به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

نام و نام خانوادگی	مرضیه حاجبی-پویا شیخ الاسلامی
شماره دانشجویی	۸۱۰۱۰۰۳۹۴-۸۱۰۱۰۱۰۰۵
تاریخ ارسال گزارش	۱۴۰۱.۰۳.۱۴

فهرست

شکل‌ها

پاسخ ۱. سامانه پرسش–پاسخ

۱-۱. مدل‌سازی مساله

در BERT دو مرحله وجود دارد: قبل از آموزش و تنظیم دقیق. در طول پیش‌آموزش، مدل بر روی داده‌های بدون برچسب در وظایف مختلف قبل از آموزش آموزش داده می‌شود. برای تنظیم دقیق، ابتدا مدل BERT با پارامترهای از پیش آموزش داده شده مقداردهی اولیه می‌شود و تمام پارامترها با استفاده از داده‌های برچسب‌گذاری شده از وظایف پایین‌دستی تنظیم می‌شوند. هر کار پایین‌دستی دارای مدل‌های تنظیم‌شده جداگانه‌ای است، حتی اگر با پارامترهای از پیش آموزش‌دیده یکسانی مقداردهی اولیه شوند.

مدل معماری مدل BERT یک رمزگذار ترنسفورمر دو طرفه چند لایه است که استفاده از Transformers رایج شده است. BERT را با استفاده از دو وظیفه بدون از قبل آموزش داده می‌شود.

وظیفه شماره ۱: LM پوشانده شده به طور شهودی، یک مدل دو جهته عمیق به شدت قدرتمندتر از مدل چپ به راست یا ترکیب کم عمق یک مدل چپ-تو راست و راست به چپ است. متأسفانه، مدل‌های زبان شرطی استاندارد را فقط می‌توان از چپ به راست یا راست به چپ آموزش داد، زیرا شرطی‌سازی دو جهته به هر کلمه اجازه می‌دهد به طور غیرمستقیم «خود» را ببیند، و مدل می‌تواند کلمه هدف را در یک زمینه چند لایه پیش‌بینی کند.

وظیفه شماره ۲: پیش بینی جمله بعدی (NSP)

بسیاری از وظایف مهم پایین دستی مانند پاسخ به سؤال (QA) و استنتاج زبان طبیعی (NLI) بر اساس درک رابطه بین دو جمله است که مستقیماً توسط مدل‌سازی زبان دریافت نمی‌شود. به منظور آموزش مدلی که روابط جملات را درک می‌کند، ما از قبل برای یک کار پیش‌بینی جمله بعدی باینریزه شده آموزش می‌دهیم که می‌تواند به‌طور بی‌اهمیت از هر پیکره تک‌زبانه‌ای تولید شود. وظیفه NSP ارتباط نزدیکی با اهداف یادگیری را دارد.

Fine tune مدل BERT ساده است زیرا مکانیزم توجه به خود در ترانسفورماتور به BERT اجازه می دهد تا بسیاری از وظایف پایین دستی را مدل سازی کند - خواه شامل متن واحد باشد یا جفت متن - با تعویض ورودی ها و خروجی های مناسب. برای برنامه‌هایی که شامل جفت‌های متنی می‌شوند، یک الگوی رایج این است که قبل از اعمال توجه متقاطع دوطرفه، به طور مستقل جفت‌های متن رمزگذاری شوند.

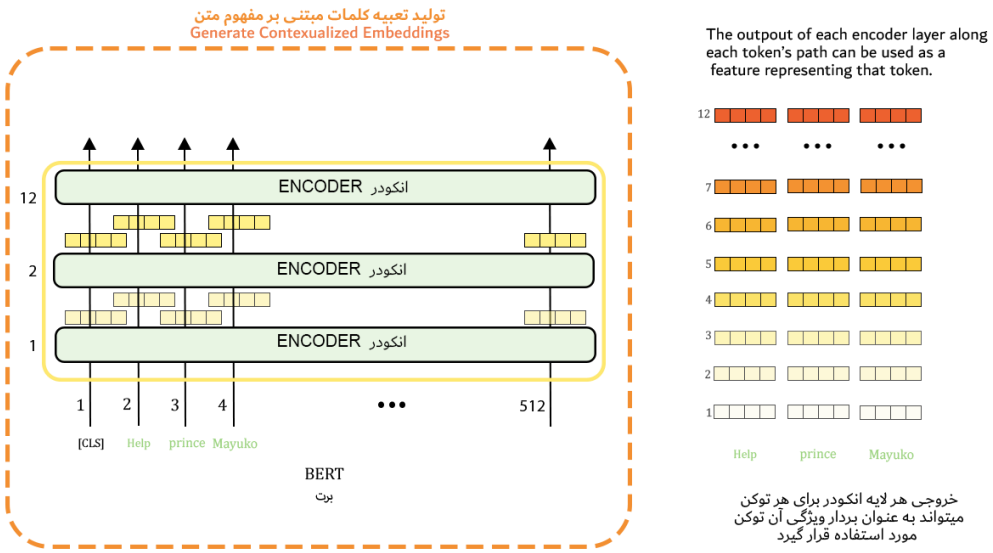
ورودی این مدل یک فهرست به طول ۵۱۲ توکن است. این توکن‌ها از ۱۲ لایه (در مدل پایه) می‌گذرند و در انتها یک بردار با طول ۷۶۸ (در مدل پایه) به‌عنوان خروجی برگردانده می‌شود. این بردار ورودی مدل دیگری برای مسئله‌ی خودمان می‌تواند باشد.

