دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین پنجم

متین بذرافشان - 810100093
شهریار عطار - 810100186

فهرست

پرسش 1. تشخیص اخبار جعلی با Transformer-ها

1-1. آشنایی با BERT و CT-BERT

I. Transfer Learning

این یک روش در یادگیری عمیق است که در آن مدلی که برای یک وظیفه خاص آموزش دیده است، برای یک وظیفه مرتبط دیگر استفاده می‌شود. این روش برای کاهش زمان و هزینه آموزش مدل‌های یادگیری عمیق کاربرد دارد.

برای مثال، فرض می‌کنیم مدلی را برای تشخیص گربه در تصاویر آموزش داده‌ایم. حالا می‌خواهیم مدلی برای تشخیص سگ در تصاویر بسازیم. به جای آموزش یک مدل کاملا جدید، می‌توانیم از مدل قبلی که برای تشخیص گربه آموزش داده‌یم، استفاده کنیم و آن را برای تشخیص سگ Fine-Tune کنیم. این کار باعث می‌شود که زمان و هزینه آموزش کاهش یابد.

یکی از مزایای استفاده از مدل‌های مانند BERT این است که می‌توانیم از آن‌ها برای یادگیری انتقالی استفاده کنیم. BERT یک مدل زبانی است که بر روی مجموعه داده بزرگی از متن‌های انگلیسی آموزش دیده است. با استفاده از BERT، می‌توانیم از دانشی که این مدل در طول فرآیند آموزش خود به دست آورده است، برای وظایف مرتبط با NLP استفاده کنیم. به عنوان مثال، می‌توانیم BERT را برای وظایفی مانند تشخیص نام‌ها، تحلیل احساسات، و ترجمه متن استفاده کنیم. این کار باعث می‌شود که نیاز به آموزش یک مدل کاملا جدید برای هر وظیفه کاهش یابد.

II. Feature-Based vs Fine-Tuning

- **Feature-based:** در این روش، ما از یک مدل پیش‌آموزش دیده برای استخراج ویژگی‌ها از داده‌ها استفاده می‌کنیم. این ویژگی‌ها سپس به عنوان ورودی به یک مدل دیگر داده می‌شوند که برای وظیفه مورد نظر ما آموزش می‌بیند. در این حالت، مدل اولیه (که ویژگی‌ها را استخراج می‌کند) تغییر نمی‌کند. این روش معمولاً زمانی استفاده می‌شود که تعداد داده‌های موجود برای آموزش کم است.
- **Fine-tuning:** در این روش، ما از یک مدل پیش‌آموزش دیده به عنوان نقطه شروع برای آموزش مدل جدید استفاده می‌کنیم. ما مدل را با داده‌های جدید Re-Tune می‌کنیم، یعنی وزن‌های مدل را با استفاده از داده‌های جدید به روز می‌کنیم. در این حالت، مدل

اولیه تغییر می‌کند و به وظیفه جدید تطبیق می‌یابد. ین روش معمولاً زمانی استفاده می‌شود که تعداد داده‌های موجود برای آموزش زیاد است و منابع محاسباتی کافی موجود است.

همچنین یک مدل دیگر با نام CT-BERT وجود دارد که برای تجزیه و تحلیل داده‌های توییتر مربوط به COVID-19 آموزش دیده است. این مدل بر روی 97 میلیون توییت (1.2 میلیارد داده آموزشی) آموزش دیده است. این مدل برای داده‌های خاص دامنه و پیام‌های شبیه به توییتر به خصوص مربوط به COVID-19، بهبود عملکرد 10 تا 30 درصدی نسبت به مدل استاندارد BERT را نشان می‌دهد.

2-1. دیتاست

مجموعه داده، شامل متون مرتبط به کووید 19 به همراه صحت آن است. هر متن می‌تولند واقعی یا جعلی باشد. همچنین این مجموعه شامل 6420 داده آموزش و 2140 داده صحت‌سنجی و 2140 داده آزمون می‌باشد.

در ابتدا باید داده‌ها پیش‌پردازش کنیم تا بتوانیم از آن‌ها استفاده کنیم. برای این کار تابعی جامع تعریف کرده‌ایم تا بتوان پردازش‌های مختلفی را روی متن انجام دهیم. پس از آزمودن پیش‌پردازش‌های مختلف به این نتیجه رسیدیم که تنها ایموجی‌ها به متون معادلشان تبدیل کنیم.[1]

```
def preprocess(x, html = False, url = False, username = False,
hashtag = False, rep = False,
                punctuation = False, demojize = True):

    if(html):
      x = re.sub(re.compile('<.*?>') , '', x)
    if(url):
      x = re.sub(r'http\S+|www\S+', '', x)
    if(username):
      x = re.sub(r'@[^ ]+', '', x)
    if(hashtag):
      x = re.sub(r'#', '', x)
    if(rep):
      x = re.sub(r'([a-zA-Z])\1{2,}', r'\1', x)
    if(demojize):
      x = emoji.demojize(x)
```

[1] انجام هر پردازش دیگری از دقت مدل می‌کاهد.

```
    if(punctuation):
      puncs = [
            '[', '!', '"', '#', '%', '\\', '\"', '(', ')', '*',
            '+', ',', '-', '.', '/', ':', ';', '<', '=', '>',
'?',
            '/', '^', '_', '`', '{', '|', '}', '~',
            ''', '"', '"', '"', ''', ']','؛','؟','«','»','‘','٪'
      ]
      pattern = r'[' + re.escape(''.join(puncs)) + r']'
      x = re.sub(pattern, ' ', x)

    return x
```

در ادامه متون را توکنایز کردیم تا آماده ورود به مدل شوند.[2] همچنین حداکثر تعداد توکن برای یک متن را 128 در نظر گرفتیم. اگر تعداد توکن‌های یک متن بیشتر از این مقدار باشد، بقیه آن‌ها را دور می‌ریزیم و همچنین اگر کمتر باشد، با حاشیه‌گذاری[3] به این مقدار می‌رسانیم.

```
def tokenize(texts, tokenizer, maxlen = MAXLEN):
    ids = []
    masks = []
    types = []
    for text in texts:
      o = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=maxlen,
            pad_to_max_length=True,
            truncation=True,
            return_tensors='tf',
            verbose=0
      )
      ids.append(o['input_ids'])
      masks.append(o['attention_mask'])
      types.append(o['token_type_ids'])

    return (
      np.array(ids).reshape(len(texts), maxlen),
      np.array(masks).reshape(len(texts), maxlen),
```

[2] علت و چگونگی این کار، در گزارش تمرین قبلی به طور کامل تشریح شده است.

[3] Padding

```
        np.array(types).reshape(len(texts), maxlen)
    )
```

در انتها با کمک تابع زیر دیتاست‌های مورد نیاز را ساختیم. این تابع، دیتاست‌های مورد نیاز را با توجه به فرمت ورودی شبکه تولید می‌کند.

1. شناسه[4]ها و ماسک‌های توکن‌های یک جمله را به عنوان ورودی شبکه در نظر می‌گیرد. همچنین واقعی بودن (1) و جعلی بودن (0) آن متن را به عنوان خروجی مورد انتظار قرار می‌دهد.
2. ورودی‌ها را بر اساس سایز بچ داده شد، دسته‌بندی می‌کند.
3. بچ‌ها را آماده ورود به تنسورها می‌کند.

```
def create_datasets(tokenizer, batch_size, maxlen = MAXLEN):
    train_tokens = tokenize(train['clean'], tokenizer, maxlen)
    valid_tokens = tokenize(valid['clean'], tokenizer, maxlen)
    test_tokens = tokenize(test['clean'], bert_tokenizer)

    train_dataset = tf.data.Dataset.from_tensor_slices((
      {'input_ids': train_tokens[0], 'attention_mask':
train_tokens[1]},
      train['label']
    ))
    train_dataset =
train_dataset.shuffle(len(train['label'])).batch(batch_size)

    valid_dataset = tf.data.Dataset.from_tensor_slices((
      {'input_ids': valid_tokens[0], 'attention_mask':
valid_tokens[1]},
      valid['label']
    ))
    valid_dataset =
valid_dataset.shuffle(len(valid['label'])).batch(batch_size)

    test_dataset = tf.data.Dataset.from_tensor_slices((
      {'input_ids': test_tokens[0], 'attention_mask':
test_tokens[1]}
    )).batch(batch_size)
```

[4] ID

```
    return train_dataset, valid_dataset, test_dataset
```

ساخت مدل

برای ساخت مدل‌ها یک تابع تعریف کردیم که شبکه مورد نیاز را بسازد. مدل ترنسفورمر را به عنوان لایه ورودی شبکه قرار می‌دهیم. این مدل به ازای هر توکن دو ورودی شناسه و ماسک را دریافت می‌کند و بردار نظیر آن توکن خروجی می‌دهد. دقت کنید این مدل دو خروجی دارد:

1. **last_hidden_state**: این متغیر، خروجی مدل به ازای هر توکن را برمی‌گرداند و ابعادی برابر `(Batch Size, Tokens, Embedding Size)` دارد.
2. **pooler_output**: این متغیر به ازای هر ویژگی بردار ویژگی‌ها، بر روی همه توکن‌ها عملیات pooling را انجام می‌دهد در نتیجه ابعاد خروجی آن `(Batch Size, Embedding Size)` دارد.

در ادامه بر اساس ورودی‌های تابع، مدل مورد نیاز ساخته می‌شود:

- `base_model`: مدل ترنسفورمر پایه. در اینجا می‌تواند BERT یا CT-BERT باشد. بردارهای تعبیه[5] بر اساس این مدل ساخته می‌شوند.
- `fine_tune`: نوع آموزش را تعیین می‌کند. در صورت `False` بودن، وزن لایه‌های مدل ترنسفورمر را فریز می‌کند و از این مدل به عنوان feature extractor استفاده می‌کند و نوع آموزش feature-based می‌شود. در صورت `True` بودن مدل ترنسفورمر fine-tune می‌شود.
- `gru`: در صورت `True` بودن بعد از آخرین لایه مدل ترنسفورمر یک لایه BiGRU با 128 واحد قرار دهد، همچنین در این حالت باید خروجی last_hidden_state را به عنوان ورودی به این لایه بدهیم، در غیر این صورت خروجی pooler_output را مستقیما به یک لایه Dense می‌دهیم.

```
def create_model(base_model, fine_tune = False, gru = False):
      input_ids = Layers.Input(shape=(MAXLEN,), dtype=tf.int32,
name="input_ids")
      attention_mask = Layers.Input(shape=(MAXLEN,),
dtype=tf.int32, name="attention_mask")
      dense_input = None
      if(gru):
            bert_output = base_model(input_ids,
```

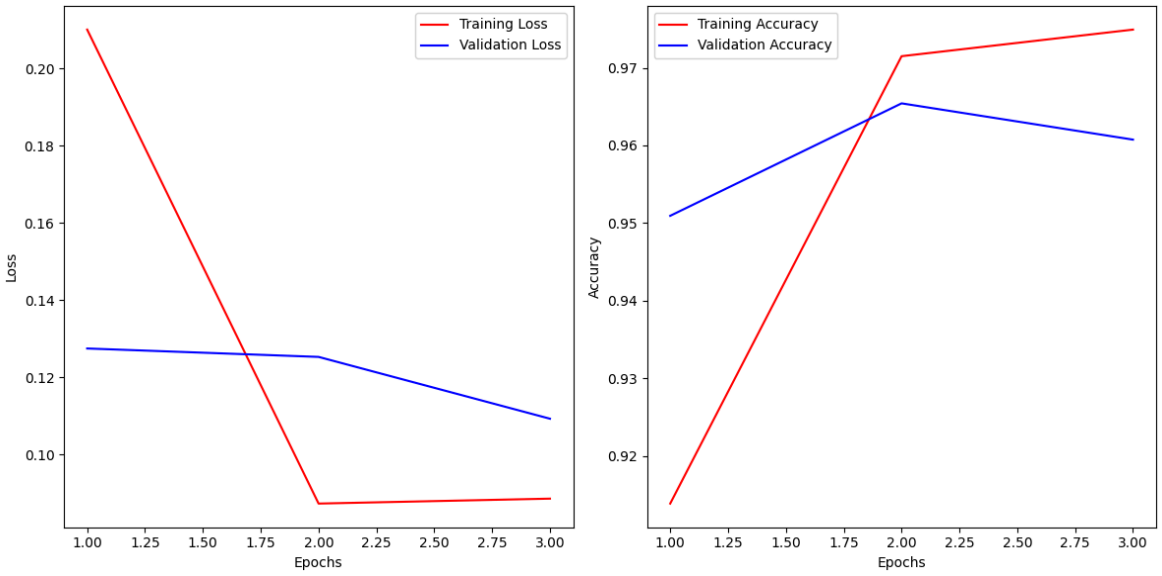
[5] Embedding

```
attention_mask=attention_mask)[0]
            dense_input =
Layers.Bidirectional(Layers.GRU(128))(bert_output)
      else:
            bert_output = base_model(input_ids,
attention_mask=attention_mask)[1]
            dense_input = bert_output
      dense_output = Layers.Dense(1,
activation='sigmoid')(dense_input)
      model = Model(inputs=[input_ids, attention_mask],
outputs=dense_output)


      model.layers[2].trainable = fine_tune
      return model
```

در ادامه تمامی مدل‌های خواسته شده را آموزش می‌دهیم. سپس نمودارهای دقت و خطا در هر دوره را برای داده‌های آموزش؛ و تمامی سنجه[6]های دسته‌بندی[7] را به همراه ماتریس آشفتگی[8] برای داده‌های آزمون نمایش می‌دهیم.

