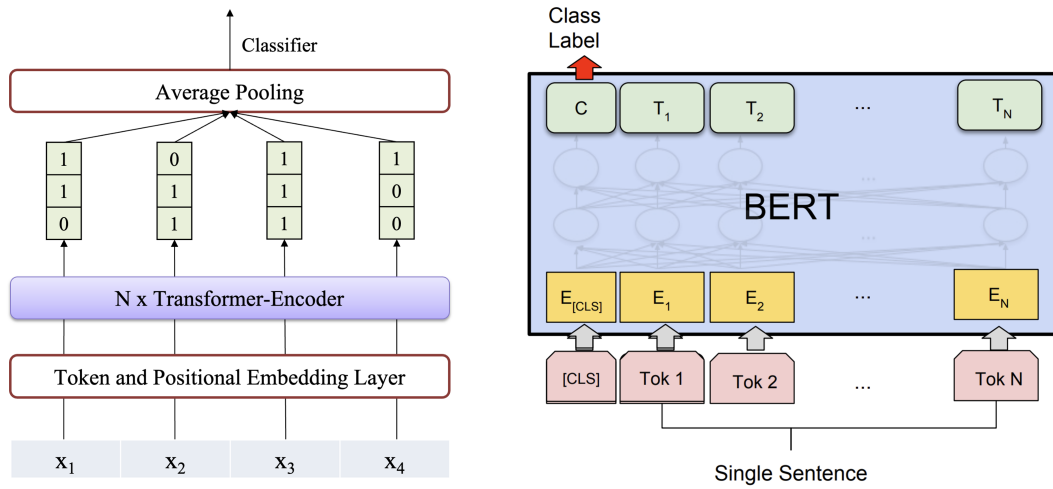


Transformer Applications

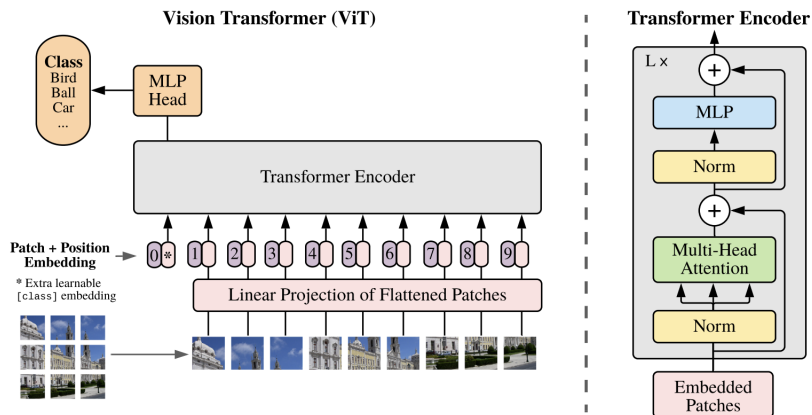
Thai Nguyen-Quoc, Vinh Dinh-Quang

Ngày 21 tháng 12 năm 2024

Phần 1. Transformer



Hình 1: Mô hình Transformer-Encoder và BERT cho bài toán phân loại văn bản.

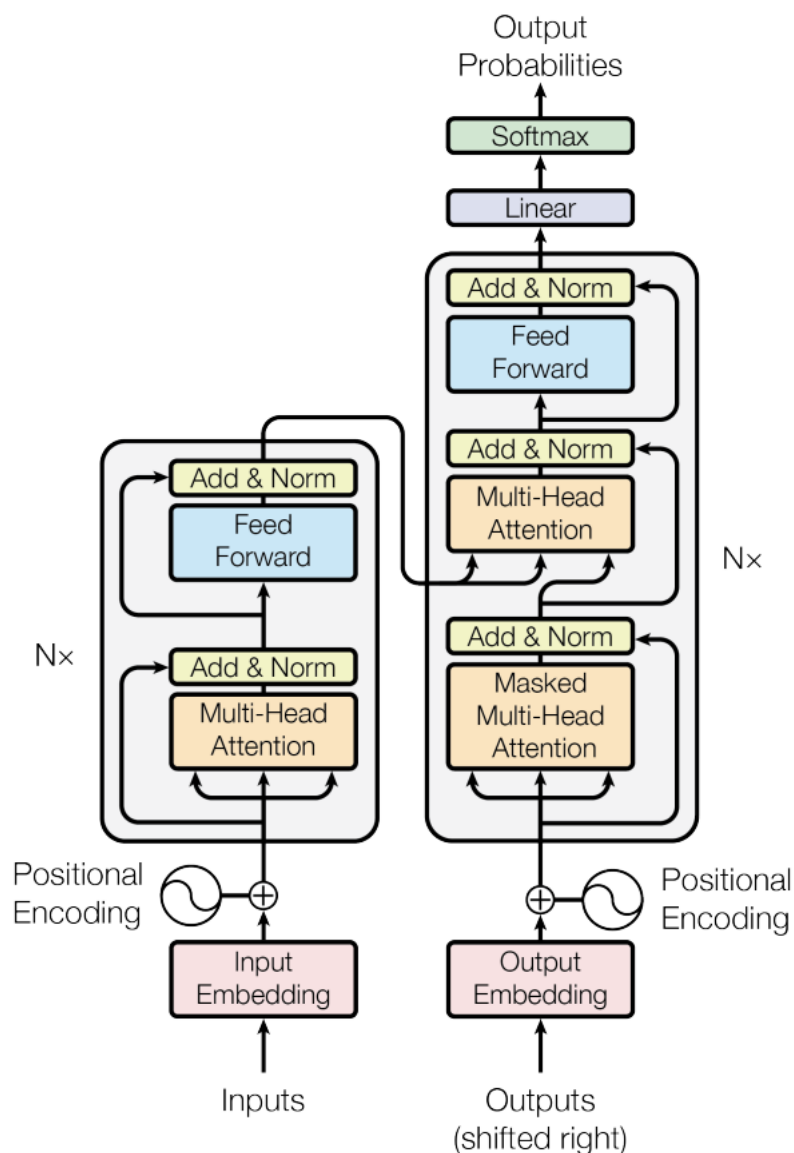


Hình 2: Mô hình Vision Transformer cho bài toán phân loại hình ảnh.

Mô hình Transformer ra đời với kỹ thuật cốt lõi và cơ chế Attention, đã đạt được những kết quả ấn tượng cho các bài toán về dữ liệu văn bản rồi từ đó mở rộng cho các kiểu dữ liệu như hình ảnh và

âm thanh,... Trong phần này, chúng ta sẽ tìm hiểu các thành phần trong mô hình Transformer và ứng dụng cho bài toán phân loại văn bản.

1.1. Kiến trúc Transformer



Hình 3: Mô hình kiến trúc **Transformer**.