در ورودی، جمله A و جمله B از قبل از آموزش مشابه با (۱) جفت جمله در paragraphing، (۲) جفت فرضیه- مقدمه در مستلزم، (۳) جفت سوال-مقطع در پاسخگویی به سوال، و (۴) یک جفت متن منحط در طبقه بندی متن یا برچسب گذاری توالی .در خروجی، نمایش‌های توکن به یک لایه خروجی برای وظایف سطح نشانه، مانند برچسب‌گذاری دنباله‌ای یا پاسخ‌گویی به سؤال، و نمایش [CLS] به لایه خروجی برای طبقه‌بندی، مانند تحلیل مستلزم یا احساسات وارد می‌شود.

سؤالاتی را که پاسخی ندارند به عنوان دارای یک دامنه پاسخ با شروع و پایان در نشانه [CLS] در نظر می گیریم. فضای احتمال برای موقعیت‌های دامنه پاسخ شروع و پایان گسترش می‌یابد تا موقعیت نشانه [CLS] را نیز در بر گیرد.

مدل BERT درواقع دسته‌ای از انکودرهای مدل ترنسفورمر (Transformer Model) است که آموزش دیده‌اند. هر دو مدل BERT تعداد زیادی لایه‌ی انکودر دارند.

مدل BERTBASE شامل ۱۲ لایه انکودر و مدل بزرگ‌تر که همان مدل BERTLARGE است شامل ۲۴ لایه انکودر است. مدل پایه در مجموع ۱۱۰میلیون پارامتر و مدل بزرگ ۳۴۵میلیون پارامتر دارد. آموزش هر یک از آن‌ها چهار روز زمان برده است. مدل پایه ۷۶۸ و مدل بزرگتر ۱۰۲۴ نود پنهان در لایه‌ی شبکه پیشخور خود دارند و تعداد لایه‌های توجه در اولی ۱۲ و در دومی ۱۶ است.



شکل ۱. مدل BERT

برای این مدل، ابتدا با پیش پردازش داده های آموزشی، تولید برچسب‌هایی برای پاسخ سؤال است که موقعیت‌های شروع و پایان نشانه‌ها مربوط به پاسخ در داخل متن خواهد بود. ابتدا باید متن ورودی را با استفاده از یک توکنایزر به شناسه هایی که مدل می تواند معنادار باشد، تبدیل کنیم یک مدل BERT را تنظیم می‌کنیم:

```
model_name_or_path = 'm3hrdadfi/albert-fa-base-v2'
tokenizer = AutoTokenizer.from_pretrained(model_name_or_path,
do_lower_case=do_lowercase, use_fast=False)
model = AutoModel.from_pretrained(model_name_or_path, config=config)
```

