AI VIET NAM
@aivietnam.edu.vn

Module 06 – Exercise Class

Convolutional Neural Network

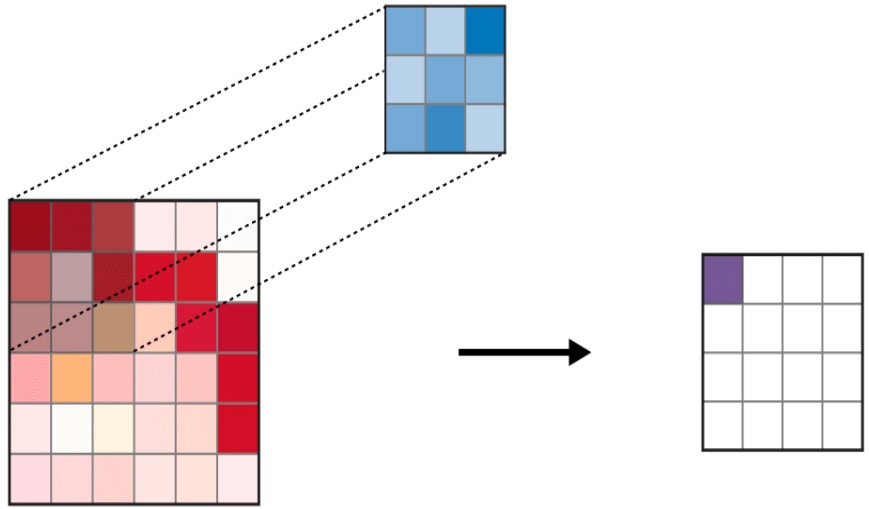
Nguyen Quoc Thai

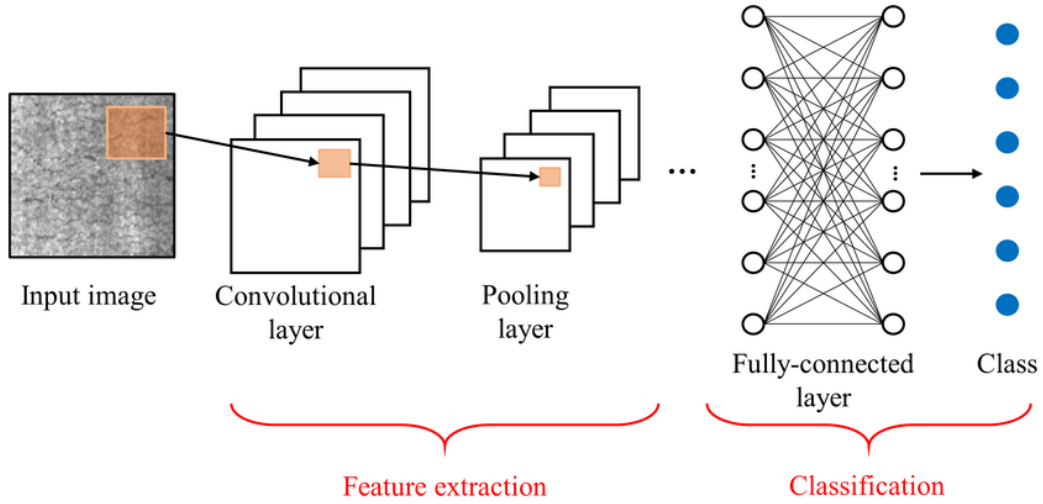
Code - Data

Objectives

CNN

- Convolution Layer, Pooling Layer
- Multiple Input – Output Channels
- LeNet Model





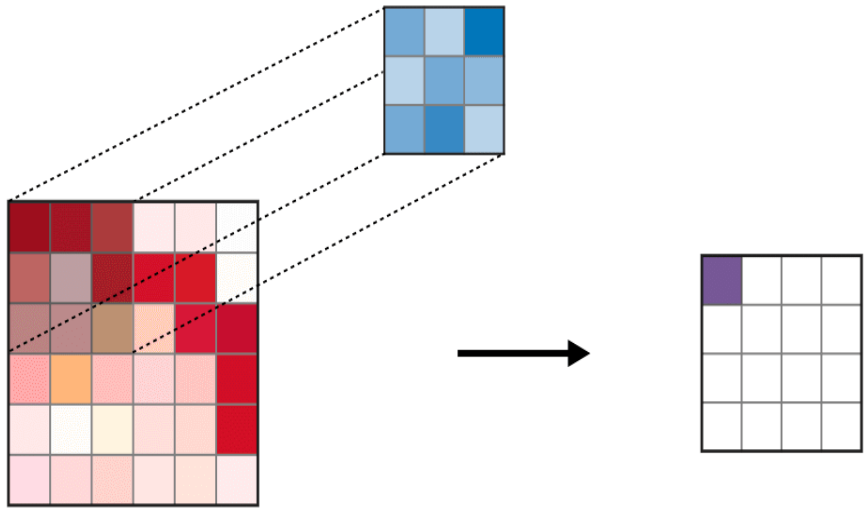
Classification

- Image Classification
- TextCNN Model
- Text Classification

Outline

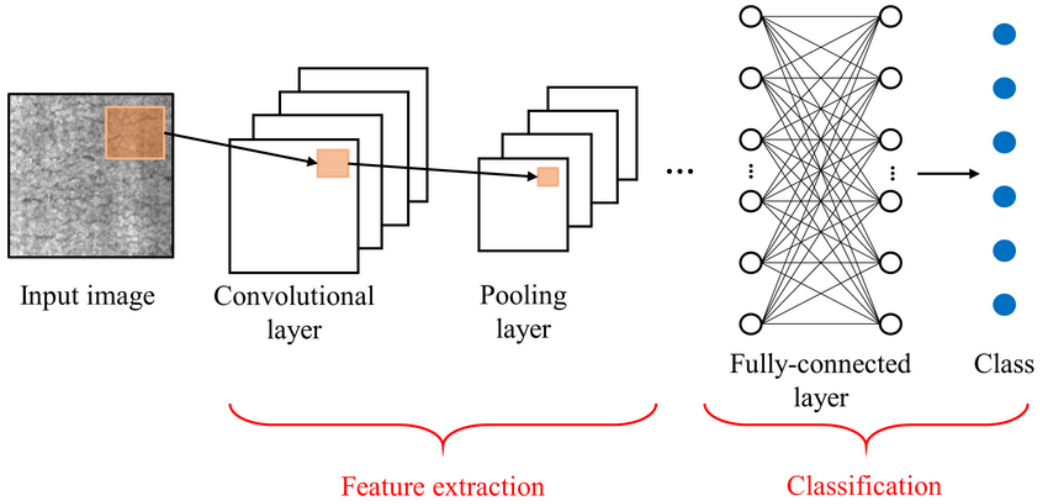
SECTION 1

CNN



SECTION 2

Applications



CNN

Convolutional Layer

Stride: (S, T)

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

Input: M x N

Padding: (P, Q)

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

Shape: (M+2P) x (N+2Q)

*

1	1	1
1	1	1
0	1	0

Kernel: K x O

1

Bias

=

7	8
15	13

$$\left\lfloor \frac{M + 2P - K}{S} + 1 \right\rfloor \times \left\lfloor \frac{N + 2Q - O}{T} + 1 \right\rfloor$$