Fine-Tuning .3-1

I. مدل پایه BERT



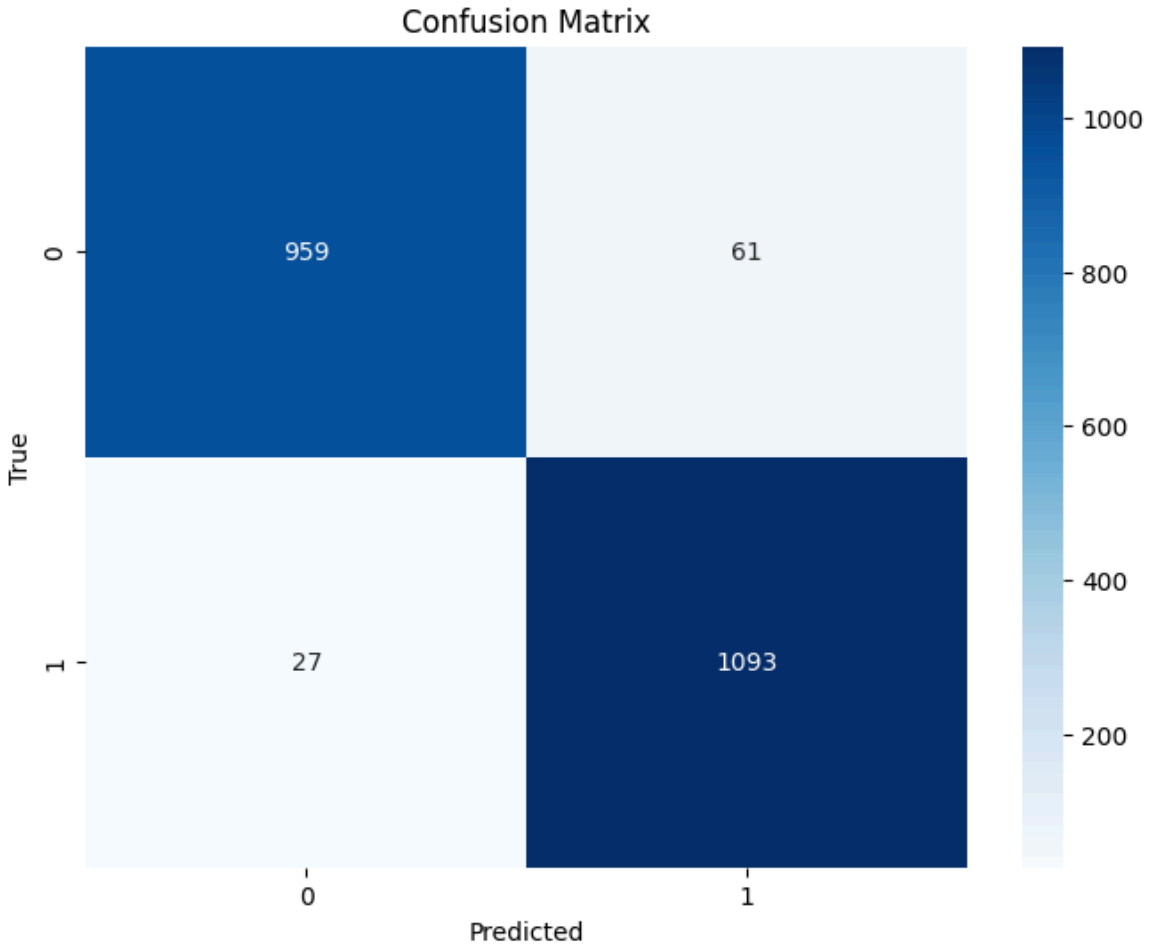
شکل 1-1. تغییرات خطا و دقت آموزش در رویکرد fine-tune مدل BERT

```
+----------+---------+
| Metric   |   Value |
+==========+=========+
| Accuracy |  0.9589 |
+----------+---------+
+---------+-------------+----------+------------+
| Class   |   Precision |   Recall |   F1-score |
+=========+=============+==========+============+
| Class 0 |      0.9726 |   0.9402 |     0.9561 |
+---------+-------------+----------+------------+
| Class 1 |      0.9471 |   0.9759 |     0.9613 |
+---------+-------------+----------+------------+
```

[6] Metric
[7] Classification
[8] Confusion Matrix

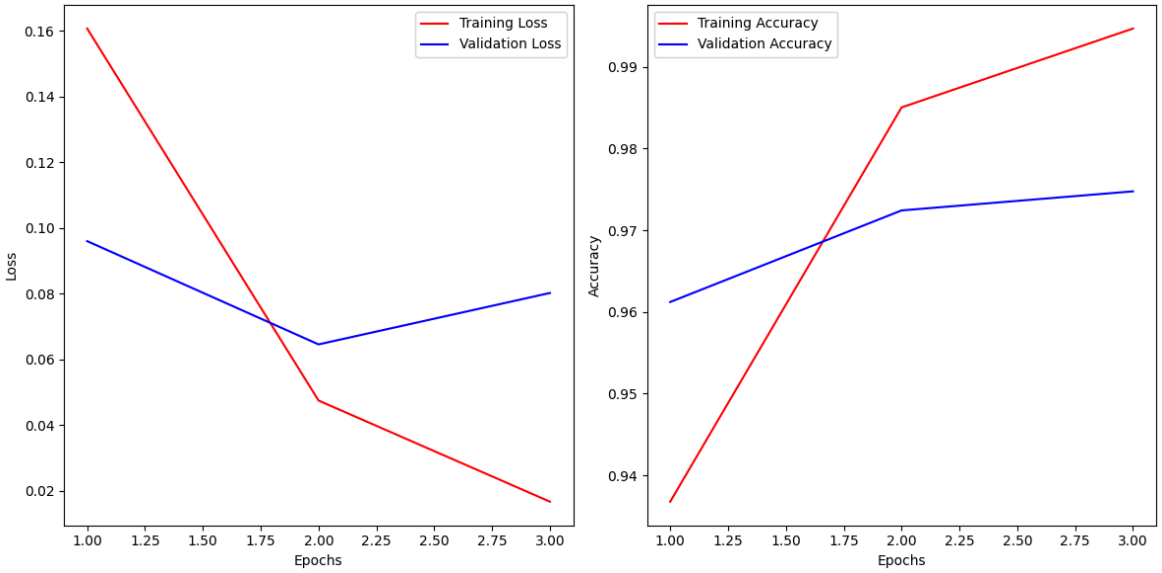
Average Type	Precision	Recall	F1-score
Macro average	0.9599	0.958	0.9587
Micro average	0.9589	0.9589	0.9589
Weighted average	0.9593	0.9589	0.9588

جدول 1-1. سنجه‌های داده‌های آزمون در رویکرد fine-tune مدل BERT



شکل 2-1. ماتریس آشفتگی در رویکرد fine-tune مدل BERT

II. مدل پایه BERT به همراه یک لایه BiGRU



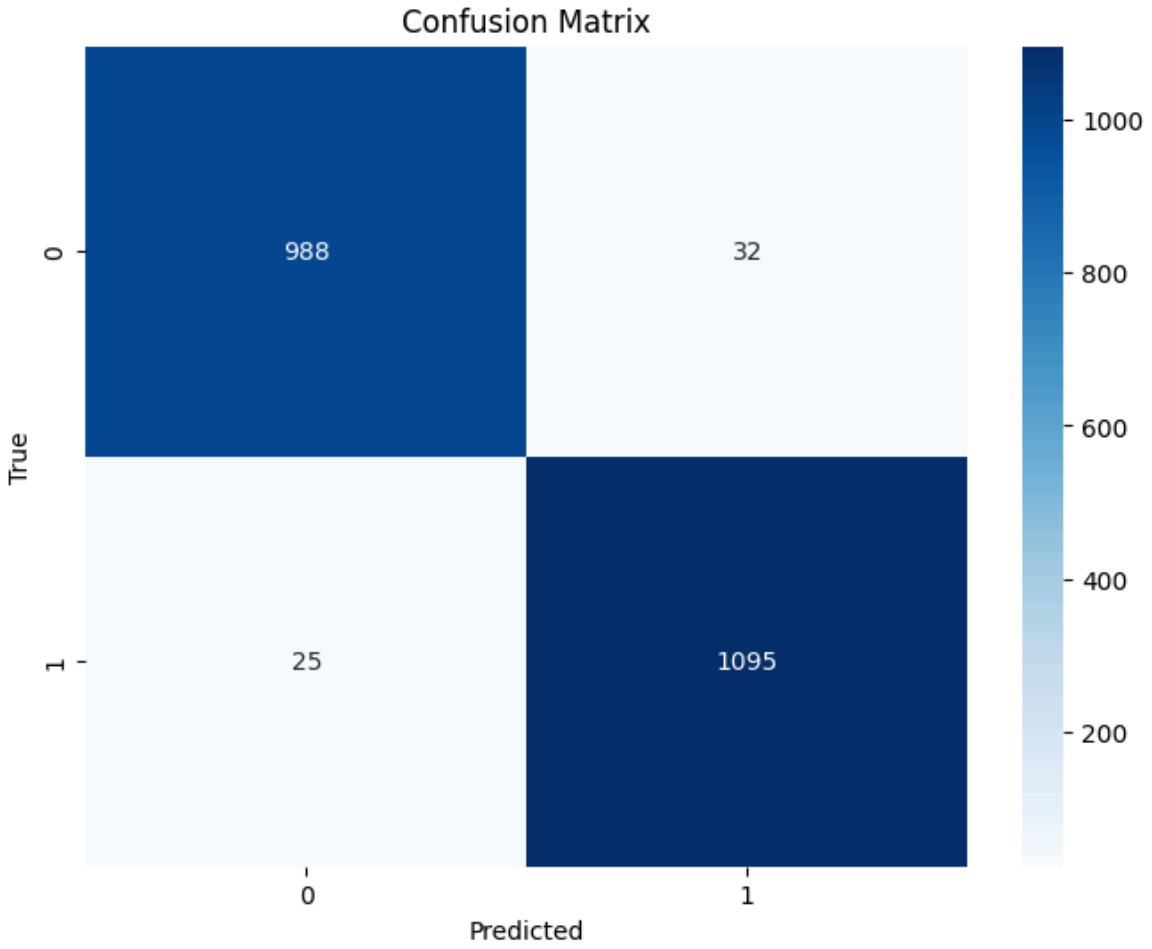
شکل 3-1. تغییرات خطا و دقت آموزش در رویکرد fine-tune مدل BERT + BiGRU

Metric	Value
Accuracy	0.9734

Class	Precision	Recall	F1-score
Class 0	0.9753	0.9686	0.972
Class 1	0.9716	0.9777	0.9746

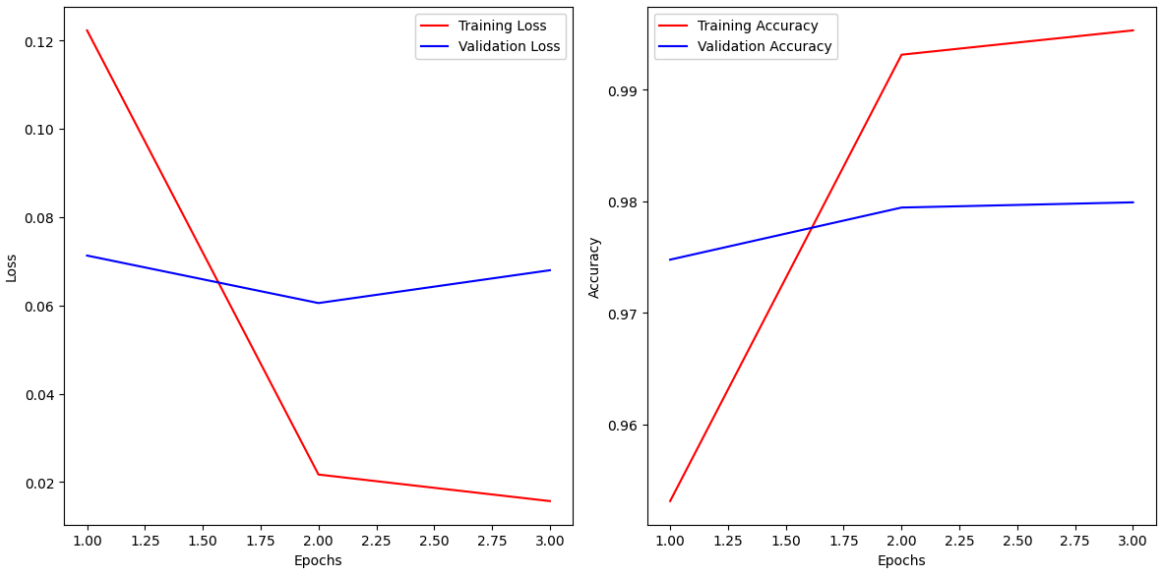
Average Type	Precision	Recall	F1-score
Macro average	0.9735	0.9732	0.9733
Micro average	0.9734	0.9734	0.9734
Weighted average	0.9734	0.9734	0.9734

