hidden_state)

        return self.sigmoid(output)
```

شکل ۴۰ - ساختار مدل **CT-BERT + BiGRU** در **Feature base**

سپس به آموزش مدل با پارامتر های مقاله به صورت زیر پرداختیم: که در این بخش آپتیمایزر تنها گرادیان پارامتر های غیر فریز شده را حساب میکند.

```
hidden_dim = 128
num_layers = 1
bidirectional=True
n_classes = 1
ct_bert_plm_model = CT_BertBiGRUClassifier(n_classes, hidden_dim, num_layers, bidirectional).to(device)
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, ct_bert_plm_model.parameters()), lr=1e-5)

ct_bert_plm_history = train(ct_bert_plm_model, loss_fn, optimizer, EPOCHS = 3)
```

شکل ۴۱ - پارامتر های مدل **CT-BERT + BiGRU** در **Feature base**

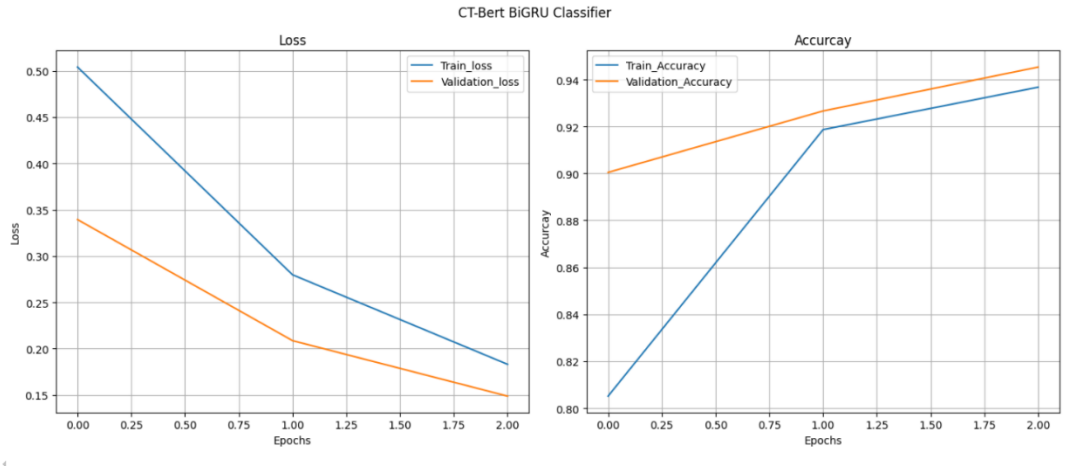
- تعداد پارامتر های آموزش این مدل عبارتند از: حدود ۸۸۶ هزار پارامتر