CNN

Max Pooling Layer

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

Input: 6 x 6

Kernel Size: 2
Stride: 2

3	3	3
4	4	4
4	4	4

Output: 3 x 3

MaxPool1d
Kernel Size: 3
Stride: 3

3	3
3	1
4	1
4	4
3	3
4	4

Output: 6 x 2

Average Pooling Layer

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

Input: 6 x 6

Kernel Size: (3, 2)
Stride: 2

2.0	1.7	0.8
1.8	1.6	1.3

Output: 2 x 3

AvgPool1d
Kernel Size: 3
Stride: 3

2.0	1.0
2.0	0.7
2.7	0.7
2.3	1.7
1.3	1.0
3.7	3.3

Output: 6 x 2

CNN

Multiple Inout Channels

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

Input Channel #1 (Red)

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

Input Channel #2 (Green)

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

Input Channel #3 (Blue)

1	1	1
1	1	1
1	0	0

Kernel Channel #1

1	1	1
1	1	1
1	1	0

Kernel Channel #2

1	0	1
1	0	1
0	1	1

Kernel Channel #3

14 + 17 + 11 + Bias

1

43	50
40	13

Multiple Input Channels

```
input = torch.randint(5, (3, 4, 4), dtype=torch.float32)
input
```

```
tensor([[[0., 0., 2., 1.],
         [4., 1., 3., 1.],
         [4., 3., 1., 4.],
         [2., 4., 2., 0.]],

        [[0., 4., 3., 4.],
         [4., 1., 2., 0.],
         [1., 2., 2., 4.],
         [2., 3., 3., 4.]],

        [[3., 2., 0., 4.],
         [0., 4., 1., 4.],
         [0., 4., 3., 3.],
         [1., 0., 0., 0.]]])
```

```
# define convolutional layer
conv_layer = nn.Conv2d(
    in_channels=3,
    out_channels=1,
    kernel_size=3,
)
```

```
# init weight
conv_layer.weight.data = init_kernel_weight
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1., 1.],
          [1., 1., 1.],
          [1., 0., 0.]],

         [[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 0.]],

         [[1., 0., 1.],
          [1., 0., 1.],
          [0., 1., 1.]]]], requires_grad=True)
```

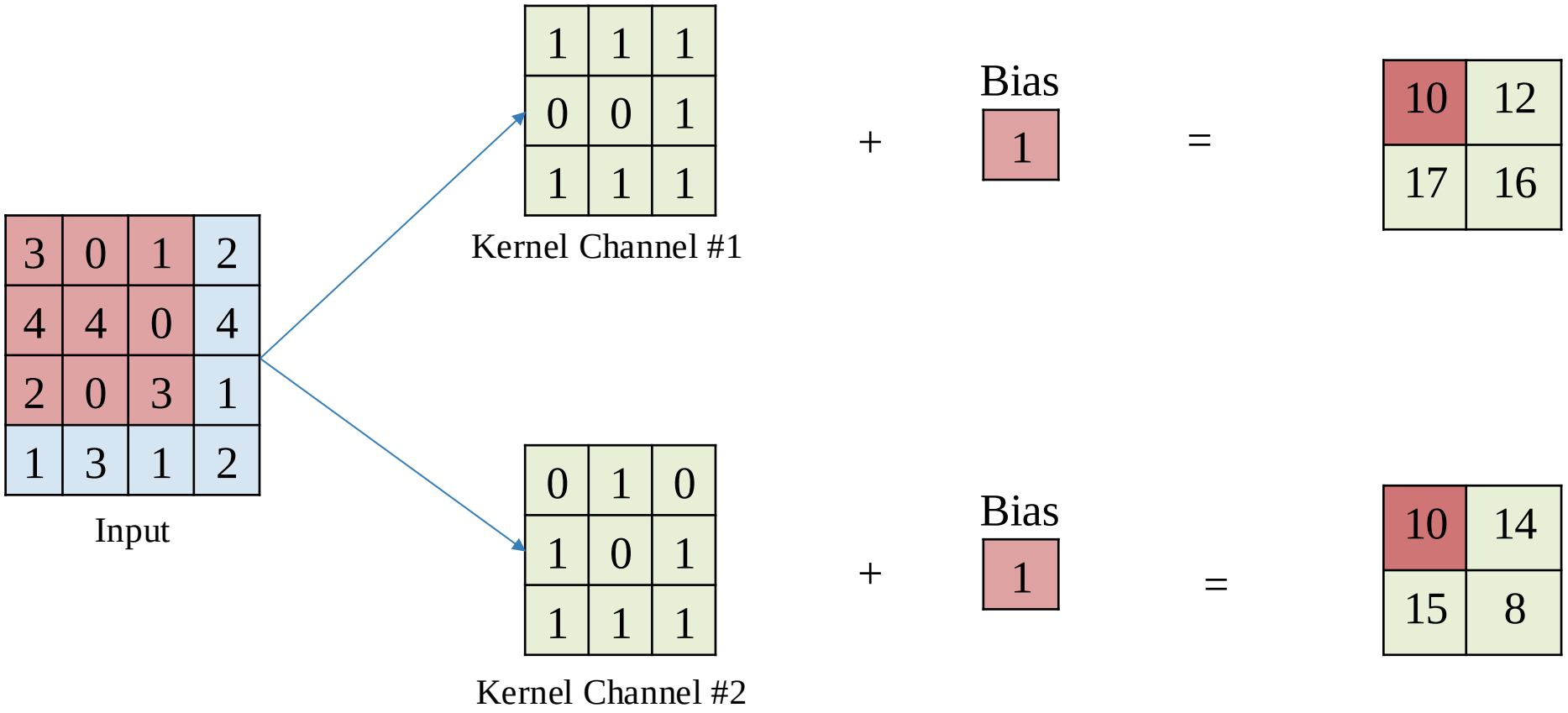
```
# init bias
conv_layer.bias =  nn.Parameter(
    torch.tensor([1], dtype=torch.float32)
)
conv_layer.bias
```

```
Parameter containing:
tensor([1.], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[43., 50.],
         [40., 50.]]], grad_fn=<SqueezeBackward1>)
```

Multiple Output Channels



Multiple Output Channels

```
input = torch.randint(5, (1, 4, 4), dtype=torch.float32)
input
```

```
tensor([[[3., 0., 1., 2.],
         [4., 4., 0., 4.],
         [2., 0., 3., 1.],
         [1., 3., 1., 2.]]])
```

```
# init weight
conv_layer.weight.data = init_kernel_weight
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1., 1.],
          [0., 0., 1.],
          [1., 1., 1.]]],


        [[[0., 1., 0.],
          [1., 0., 1.],
          [1., 1., 1.]]]], requires_grad=True)
```

```
input = torch.randint(5, (1, 4, 4), dtype=torch.float32)
input
```

```
tensor([[[3., 0., 1., 2.],
         [4., 4., 0., 4.],
         [2., 0., 3., 1.],
         [1., 3., 1., 2.]]])
```