سپس توکن روی داده‌های آموزشی اعمال و خروجی آن به مدل استفاده شده داده می‌شود که همان checkpointطور اشاره شد یکسری پارامترها از قبل مقداردهی می شوند که در این حالت پارامترهای برای حالت صفر و غیرصفر و وزن‌ها و... درنظر گرفته شده است که درنهایت مدل ذخیره شده و مقادیر آبدیت شده آن استفاده می شود.

```
def main():
    set_seed()

    global_step = ""

    model_file = os.path.join(output_dir,'pytorch_model.bin')
    do_train = not os.path.exists(model_file)
    if do_train:
        checkpoints_dir = filter(lambda x:x.startswith('checkpoint_'),
os.listdir(output_dir))
        checkpoint = max(map(lambda x:int(x[x.find('_')+1:]),
checkpoints_dir), default=0)
        if checkpoint == 0:
            config = AutoConfig.from_pretrained(model_name_or_path)
            tokenizer = AutoTokenizer.from_pretrained(model_name_or_path,
do_lower_case=do_lowercase, use_fast=False)
            train_dataset = load_and_cache_examples(tokenizer,
evaluate=False, output_examples=False)
            model = AutoModel.from_pretrained(model_name_or_path,
config=config)
        else:
            checkpoint_output_dir = os.path.join(output_dir,
'checkpoint_{}'.format(checkpoint))
            config = AutoConfig.from_pretrained(checkpoint_output_dir)
```

```
            tokenizer =
AutoTokenizer.from_pretrained(checkpoint_output_dir,
do_lower_case=do_lowercase, use_fast=False)
            train_dataset = load_and_cache_examples(tokenizer,
evaluate=False, output_examples=False)
            model = AutoModel.from_pretrained(checkpoint_output_dir,
config=config)
        model.to(device)
        global_step, tr_loss = train(train_dataset, model, tokenizer,
checkpoint)
        print(" global_step = {}, average loss = {}".format(global_step,
tr_loss))
        print("Saving model checkpoint to {}".format(output_dir))
        model_to_save = model.module if hasattr(model, "module") else model
        model_to_save.save_pretrained(output_dir)
        tokenizer.save_pretrained(output_dir)

    result = {}
    print("Evaluate the last checkpoint")
    config = AutoConfig.from_pretrained(output_dir)
    tokenizer = AutoTokenizer.from_pretrained(output_dir,
do_lower_case=do_lowercase, use_fast=False)
    model = AutoModel.from_pretrained(output_dir, config=config)
    model.to(device)
    eval_result = evaluate(model, tokenizer)
    result = dict((k, v) for k, v in eval_result.items())

    print(result)
    print("\nResult:\n\texact_match: {}\n\tf1: {}".format(result['exact'],
result['f1']))
    return result


if __name__ == "__main__":
    main()
```

۱-۲. پیش‌پردازش داده‌ها

طبق صورت سوال، داده‌های مورد نظر را از آدرس سایت داده شده دانلود و سپس در مدل استفاده شده است.

```
input_dir = os.path.join('/content/drive/MyDrive/HW_5/Dataset/',
dataset_name)
```

۱-۳. پیاده‌سازی مدل

شبکه بیان‌شده در بخش ابتدایی توسط دو مدل که قابل دسترس بودند استفاده شده است:

AlBERT لایه ها به گروه هایی تقسیم می شوند که پارامترها را به اشتراک می گذارند (برای ذخیره حافظه). پیش‌بینی جمله بعدی با پیش‌بینی ترتیب جمله جایگزین می‌شود: در ورودی‌ها، دو جمله A و B داریم (که متوالی هستند) و A را به دنبال B یا B به دنبال آن A را تغذیه می‌کنیم. مدل باید پیش‌بینی کند که آیا آنها تعویض شده‌اند یا خیر. یا نه.

```
model_name_or_path = 'm3hrdadfi/albert-fa-base-v2'
```

به عنوان بخشی از متدولوژی ParsBERT، یک پیش پردازش گسترده با ترکیب برچسب‌گذاری POS و تقسیم‌بندی WordPiece انجام شد تا مجموعه را در قالب مناسبی قرار دهد. این فرآیند بیش از ۴۰ میلیون جمله واقعی تولید می کند.