جدول 2-1. سنجه‌های داده‌های آزمون در رویکرد fine-tune مدل BERT + BiGRU



شکل 4-1. ماتریس آشفتگی در رویکرد fine-tune مدل BERT + BiGRU

III. مدل CT-BERT به همراه یک لایه BiGRU



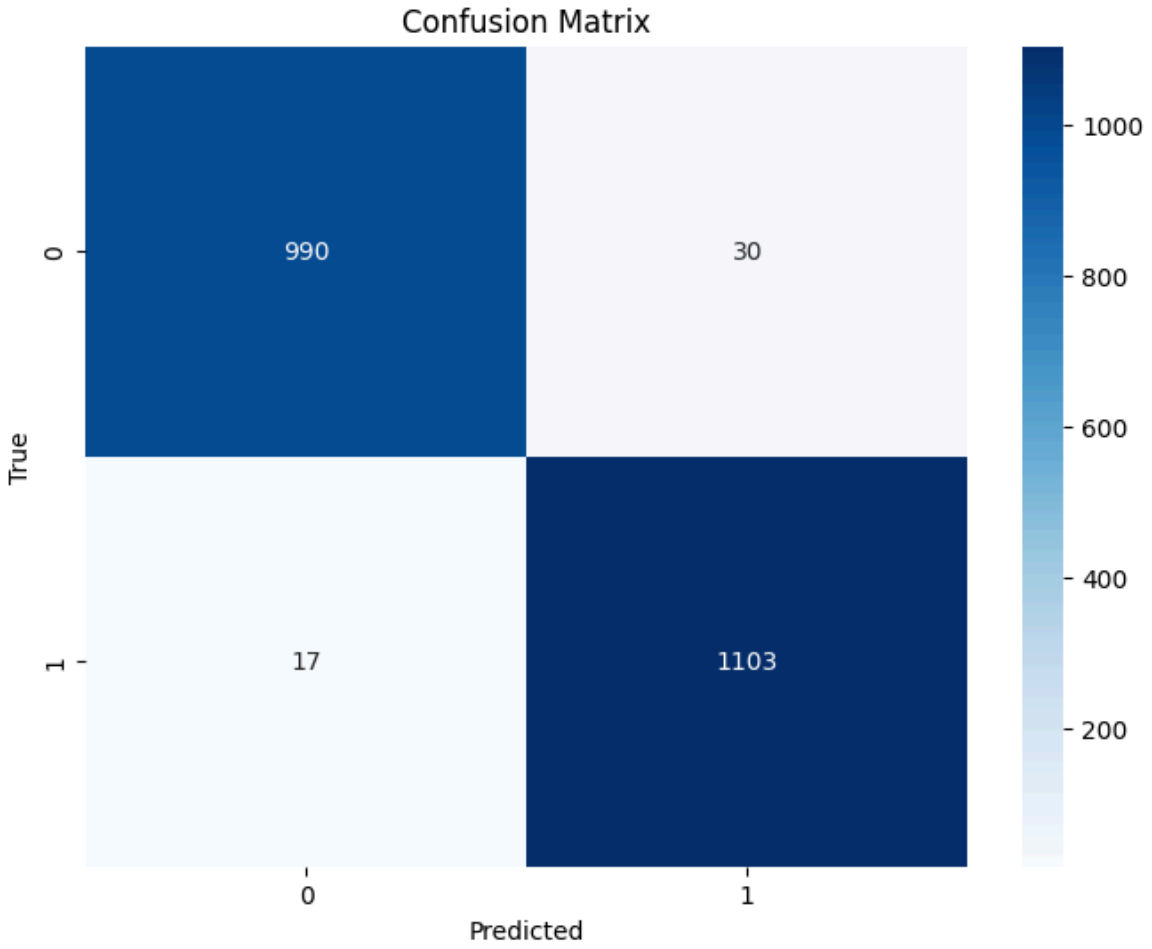
شکل 5-1. تغییرات خطا و دقت آموزش در رویکرد fine-tune مدل CT-BERT + BiGRU

Metric	Value
Accuracy	0.978

Class	Precision	Recall	F1-score
Class 0	0.9831	0.9706	0.9768
Class 1	0.9735	0.9848	0.9791

Average Type	Precision	Recall	F1-score
Macro average	0.9783	0.9777	0.978
Micro average	0.978	0.978	0.978
Weighted average	0.9781	0.978	0.978

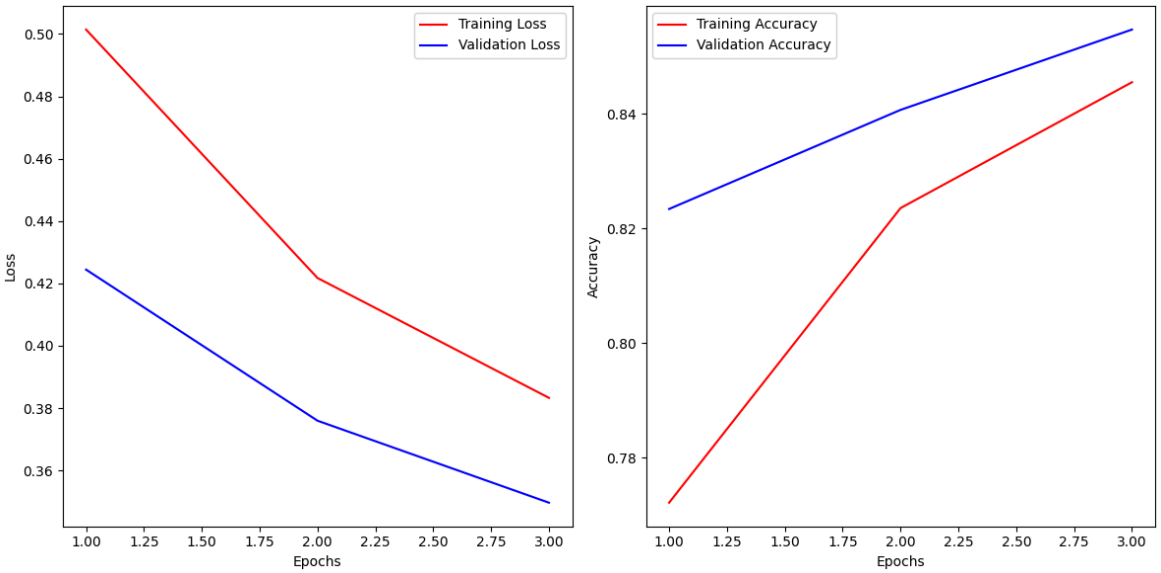
جدول 3-1. سنجه‌های داده‌های آزمون در رویکرد fine-tune مدل CT-BERT + BiGRU



شکل 6-1. ماتریس آشفتگی در رویکرد fine-tune مدل CT-BERT + BiGRU

Feature-Based .4-1

I. مدل پایه BERT



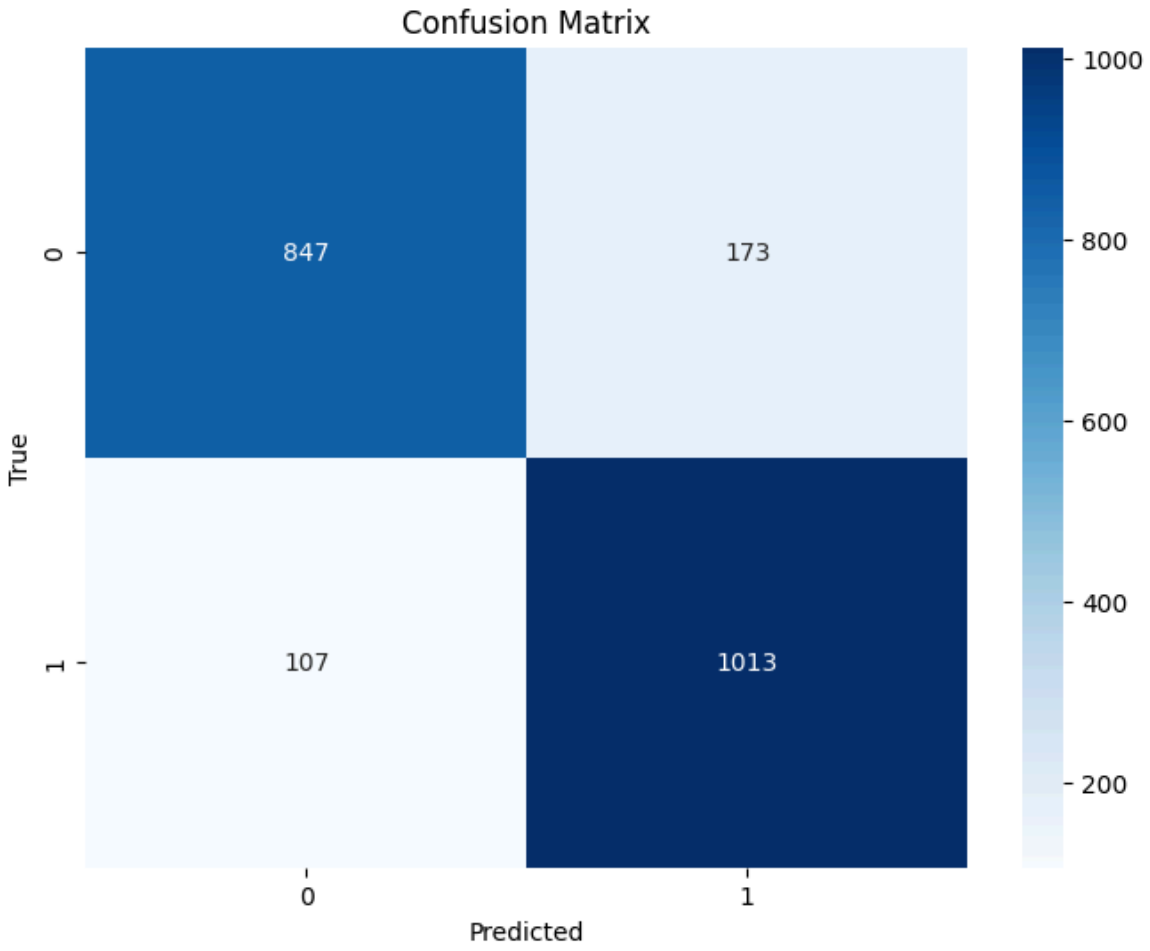
شکل 7-1. تغییرات خطا و دقت آموزش در رویکرد feature-based مدل BERT

Metric	Value
Accuracy	0.8692

Class	Precision	Recall	F1-score
Class 0	0.8878	0.8304	0.8582
Class 1	0.8541	0.9045	0.8786

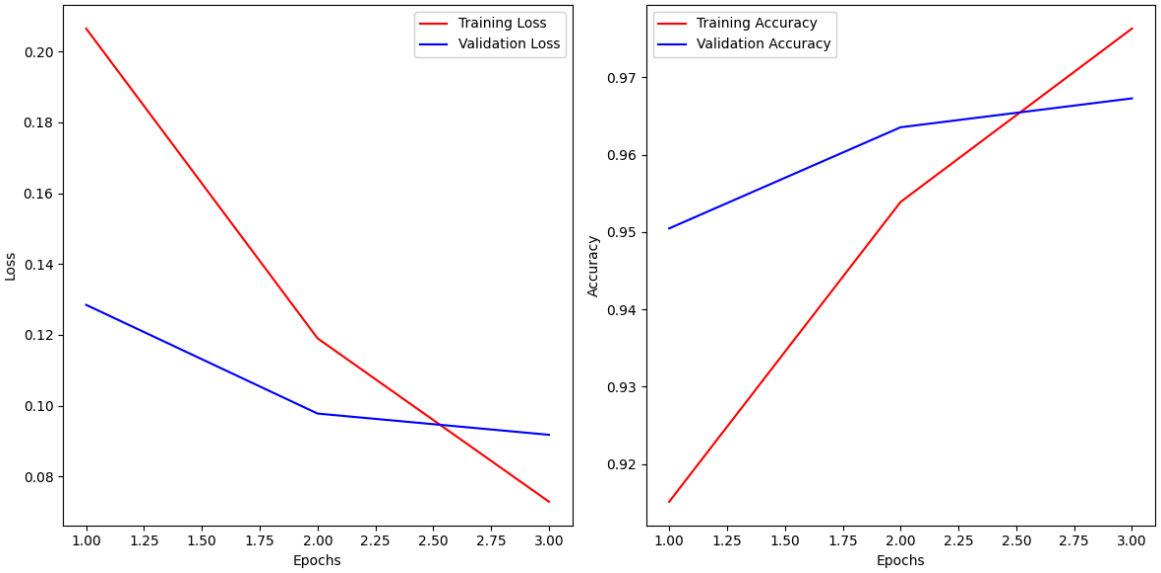
Average Type	Precision	Recall	F1-score
Macro average	0.871	0.8674	0.8684
Micro average	0.8692	0.8692	0.8692
Weighted average	0.8702	0.8692	0.8688