```
count_parameters(ct_bert_plm_model)
```

```
886529
```

شکل ۴۲ - تعداد پارامتر های آموزش مدل**CT-BERT + BiGRU** در **Fature base**

- نمودار های آموزش مدل:



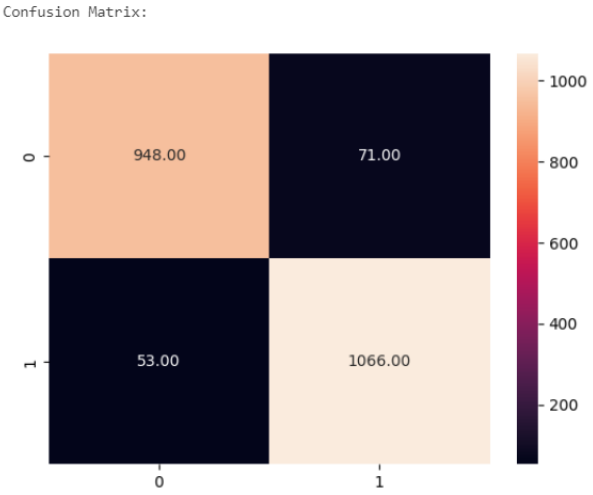
شکل ۴۳ - نمودار آموزش مدل **Ct-BERT+BiGRU** در **Feature base**

- نتایج داده های تست:

Classification Report:

	precision	recall	f1-score	support
fake	0.95	0.93	0.94	1019
real	0.94	0.95	0.95	1119
accuracy			0.94	2138
macro avg	0.94	0.94	0.94	2138
weighted avg	0.94	0.94	0.94	2138

شکل ۴۴ – گزارش طبقه بندی آموزش مدل **CT-BERT+BiGRU-Feature base**



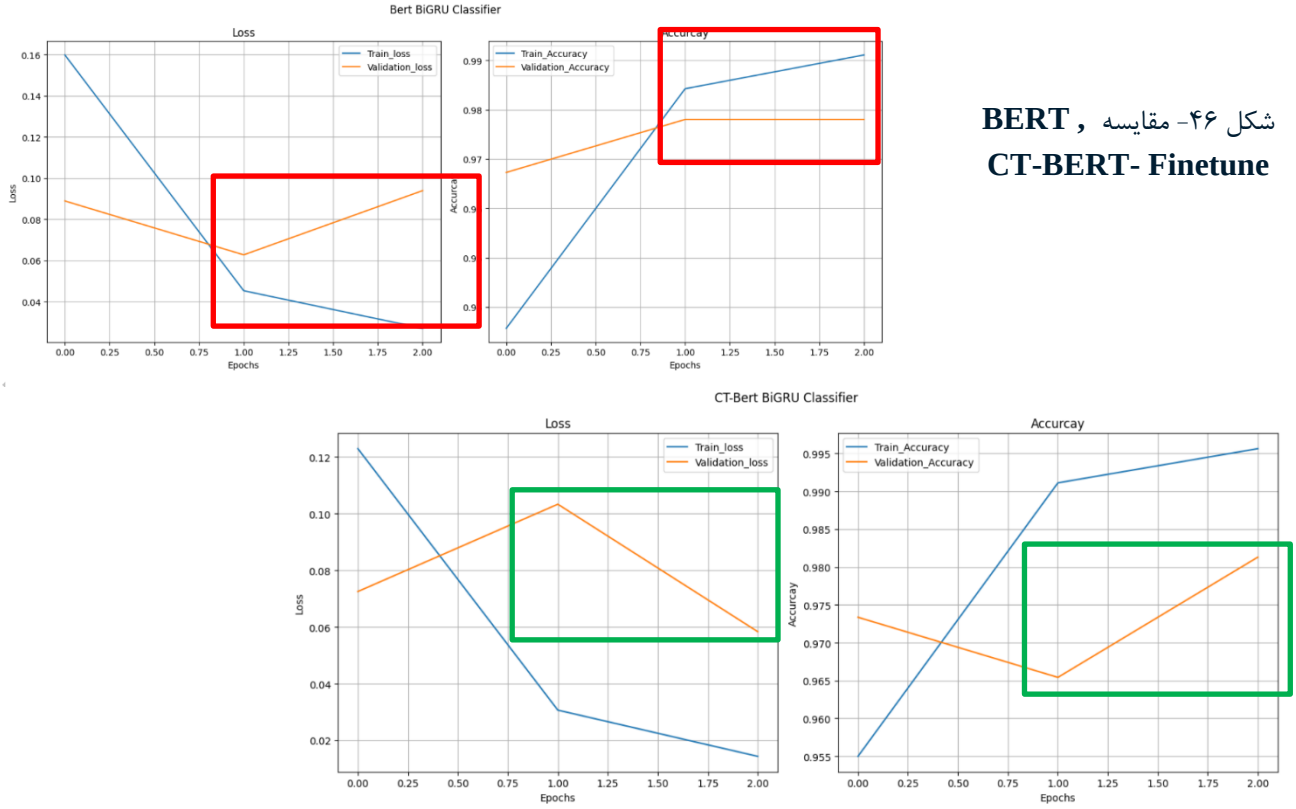
شکل ۴۵- ماتریس درهم ریختگی آموزش مدل **CT-BERT + BiGRU- Feature base**

۱-۴. تحلیل نتایج:

سوال اول: ابتدا به مقایسه مدل BERT و CT-BERT میپردازیم، این دو مقایسه یکبار در حالت Fine-tunning و بار دوم در حالت Feature-Base صورت خواهد گرفت:

- مقایسه BERT و CT-BERT در حالت Fine-Tune:

ابتدا نمودار هایی که این دو مدل به عنوان پایه برای BiGRU قرار گرفته اند را مقایسه میکنیم:



شکل ۴۶- مقایسه **BERT , CT-BERT- Finetune**

همانطور که میبینم دقت نهایی مدل و همچنین مقدار نهایی val-loss در مدلی که پایه آن CT-BERT است پایین تر است، البته وجود رفتار نوسانی بدلیل این است که تنها ۳ ایپاک به آموزش مدل ها پرداخته ایم و مدل به نظر کاملا در نمودار ها استیبل نشده است.

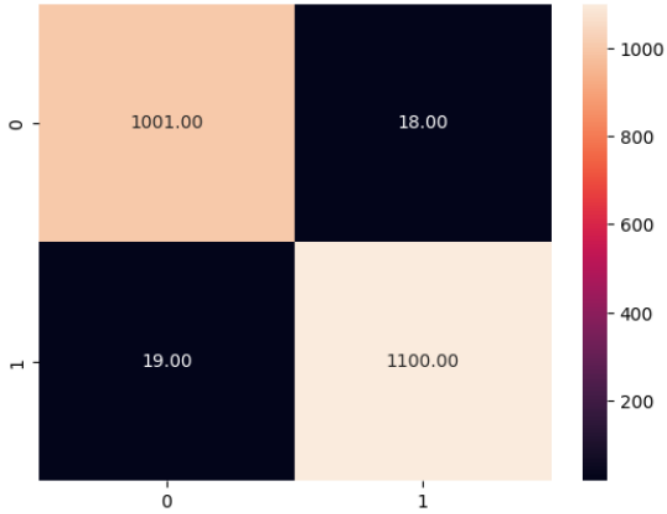
همچنین مدل CT-BERT در صورت افزایش آموزش به وضوح loss کمتر و دقت بالاتری روی مجموعه validation پیدا خواهد کرد،(کادر های سبز) در حالی که مدل با پایه Bert به نظر میرسد که دچار overfitting خواهد شد، زیرا loss ولیدیشن در حال افزایش است و دقت نیست به حد ثابتی رسیده است.(کادر های قرمز)

اگر این مقایسه را روی دقت و ماتریس آشفتگی مشاهده کنیم:

Classification Report:

	precision	recall	f1-score	support
fake	0.98	0.98	0.98	1019
real	0.98	0.98	0.98	1119
accuracy			0.98	2138
macro avg	0.98	0.98	0.98	2138
weighted avg	0.98	0.98	0.98	2138

Confusion Matrix:

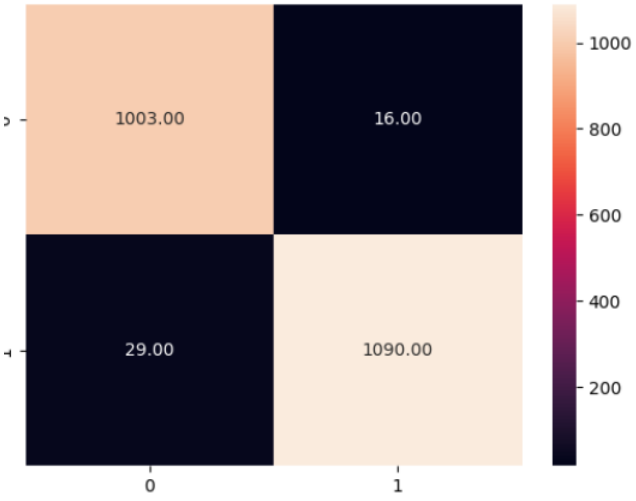


شکل ۴۸ - مدل **CT- Bert**

ssification Report:

	precision	recall	f1-score	support
fake	0.97	0.98	0.98	1019
real	0.99	0.97	0.98	1119
accuracy			0.98	2138
macro avg	0.98	0.98	0.98	2138
ghted avg	0.98	0.98	0.98	2138

fusion Matrix:



شکل ۴۷ - مدل **BERT**

در تصاویر بالا اگرچه تفاوت بسیار اندک و قابل اغماض است، اما مدل CT-BERT دارای Precision بالاتری روی داده های fake است که این بسیار مهم است، زیرا اساسا برای ما دقت روی داده های fake مهم تر است.

اما چرا اینطور است؟ دو دلیل میتواند دلیل این اتفاق باشد:

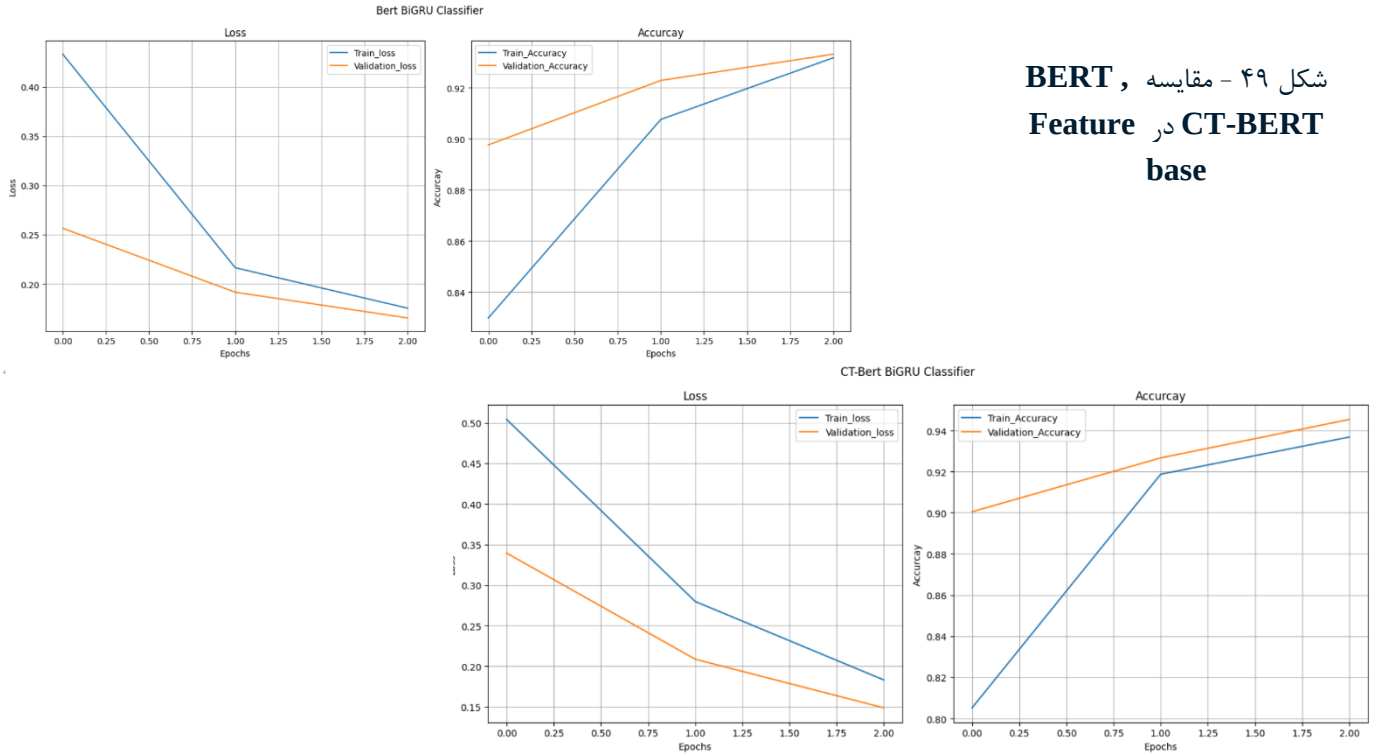
۱ – مدل CT-BERT روی همین دامنه از کلمات و توییت های آموزش دیده است، پس عملا مدل سختی کمتری برای درک توزیع این داده دارد، پس بهتر عمل میکند، همچنین به دلیل درک بالاتر از دامنه مطالب، دچار overfitting هم نمیشود و بهتر میتواند دانش کسب شده را generalize کند.

۲ – مدل CT-BERT پارامتر های آموزشی بیشتری دارد، پس توان یادگیری بالاتری را نیز از خود به نمایش میگذارد، البته باید این را در نظر گرفت که پارامتر بیشتر همیشه به نفع ما نیست، و گاهی سبب overfitting میشود، اما به دلیل یک pretrain خوب روی دامنه یکسان از مطالب، این مشکل رخ نداده است.

BERT+BiGRU	CT-BERT+BiGRU
۱۱۰,۱۷۲,۱۶۱	۳۳۶,۰۲۸,۴۱۷

جدول ۱- پارامتر های آموزش دو مدل **bert, ct-bert**

حال این دو مدل را در حالت feature base مقایسه میکنیم.



شکل ۴۹ - مقایسه **BERT ,** **CT-BERT** در **Feature base**

آنچه با مقایسه در نمودارها مشاهده میشود، اولا loss بالاتر و دقت پایین تر هر دو نسبت به حالت قبل یعنی فاین تیون می باشد، اما در این بخش نیز اگرچه هیچکدام از دو مدل دچار over-fitting نیستند، اما دقت بالاتر و loss کمتر در Ct-BERT مشهود است.

Classification Report:

	precision	recall	f1-score	support
fake	0.92	0.93	0.92	1019
real	0.93	0.93	0.93	1119
accuracy			0.93	2138
macro avg	0.93	0.93	0.93	2138
weighted avg	0.93	0.93	0.93	2138

Confusion Matrix:

شکل ۵۱ - **BERT**

Classification Report:

	precision	recall	f1-score	support
fake	0.95	0.93	0.94	1019
real	0.94	0.95	0.95	1119
accuracy			0.94	2138
macro avg	0.94	0.94	0.94	2138
weighted avg	0.94	0.94	0.94	2138

Confusion Matrix:

شکل ۵۰- **CT-BERT**

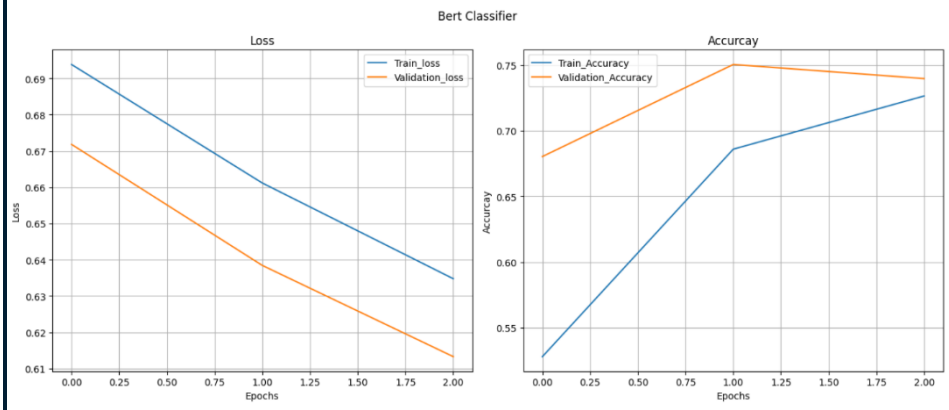
مبا مقایسه گزارش طبقه بندی و ماتریس درهم ریختگی همه چیز واضح تر میشود، میبنیم که دقت مدل ها اول حدود ۹۲ تا ۹۵ درصد است که در بهترین حالت از ۹۸ درصد مدل های قبلی دست کم ۳ درصد فاصله دارد، دوما، بین خودشان نیز مدل CT-BERT به وضوح عملکرد بهتری در همه معیار های مقایسه ای دارد.

BERT+BiGRU	CT-BERT+BiGRU
