به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

نام و نام خانوادگی	عرفان باقری سولا – محمد قره حسنلو
شماره دانشجویی	۸۱۰۱۹۸۳۶۱ - ۸۱۰۱۹۸۴۶۱
تاریخ ارسال گزارش	۱۴۰۲.۳.۱۴

فهرست

پاسخ ۱. سامانه ی پرسش–پاسخ

مدل سازی مسئله

BERT بر پایه‌ی معماری ترانسفورمر ساخته شده است که شامل پشته‌ای از لایه‌های encoder است. هر لایه انکودر دارای مکانیزم multi-head self-attention و شبکه عصبی feed-forward است. این معماری به BERT این امکان را می‌دهد تا روابط موجود بین کلمات را به صورت دوطرفه درک کند.

BERT از یک فرآیند دو مرحله‌ای استفاده می‌کند (pre-training & fine-tuning). در مرحله pre-training ، مدل بر روی مجموعه بزرگی از متون بدون برچسب با استفاده از دو تسک unsupervised learining ، یعنی masked language modeling یا (MLM) و پیش‌بینی جمله بعدی یا (NSP)، آموزش داده می‌شود. در مرحله بعد، BERT با افزودن لایه‌های مختص تسک دلواه، با استفاده از داده های برچسب دار برای آن تسک fine-tune می‌شود.

برای مثال در تسک MLM ، دنباله ها مسک می شوند (یعنی بعضی از المان های آن‌ها مخفی می‌شوند) و به عنوان ورودی به مدل داده می شوند. سپس مدل سعی می کند در خروجی دنباله اصلی را بازسازی کند. در تسک NSP نیز یک دنباله به عنوان ورودی به مدل داده می شود مدل سعی می کند جمله بعدی را در خروجی حدس بزند. به عنوان Loss function در این تسک می توان از cross entropy استفاده کرد. لایه آخر این تسک ها معمولا یک mapping از feature space به فضای توکن ها هستند و به همین دلیل با حذف کردن این لایه پس از pre-training می‌توان یک encoder بسیار کارآمد داشت.

برای تسک extractive question answering ورودی های ما دنباله سوال به همراه دنباله context خواهد بود. برای این کار با استفاده از Tokenizer مدل ترنسفورمر مورد نظر، ابتدا باید question ها و context ها را به صورت تون در کنار هم concatenate کنیم. معمولا Tokenizer به صورت خودکار از توکن های [SEP] برای جدا کردن دو دنباله استفاده می کند. خروجی مدل باید دو بردار به سایز دنباله ورودی باشد که یکی نشان دهنده مکان شروع پاسخ به سوال در دنباله ورودی و دیگری نشان دهنده محل اتمام پاسخ در دنباله ورودی می باشد.

در نتیجه با توجه به خروجی مدل که به صورت یک sequence می‌باشد، می‌توانیم دو نورون مختلف به انتهای encoder ها اضافه کنیم تا با دیدن هر المان دنباله در فضای ویژگی، یک احتمال به آن نسبت دهند که مشخص کند چقدر احتمال دارد المان فعلی، محل شروع یا پایان پاسخ سوال در دنباله ورودی باشد. سپس می توانیم از تابع هزینه cross entropy برای آموزش مدل استفاده کنیم.

پیش پردازش داده ها

برخلاف گفته مقاله، در دیتاست clone شده از گیتهاب مربوطه، سوالی با چند جواب وجود نداشت. به هر حال داده ها را که ابتدا به فرمت json هستند لود می‌کنیم و سپس آن ها را به صورت یک مجموعه داده از data point هایی متشکل از یک سوال، یک context و یک پاسخ تبدیل می‌کنیم تا راحت تر بتوانیم از آنها در هنگام پیش پردازش و آموزش استفاده کنیم. اطلاعات آماری دیتاست برای هر یک از بخش ها را در ادامه مشاهده می‌کنیم.

```
total question count = 63994
impossible to answer = 15721
answerability = 75.4 %

min context words = 7
max context words = 274
avg context words = 129.1

min question words = 2
max question words = 256
avg question words = 10.4

min answer words = 1
max answer words = 127
avg answer words = 5.2
```

شکل ۱ – اطلاعات آماری داده های آموزش

```
total question count = 7976
impossible to answer = 1981
answerability = 75.2 %

min context words = 15
max context words = 256
avg context words = 125.3

min question words = 2
max question words = 51
avg question words = 10.7

min answer words = 1
max answer words = 157
avg answer words = 6.3
```