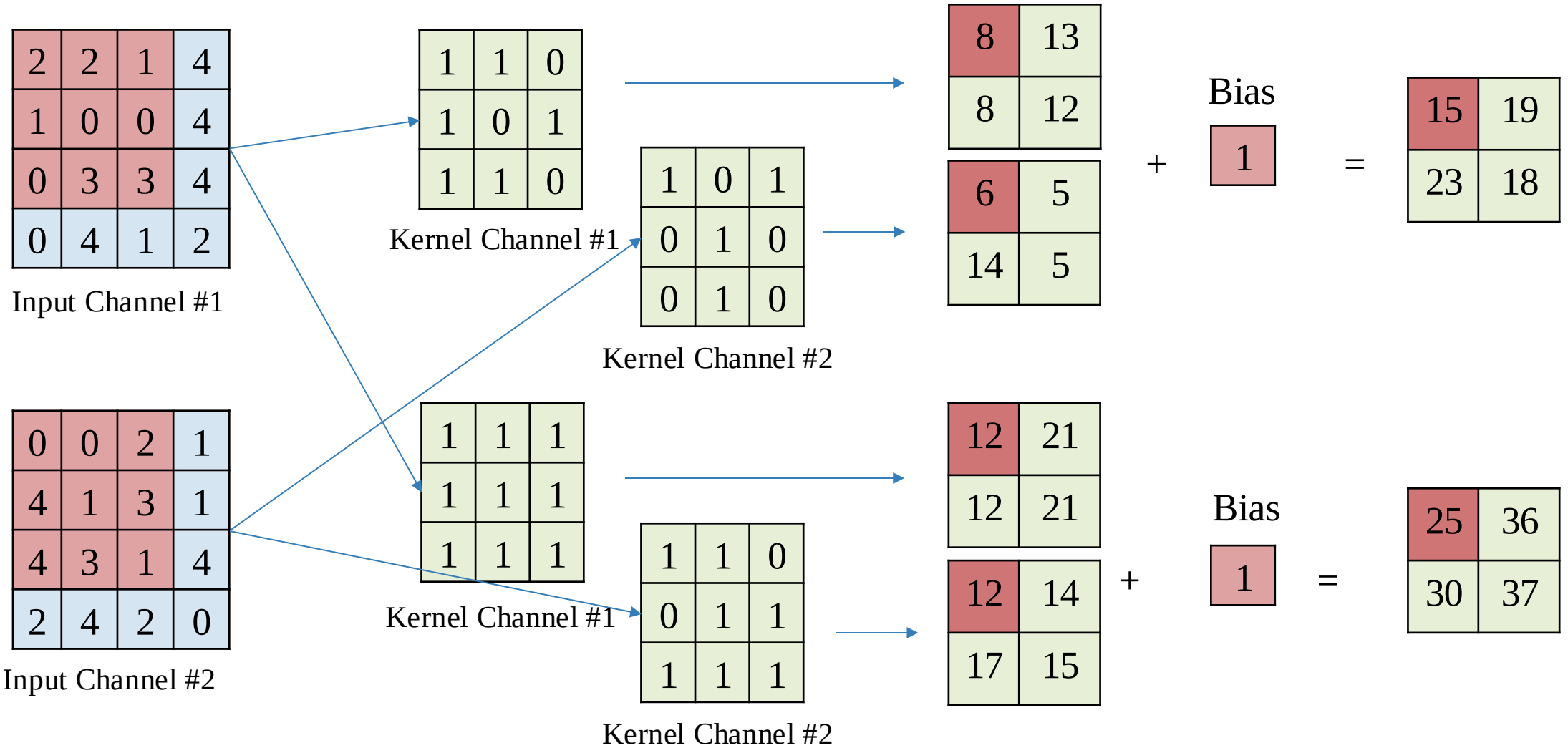
```
# init bias
conv_layer.bias =  nn.Parameter(
    torch.tensor([1, 1], dtype=torch.float32)
)
conv_layer.bias
```

```
Parameter containing:
tensor([1., 1.], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[10., 12.],
         [17., 16.]],

        [[10., 14.],
         [15.,  8.]]], grad_fn=<SqueezeBackward1>)
```

Multiple Input – Output Channels



Multiple Input – Output Channels

```
input = torch.randint(5, (2, 4, 4), dtype=torch.float32)
input
```

```
tensor([[[2., 2., 1., 4.],
         [1., 0., 0., 4.],
         [0., 3., 3., 4.],
         [0., 4., 1., 2.]],

        [[0., 0., 2., 1.],
         [4., 1., 3., 1.],
         [4., 3., 1., 4.],
         [2., 4., 2., 0.]]])
```

```
# define convolutional layer
conv_layer = nn.Conv2d(
    in_channels=2,
    out_channels=2,
    kernel_size=3,
)
```

```
# init weight
conv_layer.weight.data = init_kernel_weight
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1., 0.],
          [1., 0., 1.],
          [1., 1., 0.]],

         [[1., 0., 1.],
          [0., 1., 0.],
          [0., 1., 0.]]],


        [[[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]],

         [[1., 1., 0.],
          [0., 1., 1.],
          [1., 1., 1.]]]], requires_grad=Tru
```

```
# init bias
conv_layer.bias =  nn.Parameter(
    torch.tensor([1, 1], dtype=torch.float32)
)
conv_layer.bias
```

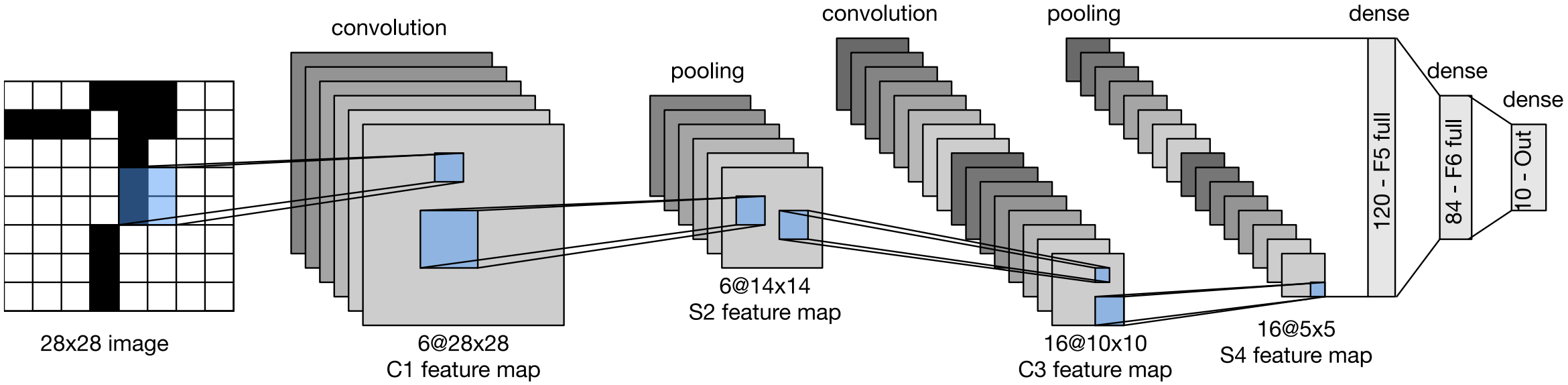
```
Parameter containing:
tensor([1., 1.], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[15., 19.],
         [23., 18.]],

        [[25., 36.],
         [30., 37.]]], grad_fn=<SqueezeBackward1>)
```

