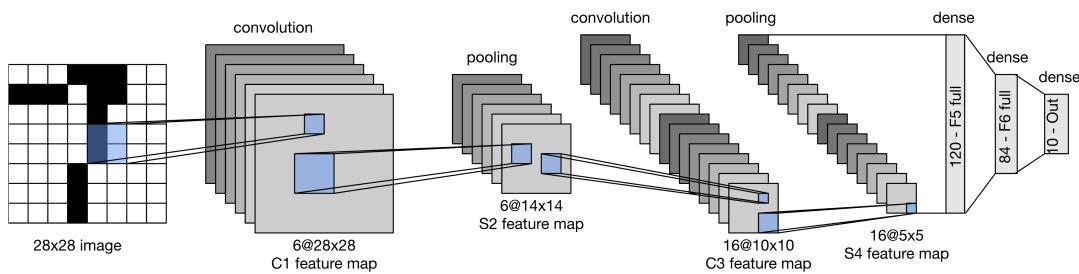


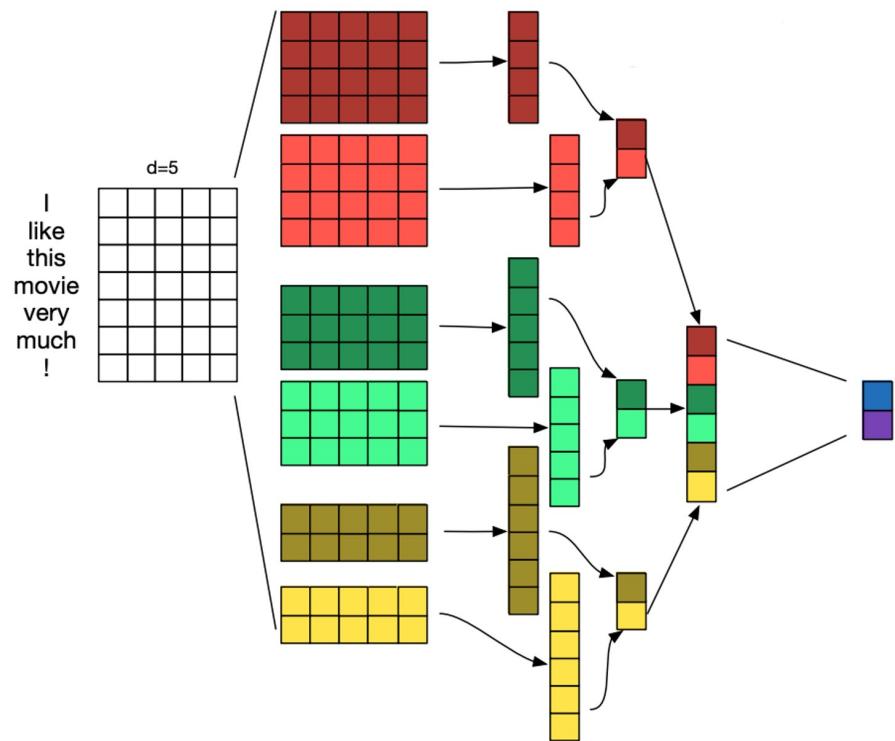
Exercise: Convolutional Neural Network

Ngày 29 tháng 11 năm 2024

Phần 1. Giới thiệu



Hình 1: Mô hình LeNet cho bài toán phân loại ảnh trên bộ dữ liệu MNIST.



Hình 2: Mô hình TextCNN cho bài toán phân loại văn bản trên bộ dữ liệu NTC-SCV.

Trong xử lý ảnh, mỗi ảnh có hàng ngàn pixels, mỗi pixel được xem như 1 feature, vậy nếu như một bức ảnh có kích thước $1000 * 1000$, thì sẽ có 1.000.000 features. Với normal feed-forward neural networks, mỗi layer là full-connected với previous input layer. Trong normal feed-forward neural network, với mỗi layer, có 1.000.000 pixels, mỗi pixel lại kết nối full-connected với 1.000.000 pixels ở layer trước, tức sẽ có 10^{12} tham số. Đây là còn số quá lớn để có thể tính được vào thời điểm đó, bởi vì với mô hình có nhiều tham số, sẽ dễ bị overfitted và cần lượng lớn data cho việc training, ngoài ra còn cần nhiều memory và năng lực tính toán cho việc training và prediction.

Vì vậy, sự ra đời CNN giúp xây dựng các model giải quyết hiệu quả với dữ liệu ảnh. Có 2 đặc tính của image hình thành nên cách hoạt động của CNN trên image, đó là feature localization và feature independence of location.

Feature localization: mỗi pixel hoặc feature có liên quan với các pixel quanh nó.

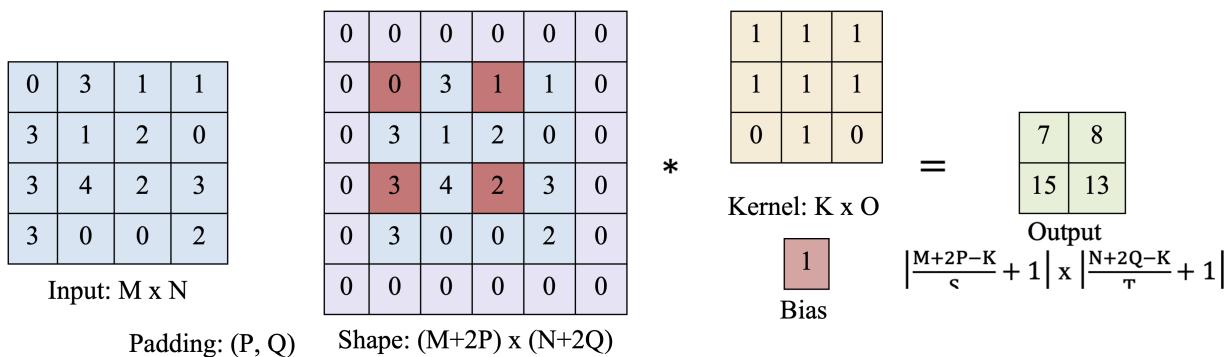
Feature Independence of location: mỗi feature dù nó có nằm ở đâu trong bức ảnh, thì nó vẫn mang giá trị của feature đó. CNN xử lý vấn đề có quá nhiều tham số với Shared parameters (feature independence of location) của Locally connected networks (feature localization), được gọi là Convolution Net.

Locally connected layer: Trong hidden layer đầu tiên, mỗi node sẽ kết nối tới một cụm nhỏ pixels của input image chứ không phải toàn bộ image, gọi là small portion. Theo cách này, ta sẽ có ít kết nối hơn, vì thế ít tham số hơn giữa input và hidden layer đầu tiên.

Shared parameters: Có những khu vực, mà việc tìm ra feature là giống nhau về cách làm, vì vậy ta có thể dùng chung bộ parameter, trong hình trên là phía phải bên trên và phía trái bên dưới. Tức ta chia sẻ bộ parameter giữa những vị trí khác nhau trong bức ảnh.

CNN có 3 thành phần chính:

- Convolution layer: gồm Filter, Padding và Stride. Filter là locally connected network, mỗi filter (kernel) sẽ học được nhiều feature trong images. Ma trận ảnh đầu vào có thể sẽ được padding thêm các giá trị padding index vào ngoài các hàng và các cột. Mỗi filter sẽ di chuyển quanh bức ảnh với bước nhảy được cấu hình trước là Stride.