جدول 4-1. سنجه‌های داده‌های آزمون در رویکرد feature-based مدل BERT



شکل 8-1. ماتریس آشفتگی در رویکرد feature-based مدل BERT

II. مدل پایه Bert به همراه یک لایه BiGRU



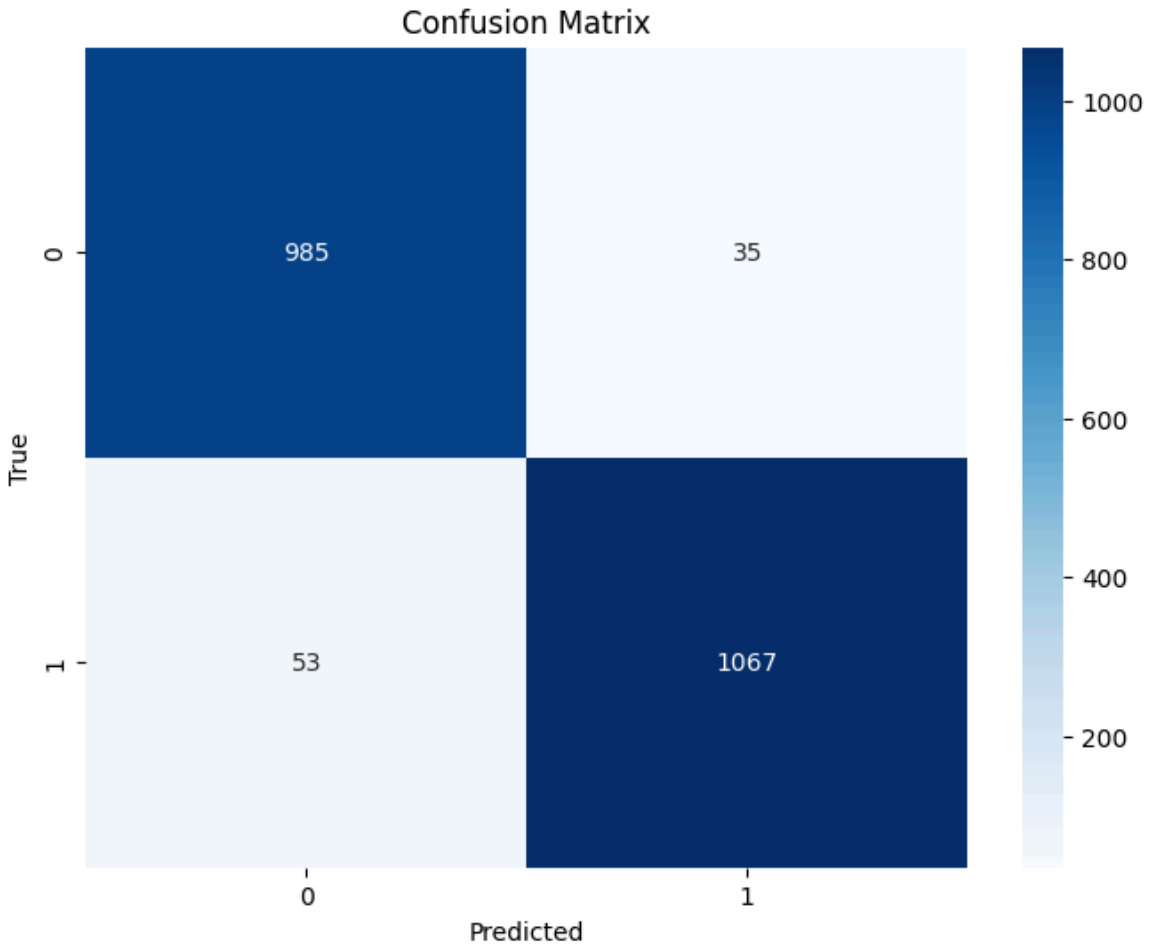
شکل 9-1. تغییرات خطا و دقت آموزش در رویکرد feature-based مدل Bert + BiGRU

Metric	Value
Accuracy	0.9589

Class	Precision	Recall	F1-score
Class 0	0.9489	0.9657	0.9572
Class 1	0.9682	0.9527	0.9604

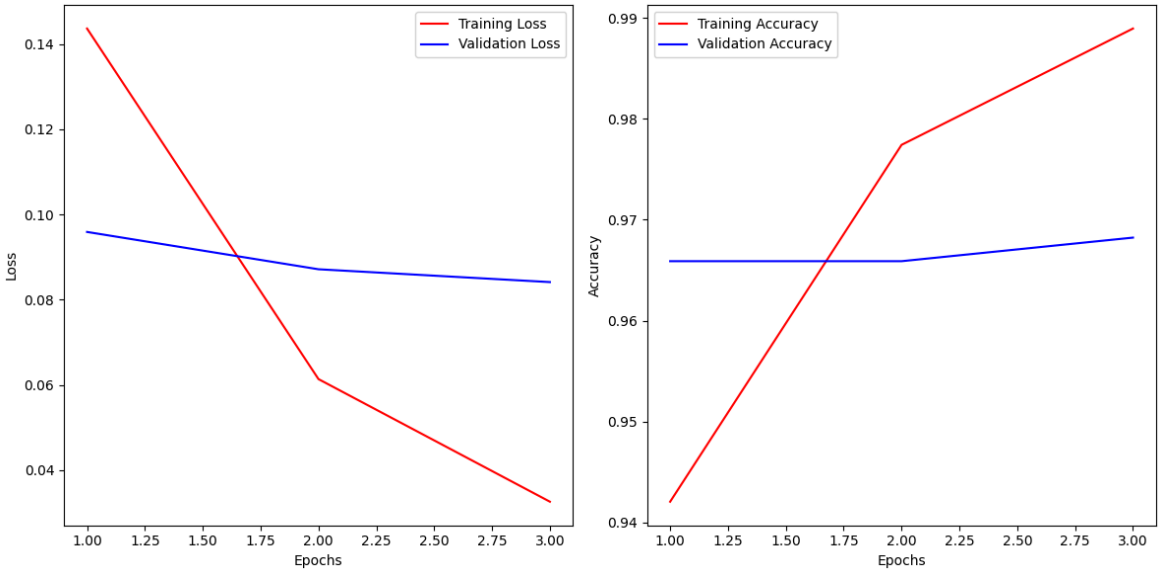
Average Type	Precision	Recall	F1-score
Macro average	0.9586	0.9592	0.9588
Micro average	0.9589	0.9589	0.9589
Weighted average	0.959	0.9589	0.9589

جدول 5-1. سنجه‌های داده‌های آزمون در رویکرد feature-based مدل Bert + BiGRU



شکل 10-1. ماتریس آشفتگی در رویکرد feature-based مدل Bert + BiGRU

III. مدل CT-BERT به همراه یک لایه BiGRU



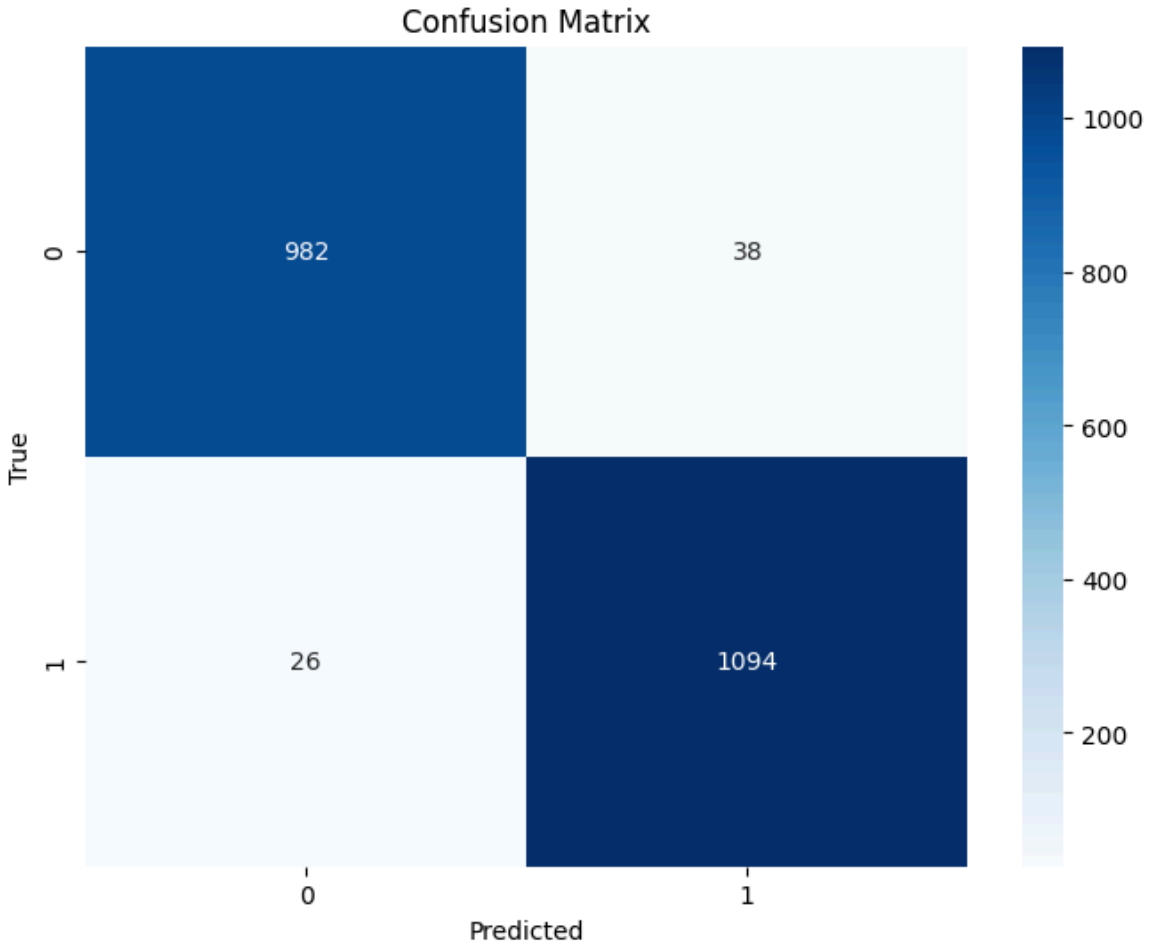
شکل 11-1. تغییرات خطا و دقت آموزش در رویکرد feature-based مدل CT-BERT + BiGRU

Metric	Value
Accuracy	0.9701

Class	Precision	Recall	F1-score
Class 0	0.9742	0.9627	0.9684
Class 1	0.9664	0.9768	0.9716

Average Type	Precision	Recall	F1-score
Macro average	0.9703	0.9698	0.97
Micro average	0.9701	0.9701	0.9701
Weighted average	0.9701	0.9701	0.9701

جدول 6-1. سنجه‌های داده‌های آزمون در رویکرد feature-based مدل CT-BERT + BiGRU



شکل 12-1. جدول آشفتگی در رویکرد feature-based مدل CT-BERT + BiGRU

5-1. نتایج

	feature-based	fine-tuning
Bert + Dense	0.8692	0.9589
Bert + BiGRU	0.9589	0.9734
CTBert + BiGRU	0.9701	0.978

جدول 7-1. مقایسه دقت تمامی مدل‌ها برای داده‌های آزمون

I. مقایسه BERT و CT-BERT

در حالت feature-based، مدل CT-BERT حدود 1.1 درصد بهتر عمل کرده است و همچنین در حالت fine-tuning این مدل حدود 0.5 درصد از مدل پایه BERT بهتر بوده است. CT-BERT به طور خاص برای تحلیل متون مرتبط با کووید-۱۹ آموزش دیده است، که شامل اصطلاحات و الگوهای خاص این حوزه است، در نتیجه عملکرد بهتری داشته است.

همچنین می‌توان مشاهده کرد که اختلاف این دو مدل در حالت fine-tuning کاهش داشته است که این نشانگر این است که fine-tuning به مدل پایه BERT کمک کرده است تا الگوهای مربوط به حوزه کووید را یاد بگیرد.

II. مقایسه مدل‌های بخش اول و دوم

همانطور که در جدول 7-1 آمده است، هر سه مدل در حالت Fine-Tune بهتر از Feature-Based عمل کرده‌اند. این اختلاف زمانی مشهودتر است که از مدل پایه BERT و بدون استفاده از یک لایه BiGRU استفاده کنیم، زیرا در این حالت وظیفه یادگیری تنها به عهده لایه Dense آخر می‌باشد و با کمک fine-tune کردن مدل BERT می‌توان الگوهای مربوط به حوزه کووید را بهتر یاد گرفت.

III. مقایسه مدل‌های مبتنی بر Fine-Tuning و Featured-Based

مدل‌های Fine-Tuned به دلیل تطبیق دقیق‌تر با داده‌های خاص یک وظیفه و بهینه‌سازی پارامترهای مدل بر اساس این داده‌ها، عملکرد بهتری نسبت به مدل‌های Feature-Based دارند. در فرآیند Fine-Tuning، مدل از پیش آموزش‌دیده (مانند

BERT) با داده‌های خاص یک وظیفه مجدداً آموزش می‌بیند، که باعث می‌شود مدل ویژگی‌ها و الگوهای مرتبط با آن وظیفه را بهتر شناسایی کند. این بهینه‌سازی دقیق‌تر منجر به افزایش دقت و کارایی مدل در وظیفه مورد نظر می‌شود، در حالی که مدل‌های Feature-Based تنها از ویژگی‌های عمومی استخراج شده استفاده می‌کنند و برای وظایف خاص بهینه نشده‌اند.