CNN

LeNet Model



CNN

LeNet Model

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

    def forward(self, inputs):
        outputs = self.conv1(inputs)
        outputs = self.avgpool1(outputs)
        outputs = F.relu(outputs)
        outputs = self.conv2(outputs)
        outputs = self.avgpool2(outputs)
        outputs = F.relu(outputs)
        outputs = self.flatten(outputs)
        outputs = self.fc_1(outputs)
        outputs = self.fc_2(outputs)
        outputs = self.fc_3(outputs)
        return outputs
```

AI VIET NAM
@aivietnam.edu.vn

Outline

SECTION 1

CNN

SECTION 2

Application

Classification

Pipeline

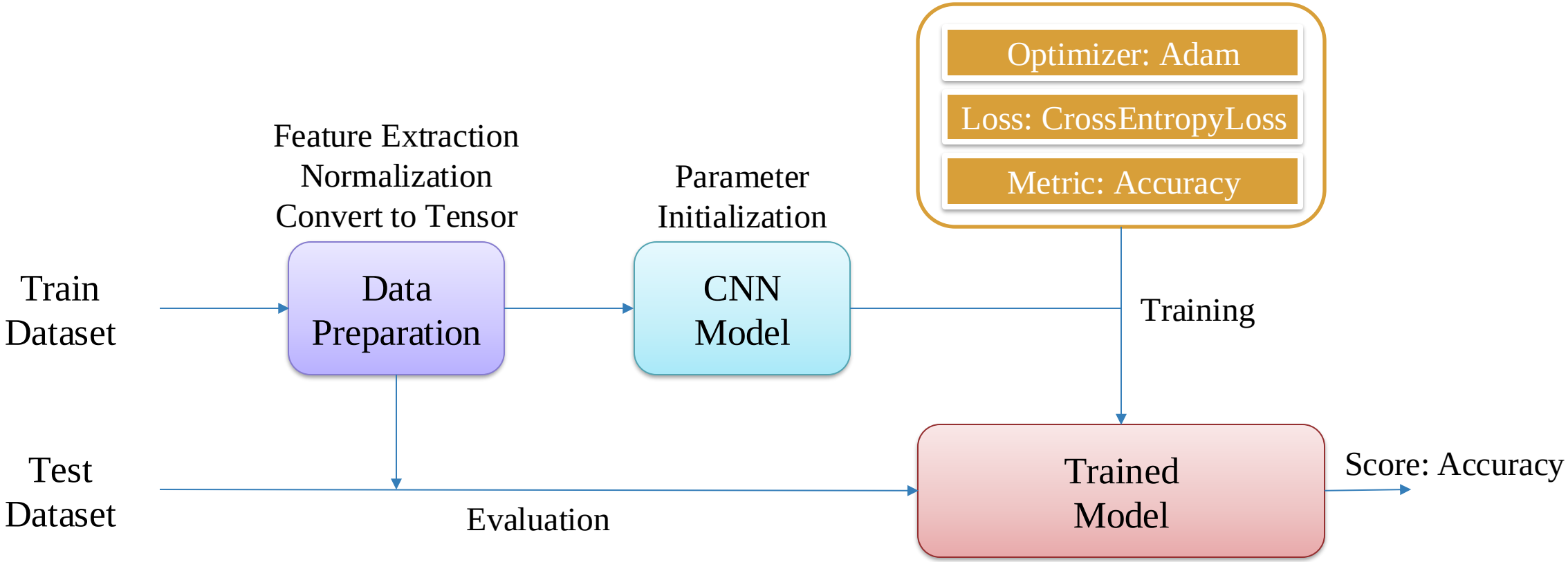


Image Classification

MNIST Dataset

- Images: 70.000
- Class: 10
- Image Size: 28 x 28

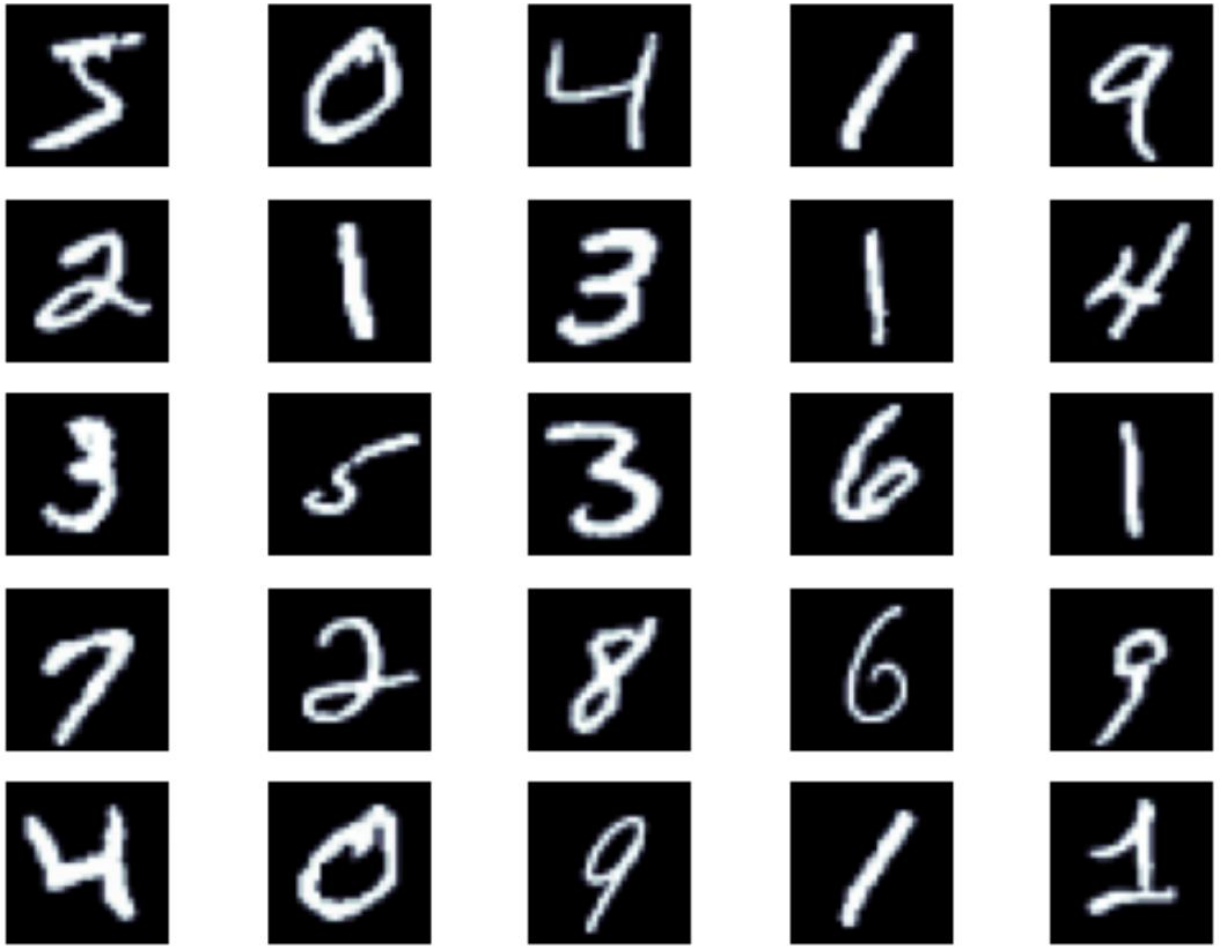


Image Classification

MNIST Dataset

- Load dataset

```
ROOT = './data'

train_data = datasets.MNIST(
    root=ROOT,
    train=True,
    download=True
)

test_data = datasets.MNIST(
    root=ROOT,
    train=False,
    download=True
    )
```

```
train_data.classes
```

```
['0 - zero',
 '1 - one',
 '2 - two',
 '3 - three',
 '4 - four',
 '5 - five',
 '6 - six',
 '7 - seven',
 '8 - eight',
 '9 - nine']
```

Image Classification

MNIST Dataset

- Preprocessing

```
VALID_RATIO = 0.9

n_train_examples = int(len(train_data) * VALID_RATIO)
n_valid_examples = len(train_data) - n_train_examples
```

```
train_data, valid_data = data.random_split(
    train_data,
    [n_train_examples, n_valid_examples]
)
```