Kiến trúc Transformer gồm các thành phần:

- **Input Embedding:** Biểu diễn các token đầu vào (Thường được tách bởi Subword-based Tokenization) thành các dense vector.
- **Positional Encoding:** Biểu diễn vị trí (thứ tự) của các token trong câu. Thường được tính dựa vào hàm sinusoid hoặc được học trong quá trình huấn luyện mô hình.
- **Các khối encoder:** Để mã hoá các tokens đầu vào thành các contextual embedding. Bao gồm: Multi-Head Attention, Add - Normalization, Feed Forward

- Các khối decoder: nhận input là các token lịch sử và trạng thái mã hoá từ encoder, giải mã dự đoán token tiếp theo. Gồm: Masked Multi-Head Attention (dựa vào token lịch sử của decoder), Multi-Head Attention (dựa vào encoder và trạng thái hiện tại tại decoder), Add - Normalization, Feed Forward
- Language Model Head: Projection và Softmax dự đoán token tiếp theo vs xác suất lớn nhất.

1. Input Embedding, Positional Encoding

```

1 class TokenAndPositionEmbedding(nn.Module):
2     def __init__(self, vocab_size, embed_dim, max_length, device='cpu'):
3         super().__init__()
4         self.device = device
5         self.word_emb = nn.Embedding(
6             num_embeddings=vocab_size,
7             embedding_dim=embed_dim
8         )
9         self.pos_emb = nn.Embedding(
10            num_embeddings=max_length,
11            embedding_dim=embed_dim
12        )
13
14    def forward(self, x):
15        N, seq_len = x.size()
16        positions = torch.arange(0, seq_len).expand(N, seq_len).to(self.device)
17        output1 = self.word_emb(x)
18        output2 = self.pos_emb(positions)
19        output = output1 + output2
20        return output

```

2. Encoder

```

1 class TransformerEncoderBlock(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
3         super().__init__()
4         self.attn = nn.MultiheadAttention(
5             embed_dim=embed_dim,
6             num_heads=num_heads,
7             batch_first=True
8         )
9         self.ffn = nn.Sequential(
10            nn.Linear(in_features=embed_dim, out_features=ff_dim, bias=True),
11            nn.ReLU(),
12            nn.Linear(in_features=ff_dim, out_features=embed_dim, bias=True)
13        )
14        self.layer_norm_1 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
15        self.layer_norm_2 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
16        self.dropout_1 = nn.Dropout(p=dropout)
17        self.dropout_2 = nn.Dropout(p=dropout)
18
19    def forward(self, query, key, value):
20        attn_output, _ = self.attn(query, key, value)
21        attn_output = self.dropout_1(attn_output)
22        out_1 = self.layer_norm_1(query + attn_output)
23        ffn_output = self.ffn(out_1)
24        ffn_output = self.dropout_2(ffn_output)
25        out_2 = self.layer_norm_2(out_1 + ffn_output)
26        return out_2
27
28 class TransformerEncoder(nn.Module):
29     def __init__(self,

```

```

30         src_vocab_size, embed_dim, max_length, num_layers, num_heads, ff_dim,
31         dropout=0.1, device='cpu'
32     ):
33         super().__init__()
34         self.embedding = TokenAndPositionEmbedding(
35             src_vocab_size, embed_dim, max_length, device
36         )
37         self.layers = nn.ModuleList(
38             [
39                 TransformerEncoderBlock(
40                     embed_dim, num_heads, ff_dim, dropout
41                 ) for i in range(num_layers)
42             ]
43         )
44
45     def forward(self, x):
46         output = self.embedding(x)
47         for layer in self.layers:
48             output = layer(output, output, output)
49         return output

```

3. Decoder

```

1 class TransformerDecoderBlock(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
3         super().__init__()
4         self.attn = nn.MultiheadAttention(
5             embed_dim=embed_dim,
6             num_heads=num_heads,
7             batch_first=True
8         )
9         self.cross_attn = nn.MultiheadAttention(
10             embed_dim=embed_dim,
11             num_heads=num_heads,
12             batch_first=True
13         )
14         self.ffn = nn.Sequential(
15             nn.Linear(in_features=embed_dim, out_features=ff_dim, bias=True),
16             nn.ReLU(),
17             nn.Linear(in_features=ff_dim, out_features=embed_dim, bias=True)
18         )
19         self.layer_norm_1 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
20         self.layer_norm_2 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
21         self.layer_norm_3 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
22         self.dropout_1 = nn.Dropout(p=dropout)
23         self.dropout_2 = nn.Dropout(p=dropout)
24         self.dropout_3 = nn.Dropout(p=dropout)
25
26     def forward(self, x, enc_output, src_mask, tgt_mask):
27         attn_output, _ = self.attn(x, x, x, attn_mask=tgt_mask)
28         attn_output = self.dropout_1(attn_output)
29         out_1 = self.layer_norm_1(x + attn_output)
30
31         attn_output, _ = self.cross_attn(
32             out_1, enc_output, enc_output, attn_mask=src_mask
33         )
34         attn_output = self.dropout_2(attn_output)
35         out_2 = self.layer_norm_2(out_1 + attn_output)
36
37         ffn_output = self.ffn(out_2)
38         ffn_output = self.dropout_3(ffn_output)

```

```

39         out_3 = self.layernorm_3(out_2 + ffn_output)
40         return out_3
41
42 class TransformerDecoder(nn.Module):
43     def __init__(self,
44                 tgt_vocab_size, embed_dim, max_length, num_layers, num_heads, ff_dim,
45                 dropout=0.1, device='cpu'
46             ):
47         super().__init__()
48         self.embedding = TokenAndPositionEmbedding(
49             tgt_vocab_size, embed_dim, max_length, device
50         )
51         self.layers = nn.ModuleList(
52             [
53                 TransformerDecoderBlock(
54                     embed_dim, num_heads, ff_dim, dropout
55                 ) for i in range(num_layers)
56             ]
57         )
58
59     def forward(self, x, enc_output, src_mask, tgt_mask):
60         output = self.embedding(x)
61         for layer in self.layers:
62             output = layer(output, enc_output, src_mask, tgt_mask)
63         return output

```

4. Transformer

```

1 class Transformer(nn.Module):
2     def __init__(self,
3                 src_vocab_size, tgt_vocab_size,
4                 embed_dim, max_length, num_layers, num_heads, ff_dim,
5                 dropout=0.1, device='cpu'
6             ):
7         super().__init__()
8         self.device = device
9         self.encoder = TransformerEncoder(
10             src_vocab_size, embed_dim, max_length, num_layers, num_heads, ff_dim
11         )
12         self.decoder = TransformerDecoder(
13             tgt_vocab_size, embed_dim, max_length, num_layers, num_heads, ff_dim
14         )
15         self.fc = nn.Linear(embed_dim, tgt_vocab_size)
16
17     def generate_mask(self, src, tgt):
18         src_seq_len = src.shape[1]
19         tgt_seq_len = tgt.shape[1]
20
21         src_mask = torch.zeros(
22             (src_seq_len, src_seq_len),
23             device=self.device).type(torch.bool)
24
25         tgt_mask = (torch.triu(torch.ones(
26             (tgt_seq_len, tgt_seq_len),
27             device=self.device)
28         ) == 1).transpose(0, 1)
29         tgt_mask = tgt_mask.float().masked_fill(
30             tgt_mask == 0, float('-inf')).masked_fill(tgt_mask == 1, float(0.0))
31         return src_mask, tgt_mask
32
33     def forward(self, src, tgt):

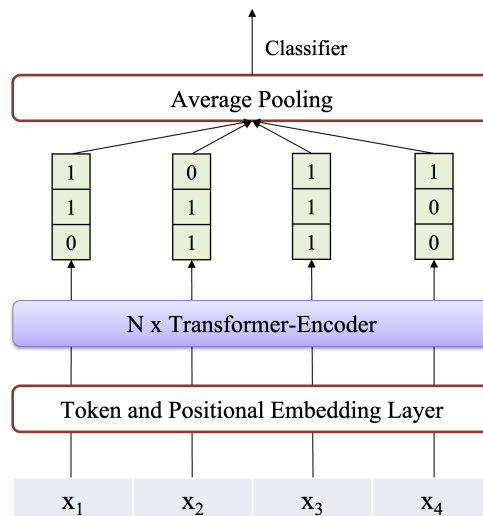
```

```
34         src_mask, tgt_mask = self.generate_mask(src, tgt)
35         enc_output = self.encoder(src)
36         dec_output = self.decoder(tgt, enc_output, src_mask, tgt_mask)
37         output = self.fc(dec_output)
38         return output
```