۶۸۹,۹۲۱	۸۸۶,۵۲۹

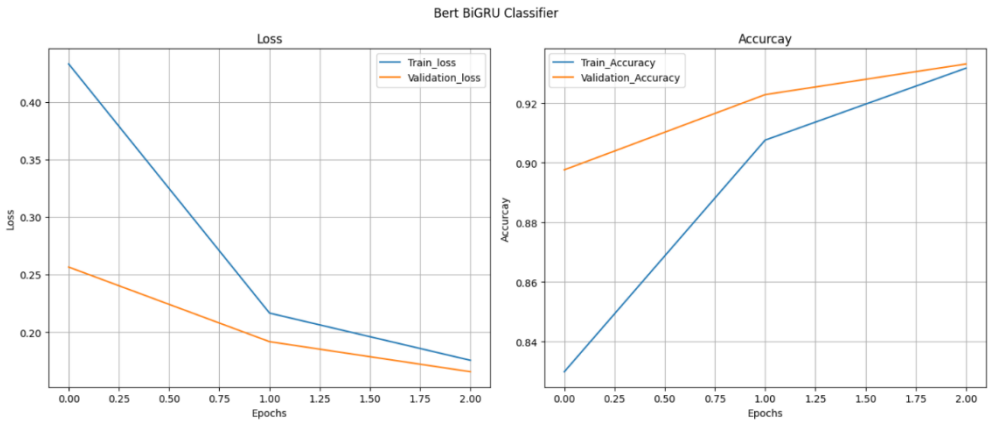
جدول ۲ -پارامتر های آموزش دو مدل **Bert+BiGRU, CT-BERT+BiGRU**

اما چرا؟ دلایل خارج از دو موردی که پیش تر بررسی کردیم نیست، چرا که هم مدل بزرگتری است و هم روی دامنه مشترکی با دیتاست pre train شده است، پس انتظار عملکرد بهتر قابل انتظار است، در این بخش نقش تفاوت BiGRU بدلیل اینکه در هر دو مدل حضور دارد، پر رنگ نیست.

سوال دوم: در اینجا به مقایسه مدل BERT و BERT + BiGRU در حالت feature base میپردازیم:



شکل ۵۲ - مقایسه **BERT - BERT+BiGRU in Feature base**

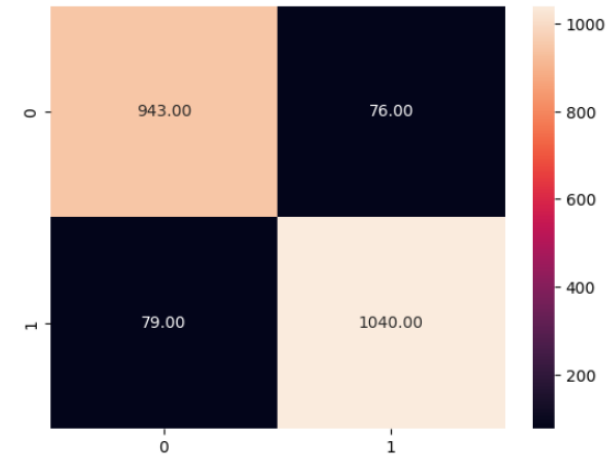


آنچه از نمودارها برداشت میشود، دقت پایین تر و loss بالاتر مدلی است که دارای BiGRU می باشد، با مقایسه دقت روی داده های تست نیز:

Classification Report:

	precision	recall	f1-score	support
fake	0.92	0.93	0.92	1019
real	0.93	0.93	0.93	1119
accuracy			0.93	2138
macro avg	0.93	0.93	0.93	2138
weighted avg	0.93	0.93	0.93	2138

Confusion Matrix:

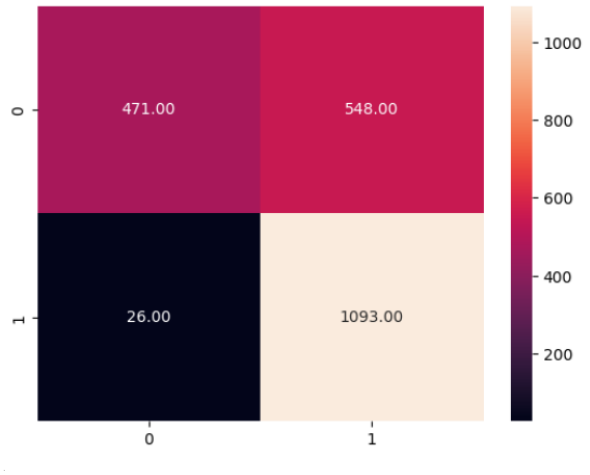


شکل ۵۳ - **Bert + BiGRU**

Classification Report:

	precision	recall	f1-score	support
fake	0.95	0.46	0.62	1019
real	0.67	0.98	0.79	1119
accuracy			0.73	2138
macro avg	0.81	0.72	0.71	2138
weighted avg	0.80	0.73	0.71	2138

Confusion Matrix:



شکل ۵۴ - **BERT**

در این بخش نیز دقت بسیار بالاتر مدل دارای لایه BiGRU مشهود است، اما دلیل چیست؟

۱ – لایه BiGRU اولا تعدادی زیاد پارامتر به مدل اضافه میکند که قدرت یادگیری مدل را به شدت افزایش میدهد، این به این معنی است که حتی اگر ساختار خاص BiGRU در پردازش متن و وابستگی کلمات پشت سر هم را نیز نبینیم از طریق تعداد پارامتر ها احتمالا درکی از اینکه کدام مدل قوی تر است خواهیم داشت.

۲- اما دلیل اصلی، به ساختار مدل های مبتنی بر RNN مثل LSTM و GRU باز میگردد، این مدل ها با پردازش sequential خروجی های BERT، از دو طرف، معنای بیشتری از لایه های عمیق تر متن استخراج میکنند و بازنمایی تولید شده dense تر و پرمحتواتر است در نتیجه بعد از تبدیل آن به یک لایه Linear و تسک classification مدل بهتر میتواند روی دیتاست عمل کند.

BERT	BERT+BiGRU
۷۶۹	۶۸۹,۹۲۱

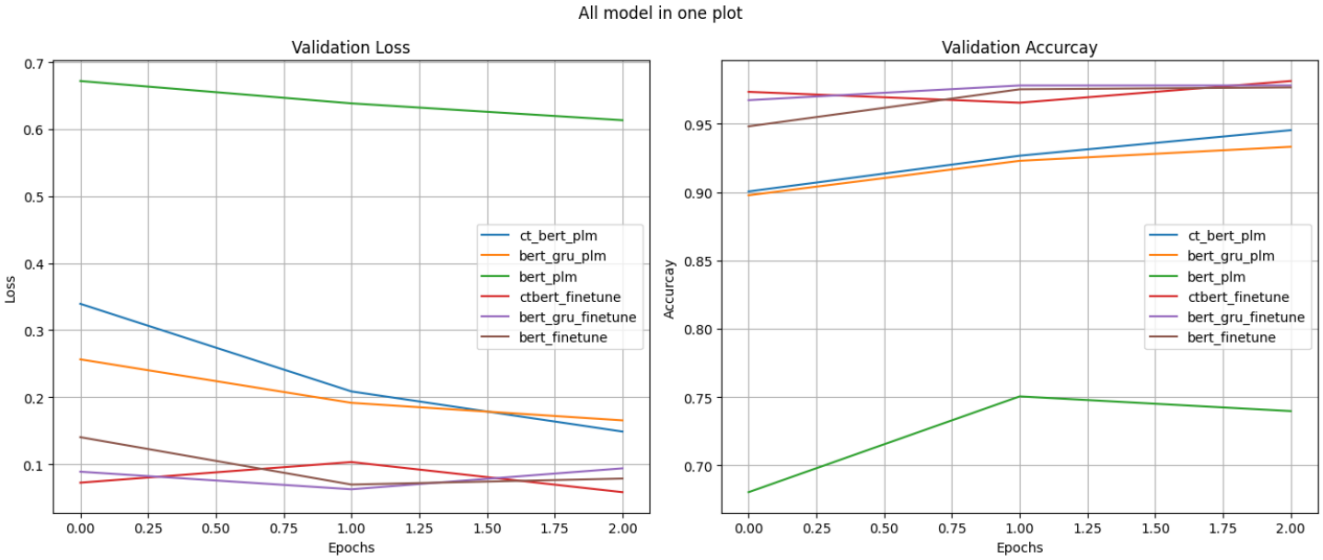
جدول ۳ -پارامتر های آموزش دو مدل **BERT, BERT+BiGRU**

سوال سوم: در این قسمت به مقایسه دو رویکرد میپردازیم:

	Accuracy, F۱-Score	Fine Tune		Feature Base	
		ACC	F۱	ACC	F۱
	BERT	۰.۹۸	۰.۹۸	۰.۷۹	۰.۶۲
	BERT + BiGRU	۰.۹۸	۰.۹۸	۰.۹۳	۰.۹۲
	CT-BERT + BiGRU	۰.۹۸	۰.۹۸	۰.۹۴	۰.۹۴

جدول ۴- مقایسه مدل های **fine-tune , feature base**

مقادیر F۱_score از داده fake که مقدار کمتری داشته، برداشت شده است.



شکل ۵۵ - نمودار مقایسه همه مدل ها

هم از جدول ارائه شده و هم از نمودار، به وضوح مشخص است که مدل های Fine-tune شده دقت بالاتری دارند و loss کمتری نیز دارند، این نمودار ها تماما روی داده های validation رسم شده است.

در نهایت نیز انتظار داشتیم چنین اتفاقی بیفتد، مدل های fine tune شده، برای یادگیری بهتر عمل میکنند و اگر دیتا و منابع محاسباتی به اندازه کافی موجود باشد، آن ها عملکرد بهتری دارند، زیرا این فرصت را پیدا میکنند که توزیع داده های دیتاست را درک کنند، در حالی که در حالت feature base مدل تسک زیربنایی را یادگرفته و تنها بر اساس یادگیری خودش، سعی میکند، feature های موثر را برای انجام تسک استخراج کند که کمتر موفق میشود.

سوال چهارم: جملات اشتباه

- مدل BERT:

۱-> Kuwait Assistant Undersecretary for Public Health Affairs Dr Buthayna Almodaf highlighted the importance of risk communication & public education in fighting #COVID19. The country increased testing capacity enabling 400000 people to be tested.
TRUE LABEL-> 1
PREDICTION-> 0.0

جمله اول: ساده است اما طولانی است و جملات طولانی به خصوص برای مدل BERT ساده، بدون BiGRU مشکل است.

۲-> Kids reach ‘f**k this shit’ stage of lockdown https://t.co/swClEhi2Iq
TRUE LABE-> 0
PREDICTION-> 1.0

جمله دوم: این جمله کوتاه است، اما گمراه کننده است، جمله دارای طنز و ساختار تو در تو است، که برای انسان ها به سادگی مشخص است یک نظر شخصی و مبتنی بر طنز است اما برای مدل درک آن مشکل است، و به نظر یک خبر واقعی بوده است.

- مدل BERT+BiGRU:

۱-> The underlying cause of death in the vast majority of death certificates that mention COVID-19 is the coronavirus. https://t.co/Hg4O4AQZ3X
TRUE LABEL-> 0
PREDICTION-> 1.0

جمله اول: دارای طنز و کنایه است و مدل به همین دلیل قادر به درک آن نشده است.

۲-> In response to the pandemic Pennsylvania Governor Tom Wolf shut down nonessential businesses and limited gatherings in May. A federal judge’s rules that these COVID restrictions were unconstitutional violating the First and Fourteenth Amendments. @thehill https://t.co/Y9fDeXe7mf