```
# compute mean and std
mean = train_data.dataset.data.float().mean() / 255
std = train_data.dataset.data.float().std() / 255
mean, std
```

```
(tensor(0.1307), tensor(0.3081))
```

```
train_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])

test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])
```

```
train_data.dataset.transform = train_transforms
valid_data.dataset.transform = test_transforms
```

Image Classification

MNIST Dataset

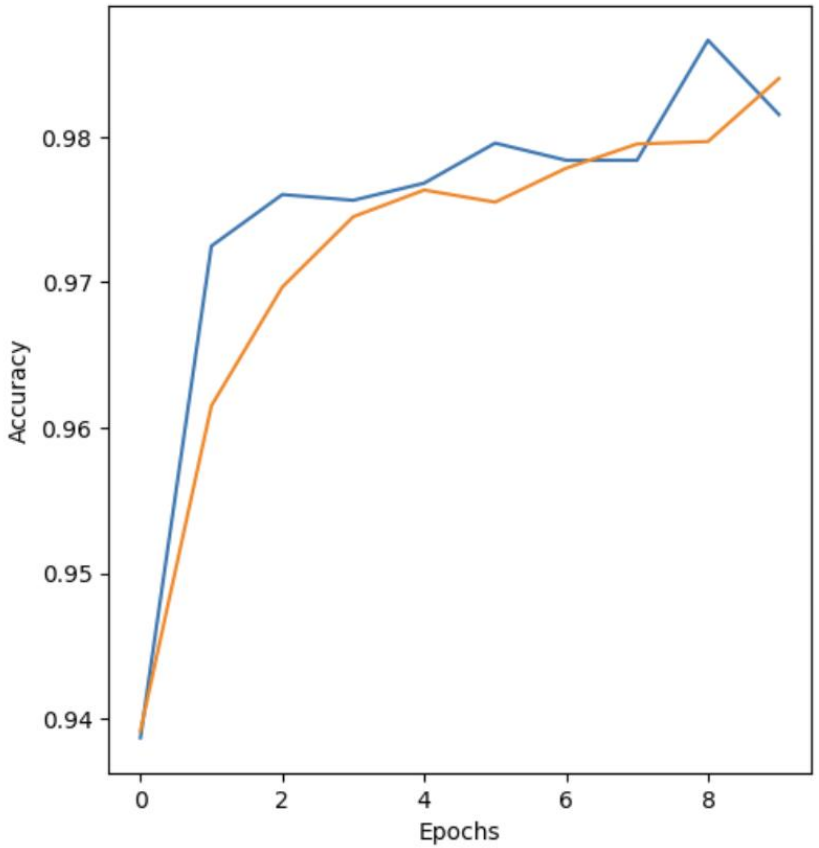
- Model

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

    def forward(self, inputs):
        outputs = self.conv1(inputs)
        outputs = self.avgpool1(outputs)
        outputs = F.relu(outputs)
        outputs = self.conv2(outputs)
        outputs = self.avgpool2(outputs)
        outputs = F.relu(outputs)
        outputs = self.flatten(outputs)
        outputs = self.fc_1(outputs)
        outputs = self.fc_2(outputs)
        outputs = self.fc_3(outputs)
        return outputs
```

Image Classification

MNIST Dataset

- Training (Test Set: 98%)



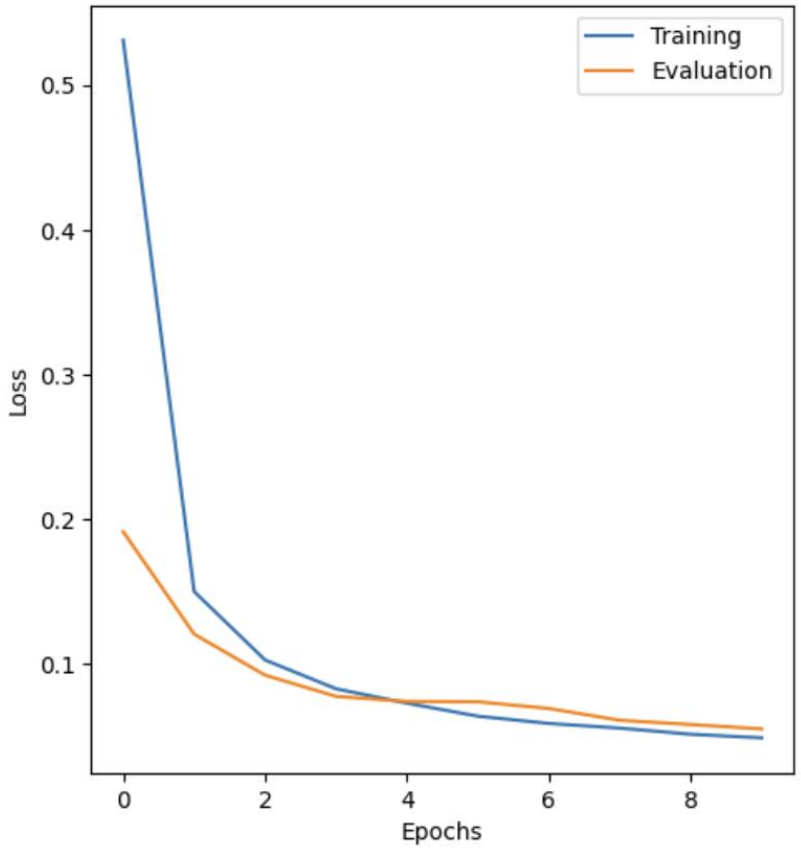
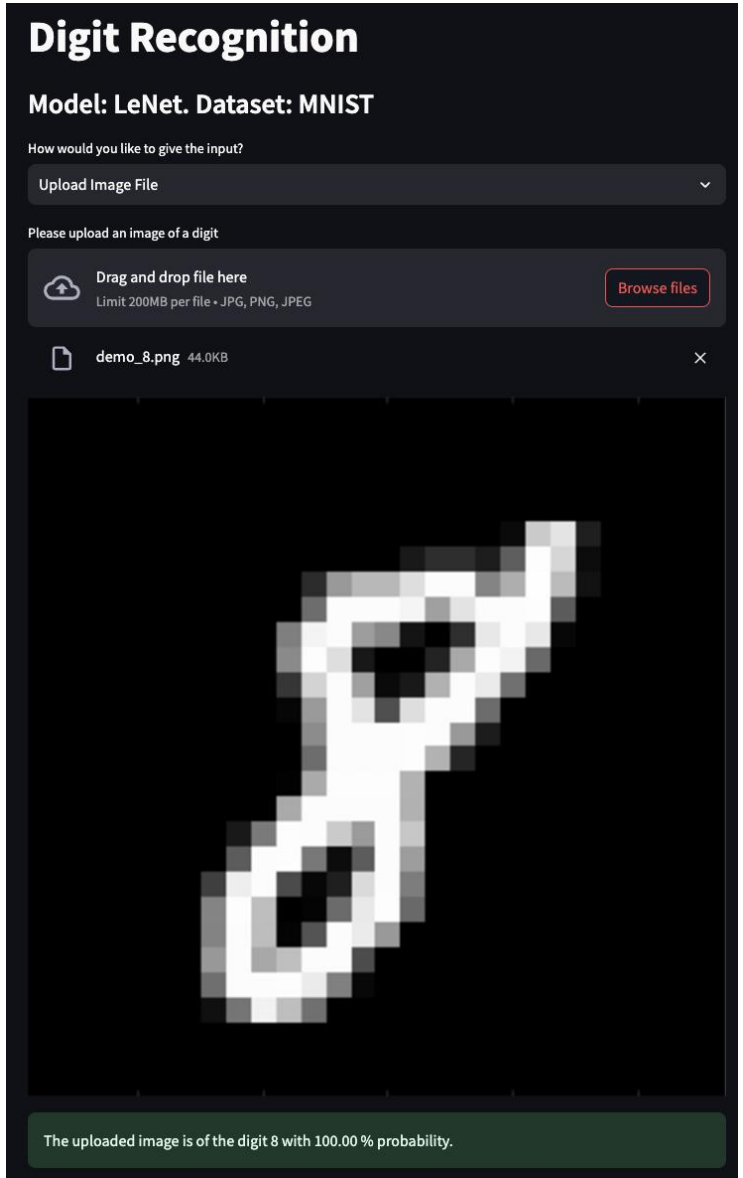