5. Thử nghiệm

```
1 batch_size = 128
2 src_vocab_size = 1000
3 tgt_vocab_size = 2000
4 embed_dim = 200
5 max_length = 100
6 num_layers = 2
7 num_heads = 4
8 ff_dim = 256
9
10 model = Transformer(
11     src_vocab_size, tgt_vocab_size,
12     embed_dim, max_length, num_layers, num_heads, ff_dim
13 )
14
15 src = torch.randint(
16     high=2,
17     size=(batch_size, max_length),
18     dtype=torch.int64
19 )
20
21 tgt = torch.randint(
22     high=2,
23     size=(batch_size, max_length),
24     dtype=torch.int64
25 )
26
27 prediction = model(src, tgt)
28 prediction.shape # batch_size x max_length x tgt_vocab_size
```

1.2. Text Classification



Hình 4: Phân loại văn bản sử dụng Transformer-Encoder

Ở phần này, chúng ta sử dụng mô hình Transformer-Encoder cho bài toán phân loại văn bản. Kiến trúc mô hình gồm các phần:

- Input Embedding và Positional Encoding
- Các lớp Encoder
- Lớp Average Pooling: lấy trung bình biểu diễn các từ thành biểu diễn cho câu
- Classifier: Linear layer

1. Load Dataset

```
1 !pip install datasets
2
3 from datasets import load_dataset
4
5 ds = load_dataset('thainq107/ntc-scv')
```

2. Preprocessing

Áp dụng hàm tiền xử lý sau trên cột 'sentence' hoặc có thể bỏ qua bước tiền xử lý khi áp dụng trên cột 'preprocessed_sentence'.

```
1 import re
2 import string
3
4 def preprocess_text(text):
5     # remove URLs https://www.
6     url_pattern = re.compile(r'https?://\s+\www\.\s+')
7     text = url_pattern.sub(" ", text)
8
9     # remove HTML Tags: <>
10    html_pattern = re.compile(r'<[<>]+>')
11    text = html_pattern.sub(" ", text)
12
```

```

13 # remove puncs and digits
14 replace_chars = list(string.punctuation + string.digits)
15 for char in replace_chars:
16     text = text.replace(char, " ")
17
18 # remove emoji
19 emoji_pattern = re.compile("[
20     u"\U0001F600-\U0001F64F" # emoticons
21     u"\U0001F300-\U0001F5FF" # symbols & pictographs
22     u"\U0001F680-\U0001F6FF" # transport & map symbols
23     u"\U0001F1E0-\U0001F1FF" # flags (iOS)
24     u"\U0001F1F2-\U0001F1F4" # Macau flag
25     u"\U0001F1E6-\U0001F1FF" # flags
26     u"\U0001F600-\U0001F64F"
27     u"\U00002702-\U000027B0"
28     u"\U000024C2-\U0001F251"
29     u"\U0001f926-\U0001f937"
30     u"\U0001F1F2"
31     u"\U0001F1F4"
32     u"\U0001F620"
33     u"\u200d"
34     u"\u2640-\u2642"
35     "]+", flags=re.UNICODE)
36 text = emoji_pattern.sub(r" ", text)
37
38 # normalize whitespace
39 text = " ".join(text.split())
40
41 # lowercasing
42 text = text.lower()
43 return text

```

3. Representation

```

1 # !pip install -q torchtext==0.17.2 (install before import torch)
2
3 def yield_tokens(sentences, tokenizer):
4     for sentence in sentences:
5         yield tokenizer(sentence)
6
7
8 # word-based tokenizer
9 from torchtext.data import get_tokenizer
10 tokenizer = get_tokenizer("basic_english")
11
12 # build vocabulary
13 from torchtext.vocab import build_vocab_from_iterator
14
15 vocab_size = 10000
16 vocabulary = build_vocab_from_iterator(
17     yield_tokens(ds['train']['preprocessed_sentence'], tokenizer),
18     max_tokens=vocab_size,
19     specials=["<pad>", "<unk>"]
20 )
21 vocabulary.set_default_index(vocabulary["<unk>"])
22
23 # convert torchtext dataset
24 from torchtext.data.functional import to_map_style_dataset
25
26 def prepare_dataset(df):
27     # create iterator for dataset: (sentence, label)

```



```

28     for row in df:
29         sentence = row['preprocessed_sentence']
30         encoded_sentence = vocabulary(tokenizer(sentence))
31         label = row['label']
32         yield encoded_sentence, label
33
34 train_dataset = prepare_dataset(ds['train'])
35 train_dataset = to_map_style_dataset(train_dataset)
36
37 valid_dataset = prepare_dataset(ds['valid'])
38 valid_dataset = to_map_style_dataset(valid_dataset)
39
40 test_dataset = prepare_dataset(ds['test'])
41 test_dataset = to_map_style_dataset(test_dataset)

```

4. Dataloader

```

1 import torch
2
3 seq_length = 100
4
5 def collate_batch(batch):
6     # create inputs, offsets, labels for batch
7     sentences, labels = list(zip(*batch))
8     encoded_sentences = [
9         sentence+([0]*(seq_length-len(sentence))) if len(sentence) < seq_length else
10        sentence[:seq_length]
11        for sentence in sentences
12    ]
13
14    encoded_sentences = torch.tensor(encoded_sentences, dtype=torch.int64)
15    labels = torch.tensor(labels)
16
17    return encoded_sentences, labels
18
19 from torch.utils.data import DataLoader
20
21 batch_size = 128
22
23 train_dataloader = DataLoader(
24     train_dataset,
25     batch_size=batch_size,
26     shuffle=True,
27     collate_fn=collate_batch
28 )
29
30 valid_dataloader = DataLoader(
31     valid_dataset,
32     batch_size=batch_size,
33     shuffle=False,
34     collate_fn=collate_batch
35 )
36
37 test_dataloader = DataLoader(
38     test_dataset,
39     batch_size=batch_size,
40     shuffle=False,
41     collate_fn=collate_batch
42 )