IV. نمونه‌های اشتباه پیش‌بینی شده توسط مدل

A. مدل BERT

```
- Fake news but detected as true:
@Juillet_Dix Sweden's covid19 cases are not spiking and they
have no face masks and no social distancing. And if you get
Covid, you just have to take Hydroxychloroquine &amp;amp; zinc and
it's gone in 6 days.

- True news but detected as fake:
States reported 630 deaths. We are still seeing a solid national
decline. Death reporting lags approximately 28 days from symptom
onset according to CDC models that consider lags in symptoms
time in hospital and the death reporting process.
```

خبر جعلی که به عنوان واقعی تشخیص داده شده: ادعای مربوط به وضعیت کووید-۱۹ در سوئد به عنوان واقعی تشخیص داده شده است. در این خبر به داروهای واقعی مانند هیدروکسی‌کلروکین و روی اشاره شده که به طور گسترده‌ای مورد بحث قرار گرفته‌اند. همچنین ایده مدیریت بیماری توسط برخی کشورها بدون اقدامات سختگیرانه ممکن است با بحث‌های معروف درباره روش ملایم‌تر سوئد همخوانی داشته باشد.

خبر واقعی که به عنوان جعلی تشخیص داده شده: گزارش مرگ‌های گزارش‌شده در ایالت‌ها به عنوان جعلی تشخیص داده شده ممکن است به دلیل چند عامل باشد: اولاً، جزئیات دقیق درباره تاخیرهای زمانی و مدل‌های آماری ممکن است بیش از حد پیچیده یا ساختگی به نظر برسد. همچنین مدل ممکن است به درستی زمینه تأخیر در گزارش مرگ و میرها را درک نکند، که منجر به تفسیر نادرست این اطلاعات به عنوان اطلاعات نادرست می‌شود.

B. مدل BERT + BiGRU

```
- Fake news but detected as true:
#Florida smashes record for most new COVID-19 cases any state
has in a single day

-True news but detected as fake:
Donald Trump has claimed he "up-played" the seriousness of the
coronavirus pandemic - despite admitting earlier this year he
had "wanted to always play it down"
```

خبر جعلی که به عنوان واقعی تشخیص داده شده: ادعای شکستن رکورد فلوریدا برای بیشترین موارد جدید کووید-۱۹ در یک روز توسط هر ایالت به عنوان واقعی تشخیص داده شده است، زیرا این نوع اخبار پرهیجان و برجسته‌سازی شده در دوران اوج همه‌گیری رایج بودند و به نظر می‌رسد که با الگوهای اخبار واقعی همخوانی داشته باشد. علاوه بر این، گزارش‌هایی درباره افزایش شدید موارد کووید-۱۹ در ایالات متحده به طور مکرر منتشر می‌شد که ممکن است باعث شده این ادعا قابل باورتر به نظر برسد.

خبر واقعی که به عنوان جعلی تشخیص داده شده: ادعای دونالد ترامپ مبنی بر اینکه او "جدیت پاندمی کرونا را بیشتر نشان داده" با وجود اینکه قبلاً اعتراف کرده بود "همیشه می‌خواسته آن را کم اهمیت جلوه دهد" به عنوان جعلی تشخیص داده شده است. این به احتمال زیاد به دلیل تضاد ظاهری در اظهارات ترامپ است که می‌تواند به نظر برسد خبری ضد و نقیض و غیرقابل باور است. همچنین جعل اخبار مربوط به تغییر موضع یا اظهارات متناقض رهبران سیاسی متداول است.

C. مدل CT-BERT + BiGRU

```
- Fake news but detected as true:
@BorisJohnson PM I think it's about time the Public were told
the truth about the origin of this virus please . Some of us
suspect it is a biological weapon created in a Chinese Lab , so
please come clean so everyone can then know how serious this
crisis is &amp;amp; we can all unite &amp;amp; fight #Covid19

- True news but detected as fake:
*DNA Vaccine: injecting genetic material into the host so that
host cells create proteins that are similar to those in the
virus against which the host then creates antibodies
```

خبر جعلی که به عنوان واقعی تشخیص داده شده: ادعای مربوط به درخواست از بوریس جانسون برای فاش کردن حقیقت درباره منشأ ویروس و بیان اینکه برخی معتقدند این ویروس یک سلاح بیولوژیکی ساخته شده در آزمایشگاهی چینی است، به عنوان واقعی تشخیص داده شده است. این احتمالاً به دلیل گسترش وسیع نظریه‌های توطئه و اطلاعات نادرست درباره منشا ویروس در دوران همه‌گیری است که باعث شده این نوع اظهارات به طور گسترده‌ای در شبکه‌های اجتماعی منتشر شود و به نظر معتبر برسد.

خبر واقعی که به عنوان جعلی تشخیص داده شده: خبر واقعی درباره واکسن DNA که در آن ماده ژنتیکی به میزبان تزریق می‌شود تا سلول‌های میزبان پروتئین‌های مشابه با ویروس تولید کنند و در نتیجه بدن پادتن ایجاد کند، به عنوان جعلی تشخیص داده شده است. این احتمالاً به دلیل پیچیدگی و تخصصی بودن اطلاعات علمی موجود در این خبر است. همچنین مفاهیم نوآورانه و پیشرفته ممکن است به دلیل کمبود داده‌های آموزشی مشابه، غیرقابل باور تشخیص داده شوند.

پرسش 2 - به‌کارگیری مدل‌های Transformer برای طبقه‌بندی تصاویر

2-1. آشنایی با مدل‌های Transformer

I. ساختار و نحوه کارکرد Vision Transformer

ViT یک مدل تبدیل کننده برای Vision است. ViT یک تصویر ورودی را به یک سری Patch-ها (مانند تقسیم متن به Token-ها) تقسیم می کند، هر Patch را به یک بردار تبدیل می کند و آن را با یک ضرب ماتریسی به یک بعد کوچکتر map می کند.

ViT از مکانیزم Self Attantion استفاده می کند که به مدل اجازه می دهد هم روابط Local و هم روابط Global در تصاویر را درک کند. این مکانیزم توجه به مدل اجازه می دهد تا اهمیت هر بخش از داده های ورودی را به طور متفاوت وزن دهد.

این مدل‌ها با الهام از موفقیت Transformer-ها در NLP ساخته شده‌اند. این مدل ها در کارایی محاسباتی و دقت تقریبا بهتر از شبکه های عصبی پیچشی (CNNs) عمل می کنند که در حال حاضر بهترین در دید کامپیوتری هستند و به طور گسترده ای برای وظایف مختلف شناسایی تصویر استفاده می شوند.

II. معماری و ساختار کلی مدل ViT

در اینجا به توضیح ساختار و نحوه کارکرد ViT می‌پردازیم:

A. پیش‌پردازش تصاویر

قبل از اعمال Transformer، تصویر ورودی باید به یک سری تکه‌های کوچک (Patch) تقسیم شود. این مراحل عبارتند از:

- تقسیم تصویر به Patch-ها: تصویر با ابعاد (H, W, C) به Patch-های مربعی کوچک با ابعاد ثابت P * P تقسیم می‌شود. بنابراین تعداد کل Patch-ها برابر خواهد بود با (W/P) * (H/P).
- تبدیل Patch-ها به توالی: هر Patch با ابعاد C * P * P به یک وکتور خطی با ابعاد C * P * P تبدیل می‌شود.

B. *Patch Embedding*

پس از تبدیل Patch-ها به وکتورهای خطی، این وکتورها از طریق یک لایه خطی به ابعاد ثابت D تبدیل می‌شوند. بنابراین هر Patch به یک وکتور D بعدی تبدیل می‌شود.

C. *Positional Encoding*

از آنجا که ترتیب و مکان Patch-ها در تصویر مهم است، به هر وکتور Patch یک بردار Positional Encoding اضافه می‌شود تا مدل بتواند اطلاعات مکانی را حفظ کند.

D. *ورود به Transformer*

توالی وکتورهای Patch همراه با Positional Encoding به عنوان ورودی به Transformer داده می‌شود. ساختار Transformer شامل چندین لایه Transformer بلوک است که هر بلوک شامل مراحل زیر می‌باشد:

- Multi-head Self-Attention Layer : که توجه هر وکتور به تمام وکتورهای دیگر را محاسبه می‌کند.
- Feed Forward: که یک شبکه عصبی دو لایه با توابع غیرخطی است.
- Layer Normalization: که به نرمال‌سازی ورودی‌ها کمک می‌کند.
- Residual Connection: که برای هر لایه، ورودی اصلی را به خروجی لایه اضافه می‌کند تا به پایداری آموزش کمک کند.

E. Classification

بعد از عبور از تمام لایه‌های Transformer، توالی وکتورها به یک وکتور واحد تبدیل می‌شود. سپس این وکتور از یک لایه خطی عبور داده می‌شود تا کلاس‌های خروجی پیش‌بینی شود.

III. معایب و بهبود معماری ViT

یکی از مهم‌ترین مشکلات Transformer-ها Computation Time می‌باشد که به دلیل پیچیدگی معماری و زیاد بودن پارامترها زمان و داده زیادی برای آموزش صرف می‌شود.

این مقاله یک معماری سلسله‌مراتبی نوآورانه برای Transformer-ها به نام FDViT پیشنهاد می‌دهد تا به چالش Computation Time بپردازد. ایده‌های کلیدی شامل موارد زیر هستند:

A. معرفی لایه کاهش‌نمونه‌گیری انعطاف‌پذیر (FD) که به stride صحیح محدود نیست و اجازه می‌دهد کاهش ابعاد فضایی به صورت هموار انجام شود تا از از دست رفتن بیش از حد اطلاعات جلوگیری شود.
B. استفاده از معماری خودرمزگذار نقاب‌دار برای تسهیل آموزش لایه‌های FD و تولید خروجی‌های اطلاعاتی.

FDViT پیشنهادی با استفاده از تعداد کمتری از FLOPs و پارامترها، عملکرد بهتری در دسته‌بندی نسبت به مدل‌های سلسله‌مراتبی موجود ترانسفورمر بصری دارد. آزمایش‌ها روی مجموعه داده‌های COCO، ImageNet و ADE20K اثربخشی این روش را نشان می‌دهند.

همچنین این مقاله یک ماژول به نام TokenLearner را مورد بحث قرار می‌دهد که می‌تواند کارایی و دقت مدل‌های ViT را بهبود بخشد. TokenLearner یک ماژول قابل یادگیری است که به جای استفاده از توکن‌سازی ثابت و یکنواخت، مجموعه‌ای کوچک‌تر از توکن‌های تطبیقی را از تصویر یا ویدئو ورودی تولید می‌کند. این کار باعث کاهش تعداد توکن‌هایی می‌شود که نیاز به پردازش توسط لایه‌های بعدی ترانسفورمر دارند و منجر به صرفه‌جویی قابل توجهی در حافظه و محاسبات می‌شود بدون اینکه عملکرد را کاهش دهد. در مقاله آزمایش‌هایی را ارائه داده شده که نشان می‌دهد با قرار دادن TokenLearner در مکان‌های مختلف داخل مدل ViT، می‌توان دقتی معادل یا بهتر از ViT پایه را به دست آورد، در حالی که هزینه محاسباتی تا دو سوم کاهش می‌یابد. TokenLearner به ویژه برای وظایف درک ویدئو بسیار مؤثر است و در چندین معیار عملکرد پیشرفته‌ای را ارائه می‌دهد.

CNN Fine-Tuning .2-2

در اینجا ما مدل VGG19 را Fine-Tune کردیم. برای این کار ابتدا مدل Pre-Trained را به شکل زیر ساختیم (نوع مدل آموزش داده شده را پارامتر weights مشخص می‌کند که مقدار imagenet برابر با مدل آموزش داده شده روی دیتاست imagenet1k می‌باشد):

```
def create_VGG19_model(input_shape: tuple, num_classes: int) ->
tuple[keras.Model, keras.Model]:
      cnn_base = VGG19(weights="imagenet", include_top=False,
input_shape=input_shape)
      model = Sequential()
      model.add(cnn_base)
      model.add(Flatten())
      model.add(Dense(256, activation="relu"))
      model.add(Dropout(0.5))
      model.add(Dense(256, activation="relu"))
      model.add(Dropout(0.5))
      model.add(Dense(num_classes, activation="softmax"))

      optimizer = Adam(learning_rate=0.0001)
      model.compile(optimizer=optimizer,
loss="categorical_crossentropy", metrics=["accuracy"])
      return cnn_base, model
```

در حالت ابتدایی مدل شامل لایه‌های زیر بود (مقدار دوم Trainable را نشان می‌دهد)

```
0 input_1 True
1 block1_conv1 True
2 block1_conv2 True
3 block1_pool True
4 block2_conv1 True
5 block2_conv2 True
6 block2_pool True
7 block3_conv1 True
8 block3_conv2 True
9 block3_conv3 True
10 block3_conv4 True
11 block3_pool True
12 block4_conv1 True
13 block4_conv2 True
14 block4_conv3 True
```

```
15 block4_conv4 True
16 block4_pool True
17 block5_conv1 True
18 block5_conv2 True
19 block5_conv3 True
20 block5_conv4 True
21 block5_pool True
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 20,224,074
Trainable params: 199,690
Non-trainable params: 20,024,384
_________________________________________________________________
```