TRUE LABE-> 1
PREDICTION-> 0.0

جمله دوم: دارای اولا دارای ساختار حقوقی است و دوما، از یک اقدام فرماندار و قانونی علیه آن صحبت میکند که درک آن مشکل است، همچنین جمله طولانی است و بخش دوم به نوعی علیه بخش اول صحبت میکند

- مدل CT-BERT+BiGRU:

۱-> The underlying cause of death in the vast majority of death certificates that mention COVID-19 is the coronavirus. https://t.co/Hg4O4AQZ3X

TRUE LABEL-> 0

PREDICTION-> 1.0

۲-> In response to the pandemic Pennsylvania Governor Tom Wolf shut down nonessential businesses and limited gatherings in May. A federal judge’s rules that these COVID restrictions were unconstitutional violating the First and Fourteenth Amendments. @thehill https://t.co/Y9fDeXe7mf

TRUE LABE-> 1
PREDICTION-> 0.0

جمله اول و دوم: نکته بسیار مهم برای من این بود که آیا این دو جمله را که مدل BERT+BIGRU موفق به پاسخگویی آن نشدند، مدل CT-BERT میتواند پاسخ دهد؟ با اینکار مقایسه ما بسیار کامل تر میشود که میبینم این مدل ها در جملات طنز آمیز و کنایی خوب عمل نمیکنند و فقط ساختار دیتاست مهم نیست بلکه پیچیدگی معماری مدل که قادر به درک این نوع از جملات باشد بسیار مهم است.

پرسش۲- به کارگیری مدلهای ترنسفرمر در طبقه بندی تصاویر

مقدمه

در این پرسش قصد بررسی مدل‌های CNN و Transformer برای طبقه‌بندی تصاویر و مقایسه آن‌ها را داریم. قبل از اینکه با ترنسفورمرهای تصویر آشنا شویم و مدل ViT را بررسی کنیم ابتدا توضیحات کلی درباره شبکه‌های CNN و Transformer می‌دهیم.

ترنسفورمرها

این معماری بر اساس استفاده از انکودرها و دیکودرها یا ترکیبی از آن‌ها طراحی شده است. در این معماری، مفهوم self-attention معرفی شد که به بررسی تأثیر یک بخش از ورودی بر سایر بخش‌ها می‌پردازد و ارتباطات بین بخش‌های مختلف را در نظر می‌گیرد. هرچند کاربرد اولیه ترنسفورمرها در پردازش زبان‌های طبیعی (NLP) بود، اما این معماری توانست در بسیاری از حوزه‌های دیگر، از جمله پردازش تصویر، جایگاه خود را پیدا کند.

شبکه‌های عصبی CNN

شبکه‌های عصبی کانولوشنی (CNN) علاوه بر وزن‌های خطی، دارای چندین فیلتر هستند که به عنوان وزن‌های اضافی در نظر گرفته می‌شوند و توسط شبکه به‌روزرسانی می‌شوند. هر فیلتر بر روی تصویر ورودی اعمال می‌شود، به طوری که این فیلترها می‌توانند ویژگی‌های مختلف تصویر را استخراج کنند. فرآیند اعمال فیلترها از طریق عملیات کانولوشنی انجام می‌شود که سبب می‌شود شبکه بتواند ویژگی‌های مختلفی از تصویر را تشخیص دهد. این شبکه‌ها از نحوه عملکرد ذهن انسان الهام گرفته‌اند، به همان صورتی که انسان با استفاده از جزئیات و ویژگی‌های مختلف، اشیاء و مناظر را تشخیص می‌دهد. به طور کلی، شبکه‌های CNN از سه بخش اصلی تشکیل شده‌اند: لایه‌های کانولوشن، لایه‌های پولینگ و لایه‌های غیرخطی.

۲-۱. آشنایی با ترنسفورمرهای تصویر

شبکه ViT بر اساس معماری ترنسفورمر طراحی شده و به جای استفاده از شبکه‌های کانولوشنی CNN، تصاویر را به مجموعه‌ای از پچ‌های کوچک‌تر تبدیل می‌کند و این پچ‌ها را به عنوان ورودی به ترنسفورمر می‌دهد.

ساختار و نحوه ی کارکرد شبکه ViT

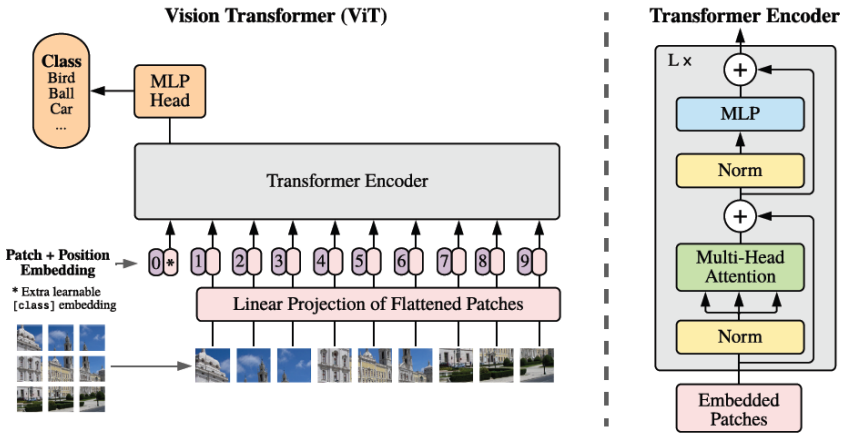
نحوه کارکرد این شبکه باتوجه به مقاله مربوطه (Dosovitskiy et al., ۲۰۲۰) به این صورت است که در اولین مرحله، تصویر ورودی به پچ‌های کوچکتری با ابعاد ثابت تقسیم شده و در ادامه هر پچ به یک بردار تبدیل می‌شود. این پچ‌ها مانند توکن‌های ورودی در NLP عمل می‌کنند. تعداد این پچ‌ها بستگی به ابعاد تصویر و اندازه پچ دارد.پس از آن هر پچ توسط یک لایه خطی به یک بردار با ابعاد مشخص تعبیه یا Embeddingمی‌شود. این بردارهای Embedding در حقیقت بردارهای ویژگی هستند که اطلاعات مربوط به هر پچ را در یک فضای ویژگی‌های مشترک قرار می‌دهند. علاوه بر این، یک تعبیه مکانی نیز به هر پچ اضافه می‌شود تا اطلاعات مکانی هر پچ حفظ شود. پس از آن پچ‌های Embedشده به همراه توکن کلاس به ورودی یک ترنسفورمر داده شده و خروجی توکن کلاس پس از عبور از لایه‌های ترنسفورمر، به یک Head طبقه‌بندی متصل می‌شود. این Head ، وظیفه طبقه‌بندی نهایی تصویر را بر عهده دارد.

مزایا و کاربرد

شبکه ViT با استفاده از داده‌های بزرگ در مرحله پیش‌آموزش، می‌تواند عملکرد بسیار خوبی در وظایف طبقه‌بندی تصویر داشته باشد. این شبکه به خصوص در شرایطی که داده‌های کافی برای آموزش وجود دارد، توانسته نتایج بهتری نسبت به شبکه‌های کانولوشنی ارائه دهد. یکی از مزایای اصلی ViT این است که به صورت بهینه‌تری با داده‌های بزرگتر کار می‌کند و نیاز به منابع محاسباتی کمتری دارد.

بخش‌های مختلف معماری ViT

معماری ViT شامل چندین بخش اصلی است که هر یک نقش مهمی در عملکرد کلی شبکه ایفا می‌کند. تصویر زیر معماری این شبکه را نشان می‌دهد.



شکل ۵۶ معماری شبکه **ViT**