Image Classification

MNIST Dataset

- Deployment (Streamlit)




QUIZ TIME

Image Classification

MNIST Dataset

- Kernel Visualization

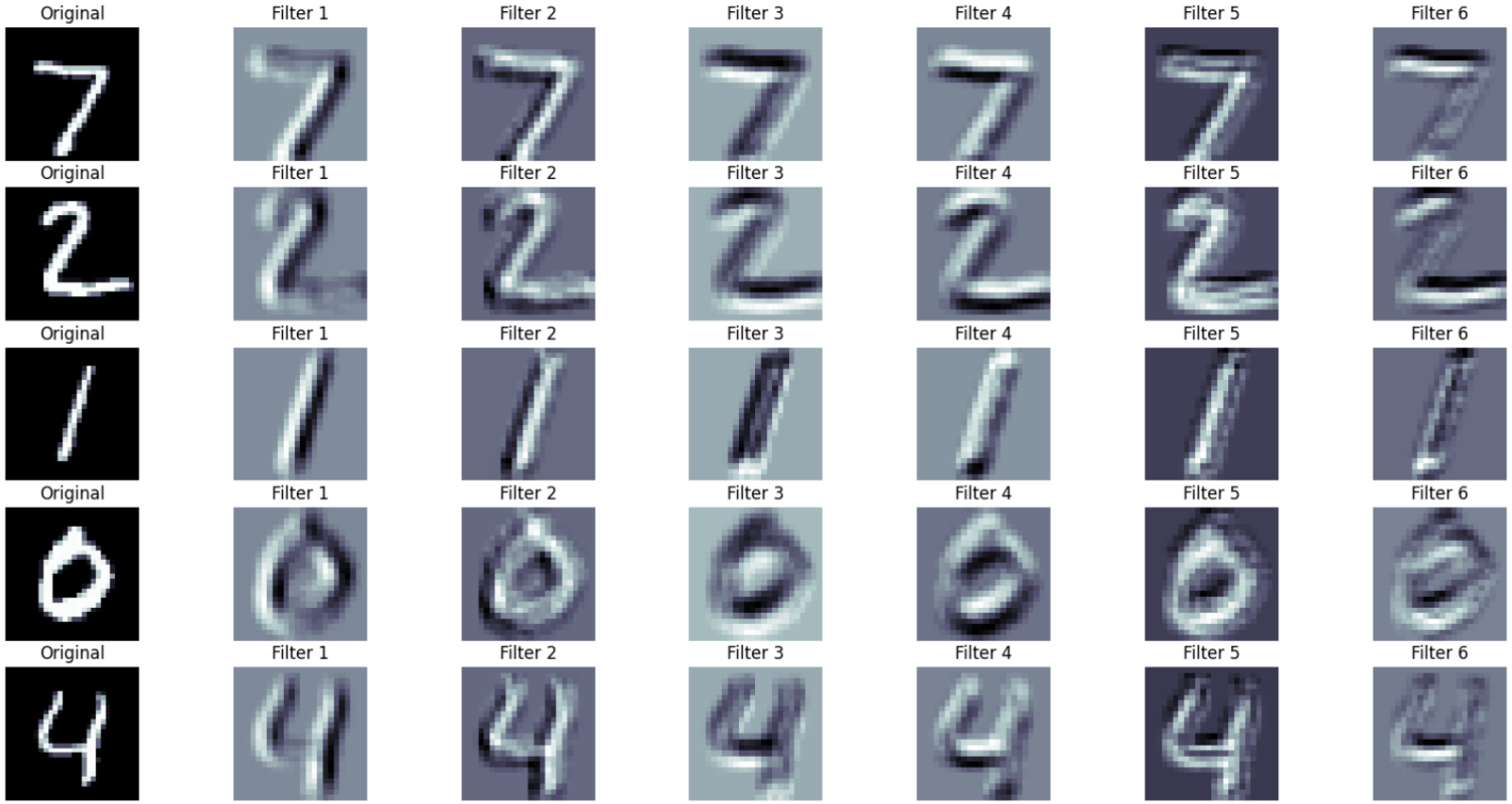


Image Classification

Cassava Leaf Disease Dataset

- Class: 5

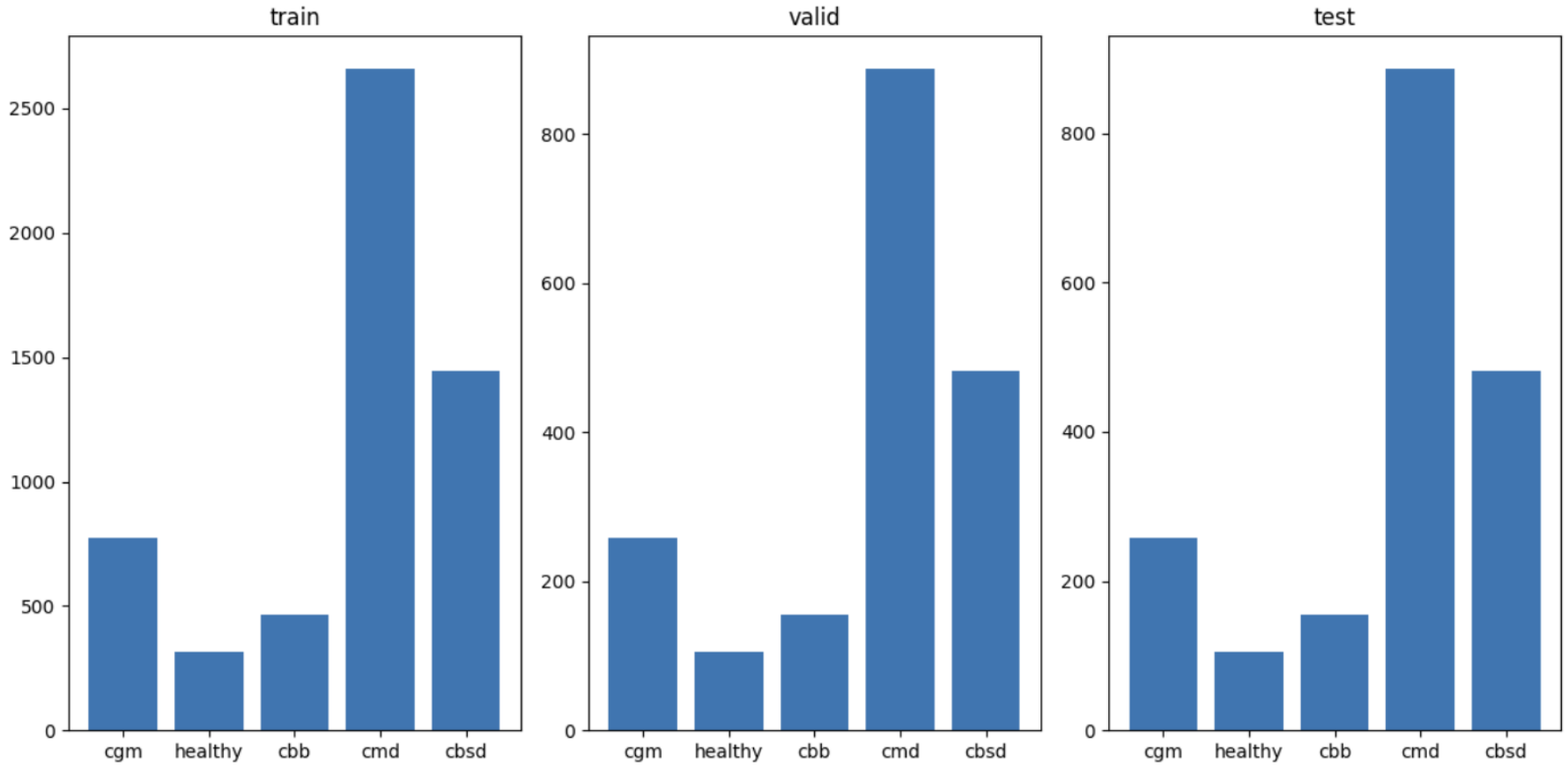


Image Classification

Cassava Leaf Disease Dataset

- Preprocessing


```
# load image from path
def loader(path):
    return Image.open(path)
```

```
img_size = 150

train_transforms = transforms.Compose([
    transforms.Resize((150, 150)),
    transforms.ToTensor(),
])
```

```
train_data = datasets.ImageFolder(
    root=data_paths['train'],
    loader=loader,
    transform=train_transforms
)
valid_data = datasets.ImageFolder(
    root=data_paths['valid'],
    transform=train_transforms
)
test_data = datasets.ImageFolder(
    root=data_paths['test'],
    transform=train_transforms
)
```

Image Classification

Cassava Leaf Disease Dataset

- Model

```
summary(lenet_model, (3, 150, 150))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 6, 150, 150]             456
         AvgPool2d-2           [-1, 6, 75, 75]               0
            Conv2d-3          [-1, 16, 71, 71]           2,416
         AvgPool2d-4          [-1, 16, 35, 35]               0
           Flatten-5               [-1, 19600]               0
            Linear-6                 [-1, 120]       2,352,120
            Linear-7                  [-1, 84]          10,164