ParsBERT بر روی سه وظیفه پایین دستی NLP ارزیابی می شود: تجزیه و تحلیل احساسات (SA)، طبقه بندی متن، و شناسایی نهاد نامگذاری شده (NER). برای این موضوع و به دلیل منابع ناکافی، دو مجموعه داده بزرگ برای SA و دو مجموعه برای طبقه‌بندی متن به صورت دستی ساخته شدند که برای استفاده عمومی و معیار در دسترس هستند. ParsBERT از تمام مدل های زبانی دیگر، از جمله BERT چند زبانه و سایر مدل های یادگیری عمیق ترکیبی برای همه کارها بهتر عمل کرد و عملکرد پیشرفته را در مدل سازی زبان فارسی بهبود بخشید.

```
model_name_or_path = 'HooshvareLab/bert-base-parsbert-uncased'
```

```
output_dir = '/content/drive/MyDrive/HW_5/Dataset'
dataset_name = ''
version_2 = False
null_score_diff_threshold = 0
max_seq_length = 512
max_query_length = 64
doc_stride = 128
do_lowercase = False
per_gpu_train_batch_size = 12
per_gpu_eval_batch_size = 8
learning_rate = 3e-5
gradient_accumulation_steps = 1
weight_decay = 0
max_grad_norm = 1
num_train_epochs = 1
```

```
warmup_steps = 0
n_best_size = 20
max_answer_length = 30
seed = 42
threads = 10

"""Parameters"""


def set_seed():
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if n_gpu > 0:
        torch.cuda.manual_seed_all(seed)


def to_list(tensor):
    return tensor.detach().cpu().tolist()


def train(train_dataset, model, tokenizer, start_epoch):
    train_sampler = RandomSampler(train_dataset)
    train_dataloader = DataLoader(train_dataset, sampler=train_sampler,
batch_size=per_gpu_train_batch_size)

    t_total = len(train_dataloader) // gradient_accumulation_steps *
num_train_epochs

    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters() if not any(nd
in n for nd in no_decay)],
            "weight_decay": weight_decay,
        },
        {"params": [p for n, p in model.named_parameters() if any(nd in n
for nd in no_decay)], "weight_decay": 0.0},
    ]

    optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
    scheduler = get_linear_schedule_with_warmup(
        optimizer, num_warmup_steps=warmup_steps,
num_training_steps=t_total
    )

    if os.path.isfile(os.path.join(output_dir, "optimizer.pt")) and
os.path.isfile(
        os.path.join(output_dir, "scheduler.pt")
```

```
    ):
        optimizer.load_state_dict(torch.load(os.path.join(output_dir,
"optimizer.pt")))
        scheduler.load_state_dict(torch.load(os.path.join(output_dir,
"scheduler.pt")))

    print("\nTraining:")

    global_step = 1
    tr_loss, logging_loss = 0.0, 0.0
    model.zero_grad()
    set_seed()

    for epoch_idx in range(num_train_epochs):
        if epoch_idx < start_epoch:
            continue
        epoch_iterator = tqdm(train_dataloader, desc="Iteration in epoch
{}".format(epoch_idx+1))
        for step, batch in enumerate(epoch_iterator):
            model.train()
            batch = tuple(t.to(device) for t in batch)
            inputs = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2],
                "start_positions": batch[3],
                "end_positions": batch[4],
            }
            outputs = model(**inputs)
            loss = outputs[0]
            if n_gpu > 1:
                loss = loss.mean()
            if gradient_accumulation_steps > 1:
                loss = loss / gradient_accumulation_steps
            loss.backward()
            tr_loss += loss.item()
            if (step + 1) % gradient_accumulation_steps == 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(),
max_grad_norm)
                optimizer.step()
                scheduler.step()
                model.zero_grad()
                global_step += 1
        checkpoint_output_dir = os.path.join(output_dir,
'checkpoint_{}'.format(epoch_idx+1))
        model_to_save = model.module if hasattr(model, "module") else model
        model_to_save.save_pretrained(checkpoint_output_dir)
        tokenizer.save_pretrained(checkpoint_output_dir)
```

```
        torch.save(optimizer.state_dict(), os.path.join(output_dir,
"optimizer.pt"))
        torch.save(scheduler.state_dict(), os.path.join(output_dir,
"scheduler.pt"))
    return global_step, tr_loss / global_step


def evaluate(model, tokenizer):
    dataset, examples, features = load_and_cache_examples(tokenizer,
evaluate=True, output_examples=True)

    eval_sampler = SequentialSampler(dataset)
    eval_dataloader = DataLoader(dataset, sampler=eval_sampler,
batch_size=per_gpu_eval_batch_size)

    print("\nEvaluation:")

    all_results = []
    start_time = timeit.default_timer()

    for batch in tqdm(eval_dataloader, desc="Evaluating"):
        model.eval()
        batch = tuple(t.to(device) for t in batch)

        with torch.no_grad():
            inputs = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2],
            }
            feature_indices = batch[3]
            outputs = model(**inputs, return_dict=False)

            for i, feature_index in enumerate(feature_indices):
                eval_feature = features[feature_index.item()]
                unique_id = int(eval_feature.unique_id)
                output = [to_list(output[i]) for output in outputs]

                start_logits, end_logits = output
                result = SquadResult(unique_id, start_logits, end_logits)