```

5. Trainer

```

1 # train epoch

```

```

2 import time
3
4 def train_epoch(model, optimizer, criterion, train_dataloader, device, epoch=0,
  log_interval=50):
5     model.train()
6     total_acc, total_count = 0, 0
7     losses = []
8     start_time = time.time()
9
10    for idx, (inputs, labels) in enumerate(train_dataloader):
11        inputs = inputs.to(device)
12        labels = labels.to(device)
13
14        optimizer.zero_grad()
15
16        predictions = model(inputs)
17
18        # compute loss
19        loss = criterion(predictions, labels)
20        losses.append(loss.item())
21
22        # backward
23        loss.backward()
24        optimizer.step()
25        total_acc += (predictions.argmax(1) == labels).sum().item()
26        total_count += labels.size(0)
27        if idx % log_interval == 0 and idx > 0:
28            elapsed = time.time() - start_time
29            print(
30                "| epoch {:3d} | {:5d}/{:5d} batches | "
31                "| accuracy {:.8.3f} |".format(
32                    epoch, idx, len(train_dataloader), total_acc / total_count
33                )
34            )
35            total_acc, total_count = 0, 0
36            start_time = time.time()
37
38    epoch_acc = total_acc / total_count
39    epoch_loss = sum(losses) / len(losses)
40    return epoch_acc, epoch_loss
41
42 # evaluate
43 def evaluate_epoch(model, criterion, valid_dataloader, device):
44     model.eval()
45     total_acc, total_count = 0, 0
46     losses = []
47
48     with torch.no_grad():
49         for idx, (inputs, labels) in enumerate(valid_dataloader):
50             inputs = inputs.to(device)
51             labels = labels.to(device)
52
53             predictions = model(inputs)
54
55             loss = criterion(predictions, labels)
56             losses.append(loss.item())
57
58             total_acc += (predictions.argmax(1) == labels).sum().item()
59             total_count += labels.size(0)
60

```

```

61     epoch_acc = total_acc / total_count
62     epoch_loss = sum(losses) / len(losses)
63     return epoch_acc, epoch_loss
64
65
66 # train
67 def train(model, model_name, save_model, optimizer, criterion, train_dataloader,
68           valid_dataloader, num_epochs, device):
69     train_accs, train_losses = [], []
70     eval_accs, eval_losses = [], []
71     best_loss_eval = 100
72     times = []
73     for epoch in range(1, num_epochs+1):
74         epoch_start_time = time.time()
75         # Training
76         train_acc, train_loss = train_epoch(model, optimizer, criterion,
77         train_dataloader, device, epoch)
78         train_accs.append(train_acc)
79         train_losses.append(train_loss)
80
81         # Evaluation
82         eval_acc, eval_loss = evaluate_epoch(model, criterion, valid_dataloader,
83         device)
84         eval_accs.append(eval_acc)
85         eval_losses.append(eval_loss)
86
87         # Save best model
88         if eval_loss < best_loss_eval:
89             torch.save(model.state_dict(), save_model + f'/{model_name}.pt')
90
91         times.append(time.time() - epoch_start_time)
92         # Print loss, acc end epoch
93         print("-" * 59)
94         print(
95             "| End of epoch {:3d} | Time: {:.5.2f}s | Train Accuracy {:.8.3f} | Train
96             Loss {:.8.3f} "
97             "| Valid Accuracy {:.8.3f} | Valid Loss {:.8.3f} ".format(
98                 epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc
99                 , eval_loss
100             )
101         )
102         print("-" * 59)
103
104         # Load best model
105         model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
106         model.eval()
107         metrics = {
108             'train_accuracy': train_accs,
109             'train_loss': train_losses,
110             'valid_accuracy': eval_accs,
111             'valid_loss': eval_losses,
112             'time': times
113         }
114     }
115     return model, metrics

```

```

111 # report
112 import matplotlib.pyplot as plt
113
114 def plot_result(num_epochs, train_accs, eval_accs, train_losses, eval_losses):
115     epochs = list(range(num_epochs))

```

```

116 fig, axs = plt.subplots(nrows = 1, ncols =2 , figsize = (12,6))
117 axs[0].plot(epochs, train_accs, label = "Training")
118 axs[0].plot(epochs, eval_accs, label = "Evaluation")
119 axs[1].plot(epochs, train_losses, label = "Training")
120 axs[1].plot(epochs, eval_losses, label = "Evaluation")
121 axs[0].set_xlabel("Epochs")
122 axs[1].set_xlabel("Epochs")
123 axs[0].set_ylabel("Accuracy")
124 axs[1].set_ylabel("Loss")
125 plt.legend()

```

6. Modeling

```

1 class TransformerEncoderCls(nn.Module):
2     def __init__(self,
3                   vocab_size, max_length, num_layers, embed_dim, num_heads, ff_dim,
4                   dropout=0.1, device='cpu'
5                   ):
6         super().__init__()
7         self.encoder = TransformerEncoder(
8             vocab_size, embed_dim, max_length, num_layers, num_heads, ff_dim, dropout,
9             device
10            )
11         self.pooling = nn.AvgPool1d(kernel_size=max_length)
12         self.fc1 = nn.Linear(in_features=embed_dim, out_features=20)
13         self.fc2 = nn.Linear(in_features=20, out_features=2)
14         self.dropout = nn.Dropout(p=dropout)
15         self.relu = nn.ReLU()
16     def forward(self, x):
17         output = self.encoder(x)
18         output = self.pooling(output.permute(0,2,1)).squeeze()
19         output = self.dropout(output)
20         output = self.fc1(output)
21         output = self.dropout(output)
22         output = self.fc2(output)
23         return output

```

7. Training

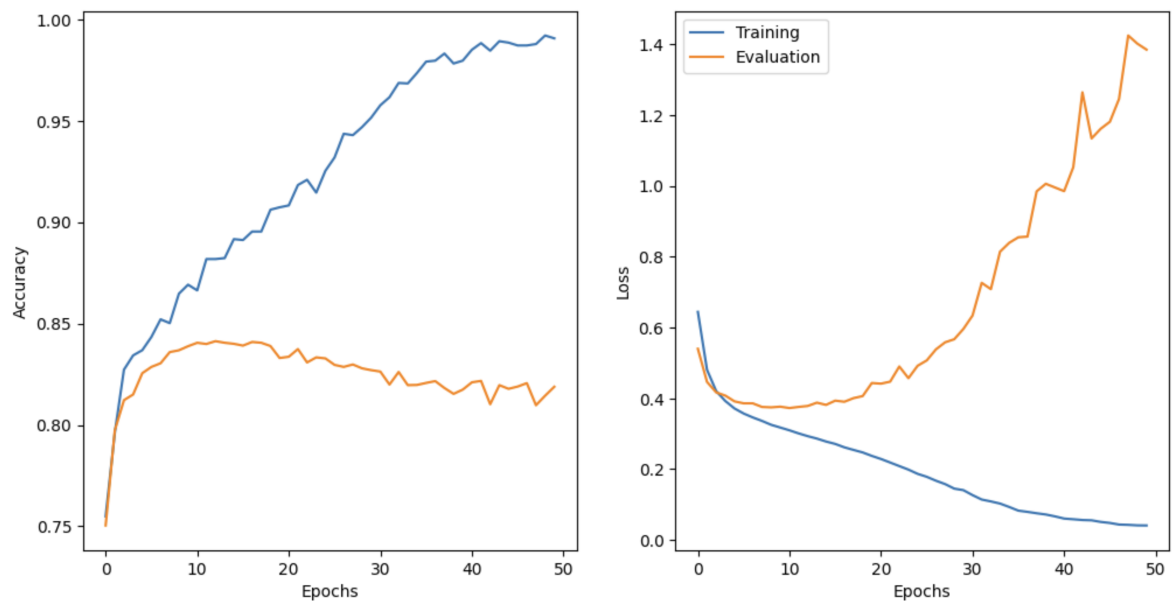
```

1 import torch.optim as optim
2
3 vocab_size = 10000
4 max_length = 100
5 embed_dim = 200
6 num_layers = 2
7 num_heads = 4
8 ff_dim = 128
9 dropout=0.1
10
11 model = TransformerEncoderCls(
12     vocab_size, max_length, num_layers, embed_dim, num_heads, ff_dim, dropout
13 )
14
15 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
16
17 model = TransformerEncoderCls(
18     vocab_size, max_length, num_layers, embed_dim, num_heads, ff_dim, dropout, device
19 )
20 model.to(device)
21
22 criterion = torch.nn.CrossEntropyLoss()
23 optimizer = optim.Adam(model.parameters(), lr=0.00005)