Hình 3: Convolutional Operation, Filter, Padding, Stride.

- Pooling layer: bao gồm max-pooling và average-pooling layer tương ứng. Max-pooling layer chọn giá trị lớn nhất từ cửa sổ tính toán, còn Average-pooling layer sẽ tính giá trị trung bình của các giá trị mỗi kernel.

Input: 6 x 6

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Kernel Size: 2
Stride: 2

3	3	3
4	4	4
4	4	4

Output: 3 x 3

Hình 4: Max Pooling.

Input: 6 x 6

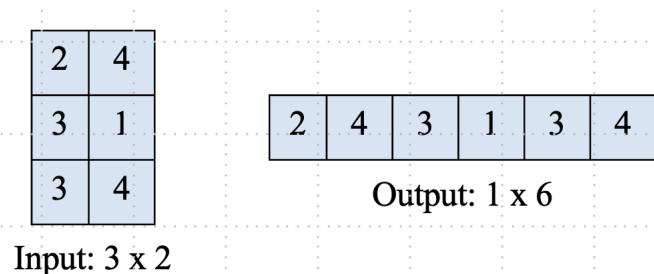
3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Kernel Size: (3, 2)
Stride: 2

2	1.7	0.8
1.8	1.6	1.3

Hình 5: Average Pooling.

- Fully-connected layer: flatten convolution layer cuối cùng duỗi các ma trận nhiều chiều trong ảnh và fully connect neuron cho output layer.

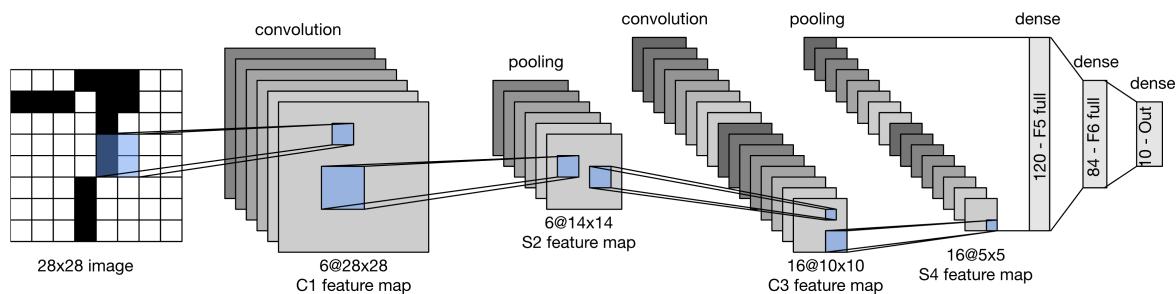


Hình 6: Flatten Layer.

Phần 2. Thực hành

2.1. Phân loại hình ảnh

Phần này chúng ta xây dựng mô hình phân loại cho dữ liệu ảnh và văn bản sử dụng mạng CNN. Một trong những mô hình mạng CNN đầu tiên và nổi bật đó là NeNet được mô tả như hình sau:



Hình 7: LeNet Model.

2.1.1. Bộ dữ liệu MNIST

Đầu tiên, mô hình phân loại văn bản sử dụng LeNet được xây dựng trên tập dữ liệu phân loại chữ số MNIST, với 70.000 ảnh và nhãn tương ứng thuộc 10 classes từ 0 đến 9.

1. Tải về bộ dữ liệu

```

1 # Import libs
2 import os
3 import random
4 import numpy as np
5
6 import torch
7 import torch.nn as nn
8 import torch.optim as optim
9 import torch.nn.functional as F
10 import torch.utils.data as data
11
12 import torchvision.transforms as transforms
13 import torchvision.datasets as datasets
14
15 from torchsummary import summary
16
17 import matplotlib.pyplot as plt
18 from PIL import Image
19
20 ROOT = './data'
21 train_data = datasets.MNIST(
22     root=ROOT,
23     train=True,
24     download=True
25 )
26 test_data = datasets.MNIST(
27     root=ROOT,
28     train=False,
29     download=True
30 )
  
```

2. Tiền xử lý dữ liệu

- Chia tỉ lệ các tập training: validation = 0.9 : 0.1
- Chuẩn hoá dữ liệu và chuyển sang tensor sử dụng torchvison.transform

```

1 # split training: validation = 0.9 : 0.1
2 VALID_RATIO = 0.9
3
4 n_train_examples = int(len(train_data) * VALID_RATIO)
5 n_valid_examples = len(train_data) - n_train_examples
6
7 train_data, valid_data = data.random_split(
8     train_data,
9     [n_train_examples, n_valid_examples]
10)
11
12 # compute mean and std for normalization
13 mean = train_data.dataset.data.float().mean() / 255
14 std = train_data.dataset.data.float().std() / 255
15
16 train_transforms = transforms.Compose([
17     transforms.ToTensor(),
18     transforms.Normalize(mean=[mean], std=[std])
19])
20 test_transforms = transforms.Compose([
21     transforms.ToTensor(),
22     transforms.Normalize(mean=[mean], std=[std])
23])
24
25 train_data.dataset.transform = train_transforms
26 valid_data.dataset.transform = test_transforms
27
28 # Create dataloader
29
30 BATCH_SIZE = 256
31
32 train_dataloader = data.DataLoader(
33     train_data,
34     shuffle=True,
35     batch_size=BATCH_SIZE
36)
37
38 valid_dataloader = data.DataLoader(
39     valid_data,
40     batch_size=BATCH_SIZE
41)
```

3. Xây dựng mô hình LeNet

```

1 class LeNetClassifier(nn.Module):
2     def __init__(self, num_classes):
3         super().__init__()
4         self.conv1 = nn.Conv2d(
5             in_channels=1, out_channels=6, kernel_size=5, padding='same'
6         )
7         self.avgpool1 = nn.AvgPool2d(kernel_size=2)
8         self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
9         self.avgpool2 = nn.AvgPool2d(kernel_size=2)
10        self.flatten = nn.Flatten()
11        self.fc_1 = nn.Linear(16 * 4 * 4, 120)
12        self.fc_2 = nn.Linear(120, 84)
13        self.fc_3 = nn.Linear(84, num_classes)
```

```
14
15     def forward(self, inputs):
16         outputs = self.conv1(inputs)
17         outputs = self.avgpool1(outputs)
18         outputs = F.relu(outputs)
19         outputs = self.conv2(outputs)
20         outputs = self.avgpool2(outputs)
21         outputs = F.relu(outputs)
22         outputs = self.flatten(outputs)
23         outputs = self.fc_1(outputs)
24         outputs = self.fc_2(outputs)
25         outputs = self.fc_3(outputs)
26     return outputs
```