                all_results.append(result)

    evalTime = timeit.default_timer() - start_time
    print("  Evaluation done in total {} secs ({} sec per
example)".format(evalTime, evalTime / len(dataset)))

    output_prediction_file = os.path.join(output_dir, "predictions.json")
    output_nbest_file = os.path.join(output_dir, "nbest_predictions.json")
```

```
    if version_2:
        output_null_log_odds_file = os.path.join(output_dir,
"null_odds.json")
    else:
        output_null_log_odds_file = None

    predictions = compute_predictions_logits(
        examples,
        features,
        all_results,
        n_best_size,
        max_answer_length,
        do_lowercase,
        output_prediction_file,
        output_nbest_file,
        output_null_log_odds_file,
        False,
        version_2,
        null_score_diff_threshold,
        tokenizer,
    )

    if output_null_log_odds_file is not None:
        filename = os.path.join(output_dir, 'null_odds.json')
        null_odds = json.load(open(filename, 'rb'))
    else:
        null_odds = None
    results = squad_evaluate(examples, predictions,
no_answer_probs=null_odds,
no_answer_probability_threshold=null_score_diff_threshold)
    return results


def load_and_cache_examples(tokenizer, evaluate=False,
output_examples=False):
    input_dir = os.path.join('/content/drive/MyDrive/HW_5/Dataset/',
dataset_name)
    cached_features_file = os.path.join(
        input_dir,
        "cached_{}".format("test" if evaluate else "train"),
    )

    if os.path.exists(cached_features_file):
        print("Loading features from cached file
{}".format(cached_features_file))
        features_and_dataset = torch.load(cached_features_file)
        features, dataset, examples = (
            features_and_dataset["features"],
```

```
            features_and_dataset["dataset"],
            features_and_dataset["examples"],
        )
    else:
        print("Creating features from dataset file at
{}".format(input_dir))
        processor = SquadV2Processor() if version_2 else SquadV1Processor()
        if evaluate:
            examples = processor.get_dev_examples(input_dir,
filename='Test.json')
        else:
            examples = processor.get_train_examples(input_dir,
filename='Train.json')
        features, dataset = squad_convert_examples_to_features(
            examples=examples,
            tokenizer=tokenizer,
            max_seq_length=max_seq_length,
            doc_stride=doc_stride,
            max_query_length=max_query_length,
            is_training=not evaluate,
            return_dataset="pt",
            threads=threads,
        )
        print("Saving features into cached file
{}".format(cached_features_file))
        torch.save({"features": features, "dataset": dataset, "examples":
examples}, cached_features_file)

    if output_examples:
        return dataset, examples, features
    return dataset


def main():
    set_seed()

    global_step = ""

    model_file = os.path.join(output_dir,'pytorch_model.bin')
    do_train = not os.path.exists(model_file)
    if do_train:
        checkpoints_dir = filter(lambda x:x.startswith('checkpoint_'),
os.listdir(output_dir))
        checkpoint = max(map(lambda x:int(x[x.find('_')+1:]),
checkpoints_dir), default=0)
        if checkpoint == 0:
            config = AutoConfig.from_pretrained(model_name_or_path)
            tokenizer = AutoTokenizer.from_pretrained(model_name_or_path,
do_lower_case=do_lowercase, use_fast=False)
```

```
            train_dataset = load_and_cache_examples(tokenizer,
evaluate=False, output_examples=False)
            model = AutoModel.from_pretrained(model_name_or_path,
config=config)
        else:
            checkpoint_output_dir = os.path.join(output_dir,
'checkpoint_{}'.format(checkpoint))
            config = AutoConfig.from_pretrained(checkpoint_output_dir)
            tokenizer =
AutoTokenizer.from_pretrained(checkpoint_output_dir,
do_lower_case=do_lowercase, use_fast=False)
            train_dataset = load_and_cache_examples(tokenizer,
evaluate=False, output_examples=False)
            model = AutoModel.from_pretrained(checkpoint_output_dir,
config=config)
        model.to(device)
        global_step, tr_loss = train(train_dataset, model, tokenizer,
checkpoint)
        print(" global_step = {}, average loss = {}".format(global_step,
tr_loss))
        print("Saving model checkpoint to {}".format(output_dir))
        model_to_save = model.module if hasattr(model, "module") else model
        model_to_save.save_pretrained(output_dir)
        tokenizer.save_pretrained(output_dir)

    result = {}
    print("Evaluate the last checkpoint")
    config = AutoConfig.from_pretrained(output_dir)
    tokenizer = AutoTokenizer.from_pretrained(output_dir,
do_lower_case=do_lowercase, use_fast=False)
    model = AutoModel.from_pretrained(output_dir, config=config)
    model.to(device)
    eval_result = evaluate(model, tokenizer)
    result = dict((k, v) for k, v in eval_result.items())

    print(result)
    print("\nResult:\n\texact_match: {}\n\tf1: {}".format(result['exact'],
result['f1']))
    return result


if __name__ == "__main__":
    main()
```