```

```
24
25 num_epochs = 50
26 save_model = './model'
27 os.makedirs(save_model, exist_ok = True)
28 model_name = 'model'
29
30 model, metrics = train(
31     model, model_name, save_model, optimizer, criterion, train_dataloader,
32     valid_dataloader, num_epochs, device
33 )
```

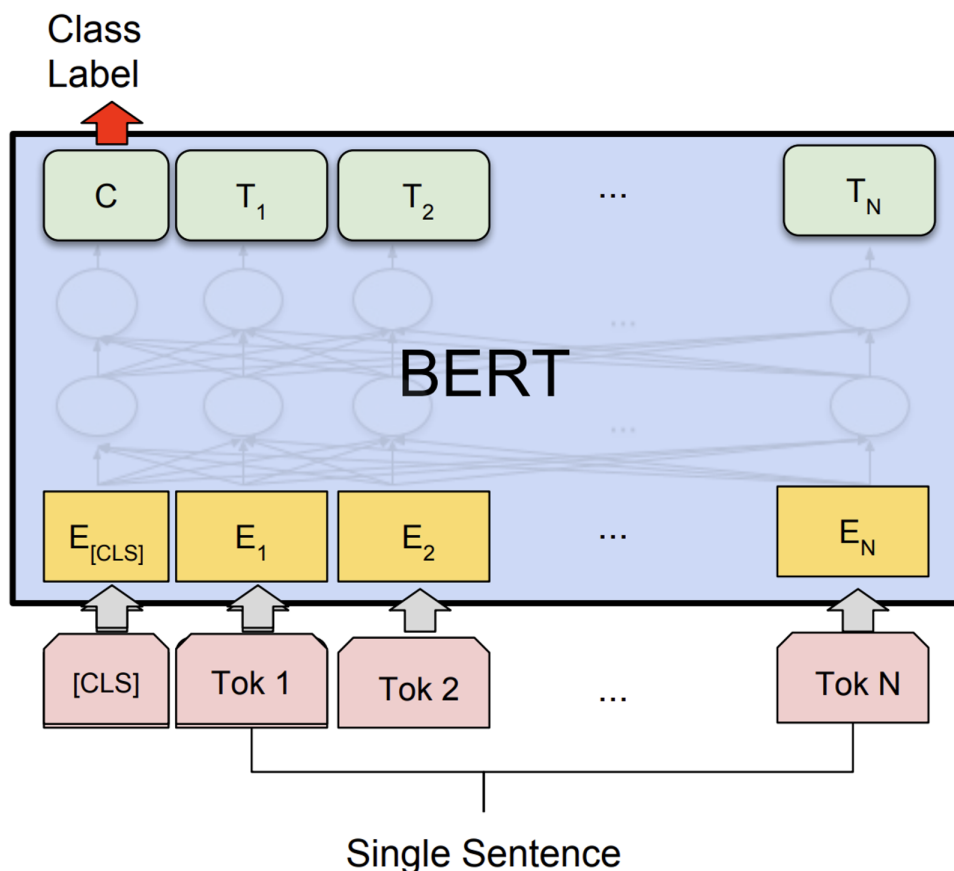
Kết quả training và accuracy trên tập test là 82%



Hình 5: Quá trình huấn luyện bộ dữ liệu NTC-SCV với mô hình Transformer-Encoder.

Phần 2. Text Classification using BERT

Một trong những mô hình pretrained đầu tiên cho dữ liệu văn bản dựa vào kiến trúc mô hình Transformer được ứng dụng cho các downstream task khác nhau đó là BERT. Trong phần này chúng ta sẽ fine tuning BERT cho bài toán phân loại trên bộ dữ liệu NTC-SCV dựa vào thư viện transformers của huggingface.



Hình 6: Fine-Tuning BERT cho bài toán phân loại văn bản.

1. Load Dataset

```
1 # install libs
2 !pip install -q -U transformers datasets accelerate evaluate
3
4 from datasets import load_dataset
5
6 ds = load_dataset('thainq107/ntc-scv')
```

2. Preprocessing

```
1 # tokenization
2 from transformers import AutoTokenizer
3
4 model_name = "distilbert-base-uncased" # bert-base-uncased
5
```

```
6 tokenizer = AutoTokenizer.from_pretrained(
7     model_name,
8     use_fast=True
9 )
10 max_seq_length = 100
11 max_seq_length = min(max_seq_length, tokenizer.model_max_length)
12
13 def preprocess_function(examples):
14     # Tokenize the texts
15
16     result = tokenizer(
17         examples["preprocessed_sentence"],
18         padding="max_length",
19         max_length=max_seq_length,
20         truncation=True
21     )
22     result["label"] = examples['label']
23
24     return result
25
26 # Running the preprocessing pipeline on all the datasets
27 processed_dataset = ds.map(
28     preprocess_function,
29     batched=True,
30     desc="Running tokenizer on dataset",
31 )
```

3. Modeling

```
1 from transformers import AutoConfig, AutoModelForSequenceClassification
2
3 num_labels = 2
4
5 config = AutoConfig.from_pretrained(
6     model_name,
7     num_labels=num_labels,
8     finetuning_task="text-classification"
9 )
10 model = AutoModelForSequenceClassification.from_pretrained(
11     model_name,
12     config=config
13 )
```

4. Metric

```
1 import numpy as np
2 import evaluate
3
4 metric = evaluate.load("accuracy")
5 def compute_metrics(eval_pred):
6     predictions, labels = eval_pred
7     predictions = np.argmax(predictions, axis=1)
8     result = metric.compute(predictions=predictions, references=labels)
9     return result
```

5. Trainer

```
1 from transformers import TrainingArguments, Trainer
2
```

```

3 training_args = TrainingArguments(
4     output_dir="save_model",
5     learning_rate=2e-5,
6     per_device_train_batch_size=128,
7     per_device_eval_batch_size=128,
8     num_train_epochs=10,
9     eval_strategy="epoch",
10    save_strategy="epoch",
11    load_best_model_at_end=True
12 )
13
14 trainer = Trainer(
15     model=model,
16     args=training_args,
17     train_dataset=processed_dataset["train"],
18     eval_dataset=processed_dataset["valid"],
19     compute_metrics=compute_metrics,
20     tokenizer=tokenizer,
21 )
22
23 trainer.train()