طبق مقاله باید لایه‌های پس از `block5_conv1` از حالت Freeze خارج می‌شدند و در حالت آموزش قرار می‌گرفتند که برای این کار از تابع زیر استفاده کردیم:

```
def froze_layers(cnn_base: keras.Model, starting_layer_name:
str):
```

```
    cnn_base.trainable = True

    trainable = False

    for layer in cnn_base.layers:
    if layer.name == starting_layer_name:
        trainable = True

    layer.trainable = trainable

froze_layers(cnn_base_vgg19, "block5_conv1")
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570
=================================================================
Total params: 20,224,074
Trainable params: 9,638,922
Non-trainable params: 10,585,152
_________________________________________________________________
```

سپس مدل را دیتاست `cifar-10` آموزش دادیم. ابتدا تعدادی از عکس‌های موجود در دیتاست را مشاهده کنیم:

شکل 2-1. تصاویر نمونه‌ای از ده کلاس موجود در دیتاست

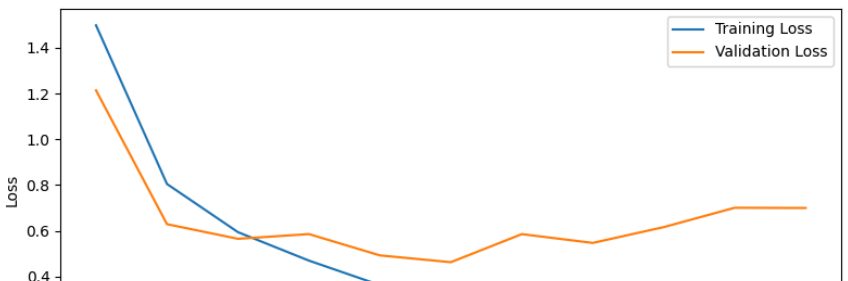
برای آموزش مدل از پارامترهای زیر استفاده شد:

```
def scheduler(epoch: int, lr: float) -> float:
      # The learning rate reduces by a factor of 0.9 and has a
minimum value of 0.0000001.
      return max(0.0000001, lr * 0.9)

LR_sched = LearningRateScheduler(scheduler)
```

```
def create_VGG19_callbacks(model_name: str) ->
list[keras.callbacks.Callback]:
      checkpoint = ModelCheckpoint(f"../models/{model_name}.h5",
monitor="val_accuracy", save_best_only=True)
      early_stopping = EarlyStopping(monitor="val_accuracy",
patience=5)
      epoch_time_callback = EpochTimeCallback()
      return [checkpoint, early_stopping, LR_sched,
epoch_time_callback]
```

```
vgg19_history = vgg_19.fit(
      x_train,
      y_train_onehot,
      batch_size=64,
      epochs=50,
      validation_data=(x_val, y_val_onehot),
      callbacks=vgg19_callbacks,
)
```

نتایج این کار به شکل زیر بود:



شکل 2-2. نمودار Accuracy و Loss برای Fine-Tune کردن مدل VGG19

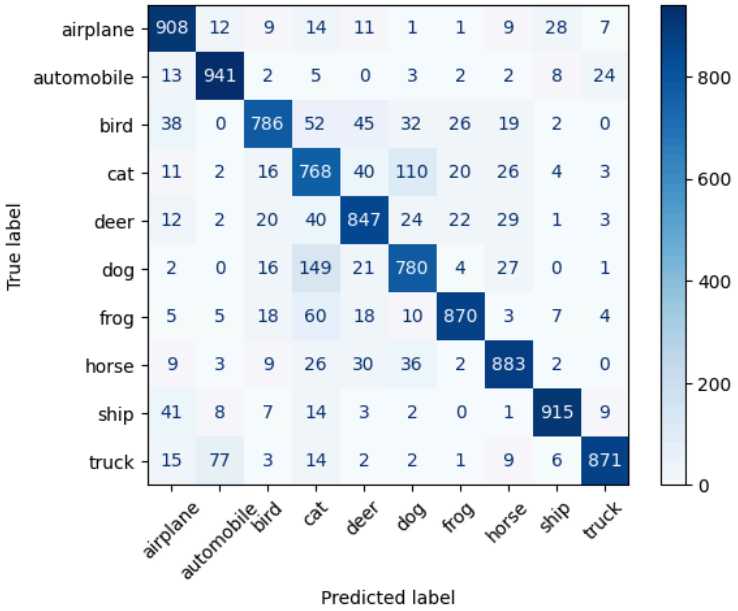
Metric	Value
Training Loss	0.06026863679289818
Validation Loss	0.7001135945320129
Training Accuracy	0.9839110970497131
Validation Accuracy	0.8539999723434448

	precision	recall	f1-score	support
airplane	0.86	0.91	0.88	1000

```
      automobile       0.90      0.94      0.92        1000
            bird       0.89      0.79      0.83        1000
             cat       0.67      0.77      0.72        1000
            deer       0.83      0.85      0.84        1000
             dog       0.78      0.78      0.78        1000
            frog       0.92      0.87      0.89        1000
           horse       0.88      0.88      0.88        1000
            ship       0.94      0.92      0.93        1000
           truck       0.94      0.87      0.91        1000

        accuracy                           0.86       10000
       macro avg       0.86      0.86      0.86       10000
    weighted avg       0.86      0.86      0.86       10000
```

```
Testing took 3.92 seconds
```

```
Average epoch time: 30.89 seconds
```



شکل 2-3. Confusion Matrix برای مدل fine-tune شده VGG19

Transformer Fine-Tuning .3-2

سپس تمام مراحل بالا را برای یک Transformer نیز پیاده‌سازی کردیم، در اینجا از مدل `DeiTBaseDistilled` استفاده کردیم. برای این کار مراحل زیر را رفتیم:

```
transform = torchvision_transforms.Compose([
      torchvision_transforms.ToPILImage(),
      torchvision_transforms.Resize((224, 224)),
      torchvision_transforms.RandomHorizontalFlip(),
      torchvision_transforms.RandomRotation(15),
      torchvision_transforms.ToTensor(),
])
```

توجه کنید که در اینجا برخلاف بخش قبل عکس‌ها را به ابعاد 224 در 224 تغییر دادیم که بتوانیم از مدل Pre-Trained استفاده کنیم. و همچنین مقدار Data Augmentation نیز در این بخش استفاده شده که مشاهده می‌کنید (در بخش قبل نیز با استفاده از `ImageDataGenerator` نیز مدل امتحان شد که تاثیر زیادی در دقت نداشت برای همین حذف شد).

سپس با کمک تابع زیر مدل را ساختیم، توجه کنید که پارامتر اول در ساخت مدل نشان‌دهنده وزن و معماری مدل Pre-Trained است و پارامتر آخر برای این است که Head مربوط به Prediction حذف شود. مقدار 768 نیز خروجی مدل قبلی و ورودی شبکه FC است.

```
def create_DeiTBaseDistilled_model(input_shape: tuple,
num_classes: int, hidden_units: int) -> tuple[nn.Module,
nn.Module]:
      base_model =
timm.create_model("deit_base_distilled_patch16_224",
pretrained=True, num_classes=0)
      model = nn.Sequential(
      nn.Linear(768, hidden_units),
      nn.ReLU(),
      nn.Dropout(0.5),
      nn.Linear(hidden_units, hidden_units),
      nn.ReLU(),
      nn.Dropout(0.5),
      nn.Linear(hidden_units, num_classes),
      )

      return base_model, model
```

حال لایه‌های مدل Transformer را بررسی می‌کنیم:

```
['cls_token', 'pos_embed', 'dist_token',
'patch_embed.proj.weight', 'patch_embed.proj.bias',
'blocks.0.norm1.weight', 'blocks.0.norm1.bias',
'blocks.0.attn.qkv.weight', 'blocks.0.attn.qkv.bias',
'blocks.0.attn.proj.weight', 'blocks.0.attn.proj.bias',
'blocks.0.norm2.weight', 'blocks.0.norm2.bias',
'blocks.0.mlp.fc1.weight', 'blocks.0.mlp.fc1.bias',
'blocks.0.mlp.fc2.weight', 'blocks.0.mlp.fc2.bias',
'blocks.1.norm1.weight', 'blocks.1.norm1.bias',
'blocks.1.attn.qkv.weight', 'blocks.1.attn.qkv.bias',
'blocks.1.attn.proj.weight', 'blocks.1.attn.proj.bias',
'blocks.1.norm2.weight', 'blocks.1.norm2.bias',
'blocks.1.mlp.fc1.weight', 'blocks.1.mlp.fc1.bias',
'blocks.1.mlp.fc2.weight', 'blocks.1.mlp.fc2.bias',
'blocks.2.norm1.weight', 'blocks.2.norm1.bias',
'blocks.2.attn.qkv.weight', 'blocks.2.attn.qkv.bias',
'blocks.2.attn.proj.weight', 'blocks.2.attn.proj.bias',
'blocks.2.norm2.weight', 'blocks.2.norm2.bias',
'blocks.2.mlp.fc1.weight', 'blocks.2.mlp.fc1.bias',
'blocks.2.mlp.fc2.weight', 'blocks.2.mlp.fc2.bias',
'blocks.3.norm1.weight', 'blocks.3.norm1.bias',
'blocks.3.attn.qkv.weight', 'blocks.3.attn.qkv.bias',
'blocks.3.attn.proj.weight', 'blocks.3.attn.proj.bias',
'blocks.3.norm2.weight', 'blocks.3.norm2.bias',
'blocks.3.mlp.fc1.weight', 'blocks.3.mlp.fc1.bias',
'blocks.3.mlp.fc2.weight', 'blocks.3.mlp.fc2.bias',
'blocks.4.norm1.weight', 'blocks.4.norm1.bias',
'blocks.4.attn.qkv.weight', 'blocks.4.attn.qkv.bias',
'blocks.4.attn.proj.weight', 'blocks.4.attn.proj.bias',
'blocks.4.norm2.weight', 'blocks.4.norm2.bias',
'blocks.4.mlp.fc1.weight', 'blocks.4.mlp.fc1.bias',
'blocks.4.mlp.fc2.weight', 'blocks.4.mlp.fc2.bias',
'blocks.5.norm1.weight', 'blocks.5.norm1.bias',
'blocks.5.attn.qkv.weight', 'blocks.5.attn.qkv.bias',
'blocks.5.attn.proj.weight', 'blocks.5.attn.proj.bias',
'blocks.5.norm2.weight', 'blocks.5.norm2.bias',
'blocks.5.mlp.fc1.weight', 'blocks.5.mlp.fc1.bias',
'blocks.5.mlp.fc2.weight', 'blocks.5.mlp.fc2.bias',
'blocks.6.norm1.weight', 'blocks.6.norm1.bias',
'blocks.6.attn.qkv.weight', 'blocks.6.attn.qkv.bias',
```

```
'blocks.6.attn.proj.weight', 'blocks.6.attn.proj.bias',
'blocks.6.norm2.weight', 'blocks.6.norm2.bias',
'blocks.6.mlp.fc1.weight', 'blocks.6.mlp.fc1.bias',
'blocks.6.mlp.fc2.weight', 'blocks.6.mlp.fc2.bias',
'blocks.7.norm1.weight', 'blocks.7.norm1.bias',
'blocks.7.attn.qkv.weight', 'blocks.7.attn.qkv.bias',
'blocks.7.attn.proj.weight', 'blocks.7.attn.proj.bias',
'blocks.7.norm2.weight', 'blocks.7.norm2.bias',
'blocks.7.mlp.fc1.weight', 'blocks.7.mlp.fc1.bias',
'blocks.7.mlp.fc2.weight', 'blocks.7.mlp.fc2.bias',
'blocks.8.norm1.weight', 'blocks.8.norm1.bias',
'blocks.8.attn.qkv.weight', 'blocks.8.attn.qkv.bias',
'blocks.8.attn.proj.weight', 'blocks.8.attn.proj.bias',
'blocks.8.norm2.weight', 'blocks.8.norm2.bias',
'blocks.8.mlp.fc1.weight', 'blocks.8.mlp.fc1.bias',
'blocks.8.mlp.fc2.weight', 'blocks.8.mlp.fc2.bias',
'blocks.9.norm1.weight', 'blocks.9.norm1.bias',
'blocks.9.attn.qkv.weight', 'blocks.9.attn.qkv.bias',
'blocks.9.attn.proj.weight', 'blocks.9.attn.proj.bias',
'blocks.9.norm2.weight', 'blocks.9.norm2.bias',
'blocks.9.mlp.fc1.weight', 'blocks.9.mlp.fc1.bias',
'blocks.9.mlp.fc2.weight', 'blocks.9.mlp.fc2.bias',
'blocks.10.norm1.weight', 'blocks.10.norm1.bias',
'blocks.10.attn.qkv.weight', 'blocks.10.attn.qkv.bias',
'blocks.10.attn.proj.weight', 'blocks.10.attn.proj.bias',
'blocks.10.norm2.weight', 'blocks.10.norm2.bias',
'blocks.10.mlp.fc1.weight', 'blocks.10.mlp.fc1.bias',
'blocks.10.mlp.fc2.weight', 'blocks.10.mlp.fc2.bias',
'blocks.11.norm1.weight', 'blocks.11.norm1.bias',
'blocks.11.attn.qkv.weight', 'blocks.11.attn.qkv.bias',
'blocks.11.attn.proj.weight', 'blocks.11.attn.proj.bias',
'blocks.11.norm2.weight', 'blocks.11.norm2.bias',
'blocks.11.mlp.fc1.weight', 'blocks.11.mlp.fc1.bias',
'blocks.11.mlp.fc2.weight', 'blocks.11.mlp.fc2.bias',
'norm.weight', 'norm.bias']
```