4. Huấn luyện mô hình

```
1 # Training function
2 def train(model, optimizer, criterion, train_dataloader, device, epoch=0, log_interval=50):
3     model.train()
4     total_acc, total_count = 0, 0
5     losses = []
6     start_time = time.time()
7
8     for idx, (inputs, labels) in enumerate(train_dataloader):
9         inputs = inputs.to(device)
10        labels = labels.to(device)
11
12        optimizer.zero_grad()
13
14        predictions = model(inputs)
15
16        # compute loss
17        loss = criterion(predictions, labels)
18        losses.append(loss.item())
19
20        # backward
21        loss.backward()
22        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
23        optimizer.step()
24        total_acc += (predictions.argmax(1) == labels).sum().item()
25        total_count += labels.size(0)
26        if idx % log_interval == 0 and idx > 0:
27            elapsed = time.time() - start_time
28            print(
29                "| epoch {:3d} | {:5d}/{:5d} batches "
30                "| accuracy {:.8.3f}|".format(
31                    epoch, idx, len(train_dataloader), total_acc / total_count
32                )
33            )
34        total_acc, total_count = 0, 0
35        start_time = time.time()
36
37    epoch_acc = total_acc / total_count
38    epoch_loss = sum(losses) / len(losses)
39    return epoch_acc, epoch_loss
40
41 # Evaluation function
42 def evaluate(model, criterion, valid_dataloader):
43     model.eval()
44     total_acc, total_count = 0, 0
```

```

45     losses = []
46
47     with torch.no_grad():
48         for idx, (inputs, labels) in enumerate(valid_dataloader):
49             inputs = inputs.to(device)
50             labels = labels.to(device)
51
52             predictions = model(inputs)
53
54             loss = criterion(predictions, labels)
55             losses.append(loss.item())
56
57             total_acc += (predictions.argmax(1) == labels).sum().item()
58             total_count += labels.size(0)
59
60     epoch_acc = total_acc / total_count
61     epoch_loss = sum(losses) / len(losses)
62     return epoch_acc, epoch_loss

```

- Training:

```

1 num_classes = len(train_data.dataset.classes)
2
3 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
4
5 lenet_model = LeNetClassifier(num_classes)
6 lenet_model.to(device)
7
8 criterion = torch.nn.CrossEntropyLoss()
9 optimizer = optim.Adam(lenet_model.parameters())
10
11 num_epochs = 10
12 save_model = './model'
13
14 train_accs, train_losses = [], []
15 eval_accs, eval_losses = [], []
16 best_loss_eval = 100
17
18 for epoch in range(1, num_epochs+1):
19     epoch_start_time = time.time()
20     # Training
21     train_acc, train_loss = train(lenet_model, optimizer, criterion, train_dataloader,
22                                   device, epoch)
23     train_accs.append(train_acc)
24     train_losses.append(train_loss)
25
26     # Evaluation
27     eval_acc, eval_loss = evaluate(lenet_model, criterion, valid_dataloader)
28     eval_accs.append(eval_acc)
29     eval_losses.append(eval_loss)
30
31     # Save best model
32     if eval_loss < best_loss_eval:
33         torch.save(lenet_model.state_dict(), save_model + '/lenet_model.pt')
34
35     # Print loss, acc end epoch
36     print("-" * 59)
37     print(
38         "| End of epoch {:3d} | Time: {:.2f}s | Train Accuracy {:.3f} | Train Loss
{:8.3f} |
| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".format(

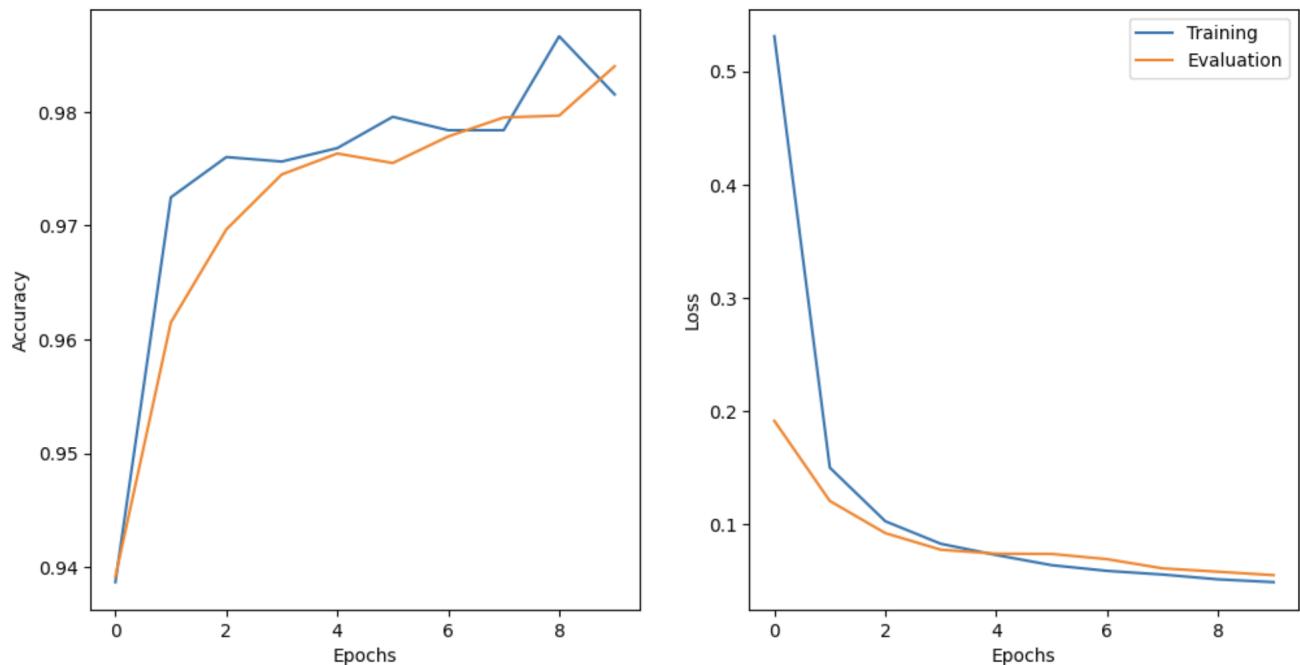
```

```

39     epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc,
40     eval_loss
41   )
42   print("-" * 59)
43
44 # Load best model
45 lenet_model.load_state_dict(torch.load(save_model + '/lenet_model.pt'))
46 lenet_model.eval()

```

- Accuracy và Loss:



Hình 8: Accracy và Loss trên tập training và validation.

5. Đánh giá mô hình trên tập test

- Dánh giá độ chính xác trên tập test

```

1 test_data.transform = test_transforms
2 test_dataloader = data.DataLoader(
3     test_data,
4     batch_size=BATCH_SIZE
5 )
6 test_acc, test_loss = evaluate(lenet_model, criterion, test_dataloader)
7 test_acc, test_loss