```

6. Training

Kết quả training và accuracy trên tập test là 85%

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.457404	0.790200
2	No log	0.432181	0.804500
3	No log	0.419964	0.812100
4	No log	0.395226	0.827100
5	0.431800	0.398116	0.831200
6	0.431800	0.388693	0.834300
7	0.431800	0.403755	0.831600
8	0.431800	0.408512	0.833300
9	0.316800	0.409772	0.835000
10	0.316800	0.410361	0.836200

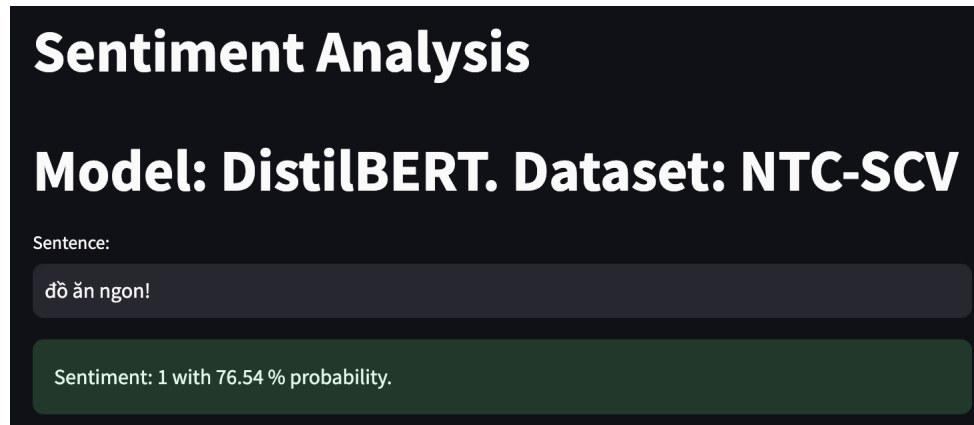
Hình 7: Quá trình huấn luyện bộ dữ liệu NTC-SCV dựa vào fine tuning BERT.

7. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)

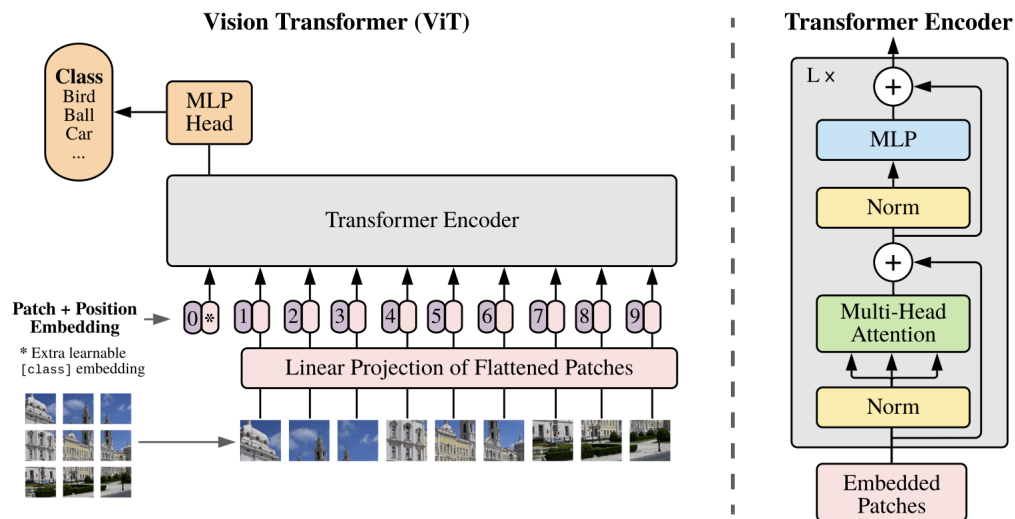
3. Giao diện:



Hình 8: Giao diện ứng dụng.

Phần 3. Vision Transformer

Trong phần này, chúng ta sẽ huấn luyện mô hình Vision Transformer cho bài toán phân loại hình ảnh.



Hình 9: Mô hình Vision Transformer cho bài toán phân loại hình ảnh.

1. Load Dataset

```
1 import torch
2 import torchvision.transforms as transforms
3 from torch.utils.data import DataLoader, random_split
4 import torch.optim as optim
5 from torchvision.datasets import ImageFolder
6 from torch import nn
7 import math
8 import os

9
10 # download
11 !wget https://www.kaggle.com/competitions/flower-photos-2021/data
12 !unzip ./flower_photos.zip
13
14 # load data
15 data_patch = "./flower_photos"
16 dataset = ImageFolder(root=data_patch)
17 num_samples = len(dataset)
18 classes = dataset.classes
19 num_classes = len(dataset.classes)
20
21 # split
22 TRAIN_RATIO, VALID_RATIO = 0.8, 0.1
23 n_train_examples = int(num_samples * TRAIN_RATIO)
24 n_valid_examples = int(num_samples * VALID_RATIO)
25 n_test_examples = num_samples - n_train_examples - n_valid_examples
26 train_dataset, valid_dataset, test_dataset = random_split(
27     dataset,
28     [n_train_examples, n_valid_examples, n_test_examples]
29 )
```

2. Preprocessing

```
1 # resize + convert to tensor
2 IMG_SIZE = 224
3
4 train_transforms = transforms.Compose([
5     transforms.Resize((IMG_SIZE, IMG_SIZE)),
6     transforms.RandomHorizontalFlip(),
7     transforms.RandomRotation(0.2),
8     transforms.ToTensor(),
9     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
10 ])
11
12 test_transforms = transforms.Compose([
13     transforms.Resize((IMG_SIZE, IMG_SIZE)),
14     transforms.ToTensor(),
15     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
16 ])
17
18 # apply
19 train_dataset.dataset.transform = train_transforms
20 valid_dataset.dataset.transform = test_transforms
21 test_dataset.dataset.transform = test_transforms
```

3. Dataloader

```
1 BATCH_SIZE = 512
2
3 train_loader = DataLoader(
4     train_dataset,
5     shuffle=True,
6     batch_size=BATCH_SIZE
7 )
8
9 val_loader = DataLoader(
10     valid_dataset,
11     batch_size=BATCH_SIZE
12 )
13
14 test_loader = DataLoader(
15     test_dataset,
16     batch_size=BATCH_SIZE
17 )
```