            Linear-8                   [-1, 5]             425
================================================================
Total params: 2,365,581
Trainable params: 2,365,581
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 2.20
Params size (MB): 9.02
Estimated Total Size (MB): 11.48
----------------------------------------------------------------
```

```
class LeNetClassifier(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.conv1 = nn.Conv2d(
            in_channels=3, out_channels=6, kernel_size=5, padding='same'
        )
        self.avgpool1 = nn.AvgPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.avgpool2 = nn.AvgPool2d(kernel_size=2)
        self.flatten = nn.Flatten()
        self.fc_1 = nn.Linear(16 * 35 * 35, 120)
        self.fc_2 = nn.Linear(120, 84)
        self.fc_3 = nn.Linear(84, num_classes)

    def forward(self, inputs):
        outputs = self.conv1(inputs)
        outputs = self.avgpool1(outputs)
        outputs = F.relu(outputs)
        outputs = self.conv2(outputs)
        outputs = self.avgpool2(outputs)
        outputs = F.relu(outputs)
        outputs = self.flatten(outputs)
        outputs = self.fc_1(outputs)
        outputs = self.fc_2(outputs)
        outputs = self.fc_3(outputs)
        return outputs
```

Image Classification

Cassava Leaf Disease Dataset

- Training (Test Set: 53%)

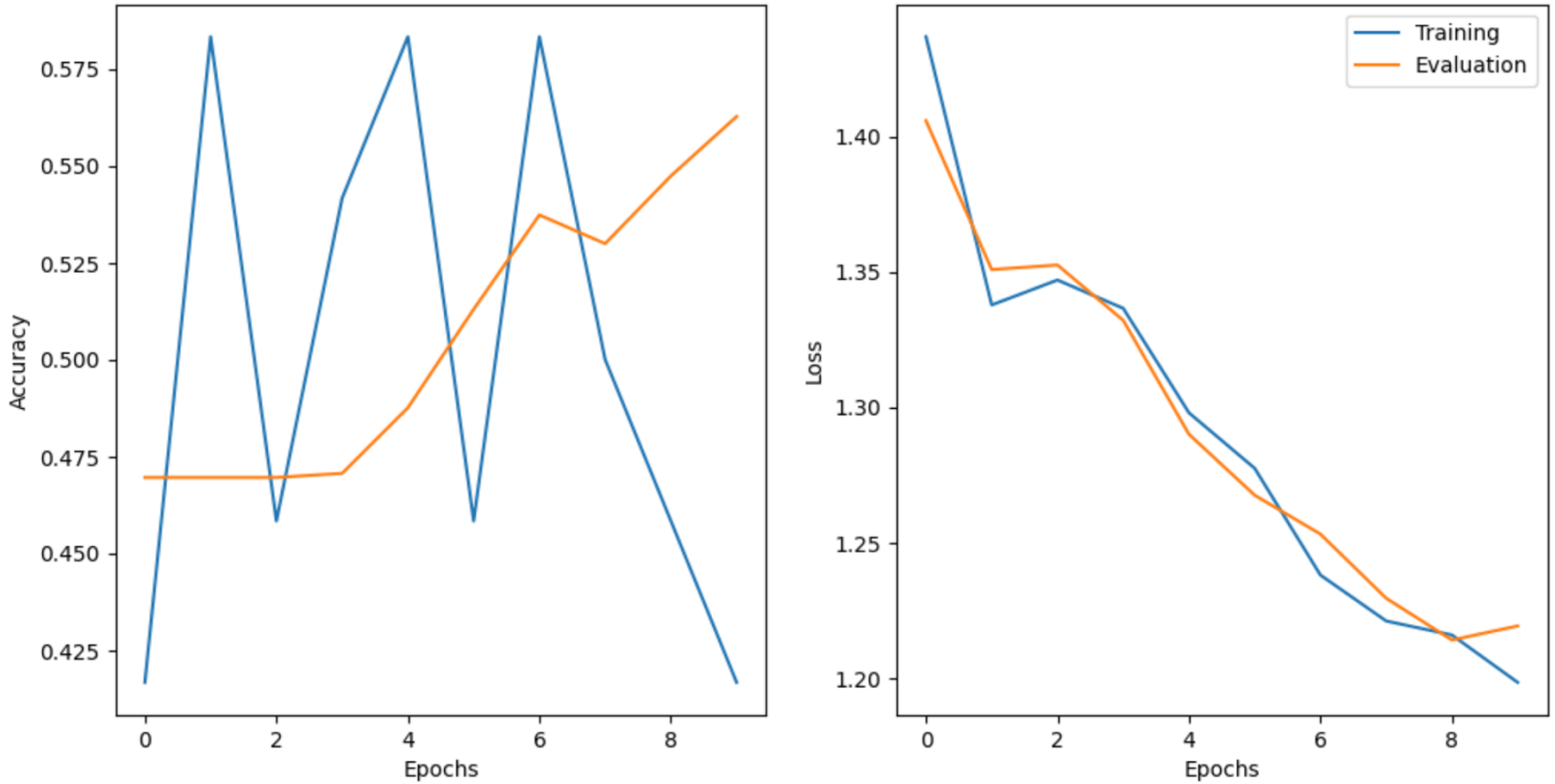
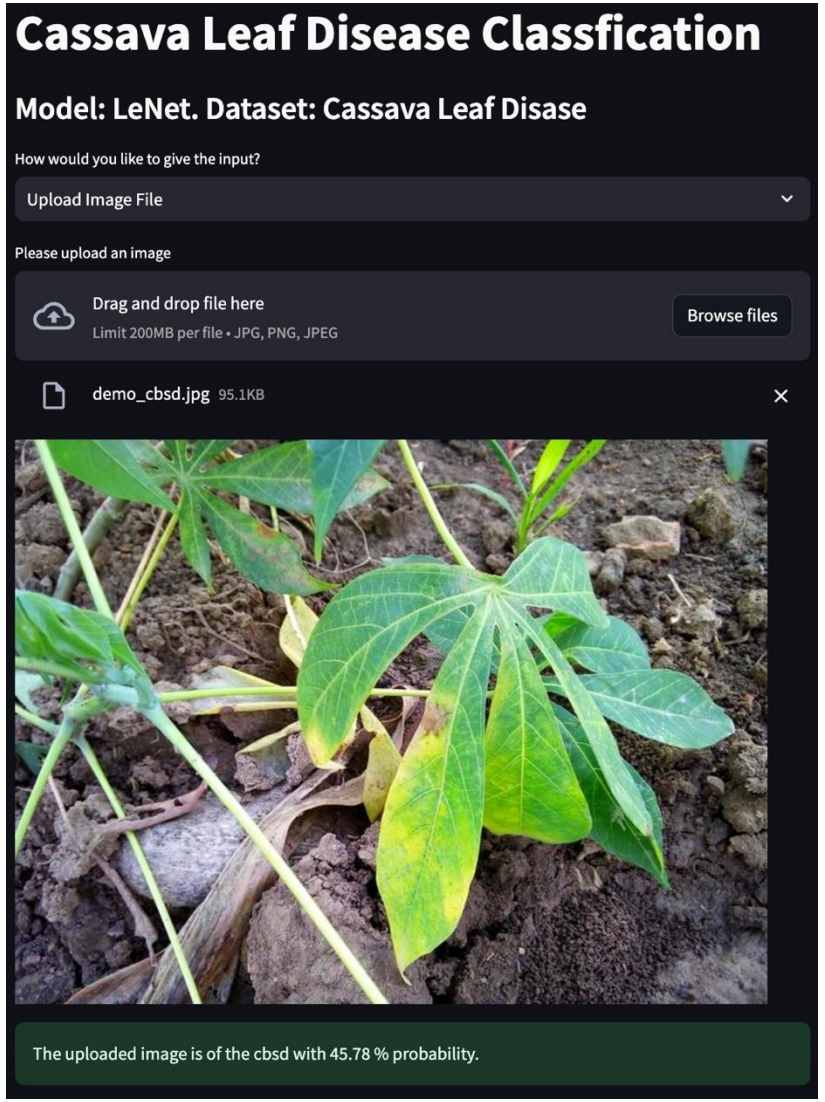