```

- Kết quả độ chính xác trên tập test là 98

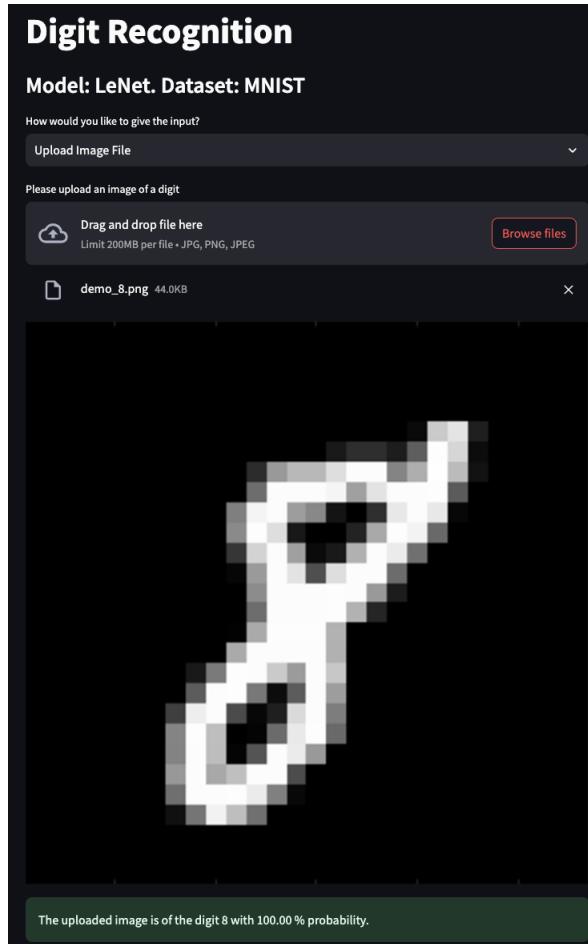
6. Triển khai mô hình

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 9: Giao diện ứng dụng.

2.1.2. Bộ dữ liệu Cassava Leaf Disease

Trong phần này chúng ta xây dựng mô hình phân loại cho bộ dữ liệu Cassava Leaf Disease với các ảnh được gán nhãn và phân loại vào 5 lớp.

1. Tải về bộ dữ liệu

```

1 !wget --no-check-certificate https://storage.googleapis.com/ucimlrepository/cassavaleafdata.zip \
2                                         -O /content/cassavaleafdata.zip
3 !unzip /content/cassavaleafdata.zip
4
5 # Import libs
6 import os
7 import random
8 import numpy as np
9
10 import torch
11 import torch.nn as nn
12 import torch.optim as optim
13 import torch.nn.functional as F
14 import torch.utils.data as data
15
16 import torchvision.transforms as transforms

```



Hình 10: Bộ dữ liệu Cassava Leaf Disease.

```

17 import torchvision.datasets as datasets
18
19 from torchsummary import summary
20
21 import matplotlib.pyplot as plt
22 from PIL import Image

```

2. Tiền xử lý dữ liệu

```

1 data_paths = {
2     'train': './train',
3     'valid': './validation',
4     'test': './test'
5 }
6
7 # load image from path
8 def loader(path):
9     return Image.open(path)

```

```

10
11 img_size = 150
12 train_transforms = transforms.Compose([
13     transforms.Resize((150, 150)),
14     transforms.ToTensor(),
15 ])
16
17 train_data = datasets.ImageFolder(
18     root=data_paths['train'],
19     loader=loader,
20     transform=train_transforms
21 )
22 valid_data = datasets.ImageFolder(
23     root=data_paths['valid'],
24     transform=train_transforms
25 )
26 test_data = datasets.ImageFolder(
27     root=data_paths['test'],
28     transform=train_transforms
29 )

```

3. Xây dựng mô hình

```

1
2 class LeNetClassifier(nn.Module):
3     def __init__(self, num_classes):
4         super().__init__()
5         self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding='same')
6             self.avgpool1 = nn.AvgPool2d(kernel_size=2)
7             self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
8             self.avgpool2 = nn.AvgPool2d(kernel_size=2)
9             self.flatten = nn.Flatten()
10            self.fc_1 = nn.Linear(16 * 4 * 4, 120)
11            self.fc_2 = nn.Linear(120, 84)
12            self.fc_3 = nn.Linear(84, num_classes)
13
14     def forward(self, inputs):
15         outputs = self.conv1(inputs)
16         outputs = self.avgpool1(outputs)
17         outputs = F.relu(outputs)
18         outputs = self.conv2(outputs)
19         outputs = self.avgpool2(outputs)
20         outputs = F.relu(outputs)
21         outputs = self.flatten(outputs)
22         outputs = self.fc_1(outputs)
23         outputs = self.fc_2(outputs)
24         outputs = self.fc_3(outputs)
25         return outputs

```

4. Huấn luyện mô hình

```

1 # Training function
2 def train(model, optimizer, criterion, train_dataloader, device, epoch=0, log_interval=50):
3     model.train()
4     total_acc, total_count = 0, 0
5     losses = []
6     start_time = time.time()
7
8     for idx, (inputs, labels) in enumerate(train_dataloader):
9         inputs = inputs.to(device)

```

```

10     labels = labels.to(device)
11
12     optimizer.zero_grad()
13
14     predictions = model(inputs)
15
16     # compute loss
17     loss = criterion(predictions, labels)
18     losses.append(loss.item())
19
20     # backward
21     loss.backward()
22     torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
23     optimizer.step()
24     total_acc += (predictions.argmax(1) == labels).sum().item()
25     total_count += labels.size(0)
26     if idx % log_interval == 0 and idx > 0:
27         elapsed = time.time() - start_time
28         print(
29             "| epoch {:3d} | {:5d}/{:5d} batches "
30             "| accuracy {:.3f}".format(
31                 epoch, idx, len(train_dataloader), total_acc / total_count
32             )
33         )
34     total_acc, total_count = 0, 0
35     start_time = time.time()
36
37 epoch_acc = total_acc / total_count
38 epoch_loss = sum(losses) / len(losses)
39 return epoch_acc, epoch_loss
40
41 # Evaluation function
42 def evaluate(model, criterion, valid_dataloader):
43     model.eval()
44     total_acc, total_count = 0, 0
45     losses = []
46
47     with torch.no_grad():
48         for idx, (inputs, labels) in enumerate(valid_dataloader):
49             inputs = inputs.to(device)
50             labels = labels.to(device)
51
52             predictions = model(inputs)
53
54             loss = criterion(predictions, labels)
55             losses.append(loss.item())
56
57             total_acc += (predictions.argmax(1) == labels).sum().item()
58             total_count += labels.size(0)
59
60     epoch_acc = total_acc / total_count
61     epoch_loss = sum(losses) / len(losses)
62     return epoch_acc, epoch_loss

```

- Huấn luyện mô hình:

```

1 num_classes = len(train_data.classes)
2
3 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
4
5 lenet_model = LeNetClassifier(num_classes)

```

```

6 lenet_model.to(device)
7
8 criterion = torch.nn.CrossEntropyLoss()
9 learning_rate = 2e-4
10 optimizer = optim.Adam(lenet_model.parameters(), learning_rate)
11
12 num_epochs = 10
13 save_model = './model'
14
15 train_accs, train_losses = [], []
16 eval_accs, eval_losses = [], []
17 best_loss_eval = 100
18
19 for epoch in range(1, num_epochs+1):
20     epoch_start_time = time.time()
21     # Training
22     train_acc, train_loss = train(lenet_model, optimizer, criterion, train_dataloader,
23         device, epoch, log_interval=10)
24     train_accs.append(train_acc)
25     train_losses.append(train_loss)
26
27     # Evaluation
28     eval_acc, eval_loss = evaluate(lenet_model, criterion, valid_dataloader)
29     eval_accs.append(eval_acc)
30     eval_losses.append(eval_loss)
31
32     # Save best model
33     if eval_loss < best_loss_eval:
34         torch.save(lenet_model.state_dict(), save_model + '/lenet_model.pt')
35
36     # Print loss, acc end epoch
37     print("-" * 59)
38     print(
39         "| End of epoch {:3d} | Time: {:.5f}s | Train Accuracy {:.3f} | Train Loss
40         {:.3f} "
41         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".format(
42             epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc,
43             eval_loss
44         )
45     )
46     print("-" * 59)
47
48     # Load best model
49     lenet_model.load_state_dict(torch.load(save_model + '/lenet_model.pt'))
50     lenet_model.eval()