4. Training from Scratch

4.1. Modeling

```
1 class TransformerEncoder(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
3         super().__init__()
4         self.attn = nn.MultiheadAttention(
5             embed_dim=embed_dim,
6             num_heads=num_heads,
7             batch_first=True
8         )
9         self.ffn = nn.Sequential(
10             nn.Linear(in_features=embed_dim, out_features=ff_dim, bias=True),
11             nn.ReLU(),
12             nn.Linear(in_features=ff_dim, out_features=embed_dim, bias=True)
```

```

13         )
14         self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
15         self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
16         self.dropout_1 = nn.Dropout(p=dropout)
17         self.dropout_2 = nn.Dropout(p=dropout)
18
19     def forward(self, query, key, value):
20         attn_output, _ = self.attn(query, key, value)
21         attn_output = self.dropout_1(attn_output)
22         out_1 = self.layernorm_1(query + attn_output)
23         ffn_output = self.ffn(out_1)
24         ffn_output = self.dropout_2(ffn_output)
25         out_2 = self.layernorm_2(out_1 + ffn_output)
26         return out_2


1 class PatchPositionEmbedding(nn.Module):
2     def __init__(self, image_size=224, embed_dim=512, patch_size=16, device='cpu'):
3         super().__init__()
4         self.conv1 = nn.Conv2d(in_channels=3, out_channels=embed_dim, kernel_size=
patch_size, stride=patch_size, bias=False)
5         scale = embed_dim ** -0.5
6         self.positional_embedding = nn.Parameter(scale * torch.randn((image_size //
patch_size) ** 2, embed_dim))
7         self.device = device
8
9     def forward(self, x):
10         x = self.conv1(x) # shape = [*, width, grid, grid]
11         x = x.reshape(x.shape[0], x.shape[1], -1) # shape = [*, width, grid ** 2]
12         x = x.permute(0, 2, 1) # shape = [*, grid ** 2, width]
13
14         x = x + self.positional_embedding.to(self.device)
15         return x


1 class VisionTransformerCls(nn.Module):
2     def __init__(self,
3         image_size, embed_dim, num_heads, ff_dim,
4         dropout=0.1, device='cpu', num_classes = 10, patch_size=16
5     ):
6         super().__init__()
7         self.embd_layer = PatchPositionEmbedding(
8             image_size=image_size, embed_dim=embed_dim, patch_size=patch_size, device=
device
9         )
10         self.transformer_layer = TransformerEncoder(
11             embed_dim, num_heads, ff_dim, dropout
12         )
13         # self.pooling = nn.AvgPool1d(kernel_size=max_length)
14         self.fc1 = nn.Linear(in_features=embed_dim, out_features=20)
15         self.fc2 = nn.Linear(in_features=20, out_features=num_classes)
16         self.dropout = nn.Dropout(p=dropout)
17         self.relu = nn.ReLU()
18     def forward(self, x):
19         output = self.embd_layer(x)
20         output = self.transformer_layer(output, output, output)
21         output = output[:, 0, :]
22         output = self.dropout(output)
23         output = self.fc1(output)
24         output = self.dropout(output)
25         output = self.fc2(output)
26         return output

```

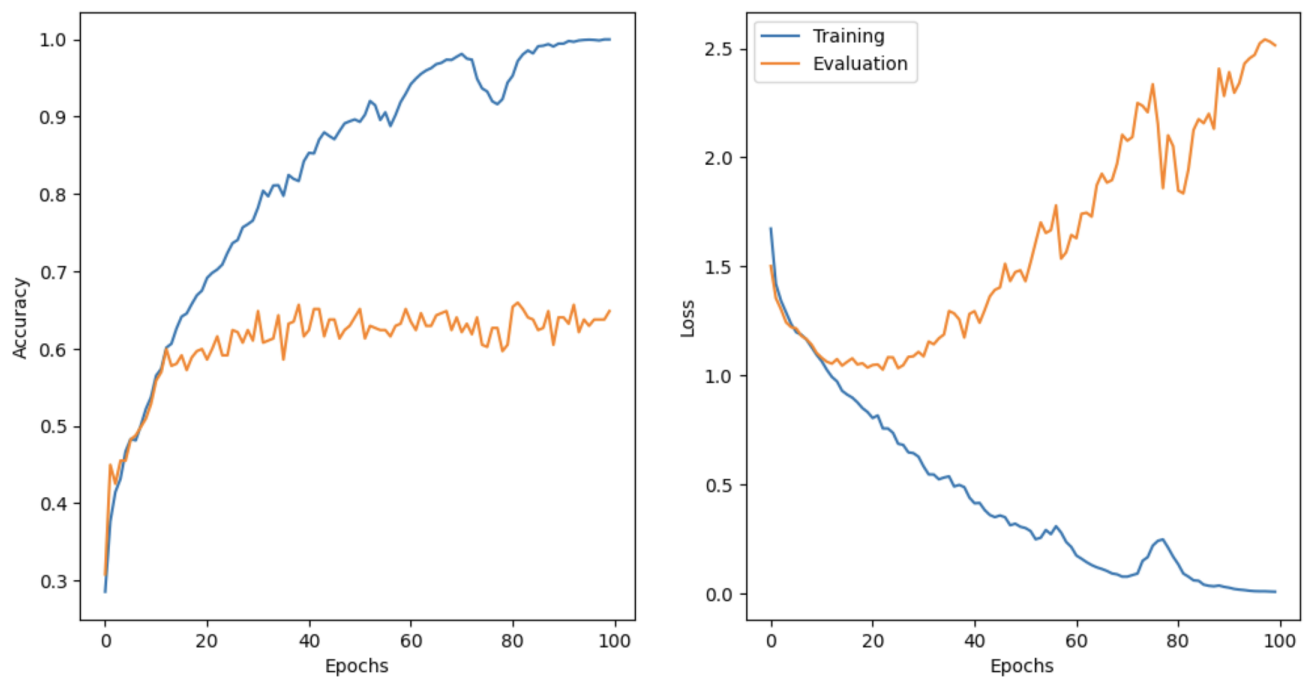
4.2. Training

```

1 image_size=224
2 embed_dim = 512
3 num_heads = 4
4 ff_dim = 128
5 dropout=0.1
6
7 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
8 model = VisionTransformerCls(
9     image_size=224, embed_dim=512, num_heads=num_heads, ff_dim=ff_dim, dropout=dropout
10    , num_classes=num_classes, device=device
11 )
12 model.to(device)
13
14 criterion = torch.nn.CrossEntropyLoss()
15 optimizer = optim.Adam(model.parameters(), lr=0.0005)
16
17 num_epochs = 100
18 save_model = './vit_flowers'
19 os.makedirs(save_model, exist_ok = True)
20 model_name = 'vit_flowers'
21
22 model, metrics = train(
23     model, model_name, save_model, optimizer, criterion, train_loader, val_loader,
24     num_epochs, device
25 )

```

Kết quả training và accuracy trên tập test là 60%



Hình 10: Quá trình huấn luyện bộ dữ liệu Flower với mô hình Vision Transformer.