شکل ۲ – اطلاعات آماری داده های ارزیابی

```
total question count = 8002
impossible to answer = 1914
answerability = 76.1 %

min context words = 17
max context words = 328
avg context words = 128.2

min question words = 2
max question words = 58
avg question words = 10.9

min answer words = 1
max answer words = 105
avg answer words = 5.5
```

شکل ۳ – اطلاعات آماری داده های آزمون

در ادامه داده ها را پیش پردازش می کنیم. توضیحات این بخش مربوط به post-processing هم می شوند چرا که همه داده های آموزش، ارزیابی و آزمون را در این بخش پیش پردازش کرده و به یک فرم استاندارد در می‌آوریم.

ابتدا داده ها را طبق توضیحات اولیه tokenize می کنیم. همانطور که در آمار مربوط به بخش های مختلف مجموعه دادگان مشاهده می کنیم، حدود ۲۵ درصد پرسش‌ها بدون پاسخ هستند و پاسخی برای آنها در context وجود ندارد. همچنین در بعضی از داده ها، مجموع طول دنباله سوال و context از حداکثر طول دنباله مجاز ورودی که در اینجا ۱۲۸ است بیشتر می‌باشد. برای حل این مشکل ابتدا به صورت یک قرارداد خروجی مربوط به محل شروع و پایان را برای تمام سوالاتی که با توجه به context خود امکان پاسخ داده شدن را ندارند برابر با محل توکن [CLS] قرار می دهیم. همچنین اگر دنباله سوال و context از اندازه دنباله ورودی مجاز مدل بیشتر شود، دنباله context را به چند بخش تقسیم می‌کنیم که به یک اندازه مشخص با یکدیگر overlap دارند و سپس هر کدام از این بخش ها را به صورت جداگانه در کنار سوال مربوطه به عنوان یک داده مجزا در نشر می گیریم. برای مشخص کردن خروجی نیز نگاه می‌کنیم در هر کدام از قسمت های context جواب به صورت کامل وجود داشته باشد محل شروع و پایان آن را به عنوان لیبل در نظر می‌گیریم و اگر جواب به طور کامل در آن قسمت نباشد، داده شامل آن قسمت از context و سوال را به عنوان پاسخ ناممکن در نظر می‌گیریم طبق قرارداد قبل عمل می کنیم.

اگر دنباله سوال و context نیز در یک داده کمتر از ورودی مدل شود، با توکن های مخصوص [PAD] ادامه آن را پر می کنیم. به این صورت همه داده ها را tokenize کرده و به فرمت مناسب برای ورودی دادن به مدل در می‌آوریم.

پیاده سازی مدل

مدل های پایه را با استفاده از TFAutoModel در محیط Tensorflow لود می کنیم. چون آموزش مدل ها بسیار زمان بر است و هر epoch تقریبا یک ساعت زمان نیاز دارد، کد های مربوط به ParsBERT و ALBERT در دو فایل مجزا پیاده سازی شده اند. البته تمام قسمت های کد در این دو فایل یکسان است و تنها تفاوت آنها مدل پایه استفاده شده در ساخت مدل QA می باشد.

```
def build_model():

    input_ids = tf.keras.Input(shape=(max_length,), dtype=tf.int32, name="input_ids")
    token_type_ids = tf.keras.Input(shape=(max_length,), dtype=tf.int32, name="token_type_ids")
    attention_mask = tf.keras.Input(shape=(max_length,), dtype=tf.int32, name="attention_mask")

    encoded = base_model(
        input_ids=input_ids,
        token_type_ids=token_type_ids,
        attention_mask=attention_mask
    )[0]

    start_positions = tf.keras.layers.Dense(1)(encoded)
    start_positions = tf.keras.layers.Flatten()(start_positions)
    start_positions = tf.keras.layers.Softmax(name='start_positions')(start_positions)

    end_positions = tf.keras.layers.Dense(1)(encoded)
    end_positions = tf.keras.layers.Flatten()(end_positions)
    end_positions = tf.keras.layers.Softmax(name='end_positions')(end_positions)

    model = tf.keras.models.Model(
        inputs=[input_ids, token_type_ids, attention_mask],
        outputs=[start_positions, end_positions]
    )

    return model
```

شکل ۴ – پیاده سازی مدل

به عنوان تابع هزینه نیز از یک تابع دستی استفاده می کنیم که در آن برای دو خروجی مدل به صورت جداگانه cross entropy را محاسبه می کنیم و Loss نهایی مدل برابر جمع این دو loss می باشد.