حال معماری اولیه بخش Transformer را نگاه می‌کنیم:

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
VisionTransformerDistilled               153,600
├─PatchEmbed: 1-1                        --
│    └─Conv2d: 2-1                       590,592
│    └─Identity: 2-2                     --
├─Dropout: 1-2                           --
├─Identity: 1-3                          --
├─Identity: 1-4                          --
├─Sequential: 1-5                        --
│    └─Block: 2-3                        --
│    │    └─LayerNorm: 3-1                    1,536
│    │    └─Attention: 3-2                    2,362,368
│    │    └─Identity: 3-3                     --
│    │    └─Identity: 3-4                     --
│    │    └─LayerNorm: 3-5                    1,536
│    │    └─Mlp: 3-6                          4,722,432
│    │    └─Identity: 3-7                     --
│    │    └─Identity: 3-8                     --
│    └─Block: 2-4                        --
│    │    └─LayerNorm: 3-9                    1,536
│    │    └─Attention: 3-10                   2,362,368
│    │    └─Identity: 3-11                    --
│    │    └─Identity: 3-12                    --
│    │    └─LayerNorm: 3-13                   1,536
│    │    └─Mlp: 3-14                         4,722,432
│    │    └─Identity: 3-15                    --
│    │    └─Identity: 3-16                    --
│    └─Block: 2-5                        --
│    │    └─LayerNorm: 3-17                   1,536
│    │    └─Attention: 3-18                   2,362,368
│    │    └─Identity: 3-19                    --
│    │    └─Identity: 3-20                    --
│    │    └─LayerNorm: 3-21                   1,536
│    │    └─Mlp: 3-22                         4,722,432
│    │    └─Identity: 3-23                    --
│    │    └─Identity: 3-24                    --
│    └─Block: 2-6                        --
│    │    └─LayerNorm: 3-25                   1,536
```

```
|     |     └─Attention: 3-26              2,362,368
|     |     └─Identity: 3-27               --
|     |     └─Identity: 3-28               --
|     |     └─LayerNorm: 3-29              1,536
|     |     └─Mlp: 3-30                    4,722,432
|     |     └─Identity: 3-31               --
|     |     └─Identity: 3-32               --
|     └─Block: 2-7                    --
|     |     └─LayerNorm: 3-33              1,536
|     |     └─Attention: 3-34              2,362,368
|     |     └─Identity: 3-35               --
|     |     └─Identity: 3-36               --
|     |     └─LayerNorm: 3-37              1,536
|     |     └─Mlp: 3-38                    4,722,432
|     |     └─Identity: 3-39               --
|     |     └─Identity: 3-40               --
|     └─Block: 2-8                    --
|     |     └─LayerNorm: 3-41              1,536
|     |     └─Attention: 3-42              2,362,368
|     |     └─Identity: 3-43               --
|     |     └─Identity: 3-44               --
|     |     └─LayerNorm: 3-45              1,536
|     |     └─Mlp: 3-46                    4,722,432
|     |     └─Identity: 3-47               --
|     |     └─Identity: 3-48               --
|     └─Block: 2-9                    --
|     |     └─LayerNorm: 3-49              1,536
|     |     └─Attention: 3-50              2,362,368
|     |     └─Identity: 3-51               --
|     |     └─Identity: 3-52               --
|     |     └─LayerNorm: 3-53              1,536
|     |     └─Mlp: 3-54                    4,722,432
|     |     └─Identity: 3-55               --
|     |     └─Identity: 3-56               --
|     └─Block: 2-10                   --
|     |     └─LayerNorm: 3-57              1,536
|     |     └─Attention: 3-58              2,362,368
|     |     └─Identity: 3-59               --
|     |     └─Identity: 3-60               --
|     |     └─LayerNorm: 3-61              1,536
|     |     └─Mlp: 3-62                    4,722,432
|     |     └─Identity: 3-63               --
```

```
│     │     └─Identity: 3-64                --
│     └─Block: 2-11                  --
│     │     └─LayerNorm: 3-65              1,536
│     │     └─Attention: 3-66              2,362,368
│     │     └─Identity: 3-67               --
│     │     └─Identity: 3-68               --
│     │     └─LayerNorm: 3-69              1,536
│     │     └─Mlp: 3-70                    4,722,432
│     │     └─Identity: 3-71               --
│     │     └─Identity: 3-72               --
│     └─Block: 2-12                  --
│     │     └─LayerNorm: 3-73              1,536
│     │     └─Attention: 3-74              2,362,368
│     │     └─Identity: 3-75               --
│     │     └─Identity: 3-76               --
│     │     └─LayerNorm: 3-77              1,536
│     │     └─Mlp: 3-78                    4,722,432
│     │     └─Identity: 3-79               --
│     │     └─Identity: 3-80               --
│     └─Block: 2-13                  --
│     │     └─LayerNorm: 3-81              1,536
│     │     └─Attention: 3-82              2,362,368
│     │     └─Identity: 3-83               --
│     │     └─Identity: 3-84               --
│     │     └─LayerNorm: 3-85              1,536
│     │     └─Mlp: 3-86                    4,722,432
│     │     └─Identity: 3-87               --
│     │     └─Identity: 3-88               --
│     └─Block: 2-14                  --
│     │     └─LayerNorm: 3-89              1,536
│     │     └─Attention: 3-90              2,362,368
│     │     └─Identity: 3-91               --
│     │     └─Identity: 3-92               --
│     │     └─LayerNorm: 3-93              1,536
│     │     └─Mlp: 3-94                    4,722,432
│     │     └─Identity: 3-95               --
│     │     └─Identity: 3-96               --
├─LayerNorm: 1-6                    1,536
├─Identity: 1-7                     --
├─Dropout: 1-8                      --
├─Identity: 1-9                     --
├─Identity: 1-10                    -
```

```
================================================================
Total params: 85,800,192
Trainable params: 85,800,192
Non-trainable params: 0
================================================================
```

طبق گفته مقاله باید از 12-امین بلاک Transfomer، شروع به Unfreeze کردن کنیم و بقیه لایه‌ها را Freeze کنیم. برای این کار به شکل زیر عمل کردیم:

```
def froze_layers_pytorch(model: nn.Module, starting_layer_name:
str):
      model.train()

      trainable = False

      for name, param in model.named_parameters():
      if starting_layer_name in name:
            trainable = True

      param.requires_grad = trainable

froze_layers_pytorch(deit_base_distilled_transformer,
"blocks.11")
```

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
VisionTransformerDistilled               153,600
├─PatchEmbed: 1-1                        --
│     └─Conv2d: 2-1                      (590,592)
│     └─Identity: 2-2                    --
├─Dropout: 1-2                           --
├─Identity: 1-3                          --
├─Identity: 1-4                          --
├─Sequential: 1-5                        --
│     └─Block: 2-3                       --
│     │     └─LayerNorm: 3-1                   (1,536)
│     │     └─Attention: 3-2                   (2,362,368)
│     │     └─Identity: 3-3                    --
│     │     └─Identity: 3-4                    --
│     │     └─LayerNorm: 3-5                   (1,536)
│     │     └─Mlp: 3-6                         (4,722,432)
│     │     └─Identity: 3-7                    --
│     │     └─Identity: 3-8                    --
│     └─Block: 2-4                       --
│     │     └─LayerNorm: 3-9                   (1,536)
│     │     └─Attention: 3-10                  (2,362,368)
```

```
|     |     └─Identity: 3-11                --
|     |     └─Identity: 3-12                --
|     |     └─LayerNorm: 3-13               (1,536)
|     |     └─Mlp: 3-14                     (4,722,432)
|     |     └─Identity: 3-15                --
|     |     └─Identity: 3-16                --
|     └─Block: 2-5                    --
|     |     └─LayerNorm: 3-17               (1,536)
|     |     └─Attention: 3-18               (2,362,368)
|     |     └─Identity: 3-19                --
|     |     └─Identity: 3-20                --
|     |     └─LayerNorm: 3-21               (1,536)
|     |     └─Mlp: 3-22                     (4,722,432)
|     |     └─Identity: 3-23                --
|     |     └─Identity: 3-24                --
|     └─Block: 2-6                    --
|     |     └─LayerNorm: 3-25               (1,536)
|     |     └─Attention: 3-26               (2,362,368)
|     |     └─Identity: 3-27                --
|     |     └─Identity: 3-28                --
|     |     └─LayerNorm: 3-29               (1,536)
|     |     └─Mlp: 3-30                     (4,722,432)
|     |     └─Identity: 3-31                --
|     |     └─Identity: 3-32                --
|     └─Block: 2-7                    --
|     |     └─LayerNorm: 3-33               (1,536)
|     |     └─Attention: 3-34               (2,362,368)
|     |     └─Identity: 3-35                --
|     |     └─Identity: 3-36                --
|     |     └─LayerNorm: 3-37               (1,536)
|     |     └─Mlp: 3-38                     (4,722,432)
|     |     └─Identity: 3-39                --
|     |     └─Identity: 3-40                --
|     └─Block: 2-8                    --
|     |     └─LayerNorm: 3-41               (1,536)
|     |     └─Attention: 3-42               (2,362,368)
|     |     └─Identity: 3-43                --
|     |     └─Identity: 3-44                --
|     |     └─LayerNorm: 3-45               (1,536)
|     |     └─Mlp: 3-46                     (4,722,432)
|     |     └─Identity: 3-47                --
|     |     └─Identity: 3-48                --
```

```
|     └─Block: 2-9                    --
|     |     └─LayerNorm: 3-49               (1,536)
|     |     └─Attention: 3-50               (2,362,368)
|     |     └─Identity: 3-51                --
|     |     └─Identity: 3-52                --
|     |     └─LayerNorm: 3-53               (1,536)
|     |     └─Mlp: 3-54                     (4,722,432)
|     |     └─Identity: 3-55                --
|     |     └─Identity: 3-56                --
|     └─Block: 2-10                   --
|     |     └─LayerNorm: 3-57               (1,536)
|     |     └─Attention: 3-58               (2,362,368)
|     |     └─Identity: 3-59                --
|     |     └─Identity: 3-60                --
|     |     └─LayerNorm: 3-61               (1,536)
|     |     └─Mlp: 3-62                     (4,722,432)
|     |     └─Identity: 3-63                --
|     |     └─Identity: 3-64                --
|     └─Block: 2-11                   --
|     |     └─LayerNorm: 3-65               (1,536)
|     |     └─Attention: 3-66               (2,362,368)
|     |     └─Identity: 3-67                --
|     |     └─Identity: 3-68                --
|     |     └─LayerNorm: 3-69               (1,536)
|     |     └─Mlp: 3-70                     (4,722,432)
|     |     └─Identity: 3-71                --
|     |     └─Identity: 3-72                --
|     └─Block: 2-12                   --
|     |     └─LayerNorm: 3-73               (1,536)
|     |     └─Attention: 3-74               (2,362,368)
|     |     └─Identity: 3-75                --
|     |     └─Identity: 3-76                --
|     |     └─LayerNorm: 3-77               (1,536)
|     |     └─Mlp: 3-78                     (4,722,432)
|     |     └─Identity: 3-79                --
|     |     └─Identity: 3-80                --
|     └─Block: 2-13                   --
|     |     └─LayerNorm: 3-81               (1,536)
|     |     └─Attention: 3-82               (2,362,368)
|     |     └─Identity: 3-83                --
|     |     └─Identity: 3-84                --
|     |     └─LayerNorm: 3-85               (1,536)
```

```
│     │     └─Mlp: 3-86                       (4,722,432)
│     │     └─Identity: 3-87                  --
│     │     └─Identity: 3-88                  --
│     └─Block: 2-14                     --
│     │     └─LayerNorm: 3-89                 1,536
│     │     └─Attention: 3-90                 2,362,368
│     │     └─Identity: 3-91                  --
│     │     └─Identity: 3-92                  --
│     │     └─LayerNorm: 3-93                 1,536
│     │     └─Mlp: 3-94                       4,722,432
│     │     └─Identity: 3-95                  --
│     │     └─Identity: 3-96                  --
├─LayerNorm: 1-6                       1,536
├─Identity: 1-7                        --
├─Dropout: 1-8                         --
├─Identity: 1-9                        --
├─Identity: 1-10                       --
==========================================================
Total params: 85,800,192
Trainable params: 7,089,408
Non-trainable params: 78,710,784
==========================================================
```

و بخش Fully Connected نیز دارای معماری زیر بود:

```
==========================================================
Layer (type:depth-idx)                 Param #
==========================================================
Sequential                             --
├─Linear: 1-1                          196,864
├─ReLU: 1-2                            --
├─Dropout: 1-3                         --
├─Linear: 1-4                          65,792
├─ReLU: 1-5                            --
├─Dropout: 1-6                         --
├─Linear: 1-7                          2,570
==========================================================
Total params: 265,226
Trainable params: 265,226
Non-trainable params: 0
==========================================================
```

حال دو مدل را با یکدیگر ترکیب می‌کنیم:

```
class DeiTBaseDistilledModel(nn.Module):
     def __init__(self, base_model: nn.Module, fc: nn.Module):
     super(DeiTBaseDistilledModel, self).__init__()
     self.base_model = base_model
     self.fc = fc

     def forward(self, x):
     x = self.base_model(x)
     x = x.view(x.size(0), -1) # Flatten
     x = self.fc(x)
     return x

deit_base_distilled_model =
DeiTBaseDistilledModel(deit_base_distilled_transformer,
deit_base_distilled_fc)
deit_base_distilled_model.to(device)
```