5. Fine Tuning

5.1. Modeling

```

1 from transformers import ViTForImageClassification
2
3 id2label = {id:label for id, label in enumerate(classes)}
4 label2id = {label:id for id,label in id2label.items()}
5
6 model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224-in21k',
7                                                  num_labels=num_classes,
8                                                  id2label=id2label,
9                                                  label2id=label2id)
10 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
11 model.to(device)

```

5.2. Metric

```

1 import evaluate
2 import numpy as np
3
4 metric = evaluate.load("accuracy")
5
6 def compute_metrics(eval_pred):
7     predictions, labels = eval_pred
8     predictions = np.argmax(predictions, axis=1)
9     return metric.compute(predictions=predictions, references=labels)

```

5.3. Trainer

```

1 import torch
2 from transformers import ViTImageProcessor
3 from transformers import TrainingArguments, Trainer
4
5 feature_extractor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224-in21k")
6
7 metric_name = "accuracy"
8
9 args = TrainingArguments(
10     f"vit_flowers",
11     save_strategy="epoch",
12     evaluation_strategy="epoch",
13     learning_rate=2e-5,
14     per_device_train_batch_size=32,
15     per_device_eval_batch_size=32,
16     num_train_epochs=10,
17     weight_decay=0.01,
18     load_best_model_at_end=True,
19     metric_for_best_model=metric_name,
20     logging_dir='logs',
21     remove_unused_columns=False,
22 )
23
24 def collate_fn(examples):
25     # example => Tuple(image, label)
26     pixel_values = torch.stack([example[0] for example in examples])
27     labels = torch.tensor([example[1] for example in examples])
28     return {"pixel_values": pixel_values, "labels": labels}
29

```

```
30 trainer = Trainer(  
31     model,  
32     args,  
33     train_dataset=train_dataset,  
34     eval_dataset=valid_dataset,  
35     data_collator=collate_fn,  
36     compute_metrics=compute_metrics,  
37     tokenizer=feature_extractor,  
38 )
```

5.4. Training

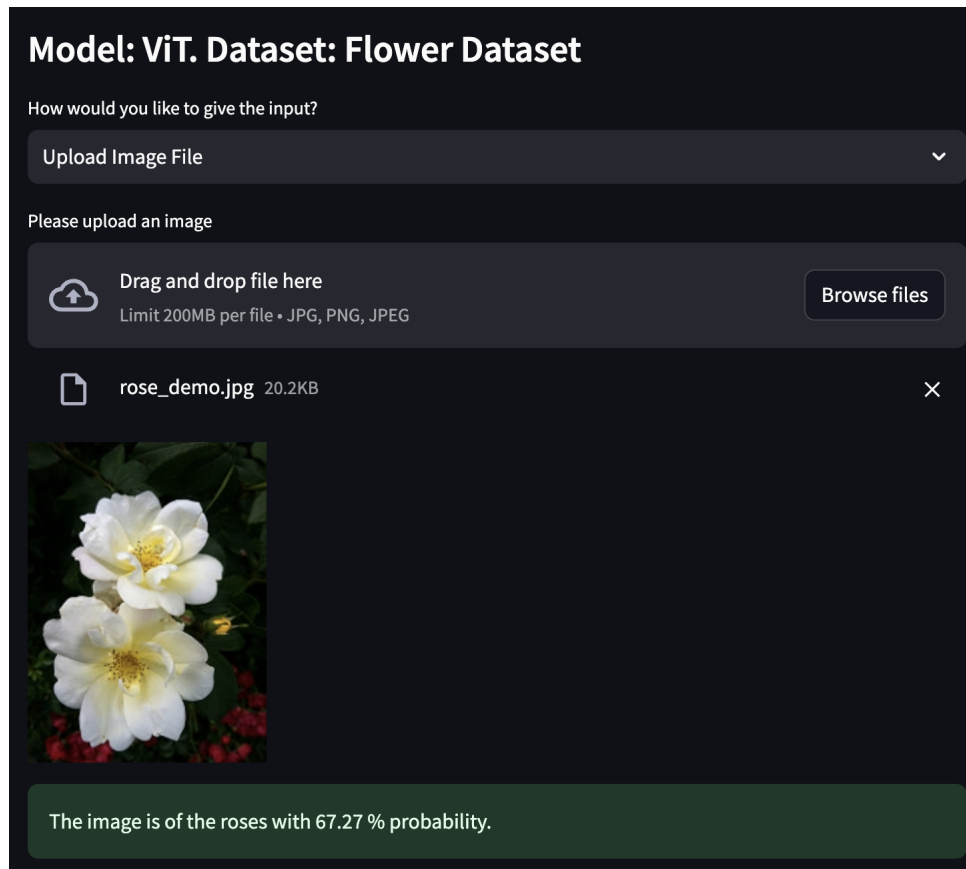
```
1 trainer.train()  
2 outputs = trainer.predict(test_dataset)  
3 outputs.metrics
```

Kết quả training và accuracy trên tập test là 97%

5.5. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)
3. Giao diện:



Hình 11: Giao diện ứng dụng.

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Lớp Positional Encoding trong kiến trúc mô hình Transformer dùng để làm gì?

- a) Biểu diễn vị trí của các tokens
- b) Tính Attention
- c) Dự đoán token tiếp theo
- d) Tính Loss

Câu hỏi 2 Layer nào sau đây không có trong Transformer-Encoder?

- a) Masked Multi-Head Attention
- b) Multi-Head Attention
- c) Layer Normalization
- d) Feed Forward

Câu hỏi 3 Layer nào sau đây không có trong Transformer-Decoder?

- a) Masked Multi-Head Attention
- b) Multi-Head Attention
- c) CNN
- d) Feed Forward

Câu hỏi 4 Masked Multi-Head Attention được sử dụng để làm gì?

- a) Mask những token lịch sử
- b) Mask vị trí các token lịch sử
- c) Mask những token trong tương lai
- d) Mask cả những token lịch sử và token trong tương lai

Câu hỏi 5 Phát biểu nào sau đây là đúng về BERT?

- a) Bao gồm các khối Transformer-Encoder
- b) Bao gồm các khối Transformer-Decoder
- c) Cả 2 đáp án a và b đều đúng
- d) Cả 2 đáp án a và b đều sai

Câu hỏi 6 Objective Function của BERT là?

- a) Masked Language Model
- b) Next Sentence Prediction
- c) Cả a và b đều đúng
- d) Cả a và b đều sai

Câu hỏi 7 Thành phần nào không có trong các giá trị Input của BERT

- a) Token Embeddings
- b) Segment Embeddings

- c) Position Embeddings
 d) Language Model Head

Câu hỏi 8 Số lượng tham số của mô hình BERT-base và BERT-large là?

- a) 340M và 110M
 b) 110M và 110M
 c) 340M và 340M
 d) 110M và 340M

Câu hỏi 9 Các Patch trong Vision Transformer được xác định như thế nào?

- a) Flatten ảnh đầu vào
 b) Trung bình các giá trị điểm ảnh theo hàng
 c) Trung bình các giá trị điểm ảnh theo cột
 d) Chia ảnh đầu vào thành các Patch với kích thước các mảng cố định

Câu hỏi 10 Bộ dữ liệu nào sau đây được sử dụng trong phần thực nghiệm so sánh kết quả Vision Transformer với các phương pháp huấn luyện từ đầu và sử dụng pretrained?

- a) CIFAR10
 b) Flower
 c) CIFAR100
 d) MNIST

Phần 5. Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải về tại đây:

- [Transformer-For-Text](#)
- [Transformer-For-Image](#)

2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. **Rubric:**

Phần	Kiến Thức	Đánh Giá
1	- Hiểu rõ kiến trúc mô hình Transformer - Hiểu rõ mô hình BERT - Áp dụng Transformer-Encoder và BERT cho bài toán phân loại văn bản	- Xây dựng mô hình phân loại văn bản sử dụng Transformer-Encoder và BERT
2.	- Hiểu rõ kiến trúc mô hình Vision Transformer	- Phân loại hình ảnh sử dụng mô hình Vision Transformer và các mô hình tiền huấn luyện cho Vision Transformer

- Hết -