```

5. Đánh giá mô hình

- Kết quả trên tập test: 56

```

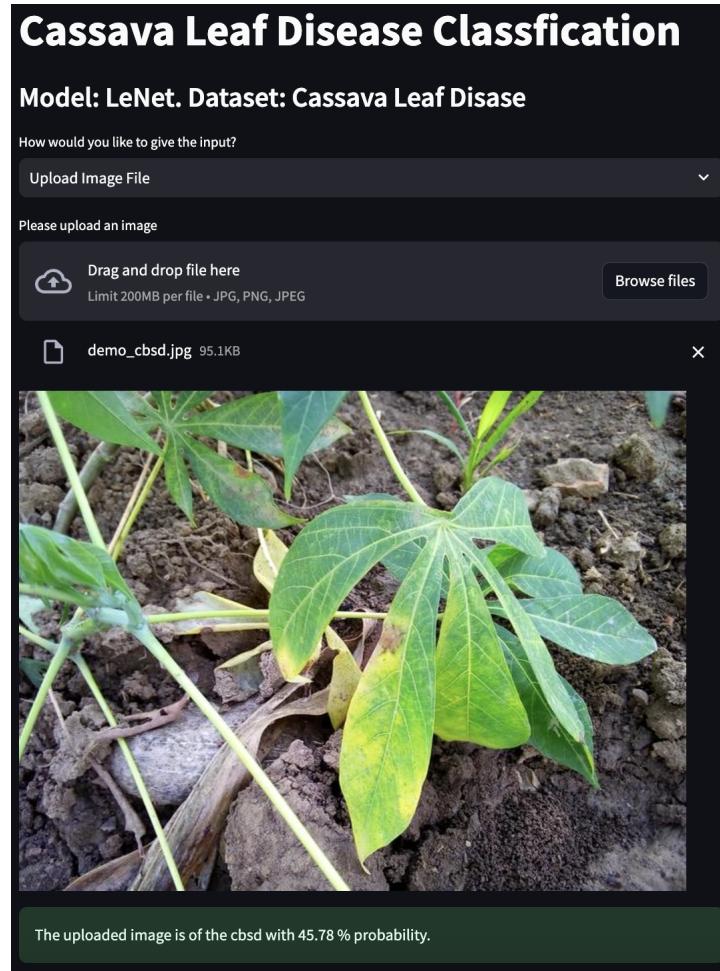
1 test_acc, test_loss = evaluate(lenet_model, criterion, test_dataloader)
2 test_acc, test_loss

```

6. 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.

2.2. Phân loại văn bản

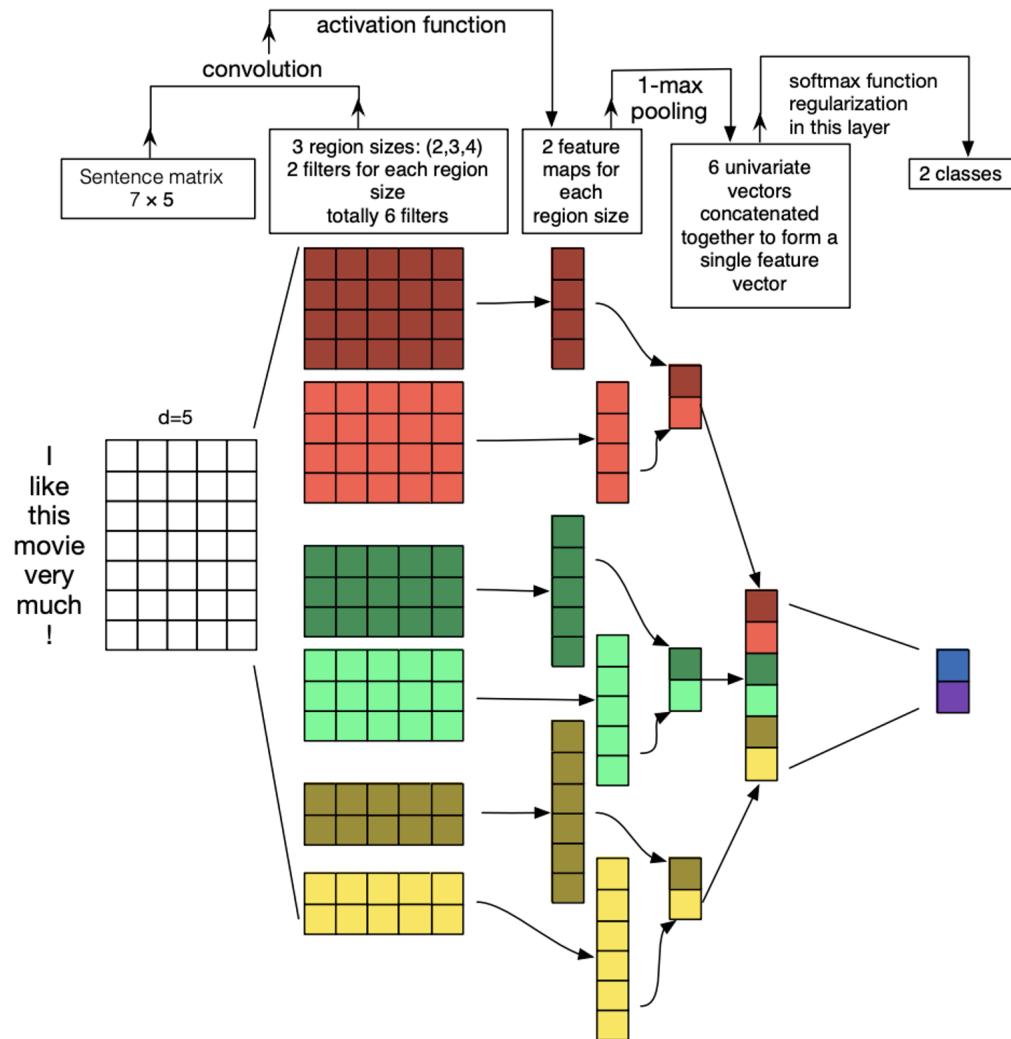
Trong phần này chúng ta xây dựng mô hình phân loại văn bản sử dụng kiến trúc mạng TextCNN trên bộ dữ liệu NTC-SCV:

1. Tải về bộ dữ liệu

```

1 !git clone https://github.com/congnggia0609/ntc-scv.git
2 !unzip ./ntc-scv/data/data_test.zip -d ./data
3 !unzip ./ntc-scv/data/data_train.zip -d ./data
4 !rm -rf ./ntc-scv
5
6 # load data from paths to dataframe
7 import os
8 import pandas as pd
9
10 def load_data_from_path(folder_path):
11     examples = []
12     for label in os.listdir(folder_path):
13         full_path = os.path.join(folder_path, label)
14         for file_name in os.listdir(full_path):
15             file_path = os.path.join(full_path, file_name)
16             with open(file_path, "r", encoding="utf-8") as f:

```



Hình 12: TextCNN Model.

```

17     lines = f.readlines()
18     sentence = " ".join(lines)
19     if label == "neg":
20         label = 0
21     if label == "pos":
22         label = 1
23     data = {
24         'sentence': sentence,
25         'label': label
26     }
27     examples.append(data)
28 return pd.DataFrame(examples)
29
30 folder_paths = {
31     'train': './data/data_train/train',
32     'valid': './data/data_train/test',
33     'test': './data/data_test/test'
34 }
35
36 train_df = load_data_from_path(folder_paths['train'])

```

```
37 valid_df = load_data_from_path(folder_paths['valid'])
38 test_df = load_data_from_path(folder_paths['test'])
```

2. Tiền xử lý dữ liệu

Với bộ dữ liệu NTC-SCV có một số bình luận viết bằng tiếng anh hoặc tiếng pháp và chứa các thẻ HTML, đường dẫn URLs. Tiền xử lý dữ liệu gồm 2 bước:

- Xoá bỏ những bình luận không phải tiếng việt

Sử dụng thư viện langid được mô tả như sau:

```
1 # Install library
2 !pip install langid
3
4 from langid.langid import LanguageIdentifier, model
5 def identify_vn(df):
6     identifier = LanguageIdentifier.from_modelstring(model, norm_probs=True)
7     not_vi_idx = set()
8     THRESHOLD = 0.9
9     for idx, row in df.iterrows():
10         score = identifier.classify(row["sentence"])
11         if score[0] != "vi" or (score[0] == "vi" and score[1] <= THRESHOLD):
12             not_vi_idx.add(idx)
13     vi_df = df[~df.index.isin(not_vi_idx)]
14     not_vi_df = df[df.index.isin(not_vi_idx)]
15     return vi_df, not_vi_df
16
17 train_df_vn, train_df_other = identify_vn(train_df)
```

- Làm sạch dữ liệu

Các bước tiền làm sạch liệu:

- Xoá bỏ thẻ HTML, đường dẫn URL
- Xoá bỏ dấu câu, số
- Xoá bỏ các ký tự đặc biệt, emoticons,...
- Chuẩn hoá khoảng trắng
- Chuyển sang viết thường

```
1 import re
2 import string
3
4 def preprocess_text(text):
5
6     url_pattern = re.compile(r'https?://\s+\www\.\s+')
7     text = url_pattern.sub(" ", text)
8
9     html_pattern = re.compile(r'<[^>]+>')
10    text = html_pattern.sub(" ", text)
11
12    replace_chars = list(string.punctuation + string.digits)
13    for char in replace_chars:
14        text = text.replace(char, " ")
15
16    emoji_pattern = re.compile([
17        u"\U0001F600-\U0001F64F" # emoticons
18        u"\U0001F300-\U0001F5FF" # symbols & pictographs
19        u"\U0001F680-\U0001F6FF" # transport & map symbols
20    ])
```

```

20         u"\U0001F1E0-\U0001F1FF" # flags (iOS)
21         u"\U0001F1F2-\U0001F1F4" # Macau flag
22         u"\U0001F1E6-\U0001F1FF" # flags
23         u"\U0001F600-\U0001F64F"
24         u"\U00002702-\U000027B0"
25         u"\U000024C2-\U0001F251"
26         u"\U0001f926-\U0001f937"
27         u"\U0001F1F2"
28         u"\U0001F1F4"
29         u"\U0001F620"
30         u"\u200d"
31         u"\u2640-\u2642"
32     "+" , flags=re.UNICODE)
33     text = emoji_pattern.sub(r" ", text)
34
35     text = " ".join(text.split())
36
37     return text.lower()
38
39 train_df_vn['preprocess_sentence'] = [
40     preprocess_text(row['sentence']) for index, row in train_df_vn.iterrows()
41 ]
42 valid_df['preprocess_sentence'] = [
43     preprocess_text(row['sentence']) for index, row in valid_df.iterrows()
44 ]
45 test_df['preprocess_sentence'] = [
46     preprocess_text(row['sentence']) for index, row in test_df.iterrows()
47 ]

```

3. Biểu diễn văn bản thành vector

Để biểu diễn dữ liệu văn bản thành các đặc trưng (vectors), chúng ta sử dụng thư viện torchtext.

```

1 !pip install -q torchtext==0.16.0
2
3 # word-based tokenizer
4 from torchtext.data.utils import get_tokenizer
5 tokenizer = get_tokenizer("basic_english")
6
7 # create iter dataset
8 def yield_tokens(sentences, tokenizer):
9     for sentence in sentences:
10         yield tokenizer(sentence)
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(train_df_vn['preprocess_sentence'], tokenizer),
18     max_tokens=vocab_size,
19     specials=["<pad>", "<unk>"]
20 )
21 vocabulary.set_default_index(vocabulary["<unk>"])
22
23 # convert iter into torchtext dataset
24 from torchtext.data.functional import to_map_style_dataset
25 def prepare_dataset(df):
26     for index, row in df.iterrows():
27         sentence = row['preprocess_sentence']
28         encoded_sentence = vocabulary(tokenizer(sentence))

```

```

29     label = row['label']
30     yield encoded_sentence, label
31
32 train_dataset = prepare_dataset(train_df_vn)
33 train_dataset = to_map_style_dataset(train_dataset)
34
35 valid_dataset = prepare_dataset(valid_df)
36 valid_dataset = to_map_style_dataset(valid_dataset)

1 import torch
2 from torch.utils.data import DataLoader
3
4 def collate_batch(batch):
5     # create inputs, offsets, labels for batch
6     encoded_sentences, labels = [], []
7     for encoded_sentence, label in batch:
8         labels.append(label)
9         encoded_sentence = torch.tensor(encoded_sentence, dtype=torch.int64)
10        encoded_sentences.append(encoded_sentence)
11
12    labels = torch.tensor(labels, dtype=torch.int64)
13    encoded_sentences = pad_sequence(
14        encoded_sentences,
15        padding_value=vocabulary["<pad>"]
16    )
17
18    return encoded_sentences, labels
19
20 batch_size = 128
21 train_dataloader = DataLoader(
22     train_dataset,
23     batch_size=batch_size,
24     shuffle=True,
25     collate_fn=collate_batch
26 )
27 valid_dataloader = DataLoader(
28     valid_dataset,
29     batch_size=batch_size,
30     shuffle=False,
31     collate_fn=collate_batch
32 )

```

4. Xây dựng mô hình TextCNN

```

1 class TextCNN(nn.Module):
2     def __init__(self,
3                  vocab_size, embedding_dim, kernel_sizes, num_filters, num_classes):
4         super(TextCNN, self).__init__()
5
6         self.vocab_size = vocab_size
7         self.embedding_dim = embedding_dim
8         self.kernel_sizes = kernel_sizes
9         self.num_filters = num_filters
10        self.num_classes = num_classes
11        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
12        self.conv = nn.ModuleList([
13            nn.Conv1d(
14                in_channels=embedding_dim,
15                out_channels=num_filters,
16                kernel_size=k,
17