مدل نهایی دارای معماری و مشخصات زیر است:

```
DeiTBaseDistilledModel(
  (base_model): VisionTransformerDistilled(
     (patch_embed): PatchEmbed(
     (proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16,
16))
     (norm): Identity()
     )
     (pos_drop): Dropout(p=0.0, inplace=False)
     (patch_drop): Identity()
     (norm_pre): Identity()
     (blocks): Sequential(
     (0): 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()
     )
     (1): 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()
     )
     (2): 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()
     )
     (3): 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()
      )
      (4): 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()
      )
      (5): 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()
      )
      (6): 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()
      )
      (7): 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()
     )
     (8): 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()
     )
     (9): 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()
     )
     (10): 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()
      )
      (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()
```

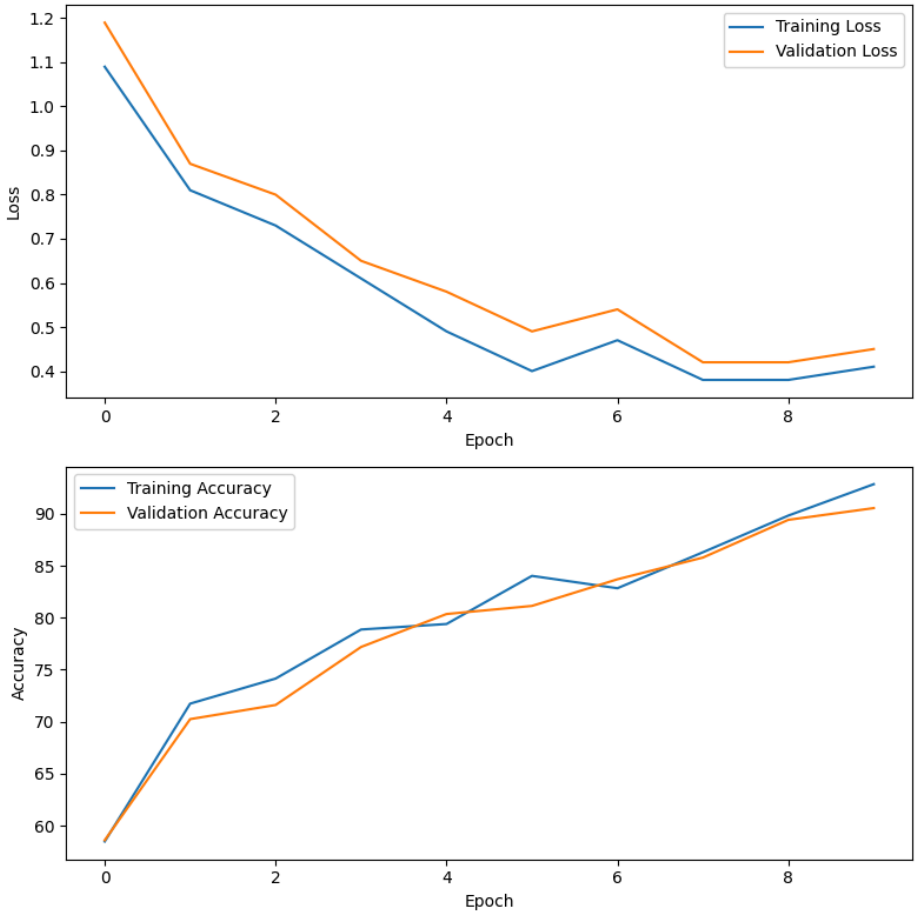
```
      (norm2): LayerNorm((768,), eps=1e-06,
elementwise_affine=True)
      (mlp): Mlp(
            (fc1): Linear(in_features=768, out_features=3072,
bias=True)
            (act): GELU(approximate='none')
            (drop1): Dropout(p=0.0, inplace=False)
            (norm): Identity()
            (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): Identity()
      (head_dist): Identity()
  )
  (fc): Sequential(
      (0): Linear(in_features=768, out_features=256, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=256, out_features=256, bias=True)
      (4): ReLU()
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=256, out_features=10, bias=True)
  )
)
```

سپس به آموزش مدل با پارامترهای زیر پرداختیم (به خاطر محدودیت سیستم از batch_size با اندازه 64 استفاده شد و تنها 10 Epoch مدل آموزش داده شد):

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(deit_base_distilled_model.parameters(),
lr=0.0001)
def train_model(
      model: nn.Module,
      criterion: nn.Module,
      optimizer: optim.Optimizer,
      train_loader: DataLoader,
      val_loader: DataLoader,
      num_epochs: int,
) -> tuple[list, list, list, list]:
      train_loss = []
      val_loss = []
      train_acc = []
      val_acc = []
      epoch_times = []

      for epoch in range(num_epochs):
      model.train()

      start = time.time()

      for x, y in train_loader:
            x, y = x.to(device), y.to(device)

            optimizer.zero_grad()
            y_pred = model(x)

            loss = criterion(y_pred, y)
            loss.backward()
            optimizer.step()

            train_loss.append(loss.item().__round__(2))
            train_acc.append((y_pred.argmax(dim=1) ==
y).float().mean().item().__round__(2))

      model.eval()
```

```
      with torch.no_grad():
            losses = []
            accuracies = []

            for x, y in val_loader:
                  x, y = x.to(device), y.to(device)
                  y_pred = model(x)

                  loss = criterion(y_pred, y)
                  losses.append(loss.item())
                  accuracies.append((y_pred.argmax(dim=1) ==
y).float().mean().item())

            val_loss.append(np.mean(losses).__round__(2))
            val_acc.append(np.mean(accuracies).__round__(2))

      finish = time.time()
      epoch_times.append((finish - start).__round__(2))

      print(f"Epoch {epoch+1} took {finish - start:.2f} seconds,
Train Loss: {train_loss[-1]}, Train Accuracy: {train_acc[-1]},
Validation Loss: {val_loss[-1]}, Validation Accuracy:
{val_acc[-1]}")

      return train_loss, val_loss, train_acc, val_acc,
epoch_times
deit_base_distilled_train_loss, deit_base_distilled_val_loss,
deit_base_distilled_train_acc, deit_base_distilled_val_acc,
deit_base_distilled_epoch_times =
train_model(deit_base_distilled_model, criterion, optimizer,
train_loader, val_loader, 10)
```

نتایج به شکل زیر بود:



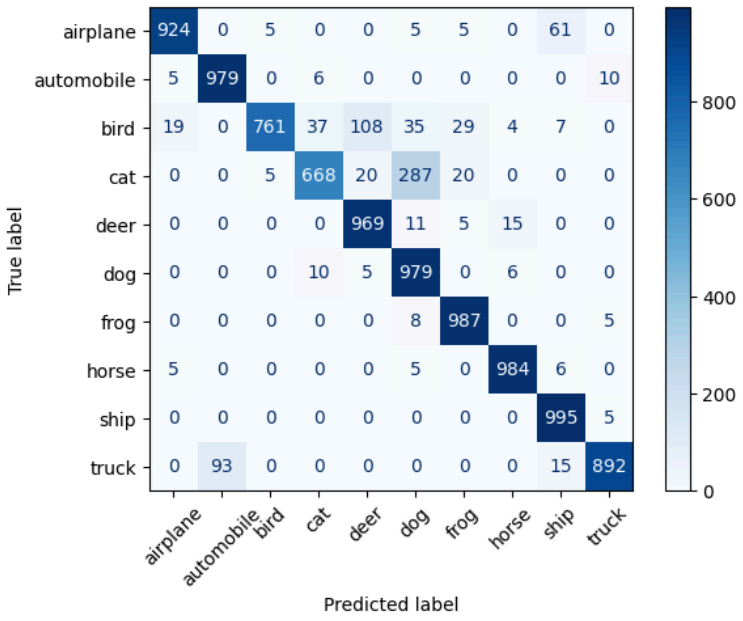
شکل 2-4. نمودار Accuracy و Loss برای Fine-Tune کردن مدل DeiTBaseDistilled

Metric	Value
Training Loss	0.41
Validation Loss	0.45
Training Accuracy	92.85
Validation Accuracy	90.55

```
              precision    recall  f1-score   support

    airplane       0.97      0.92      0.95      1000
  automobile       0.91      0.98      0.94      1000
        bird       0.99      0.76      0.86      1000
         cat       0.93      0.67      0.78      1000
        deer       0.88      0.97      0.92      1000
         dog       0.74      0.98      0.84      1000
        frog       0.94      0.99      0.96      1000
       horse       0.98      0.98      0.98      1000
        ship       0.92      0.99      0.95      1000
       truck       0.98      0.89      0.93      1000

    accuracy                           0.91     10000
   macro avg       0.92      0.91      0.91     10000
weighted avg       0.92      0.91      0.91     10000
```

```
Average epoch time: 239.05 seconds
```

```
Time taken: 4.02 seconds
```



شکل 2-5. Confusion Matrix برای مدل fine-tune شده DeiTBaseDistilled

4-2. نتایج

حال به بررسی عملکرد مدل‌ها و مقایسه بین آن‌ها می‌پردازیم، توجه داشته باشید که مدل `DenseNet201` نیز (که جزو مدل‌های CNN-ای می‌باشد) نیز آموزش داده شد که می‌توانید در کد نتایج را مشاهده کنید.

با توجه به Classification Report می‌توان نتیجه گرفت که هر دو مدل با دقت خوبی در حال طبقه‌بندی می‌باشند. دقت مدل CNN حدود 86 درصد و دقت مدل Transformer حدود 81 درصد بود که نشان‌دهنده این موضوع می‌باشد که مدل Transformer با وجود اینکه پارامتر کمتری داشت (البته پارامتر Trainable، ولی در کل پارامترهای بسیار بیشتری داشت) دقت و عملکرد بهتری دارد. این فرضیه را می‌توان از بررسی F1 و Recall نیز تایید کرد. با بررسی Confusion Matrix-ها نیز می‌توان دید که مدل Transformer به جز برای کلاس‌های `bird` و `cat` که به ترتیب با `deer` و با `dog` و همچنین اشتباه گرفتن کلاس `truck` با `automobile` مقداری را اشتباه تشخیص داده بقیه کلاس‌ها دقت بسیار خوبی دارند و تقریبا به طور کامل توانسته تشخیص دهد. مدل CNN نیز در این موارد مشکل داشته البته به طور کلی در طبقه‌بندی `bird` ضعیف عمل کرده و بسیاری از عکس‌ها را اشتباه تشخیص داده است. نمودارهای Accuracy و Loss و بقیه پارامترها نیز برتری مدل Transformer را نشان می‌دهد هر چند مدل CNN نیز دقت قابل‌قبولی دارد.

مدل Transformer با اینکه دقت بهتری دارد اما زمان بسیار بیشتری (حدود 7 برابر) در هر آموزش زمان برد و بسیار نیز سنگین‌تر بود و حجم و مموری و میزان بیشتری از GPU درگیر شد (البته از دو کتابخانه متفاوت استفاده شده و سیستم در شرایط کاملا یکسان نبوده و به طور دقیق نتایج قابل مقایسه نیست اما به طور کلی می‌توان دید کلی از نسبت و مقایسه این دو به دست آورد). زمان تست هر دو تا حدی نزدیک بود (البته مال Transformer بیشتر بود که با توجه به چهار برابر بودن تعداد پارامترها منطقی‌ست) و البته احتمالا در صورت تکرار آزمایش اختلاف زمان‌ها بیشتر نیز خواهد بود.

یک نکته مهم در مقایسه عملکرد این دو مدل این است که مدل CNN دارای کمی Overfitting شده، هر چند مدل‌های Transformer به طور بسیار آسیب‌پذیرتر نسبت به این موضوع می‌باشند اما همانطور که گفته شد به دلیل محدودیت سیستم امکان آموزش بیشتر مدل فراهم نبود ولی احتمالا در صورت ادامه مدل کمی Overfitting می‌کرد که به عنوان راه‌حل می‌توانستیم از روش‌هایی مانند Regularization استفاده کنیم که چون در مقاله گفته نشده بود برای یکسان نگه داشتن شرایط در اینجا استفاده نشد، هر چند از Early Stopping برای مدل CNN برای جلوگیری از شدت Overfitting استفاده شد.

دلیل اختلاف حدودا 5 درصدی بین نتایج به دست آمده و مقاله نیز می‌تواند شامل این باشد که مدل ما از `batch_size` کوچک‌تری (زمانی که با 512 مانند مقاله استفاده کردیم مدل اصلا دقت خوبی نداشت) و همچنین استفاده از Learning Rate متفاوت (در مقاله از ضریب 0.6 استفاده می‌شد ولی ما از ضریب 0.9 برای کاهش این پارامتر استفاده کردیم) و احتمالا Pre-Processing متفاوت است که در مقاله هیچ اشاره‌ای به آن نشده بود. دلایل دیگری نیز مانند استفاده از پیاده‌سازی‌های متفاوت و ... نیز ممکن است باعث این اختلاف باشد ولی خب اختلاف طبیعی‌ست و تقریبا در همه حالت‌ها رخ می‌دهد.