۱-۳. ارزیابی و پس‌پردازش (Postprocessing)

پس‌پردازش توکن مجدد روی داده‌ها انجام می‌گردد برای پیش‌بینی جمله بعدی از پیش آموزش دیده شده انجام می‌گردد.

نتایج برای ارزیابی روی دو مدل گفته شده در صورت سوال پیاده شده است که نتایج به صورت زیر است:

نتایج مدل AlBERT:

```
Evaluating:   0%|          | 0/1021 [00:00<?, ?it/s]

  Evaluation done in total 213.4106203199999 secs (0.026137246824249834 sec per example)
{'exact': 50.44988752811797, 'f1': 64.65321757161851, 'total': 8002, 'HasAns_exact': 66.22864651773982,

Result:
        exact_match: 50.44988752811797
        f1: 64.65321757161851
```

نتایج مدل ParBERT:

```
Evaluating:   0%|          | 0/1021 [00:00<?, ?it/s]

  Evaluation done in total 219.36565646099996 secs (0.026866583767421917 sec per example)
{'exact': 50.44988752811797, 'f1': 64.65321757161851, 'total': 8002, 'HasAns_exact': 66.22864651773982,

Result:
        exact_match: 50.44988752811797
        f1: 64.65321757161851
```

مشاهده می‌شود که نتایج ازیابی دو مدل یکسان است که باید دقت مدل ParsBERT از AlBERT بهتر باشد.

پاسخ ۲ – استفاده از Vision Transformer برای طبقه‌بندی تصاویر

۲-۱. لود کردن دیتاست و انجام پیش‌پردازش‌های لازم

ابتدا داده‌های cifar10 را لود کرده و سپس به داده‌های تست، ترین و ولید تقسیم، کلاس‌بندی و پیش‌پردازش نرمالایزیشن روی این داده‌ها انجام شد سپس برای تغییر سایز داده‌ها از ۳۲*۳۲ به ۲۲۴*۲۲۴ از Lambda استفاده گردید.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

تغییر سایز از ۳۲*۳۲ به ۲۲۴*۲۲۴:

```
input_shape = (32, 32, 3) #Cifar10 image size
image_size = 256 #size after resizing image
num_classes = 10

def build_model():
    inputs = Input(shape=input_shape)
    x = tf.keras.layers.Lambda(lambda image: tf.image.resize(image,
(image_size, image_size)))(inputs) #Resize image to  size 224x224
```

۲-۲. شبکه کانولوشنی

EfficientB7 این تابع یک مدل طبقه‌بندی تصویر Keras را برمی‌گرداند که به صورت اختیاری با وزن‌هایی که از قبل در ImageNet آموزش داده شده است، بارگذاری شده است.

شبکه کاملا کانولوشنی استفاده شده طبق مقاله از EfficientB7 استفاده گردیده است که ورودی ۳۲*۳۲ را دریافت و مدل به ۲۲۴*۲۲۴ تبدیل و سپس به شبکه EfficientNetB7 به عنوان ورودی داده شده است و سپس مجدد روی خروجی Flatten و batch_normalization و به لایه با خروجی ۱۰ نورون و تابع فعال‌ساز softmax داده شده است.

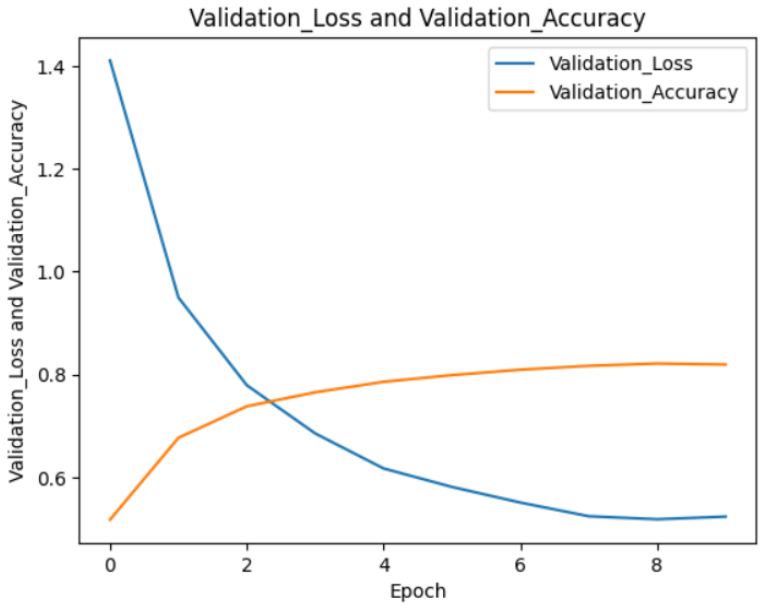
دو شبکه در حالت فاین تیون و بدون فاین تیون انجام شده است و نتایج به صورت زیر حاصل شده است:

با فاین تیون:

```
input_shape = (32, 32, 3) #Cifar10 image size
image_size = 224 #size after resizing image
num_classes = 10
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
BatchNormalization, Flatten, Dropout, Activation, Input
import tensorflow as tf

inputs = Input(shape=input_shape)
x = tf.keras.layers.Lambda(lambda image: tf.image.resize(image,
(image_size, image_size)))(inputs) #Resize image to  size 224x224

model = enet.EfficientNetB7(include_top=False, input_shape=(224,224,3),
pooling='avg', weights='imagenet')

x = model(x)

x = BatchNormalization()(x)
x = Dropout(0.7)(x)

x = Dense(512)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

predictions = Dense(10, activation="softmax")(x)

model_final = Model(inputs = model.input, outputs = predictions)

model_final.summary()
```

```
for layer in model_final.layers:
```

```
    layer.trainable = True
model_final.compile(loss='categorical_crossentropy',
              optimizer=Adam(0.0001),
              metrics=['accuracy'])

mcp_save = ModelCheckpoint('EnetB7_CIFAR10_TL.h5', save_best_only=True,
monitor='val_acc')
reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=2,
verbose=1,)

History = model_final.fit(x_train, y_train,
              batch_size=32,
              epochs=10,
              validation_split=0.1,
              callbacks=[mcp_save, reduce_lr],
              shuffle=True,
              verbose=1)
```



شکل ۲. نمودار Validation _Accuracy و Validation _Loss

۲-۲. شبکه ViT(تبدیل‌کننده تصویر)

Vision Transformer یا ViT مدلی برای طبقه بندی تصاویر است که از معماری ترانسفورماتور مانند روی تکه های تصویر استفاده می کند. یک تصویر به تکه‌های با اندازه ثابت تقسیم می‌شود، سپس هر یک به صورت خطی جاسازی می‌شوند، جاسازی‌های موقعیت اضافه می‌شوند و دنباله بردارها به یک رمزگذار استاندارد تبدیل می‌شوند. برای انجام طبقه‌بندی، از رویکرد استاندارد افزودن یک «نشانه طبقه‌بندی» قابل یادگیری اضافی به دنباله استفاده می‌شود.

ورودی ۳۲*۳۲ را دریافت و مدل به ۲۲۴*۲۲۴ تبدیل و سپس به شبکه ViT به عنوان ورودی داده شده است و سپس مجدد روی خروجی Flatten و batch_normalization و به لایه با خروجی ۱۰ نورون و تابع فعال‌ساز softmax داده شده است.

دو شبکه در حالت فاین تیون و بدون فاین تیون انجام شده است و نتایج به صورت زیر حاصل شده است:

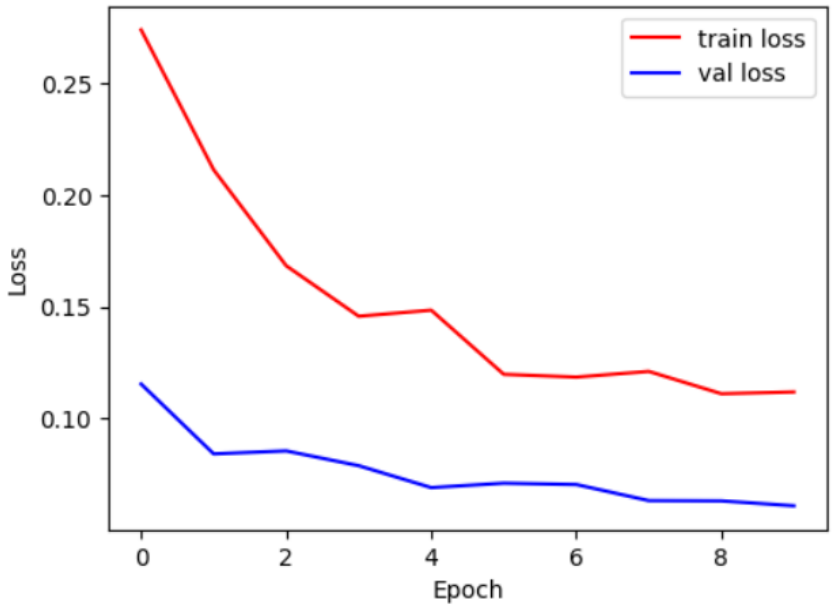
بدون فاین تیون:

```
input_shape = (32, 32, 3) #Cifar10 image size
image_size = 256 #size after resizing image
num_classes = 10

def build_model():
    inputs = Input(shape=input_shape)
    x = tf.keras.layers.Lambda(lambda image: tf.image.resize(image,
(image_size, image_size)))(inputs) #Resize image to  size 224x224
    base_model = vit.vit_b16(image_size=image_size, activation="sigmoid",
pretrained=True,
                             include_top=False, pretrained_top=False)

    base_model.trainable = False #Set false for transfer learning
    x = base_model(x)
    x = Flatten()(x)
    x = BatchNormalization()(x)
    x = Dense(32, activation=tfa.activations.gelu)(x)
    x = BatchNormalization()(x)
    outputs = Dense(num_classes, activation="softmax")(x)

    model_final = Model(inputs=inputs, outputs=outputs)
    return model_final
```