```

```

18         stride=1
19     ) for k in kernel_sizes])
20 self.fc = nn.Linear(len(kernel_sizes) * num_filters, num_classes)
21
22 def forward(self, x):
23     batch_size, sequence_length = x.shape
24     x = self.embedding(x.T).transpose(1, 2)
25     x = [F.relu(conv(x)) for conv in self.conv]
26     x = [F.max_pool1d(c, c.size(-1)).squeeze(dim=-1) for c in x]
27     x = torch.cat(x, dim=1)
28     x = self.fc(x)
29     return x

```

5. Huấn luyện mô hình

```

1 # Training
2 import time
3
4 def train(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        # zero grad
15        optimizer.zero_grad()
16
17        # predictions
18        predictions = model(inputs)
19
20        # compute loss
21        loss = criterion(predictions, labels)
22        losses.append(loss.item())
23
24        # backward
25        loss.backward()
26        optimizer.step()
27        total_acc += (predictions.argmax(1) == labels).sum().item()
28        total_count += labels.size(0)
29        if idx % log_interval == 0 and idx > 0:
30            elapsed = time.time() - start_time
31            print(
32                "| epoch {:3d} | {:5d}/{:5d} batches "
33                "| accuracy {:.8.3f}").format(
34                    epoch, idx, len(train_dataloader), total_acc / total_count
35                )
36        total_acc, total_count = 0, 0
37        start_time = time.time()
38
39    epoch_acc = total_acc / total_count
40    epoch_loss = sum(losses) / len(losses)
41    return epoch_acc, epoch_loss
42
43 # Evaluation
44 def evaluate(model, criterion, valid_dataloader):

```

```

46     model.eval()
47     total_acc, total_count = 0, 0
48     losses = []
49
50     with torch.no_grad():
51         for idx, (inputs, labels) in enumerate(valid_dataloader):
52             inputs = inputs.to(device)
53             labels = labels.to(device)
54             # predictions
55             predictions = model(inputs)
56
57             # compute loss
58             loss = criterion(predictions, labels)
59             losses.append(loss.item())
60
61             total_acc += (predictions.argmax(1) == labels).sum().item()
62             total_count += labels.size(0)
63
64     epoch_acc = total_acc / total_count
65     epoch_loss = sum(losses) / len(losses)
66     return epoch_acc, epoch_loss

```

- Huấn luyện mô hình

```

1 num_class = 2
2 vocab_size = len(vocabulary)
3 embedding_dim = 100
4 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
5
6 model = TextCNN(
7     vocab_size=vocab_size,
8     embedding_dim=embedding_dim,
9     kernel_sizes=[3, 4, 5],
10    num_filters=100,
11    num_classes=2
12 )
13 model.to(device)
14
15 criterion = torch.nn.CrossEntropyLoss()
16 optimizer = torch.optim.Adam(model.parameters())
17
18 num_epochs = 10
19 save_model = './model'
20
21 train_accs, train_losses = [], []
22 eval_accs, eval_losses = [], []
23 best_loss_eval = 100
24
25 for epoch in range(1, num_epochs+1):
26     epoch_start_time = time.time()
27     # Training
28     train_acc, train_loss = train(model, optimizer, criterion, train_dataloader,
29                                   device, epoch)
30     train_accs.append(train_acc)
31     train_losses.append(train_loss)
32
33     # Evaluation
34     eval_acc, eval_loss = evaluate(model, criterion, valid_dataloader)
35     eval_accs.append(eval_acc)
36     eval_losses.append(eval_loss)

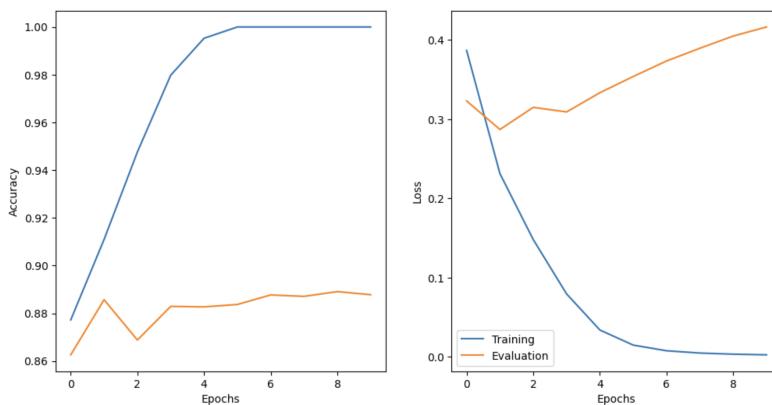
```

```

37     # Save best model
38     if eval_loss < best_loss_eval:
39         torch.save(model.state_dict(), save_model + '/text_cnn_model.pt')
40
41     # Print loss, acc end epoch
42     print("-" * 59)
43     print(
44         "| End of epoch {:3d} | Time: {:.5f}s | Train Accuracy {:.3f} | Train Loss
45         {:.3f}" +
46         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".format(
47             epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc,
48             eval_loss
49         )
50     )
51     print("-" * 59)
52
53     # Load best model
54     model.load_state_dict(torch.load(save_model + '/text_cnn_model.pt'))
55     model.eval()

```

- Kết quả huấn luyện mô hình:



Hình 13: Accuracy và Loss cho tập training và validation.

6. Đánh giá mô hình

```

1 test_dataset = prepare_dataset(test_df)
2 test_dataset = to_map_style_dataset(test_dataset)
3
4 test_dataloader = DataLoader(
5     test_dataset,
6     batch_size=batch_size,
7     shuffle=False,
8     collate_fn=collate_batch
9 )
10
11 test_acc, test_loss = evaluate(model, criterion, valid_dataloader)
12 test_acc, test_loss

```

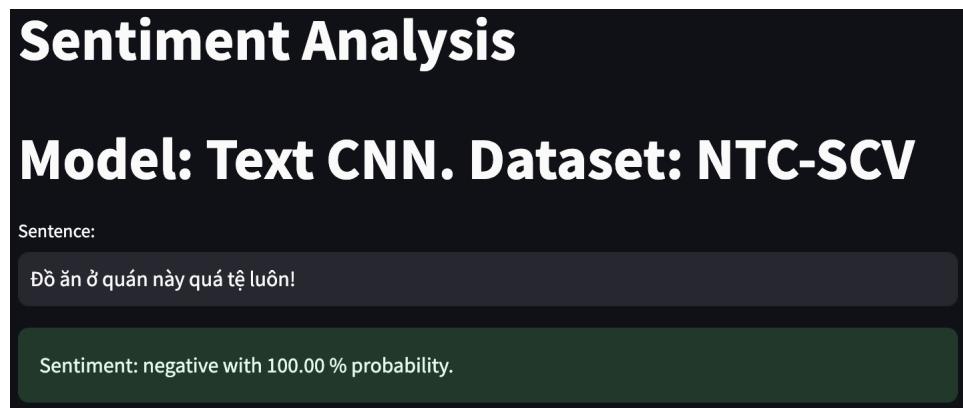
- Độ chính xác trên tập test là 88.78

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 14: Giao diện ứng dụng.