در این تسک چون از مدل های pre-trained استفاده می‌کنیم، هدف ما آموزش مدل از ابتدا نیست و در اصل می خواهیم fine-tuning انجام دهیم. به همین دلیل نرخ یادگیری را کوچک در نظر می‌گیریم. طبق آزمایشات ما اعدادی در حدود 10^{-4} و 10^{-5} نتایج خوبی را حاصل می کنند.

همچنین به علت بزرگ تر بودن مدل ParsBERT نسبت به ALBERT ، سرعت همگرایی در ALBERT بیشتر از ParsBERT می باشد و به همین علت مدل ParsBERT را در دو epoch و مدل ALBERT را در یک epoch آموزش دادیم.

ارزیابی و پس پردازش

پس از اتمام آموزش مدل ها، با استفاده از معیار های ذکر شده و مجموعه دادگان تست، عملکرد مدل ها را ارزیابی می کنیم. در جدول زیر نتایج مدل ها را با نتایج مقاله مقایسه می کنیم.

جدول ۱ – مقایسه نتایج مدل با نتایج مقاله

Model	Exact Match	F1 Score	EM - Paper	F1 - Paper
ParsBERT	67.8%	74.4%	68.1%	82.0%
ALBERT	69.2%	76.2%	-	-

در ادامه نیز نمونه هایی از پاسخ مدل به داده های آزمون را مشاهده می کنیم. جواب هر دو مدل به این سوالات یکسان می باشد.

```
question : کتاب مقدس دین اسلام چیست؟
answer : قران
```

شکل ۵ – نمونه خروجی مدل

```
question : از آغاز تأسیس دارالفنون چه کسانی به مخالفت با آن پرداختند؟
answer : برخی از درباریان
```

شکل ۶ – نمونه خروجی مدل

```
question : افزوده شدن شمشیر به نقش شیر و خورشید به چه زمانی برمی‌گردد؟
answer : دوران قاجار
```

شکل ۷ – نمونه خروجی مدل

پاسخ ۲. استفاده از Transformer Vision برای طبقه بندی تصاویر

لود کردن دیتاست و انجام پیش پردازش های لازم

در ابتدا با استفاده از transforms، پیش پردازش های گفته شده در مقاله را انجام میدهیم. در ابتدا عکس ها را up-scale میکنیم و از (32, 32, 3) به (224, 224, 3) میبریم. سپس عکس ها ار به صورت رندوم به صورت horizontal، flip میکنیم که سایز دیتاست افزایش پیدا میکند. داده ها را به Tensor برده و در آخر آن را normalize میکنیم. چیش چردازش های گفته شده به ترتیب در عکس زیر قابل مشاهده است.

```
mean, std = 0.5, 0.5

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((mean, mean, mean), (std, std, std))
])
```

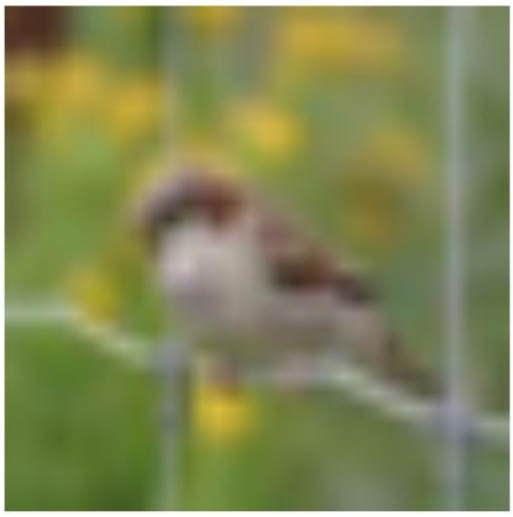
شکل ۸ – پیش پردازش های گفته شده در مقاله

حال دیتاست CIFAR10 را با استفاده از torchvision، لود میکنیم. دیتاهارا در دسته داده آموزش، شافل میکنیم تا بایاسی با توجه به ترتیب عکس ها نداشته باشیم.