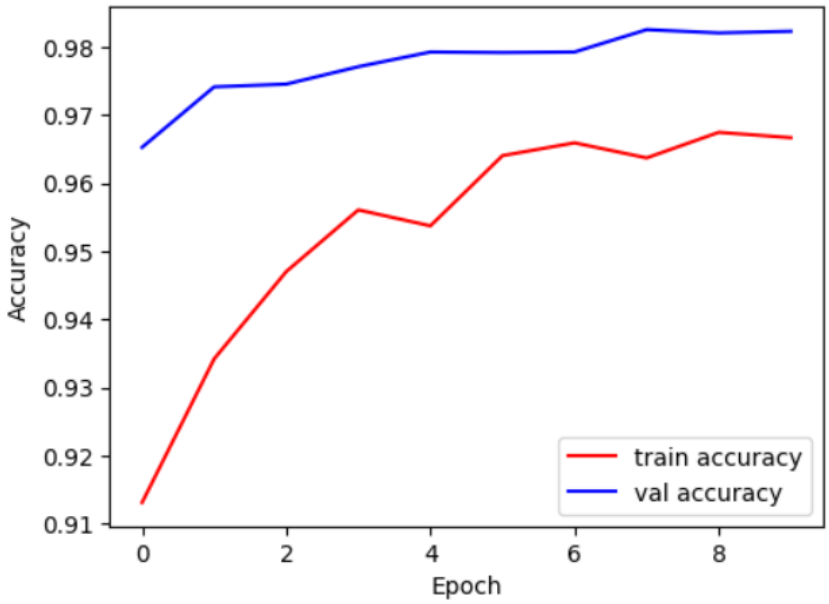
```
model = build_model()
model.compile(optimizer=optimizers.SGD(learning_rate=0.01, momentum=0.9),
loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
print("\n")
model.fit(train_generator,
          steps_per_epoch=200,
          epochs=2,
          validation_data=(X_valid, y_valid),)
gc.collect()
```

با فاین تیون:

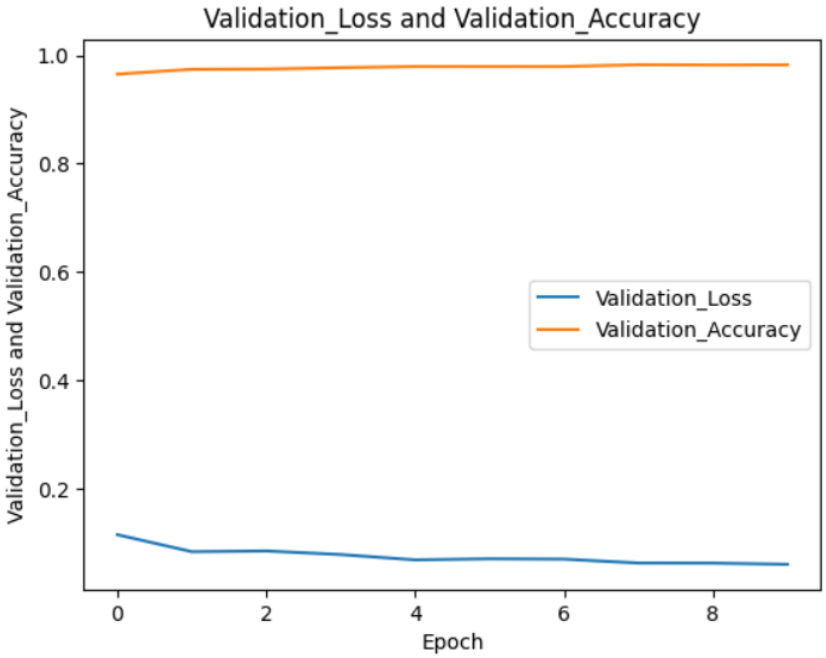
```
plateau = ReduceLROnPlateau(monitor="val_loss", factor=0.7, patience=1,
verbose=1)
earlystopping = EarlyStopping(monitor="val_loss", patience=3, verbose=1)

for layer in model.layers:
    layer.trainable = True

model.compile(optimizer=optimizers.SGD(learning_rate=0.001, momentum=0.9),
loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
print("\n")
history = model.fit(train_generator,
                    steps_per_epoch=30,.
                    epochs=50,
                    validation_data=(X_valid, y_valid),
                    callbacks=[plateau, earlystopping]
                   )
print("\nTest Accuracy: ", accuracy_score(np.argmax(test_label, axis=1),
np.argmax(model.predict(test_data), axis=1)))
```



شکل ۳. نمودار Loss



شکل ۴. نمودار Accuracy



شکل ۵. نمودار Validation _Accuracy و Validation _Loss

با توجه به نتایج مقاله، مشاهده می‌شود که مقدار دقت برای این مدل از نتایج مقاله بهتر حاصل شده است.