Phần 3. Câu hỏi trắc nghiệm

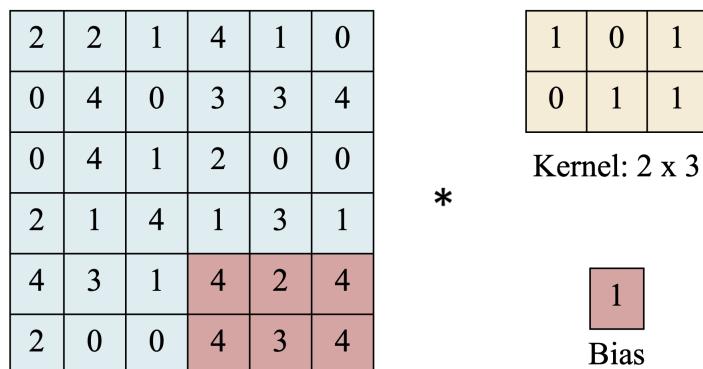
Câu hỏi 1 Phép toán sau đây không thuộc lớp tích chập (Convolutional Layer)?

- a) Covolutional Operation
- b) Padding
- c) Stride
- d) Max Pooling

Câu hỏi 2 Lớp Pooling có tác dụng?

- a) Tăng kích thước ảnh
- b) Giảm kích thước ảnh
- c) Cả hai đáp án trên đều đúng
- d) Cả hai đáp án trên đều sai

Trả lời câu hỏi 3 và 4 dựa vào ma trận input và kernel được biểu diễn như hình sau:



Hình 15: Phép convolution với kernel size (2, 3).

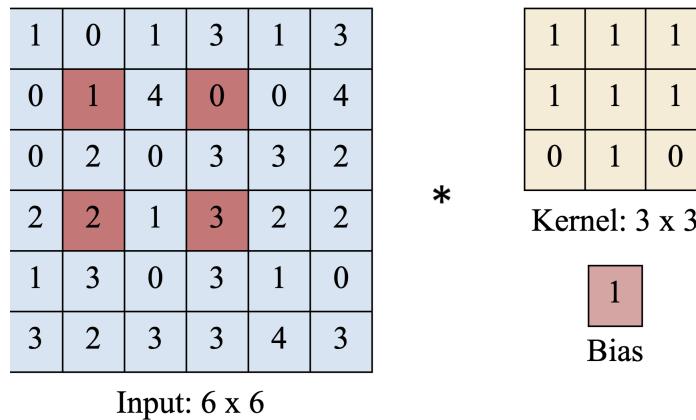
Câu hỏi 3 Phép tính convolutional với kernel trên cho kết quả có kích thước là?

- a) 4 x 3
- b) 5 x 4
- c) 6 x 5
- d) 5 x 6

Câu hỏi 4 Kết quả giá trị cuối cùng theo cả hàng và cột của ví dụ trên (tương ứng là phép tính tại vị trí màu hồng) là?

- a) 4
- b) 8
- c) 16
- d) 32

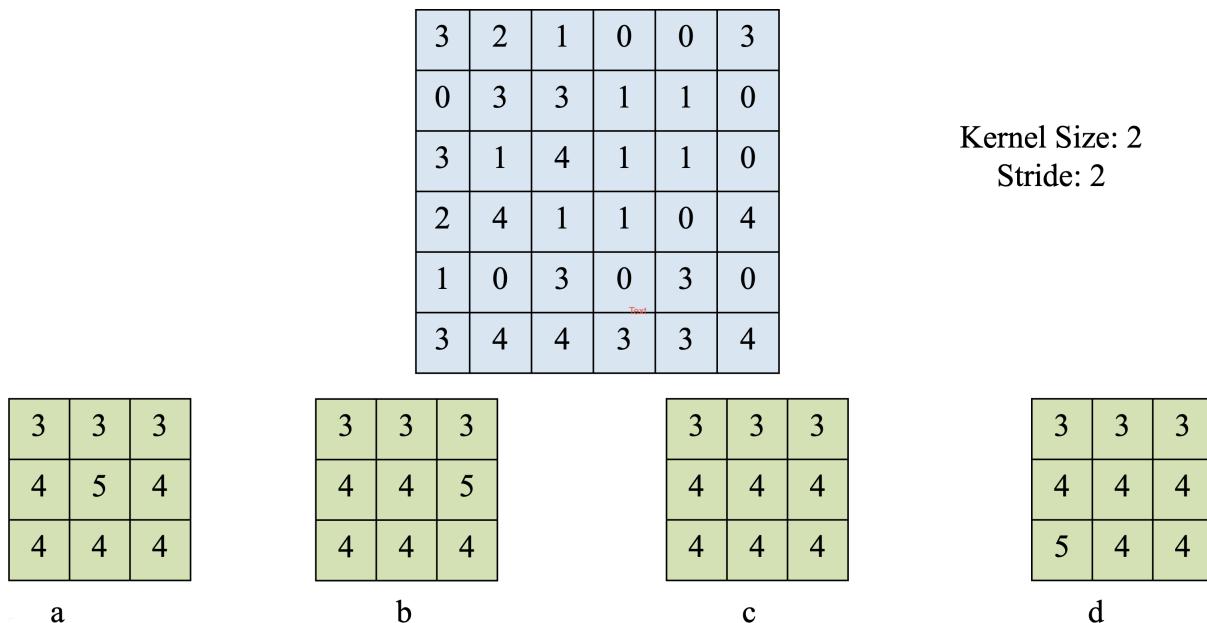
Câu hỏi 5 Cho ma trận input và kernel size: 3, stride: 2



Hình 16: Phép convolution với kernel size: 3, stride: 3.

- Kết quả của phép tính trên có kích thước là?
- 4 x 4
 - 3 x 3
 - 2 x 2
 - 1 x 1

Câu hỏi 6 Chọn đáp án đúng cho phép max pooling sau đây?



Hình 17: Phép max pooling với kernel size: 2 và stride: 2.

Câu hỏi 7 Bộ dữ liệu được sử dụng cho phân loại hình ảnh trong bài là?

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

Câu hỏi 8 Bộ dữ liệu được sử dụng cho phân loại văn bản được sử dụng trong bài là?

- a) C4
- b) MNIST
- c) NTC-SCV
- d) Tweets

Câu hỏi 9 Mô hình phân loại hình ảnh được sử dụng trong bài là?

- a) LeNet
- b) VGG
- c) GoogleNet
- d) RCNN

Câu hỏi 10 Mô hình phân loại văn bản được sử dụng trong bài là?

- a) RNN
- b) TextCNN
- c) Neural Network
- d) Logistic Regression

Phần 4. Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải về [tại đây](#)
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	<ul style="list-style-type: none"> - Nắm vững kiến thức về bài toán phân loại hình ảnh - Kiến trúc mạng LeNet - Các thành phần quan trọng xây dựng mô hình LeNet như: Lớp tích chập, lớp pooling, lớp linear 	<ul style="list-style-type: none"> - Xây dựng mô hình phân loại hình ảnh sử dụng mạng LeNet trên bộ dữ liệu MNIST và Cassava Leaf Disease
2.	<ul style="list-style-type: none"> - Hiểu về mô hình phân loại văn bản sử dụng TextCNN - Các lớp xử lý cho văn bản như Embedding thông qua thư viện torchtext 	<ul style="list-style-type: none"> - Xây dựng mô hình phân loại văn bản sử dụng TextCNN - Bộ dữ liệu phân loại văn bản NTC-SCV

- *Hết* -