مقدار batch_size، با توجه به گفته مقاله در ابتدا ۵۱۲ گرفته شد انا در هنگام آموزش مدل به ارور memory fragmentation خوردیم و با آزمون و خطا و کاهش batch_size به ۳۲ این مشکل برطرف شد و برای هر دو مدل این مقدار در نظر گرفته شد تا امکان مقایسه وجود داشته باشد.

نمونه ای از دیتاست CIFAR10 همراه با اندازه تصویر در شکل زیر آمده است:

```
train_features_batch.shape = torch.Size([32, 3, 224, 224]), train_labels_batch.shape = torch.Size([32])
Image size: torch.Size([3, 224, 224])
Label: 2
```



شکل ۹ – نمونه از دیتاست **CIFAR10**

شبکه کانولوشنی

در این مقاله از چندین شبکه کانولوشنی نام برده شده که یکی از آنها vgg19 است که در آن یک conv1 از بلاک پنجم قابل آموزش و unfreeze شده است که برای همین به شکل زیر، لایه ۲۹ ام unfreeze شده است.

```python
# Define number of classes from each dataset
num_classes_CIFAR10 = len(train_dataset_CIFAR10.classes)

# Load VGG19 model
vgg19_3 = models.vgg19(pretrained=True)

# Freeze all layers except block5_conv1
for param in vgg19_3.features.parameters():
    param.requires_grad = False
for param in vgg19_3.classifier.parameters():
    param.requires_grad = False
for param in vgg19_3.features[28].parameters(): # block5_conv1
    param.requires_grad = True

# Replace classifier with new one
vgg19_3.classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(25088, 256), # 25088 is output of block5_conv1
    nn.ELU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear(256, num_classes_CIFAR10)
)

vgg19_3
```

شکل ۱۰ – مدل کانولوشنی **vgg19** استفاده شده برای **transfer learning**

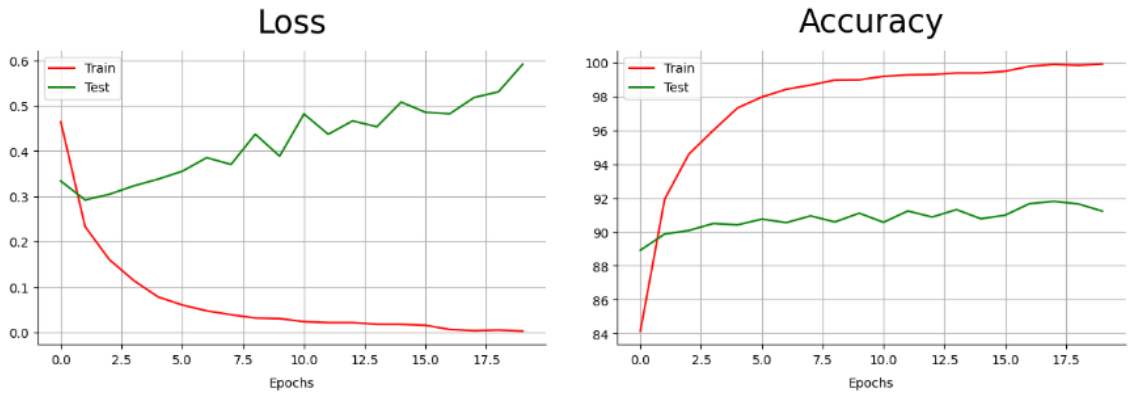
```
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace=True)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace=True)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace=True)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=25088, out_features=256, bias=True)
    (2): ELU(alpha=1.0, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=10, bias=True)
  )
)
```

شکل ۱۱ – انتهای مدل **vgg19**، **fine-tune** شده

همچنین بعد از آخزین لایه برای اینکه طبقه بندی به درستی انجام شود، یک Sequential تعریف میکنیم که در آن ابتدا انتهای شبکه کانولوشنی را flat میکنیم، سپس خروجی آن که ۲۵۰۸۸ نورون بعد از flat کردن میشود، به ۲۵۶ نورون وصل میکنیم. برای لایه غیرخطی از ELU استفاده میکنیم، یک Dropout با احتمال ۵۰ درصد گذاشته و در آخر۲۵۶ نورون را به ۱۰ نورون(به تعداد کلاس های CIFAR10) وصل میکنیم. قسمت softmax در accuracy_fn پیاده سازی شده است تا احتمال قرار گرفتن در هر کلاس مشخص شده باشد.