در اولین گام، تصویر ورودی به مجموعه‌ای از پچ‌های کوچک‌تر با ابعاد ثابت تقسیم می‌شود. به عنوان مثال، یک تصویر ۲۲۴×۲۲۴ به پچ‌های ۱۶ × ۱۶ تقسیم می‌شود که در نتیجه ۱۹۶ پچ به دست می‌آید. این پچ‌ها به عنوان توکن‌های ورودی برای شبکه عمل می‌کنند. نحوه دقیق ایجاد ورودی به شکل زیر است:

تصویر ورودی با ابعاد $H \times W \times C$ (ارتفاع، عرض و تعداد کانال‌ها) به پچ‌های مربعی با ابعاد $P \times P$ تقسیم می‌شوند. تعداد پچ‌ها N بوده که برابر با $\frac{H \times W}{P^2}$ می‌باشد.

3.1 VISION TRANSFORMER (VIT)

An overview of the model is depicted in Figure 1. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.

شکل ۵۷ ورودی شبکه **ViT**

در ادامه هر پچ با استفاده از یک لایه خطی به یک بردار تعبیه یا Embedding تبدیل می‌شود. این بردار تعبیه شامل اطلاعات ویژگی‌های هر پچ است و به یک فضای با ابعاد مشخص (معمولاً D) نگاشت می‌شود. علاوه بر این، به هر بردار تعبیه یک تعبیه مکانی یا Position Embedding اضافه می‌شود تا اطلاعات مکانی هر پچ نیز در شبکه وارد شود.

پچ‌هایی که به بردار تعبیه تبدیل شد به همراه توکن کلاس به ورودی یک ترنسفورمر انکودر داده می‌شود. ترنسفورمر انکودر شامل لایه‌های متعددی از بلاک‌های Multi-Head Self-Attention و شبکه‌های MLP است. هر بلاک شامل یک لایه نرمال‌سازی (Layer Normalization) قبل از هر بلاک و بعد از هر بلاک است. در نهایت خروجی توکن کلاس پس از عبور از لایه‌های ترنسفورمر به یک Head طبقه‌بندی متصل می‌شود.

ایرادات وارد به ترنسفورمر ViT

- **نیاز به داده‌های بزرگ برای پیش‌آموزش:** یکی از بزرگترین ایرادات ViT این است که برای دستیابی به عملکرد بهینه نیاز به مجموعه داده‌های بسیار بزرگ دارد. بدون داده‌های کافی، عملکرد آن نسبت به شبکه‌های کانولوشنی کمتر است.
- **مصرف منابع زیاد:** عملیات self-attention در ViT به حافظه زیادی نیاز دارد، به خصوص وقتی که اندازه تصاویر ورودی بزرگ باشد. این مسئله می‌تواند به افزایش زمان و منابع محاسباتی مورد نیاز منجر شود.

- **Inductive bias** : برخلاف CNNها که بایاس‌های مکانی دارند، ViT این بایاس‌ها را به طور ذاتی ندارد. این موضوع می‌تواند باعث شود که شبکه نتواند ویژگی‌های مکانی مهم تصویر را به خوبی یاد بگیرد.

بهبود ViT

- **استفاده از مدل‌های هیبریدی:** ترکیب ViT با لایه‌های کانولوشنی می‌تواند به بهبود عملکرد کلی مدل کمک کند. برای مثال، استفاده از CNN برای استخراج ویژگی‌های اولیه و سپس استفاده از ViT برای پردازش‌های پیشرفته‌تر.
- **استفاده از Data Augmentation :** استفاده از روش‌های داده‌افزایی می‌تواند به بهبود عملکرد ViT کمک کند. تکنیک‌هایی مانند Random Flip که در این تمرین استفاده شده می‌توانند تنوع داده‌های آموزشی را افزایش دهند و به شبکه کمک کنند تا تعمیم بهتری داشته باشد.
- **کاهش مصرف حافظه:** استفاده از مکانیزم‌های self-attention محلی می‌تواند به کاهش مصرف حافظه و افزایش کارایی کمک کند. مثل روش‌های Swin Transformer

منابع استفاده شده در این قسمت:

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (۲۰۲۰, November). A simple framework for contrastive learning of visual representations. In International conference on machine learning (pp. ۱۵۹۷-۱۶۰۷). PMLR.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (۲۰۲۱). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. ۱۰۰۱۲-۱۰۰۲۲).

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (۲۰۲۰). An image is worth ۱۶x۱۶ words: Transformers for image recognition at scale. arXiv preprint arXiv:۲۰۱۰,۱۱۹۲۹.

۲-۲ . لود و پیش پردازش دیتاست

مدل‌های مورد استفاده در مقاله روی دیتاست image۱k از پیش آموزش دیده بودند (به جز مدل ViT-L که با image۲۱k آموزش دیده بود). دیتاست image۱k شامل ۱.۲ میلیون تصویر با وضوح بالا است. برای ارزیابی نتایج مقایسه مدل‌ها در شرایط مورد اشاره در مقاله، از دیتاست CIFAR۱۰ استفاده شده است. این دیتاست دارای ۱۰ کلاس و هر کلاس شامل ۶۰۰۰ تصویر رنگی (مجموعاً ۶۰۰۰۰ تصویر) با ابعاد

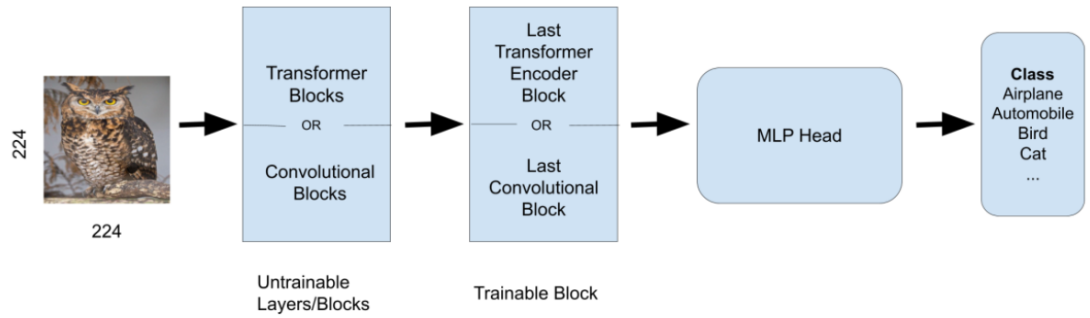
۳۲ در ۳۲ پیکسل است. از این تصاویر، ۵۰۰۰ تصویر برای آموزش و ۱۰۰۰ تصویر برای تست مدل استفاده شده‌اند. همچنین برای ورود تصاویر به مدل‌ها، لازم است ابعاد آن‌ها به ۲۲۴ در ۲۲۴ پیکسل تغییر یابد.

در این مقاله دو روش پیش‌پردازش ساده معرفی شده‌اند. اولین روش، که قبلاً نیز به آن اشاره شد، تغییر اندازه تصاویر دیتاست CIFAR۱۰ از ابعاد اصلی ۳۲ در ۳۲ به ۲۲۴ در ۲۲۴ است. دومین روش، شامل استفاده از چرخش افقی تصاویر است که در آن هر تصویر با احتمال ۵۰ درصد به صورت افقی چرخانده می‌شود و یا به حالت اولیه خود باقی می‌ماند. تصاویر به اندازه میانگین و انحراف معیار دیتاست CIFAR۱۰ نیز نرمال‌سازی می‌شوند. این موارد با کمک Transform پیاده‌سازی تا به صورت On the fly در هنگام آموزش شبکه اعمال شوند:

```
transform_train = transforms.Compose([
    transforms.Resize((224, 224), antialias= None, interpolation=transforms.InterpolationMode.BILINEAR),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.49139968, 0.48215827, 0.44653124), (0.24703233, 0.24348505, 0.26158768))
])
```

پیاده‌سازی معماری مقاله

در این مقاله از شیوه‌ای خاص جهت finetuning استفاده شده است. پس از مدل از پیش آموزش دیده شده، از یک لایه‌ی flatten و سپس یک لایه‌ی خطی با تابع فعال سازی ELU استفاده شده است. اندازه‌ی خروجی لایه‌ی خطی ۲۵۶ است و روی خروجی dropout با احتمال ۰.۵٪ اعمال می‌گردد. سپس یک لایه‌ی خطی دیگر با خروجی به اندازه‌ی ۱۰ نورون (که هر نورون نماینده‌ی یکی از کلاس‌های CIFAR۱۰ است) به انتهای مدل اضافه کردیم. فعال سازی این لایه نیز از نوع softmax تعیین گردید.



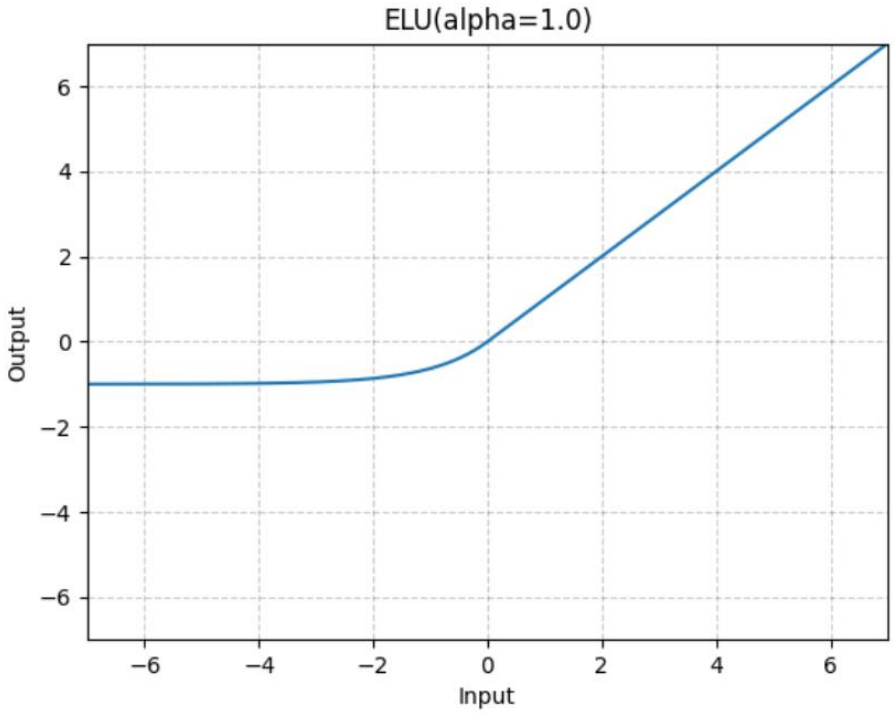
شکل ۵۸ معماری پیشنهادی مقاله برای **Finetuning**

یکی از ابهاماتی که در این مورد در مقاله وجود داشت دقیقا مشخص نشده بود که Head مدل‌های اولیه باید حذف شوند و یا آن‌ها را آنفریز کرده و از آن استفاده کنیم. در این تمرین ما هر دو حالت را بررسی کردیم و تفاوتی در دقت ایجاد نشد. فقط با اضافه کردن آن تعداد پارامترها افزایش می یافت. پس در فایل نهایی فقط حالتی که به پیاده‌سازی مقاله نزدیک‌تر است را قرار دادیم.

فعال ساز ELU نیز به صورت زیر است:

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$

شکل ۵۹ تابع فعال‌ساز **ELU** استفاده شده در مقاله



شکل ۶۰ نمودار تابع فعال‌ساز **ELU**

هایپرپارامترهای استفاده شده در مقاله

تعداد ایپاک: حداکثر ۲۰ ایپاک

نرخ یادگیری: ۰.۰۰۰۱ . درصورتی که دقت داده‌های validation پیشرفت نکند، ضریب آموزش در ۰.۶ ضرب می‌گردد تا حداقل به ۰.۰۰۰۰۰۰۱ برسد. این مورد در حلقه آموزش پیاده‌سازی شد.

اندازه‌ی بچ: ۵۱۲ .

تابع loss: cross entropy

در جدول زیر نیز مدل‌هاس استفاده شده در مقاله را مشاهده می‌کنیم. که در ادامه ۲ مدل از این مدل‌ها که به نسبت پارامتر و دقت مناسب هستند را برای این تمرین انتخاب می‌کنیم.

Model	Model type	Trainable Parameters	Validation Accuracy(%)	Pretrained on
Resnet152V2[9]	CNN	15,497,994	83.583	ImageNet1K
EfficientNetV2L[21]	CNN	7,020,960	90.130	ImageNet1K
VGG-19[17]	CNN	9,573,130	92.784	ImageNet1K
NFNetF6[3]	CNN	12,012,043	94.350	ImageNet1K
Densenet201[10]	CNN	6,982,400	94.757	ImageNet1K
ViT-L32[7]	Transformer	12,863,242	95.467	ImageNet21K
Swin-B224[15]	Transformer	12,868,650	93.580	ImageNet1K
CoaT-LiteSmall[25]	Transformer	3,308,298	94.100	ImageNet1K
CaiTS24[26]	Transformer	**1,877,130**	96.000	ImageNet1K
DeiTBaseDistilled[22]	Transformer	7,488,276	**96.450**	ImageNet1K

شکل ۶۱ مدل‌های استفاده شده در مقاله و تعداد دقت و تعداد پارامتر آن‌ها

برای پیاده‌سازی این تمرین پس از دریافت دیتاست و تعریف دیتالودرها، معماری پیشنهادی مقاله را به صورت یک مدل پیاده‌سازی کردیم. این Custom Model مدل پایه را دریافت کرده و تمام موارد گفته شده را بر روی خروجی آن اعمال کرده و در نهایت خروجی کلی را می‌دهد.

۳-۲. fine-tuning شبکه کانولوشنی

انتخاب مدل و Unfreeze کردن لایه‌ها

برای مدل CNN از DenseNet۲۰۱ و برای مدل Transformer از CaiT-S۲٤ استفاده کردیم. دلیل انتخاب این دو مدل، تعادل میان دقت و تعداد پارامترهای آنها بود. این مدل‌ها نه تنها دقت بالایی دارند که آن‌ها را در میان بهترین‌ها قرار می‌دهد، بلکه در مقایسه با مدل‌هایی با دقت مشابه، تعداد پارامترهای به مراتب کمتری دارند.

پس از دریافت مدل و بارگذاری آن با وزن‌های اولیه و خروجی گرفتن از آن لایه‌های مختلف را بررسی کردیم. پس از آن از تمامی لایه‌ها خروجی گرفتیم و پس از اینکه تمامی لایه‌ها را فریز کردیم، باتوجه مقاله لایه‌های conv۵ به بعد را آنفریز کردیم. توجه شود که برای اینکه لایه Head را جایگزین کنیم باید آن را غیرفعال کنیم. پس تابع همانی را بجای لایه آخر یعنی Classifier قرار می‌دهیم:

```
base_cnn_model.classifier = torch.nn.Identity()
```

Unfreeze Selected Layers in Paper

```
for param in base_cnn_model.parameters():
    param.requires_grad = False

for param in base_cnn_model.features.denseblock4.parameters():
    param.requires_grad = True

for param in base_cnn_model.features.norm5.parameters():
    param.requires_grad = True
```

شکل ۶۲ لایه‌های **Unfreeze** شده در مدل کانولوشنی

در مقاله گفته شده که لایه‌های پس از بلوک مشخص شده مثل Norm و Pooling هم باید در نظر گرفته شده و Unfreeze شوند.

تعداد پارامترهای Trainable

پس از اینکه مدل پایه را به مدل Custom دادیم، تعداد پارامترهای آموزشی را هم برای مدل پایه و هم مدل نهایی محاسبه کردیم که نتیجه آن به صورت زیر است:

Number of trainable parameters (Base Model): ٦٩٨٢٤٠٠

Number of trainable parameters (Custom Model): ٧٤٧٦٧٤٦

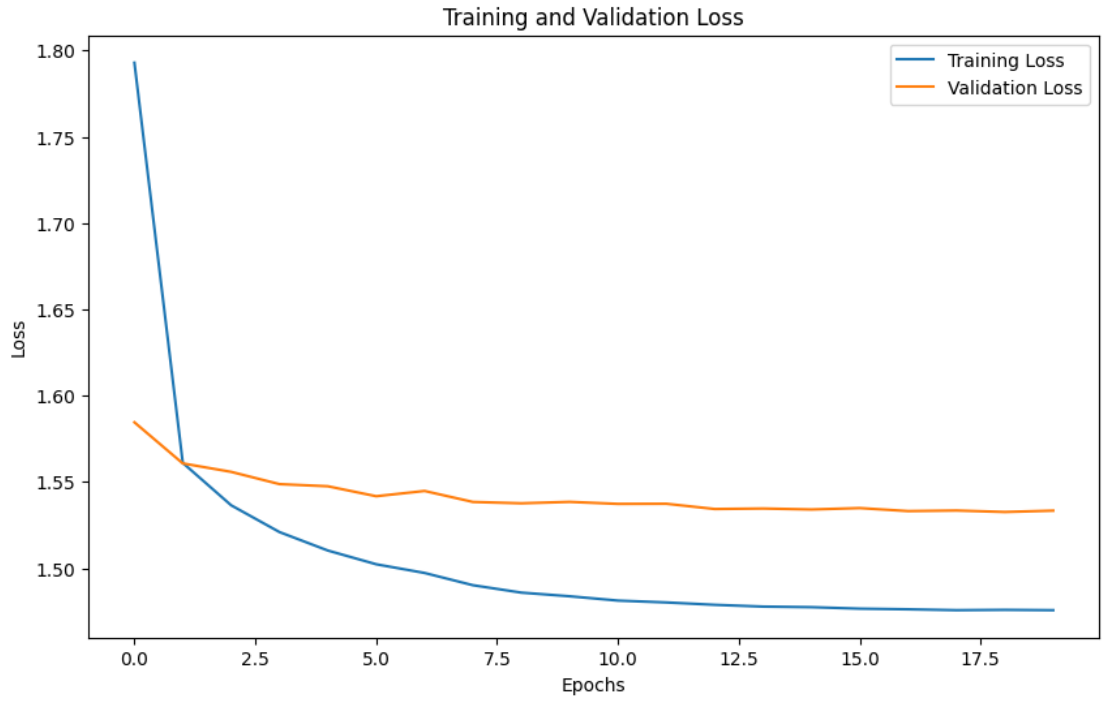
همانطور که مشاهده می‌شود تعداد پارامترهای Trainable مدل پایه ۶۹۸۲۴۰۰ بوده و برابر با تعداد ذکرشده در جدول مقاله است. با پیاده‌سازی Head جدید طبیعتا پارامترها افزایش می‌یابد.

نتیجه فاین‌تیون با کمک شبکه کانولوشنی

پس از اینکه مدل را با هایپرپارامترهای مذکور آموزش دادیم نتیجه زیر حاصل شد:

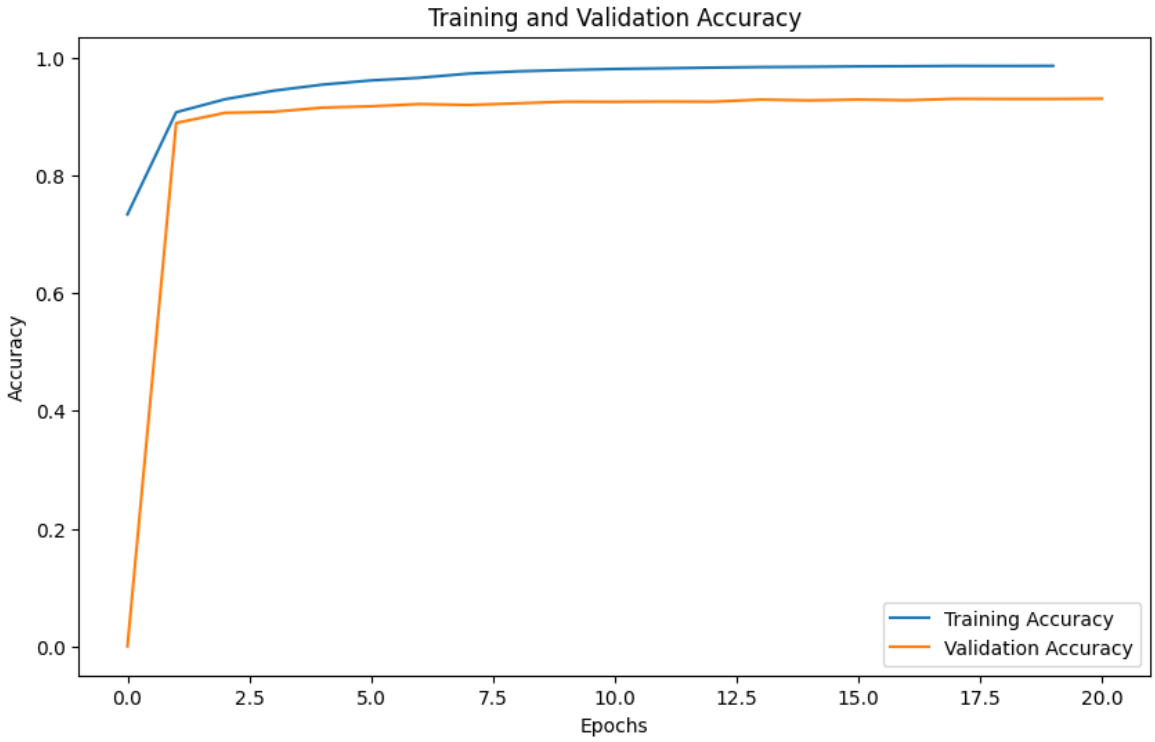
```
Final Training Accuracy: 0.9860

Final Validation Accuracy: 0.9303
```

همانطور که مشاهده می‌شود این مقدار بسیار نزدیک به پیاده‌سازی مقاله است.



شکل ۶۳ نمودار **Loss** برای داده‌های آموزش و ارزیابی برای مدل کانولوشنی

همانطور که مشاهده می‌شود، مقدار Loss به خوبی برای هر دو مجموعه داده روند کاهشی داشته است.



شکل ۶۴ نمودار **Accuracy** برای داده‌های آموزش و ارزیابی برای مدل کانولوشنی

مقدار دقت نیز به خوبی افزایش یافته و به مقدار قابل قبولی رسیده است که نشان می‌دهد شبکه به خوبی روی داده‌ها آموزش دیده است.

مدت زمان آموزش و اعتبارسنجی

هم برای مرحله آموزش و هم ارزیابی مقدار زمان هر ایپاک چاپ شد که در نوت بوک قابل مشاهده است. با تجمیع کل زمان‌ها و تقسیم بر تعداد ایپاک‌ها نیز مقادیر زیر بدست آمد که در قسمت آخر تمرین آن‌ها را بررسی و تحلیل می‌کنیم:

Average Training Time per Epoch: ۱۷۳٫۱۹ seconds

Average Validation Time per Epoch: ۲۷٫۳۷ seconds

۴-۲. fine-tuning شبکه ترنسفورمر

انتخاب مدل و Unfreeze کردن لایه‌ها

همانطور که گفته شد، برای مدل Transformer از CaiT-S۲٤ استفاده کردیم. دلیل انتخاب این مدل، تعادل میان دقت و تعداد پارامترهای آن بود.

پس از دریافت مدل و بارگذاری آن با وزن‌های اولیه و خروجی گرفتن از آن لایه‌های مختلف را بررسی کردیم. پس از آن از تمامی لایه‌ها خروجی گرفتیم و پس از اینکه تمامی لایه‌ها را فریز کردیم، باتوجه مقاله بلوک blocks token only به بعد را آنفریز کردیم. توجه شود که برای اینکه لایه Head را جایگزین کنیم باید آن را غیرفعال کنیم. پس هنگام بارگذاری این مدل تعداد کلاس را صفر می‌گیریم تا در Head تابع همانی جایگزین شود.

```
model_name = "cait_s24_224"
base_transformer_model = timm.create_model(model_name, pretrained=True, num_classes=0)
```

Unfreeze Layers

+ Code + Markdown

```
for param in base_transformer_model.parameters():
    param.requires_grad = False

for param in base_transformer_model.blocks_token_only[1].parameters():
    param.requires_grad = True