برای محاسبه loss در هر دفعه، با توجه به نوع مسئله که طبقه بندی است، از crossEntropy در مقاله گفته شده که استفاده شود. همچنین Learning Scheduler طبق گفته مقاله استفاده شده است.

نتیجه خروجی به شکل زیر شده است که مدل کانولوشنی پس از fine-tune به دقت ۹۹.۹ درصد برای داده های آموزش و دقت ۹۱.۲ در داده های ارزیابی زسیده است. این کار برای ۲۰ ایپاک در ۷۲۹۴ ثانیه یعنی حدود ۲ ساعت انجام شده است. نمودار accuracy و loss برای هر دو داده آموزش و ارزیابی به شکل زیر میباشد:



شکل ۱۲ – نمودار دقت و خطا برای مدل کانولوشنی **vgg19** در ۲۰ ایپاک

شبکه کانولوشنی بدون unfreeze کردن

به طور کلی این قسمت اتفاقی انجام و اضافه شده است و دلیل این آن است که در اشتباه لایه اشتباهی(یک relu از بلاک چهارم) unfreeze شده بود. این قسمت خواسته نشده است.

در این مقاله از چندین شبکه کانولوشنی نام برده شده که یکی از آنها vgg19 است که در آن یک relu قابل آموزش و unfreeze شده است که برای همین به شکل زیر، لایه ۲۵ ام unfreeze شده است.

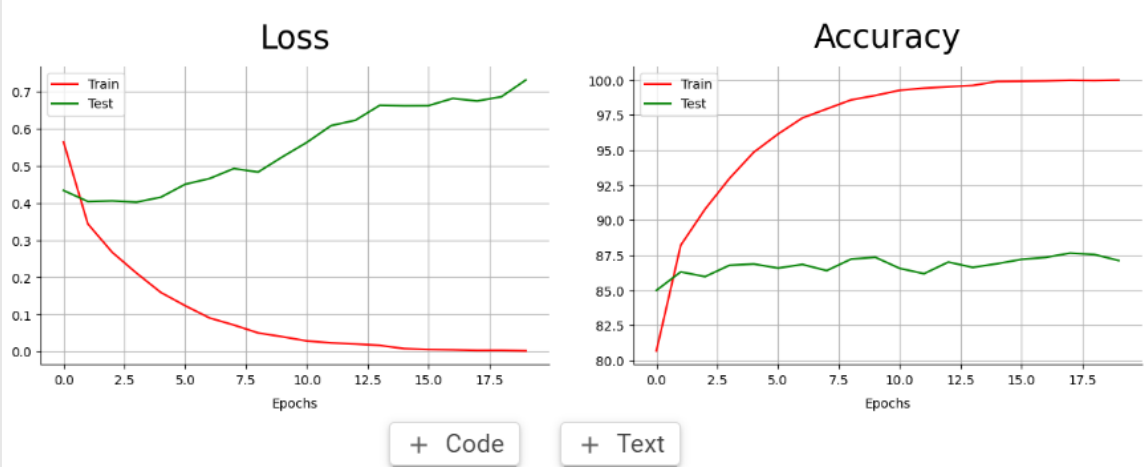
```
# Define number of classes from each dataset
num_classes_CIFAR10 = len(train_dataset_CIFAR10.classes)

# Load VGG19 model
vgg19_2 = models.vgg19(pretrained=True)

# Freeze all layers except block5_conv1
for pafram in vgg19_2.features.parameters():
    param.requires_grad = False
for param in vgg19_2.classifier.parameters():
    param.requires_grad = False
for param in vgg19_2.features[24].parameters(): # ReLU
    param.requires_grad = True

# Replace classifier with new one
vgg19_2.classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(25088, 256),
    nn.ELU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear(256, num_classes_CIFAR10)
)

vgg19_2
```

شکل ۱۳ – مدل کانولوشنی **vgg19** بدون **unfreezing** استفاده شده برای **transfer learning**

همچنین بعد از آخرین لایه برای اینکه طبقه بندی به درستی انجام شود، یک Sequential تعریف میکنیم که در آن ابتدا انتهای شبکه کانولوشنی را flat میکنیم، سپس خروجی آن که ۲۵۰۸۸ نورون بعد از flat کردن میشود، به ۲۵۶ نورون وصل میکنیم. برای لایه غیرخطی از ELU استفاده میکنیم، یک Dropout با احتمال ۵۰ درصد گذاشته و در آخر۲۵۶ نورون را به ۱۰ نورون(به تعداد کلاس های CIFAR10) وصل میکنیم. قسمت softmax در accuracy_fn پیاده سازی شده است تا احتمال قرار گرفتن در هر کلاس مشخص شده باشد.

برای محاسبه loss در هر دفعه، با توجه به نوع مسئله که طبقه بندی است، از crossEntropy در مقاله گفته شده که استفاده شود. همچنین Learning Scheduler طبق گفته مقاله استفاده شده است.

نتیجه خروجی به شکل زیر شده است که مدل کانولوشنی پس از fine-tune به دقت ۱۰۰ درصد برای داده های آموزش و دقت ۸۷.۱ در داده های ارزیابی زسیده است. این کار برای ۲۰ ایپاک در 7290 ثانیه یعنی حدود ۲ ساعت انجام شده است. دقت شود که به طور ضمنی در اینجا unfreeze کردن انجام نشده است. نمودار accuracy و loss برای هر دو داده آموزش و ارزیابی به شکل زیر میباشد:



شکل ۱۴ – نمودار دقت و خطا برای مدل کانولوشنی **vgg19** در ۲۰ ایپاک برای مدل **unfreeze** نشده

شبکه ViT (تبدیل کننده تصویر)

در اینجا از Deit استفاده کردیم و طبق گفته مقاله، ۱۲ امین بلاک را طبق شکل زیر unfreeze کرده ایم:

```
[ ]  # Define number of classes from each dataset
     num_classes_CIFAR10 = len(train_dataset_CIFAR10.classes)

     # Define the Transformer model
     transformer_model = torch.hub.load('facebookresearch/deit:main', 'deit_base_patch16_224', pretrained=True)

     transformer_model.head = nn.Sequential(
         nn.Flatten(),
         nn.Linear(transformer_model.head.in_features, 256),
         nn.ELU(inplace=True),
         nn.Dropout(p=0.5),
         nn.Linear(256, num_classes_CIFAR10)
     )

     # Unfreeze the 12th Transformer Block
     for param in transformer_model.blocks[11].parameters():
         param.requires_grad = True

     transformer_model
```

شکل ۱۵ – مدل ترنسفورمری **DeiT** استفاده شده برای **transfer learning**

```
    (11): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (q_norm): Identity()
        (k_norm): Identity()
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (ls1): Identity()
      (drop_path1): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU(approximate='none')
        (drop1): Dropout(p=0.0, inplace=False)
        (norm): Identity()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop2): Dropout(p=0.0, inplace=False)
      )
      (ls2): Identity()
      (drop_path2): Identity()
    )
  )
  (norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (fc_norm): Identity()
  (head_drop): Dropout(p=0.0, inplace=False)
  (head): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=768, out_features=256, bias=True)
    (2): ELU(alpha=1.0, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=10, bias=True)
  )
```

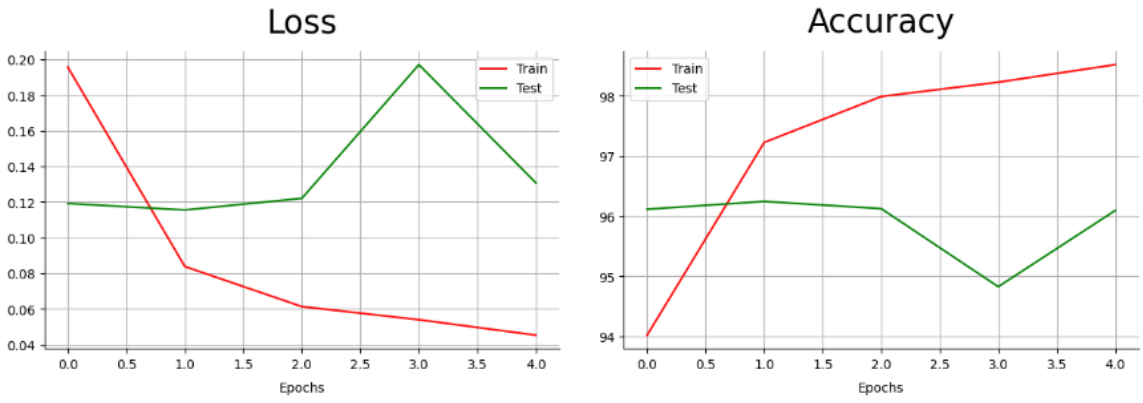
شکل ۱۶ – انتهای مدل **DeiT**، **fine-tune** شده