for param in base_transformer_model.norm.parameters():
    param.requires_grad = True
```

شکل ۶۵ لایه‌های **Unfreeze** شده در مدل ترنسفورمر

در مقاله گفته شده که لایه‌های پس از بلوک مشخص شده مثل Norm هم باید در نظر گرفته شده و Unfreeze شوند.

تعداد پارامترهای Trainable

پس از اینکه مدل پایه را به مدل Custom دادیم، تعداد پارامترهای آموزشی را هم برای مدل پایه و هم مدل نهایی محاسبه کردیم که نتیجه آن به صورت زیر است:

Number of trainable parameters (Base Model): ۱۷۷٦۰۰۰

Number of trainable parameters (Custom Model): ۱۸۷۷۱۳۰

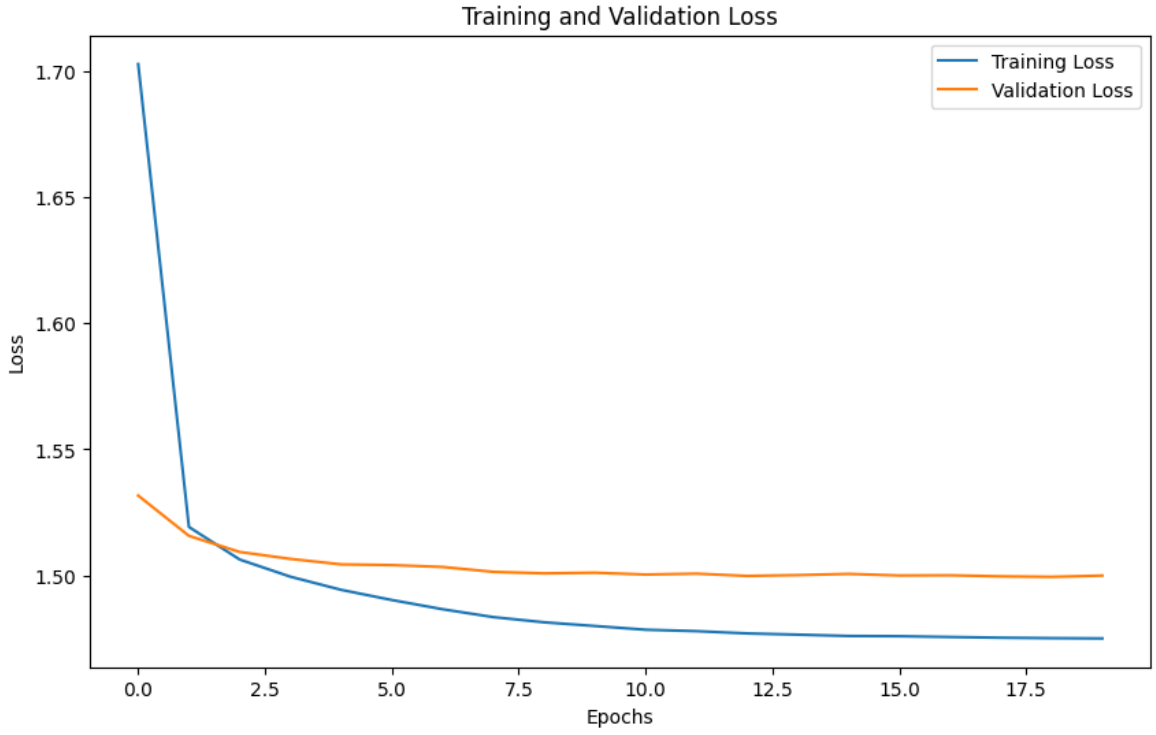
همانطور که مشاهده می‌شود تعداد پارامترهای Trainable مدل پایه ۱۷۷۶۰۰۰ بوده و کمتر از مقدار ذکر شده در جدول است. نکته این مدل این است وقتی که تعداد پارامترهای پس از اعمال معماری مقاله را خروجی گرفتیم برابر با تعداد جدول و ۱۸۷۷۱۳۰ شد. . با پیاده‌سازی Head جدید طبیعتا پارامترها افزایش می‌یابد. علت این تفاوت مشخص نشد. پارامترها با حالات مختلف با و بدون Head و لایه Norm نیز اندازه گیری شد ولی ظاهرا مقدار جدول در روش ترنسفورمر منظور پس از اعمال Head جدید است.

نتیجه فاین‌تیون با کمک شبکه ترنسفورمر

پس از اینکه مدل را با هایپرپارامترهای مذکور آموزش دادیم نتیجه زیر حاصل شد:

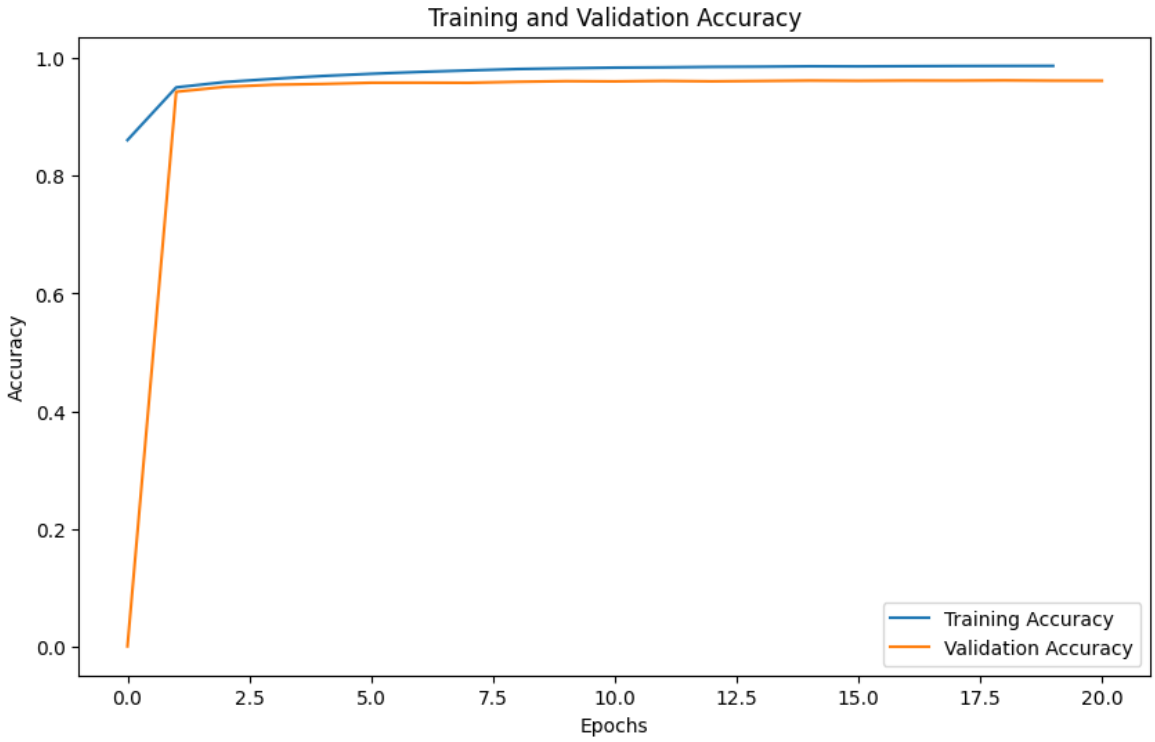
```
Final Training Accuracy: 0.9872

Final Validation Accuracy: 0.9619
```

همانطور که مشاهده می‌شود این مقدار بسیار نزدیک به پیاده‌سازی مقاله بوده و حتی کمی بیشتر است.



شکل ۶۶ نمودار **Loss** برای داده‌های آموزش و ارزیابی برای مدل ترنسفورمر

همانطور که مشاهده می‌شود، مقدار Loss به خوبی برای هر دو مجموعه داده روند کاهشی داشته است.



شکل ۶۷ نمودار **Accuracy** برای داده‌های آموزش و ارزیابی برای مدل ترنسفورمر

مقدار دقت نیز به خوبی افزایش یافته و به مقدار قابل قبولی رسیده است که نشان می‌دهد شبکه به خوبی روی داده‌ها آموزش دیده است.

مدت زمان آموزش و اعتبارسنجی

هم برای مرحله آموزش و هم ارزیابی مقدار زمان هر ایپاک چاپ شد که در نوت بوک قابل مشاهده است. با تجمیع کل زمان‌ها و تقسیم بر تعداد ایپاک‌ها نیز مقادیر زیر بدست آمد که در قسمت آخر تمرین آن‌ها را بررسی و تحلیل می‌کنیم:

Average Training Time per Epoch: ۳۰۸٫۳٤ seconds

Average Validation Time per Epoch: ٦۲٫۰۷ seconds

حال این مورد را با بخش قبلی مقایسه می‌کنیم:

مدل CNN:

میانگین مدت زمان آموزش به ازای هر ایپاک: ۱۷۳.۱۹ ثانیه

میانگین مدت زمان اعتبارسنجی به ازای هر ایپاک: ۲۷.۳۷ ثانیه

مدل ترنسفورمر:

میانگین مدت زمان آموزش به ازای هر ایپاک: ۳۰۸.۳۴ ثانیه

میانگین مدت زمان اعتبارسنجی به ازای هر ایپاک: ۶۲.۰۷ ثانیه

مقایسه این زمان‌ها نشان می‌دهد که مدت زمان آموزش و اعتبارسنجی برای مدل ترنسفورمر به طور قابل توجهی بیشتر از مدل CNN است. این امر به دلیل پیچیدگی بالاتر مدل ترنسفورمر و نیاز به محاسبات بیشتر در هر ایپاک می‌باشد. البته باید تاکید شود که این نتیجه مربوط به مدل‌های انتخاب شده فعلی و بر اساس شرایط این سوال است. در ادامه مقایسه دقیق‌تری بر حسب دقت خواهیم داشت.

۵-۲. مقایسه نتایج

دقت مقاله و دقت مدل پیاده‌سازی شده در جدول زیر قابل مشاهده است.

Model	Accuracy	Parameters(All)	Time per epoch	Accuracy(Paper)
DenseNet۲۰۱	**۰.۹۳۰۳**	۷۴۷۶۷۴۶	۱۷۳ s	**۹۴.۷۵۷**
CaiTS۲٤	**۰.۹۶۱۹**	۱۸۷۷۱۳۰	۳۰۸ s	**۹۶.۰۰۰**

جدول ۵ جدول مقایسه دقت مدل پیاده‌سازی شده با مقاله

این اختلاف‌های جزئی احتمالاً به دلیل تصادفی بودن و تقسیم داده‌ها در بچ‌های مختلف است. نتایج اجراهای مختلف می‌توانند دقت‌های متفاوتی داشته باشند، که نشان‌دهنده نوسانات طبیعی در فرآیند آموزش مدل‌ها است. در برخی اجراها دقت مدل ترنسفورمر به حدود ۹۷ هم نزدیک شد. همچنین برای مدل کانولوشنی ممکن است از کتابخانه دیگری استفاده شده باشد. یک مورد دیگر هم این است که مقاله در مورد لایه Head خود مقاله ابهام دارد. در یک جا گفته شده تا آخرین لایه آنفریز می‌شوند و در جای دیگر در جدول مشاهده می‌شود که تعداد پارامترها بدون Head است. همچنین ممکن است روش پیش‌پردازش‌ها به طور غیرمحسوسی متفاوت باشند. به هر حال دقت‌ها بسیار نزدیک به هم هستند و در مورد دوم تفاوت خاصی وجود ندارد.

ولی به طور کلی می‌توان نتیجه گرفت که با توجه به نتایج به دست آمده و همچنین جدول مقایسه‌ی مدل‌ها ، می‌توان نتیجه‌گیری کرد که مدل‌های ترنسفورمری به طور کلی عملکرد بهتری نسبت به مدل‌های کانولوشنی داشتند، حتی با وجود تعداد پارامترهای کمتر. این نکته نشان‌دهنده‌ی کارایی بالای مدل‌های ترنسفورمری در پردازش تصویر است. با این حال، باید توجه داشت که زمان آموزش مدل‌های ترنسفورمری نسبت به مدل‌های کانولوشنی بیشتر بود.