همچنین بعد از آخزین لایه برای اینکه طبقه بندی به درستی انجام شود، یک Sequential تعریف میکنیم که در آن ابتدا انتهای شبکه ترنسفورمری را flat میکنیم، سپس خروجی آن به ۲۵۶ نورون وصل میکنیم. برای لایه غیرخطی از ELU استفاده میکنیم، یک Dropout با احتمال ۵۰ درصد گذاشته تا جلوی overfitting را تا حدی بگیرد و در آخر۲۵۶ نورون را به ۱۰ نورون(به تعداد کلاس های CIFAR10) وصل میکنیم. قسمت softmax در accuracy_fn پیاده سازی شده است تا احتمال قرار گرفتن در هر کلاس مشخص شده باشد.

برای محاسبه loss در هر دفعه، با توجه به نوع مسئله که طبقه بندی است، از crossEntropy در مقاله گفته شده که استفاده شود. همچنین Learning Scheduler طبق گفته مقاله استفاده شده است.

با توجه به اینکه هر epoch در تقریبا نیم ساعت انجام میشد، ۲۰ ایپاک ۱۰ ساعت زمان میگرفت، به همین دلیل تنها در ۵ ایپاک ران شده است که به طور کلی ۸۵۰۰ ثانیه یعنی حدود دو ساعت و بیست دقیقه وقت میگرفت. مدل های ترنسفورمری به طور کلی به دلیل ساختار بزرگ و پیچیده و همچنین تعداد پارامترهای زیادی دارند، زمان زیادی برای آموزش نیاز دارند.

نتیجه خروجی به شکل زیر شده است که مدل ترنسفورمری پس از fine-tune به دقت ۹۸.۵ درصد برای داده های آموزش و دقت ۹۶.۱ درصدی در داده های ارزیابی زسیده است. این کار برای ۵ ایپاک در ۸۵۰۱ ثانیه یعنی حدود ۲ ساعت و ۲۰ دقیقه انجام شده است. نمودار accuracy و loss برای هر دو داده آموزش و ارزیابی به شکل زیر میباشد:



شکل ۱۷ – نمودار دقت و خطا برای مدل ترنسفورمری **DeiT** در ۵ ایپاک

مقایسه

جدول ۲ مقایسه نتایج به دست آمده از شبکه کانولوشنی و ترنسفورمری در مقایسه با نتایج **paper**

Model	Model Type	Validation Accuracy (%) in Paper	# of Epochs in Paper	Validation Accuracy (%) in Project	# of Epochs in Project	Consumed Time (second)
VGG-19(With Unfreezing)	CNN	92.784	۲۰	۹۱.۲	20	7294
VGG-19(Without Unfreezing)	CNN	-	_	87.1	20	7290
Deit	Transformer	96.450	۲۰	۹۶.۱	۵	8501

همانطور که مشخص است دقت مدل ترنسفورمری از دقت مدل کانولوشنی بیشتز شده است اما در تعداد ایپاک کمتری مدل DeiT نسبت به مقاله ران شده است که اختلاف ۰.۳۵ با مقاله به دست آمده است.

علی رغم اینکه در شرایط برابر(یعنی epoch برابر) زمان بسیاز زیادی مدل ترنسفورمری طول خواهد کشید اما در شرایط بالا که زمان تقریبا نزدیک است(یعنی زمان مدل vgg-19 در ۲۰ لیپاک حدودا ۲۰ دققه کمتر از DeiT در ۵ ایپاک است.) مدل DeiT با اختلاف حدود ۵ درصدی عملکرد بهتری داشته است که به نوبه خود چشمگیر است.

همچنین بدون unfreeze کردن جواب اختلاف فاحشی با freeze کردن در مدل vgg-19 در شرایط برابر دارد(حدود ۴ درصد) که میتوان به اهمیت fine-tune کردن بدین وسیله پی برد.

در مقایسه با مقاله، مدل vgg-19 در شرایط برابر، اختلاف حدود ۱.۶ داشته است. مدل DeiT با ۱۵ ایپاک کمتر اختلاف ۰.۳۵ دارد.