Image Classification

Cassava Leaf Disease Dataset

- Deployment (Streamlit)



Text Classification

NTC-SCV Dataset

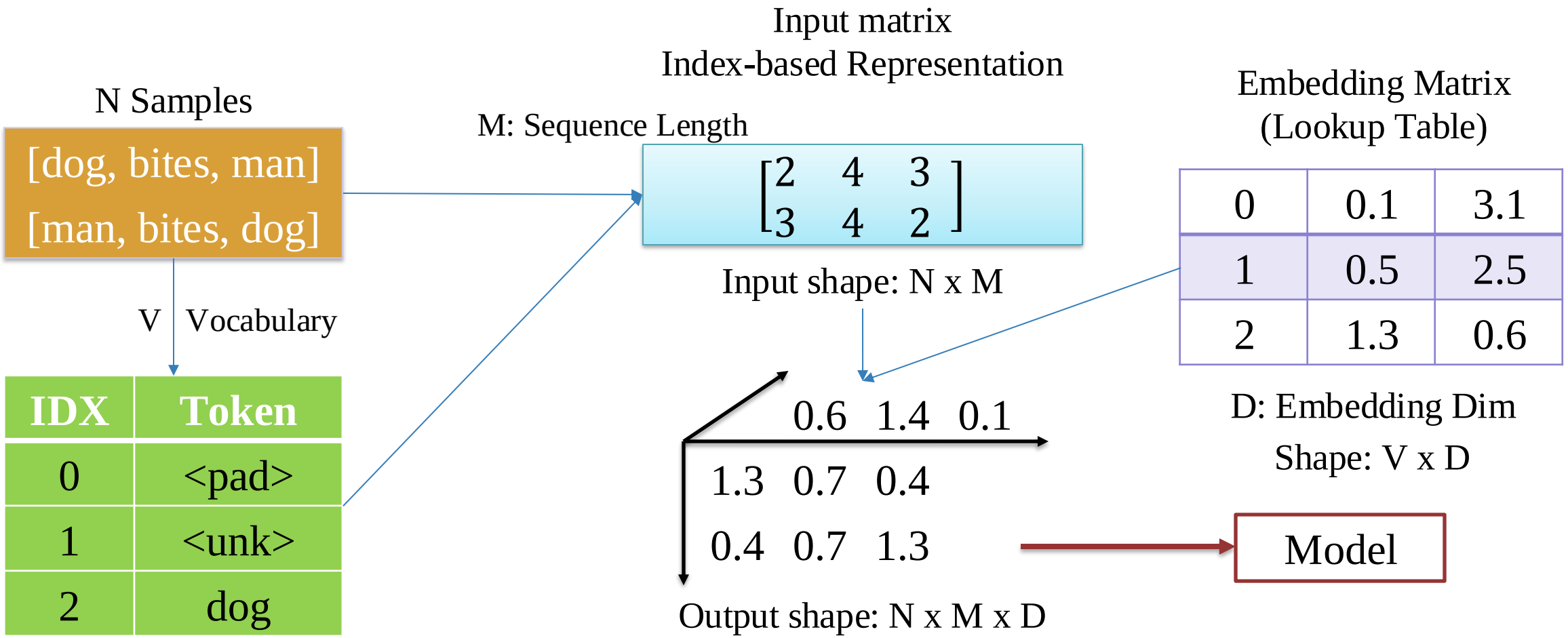
- Sentiment Analysis

Positive Example	Negative Example
Mình được 1 cô bạn giới_thiệu đến đây , tìm địa_chỉ khá dễ . Menu nước uống chất khỏi nói . Mình muốn cũng đc 8 loại nước ở đây , món nào cũng ngon và bổ_dưỡng cả .	Quán chế_biến đồ_ăn lâu , Cá_Sapa nướng ướp rất dở , sò Lông ko tươi , nước_chấm ko ngon\n Tóm_lại sẽ ko bao_giờ ghé nữa , ăn_dở mà uổng tiền
Mỗi lần thèm trà sữa là làm 1 ly . Quán dễ kiếm , không_gian lại rộng_rãi . Nhân_viên thì dễ_thương gần_gũi . Nói_chung thèm trà sữa là mình ghé Quán ở đây vì gần nhà .	Quán này thấy khá nhiều người bảo mình nên mình đã đi ăn thử , nhưng thực_sự ăn xong thấy không được như mong_đợi lắm .

Text Classification

NTC-SCV Dataset

- Data Representation

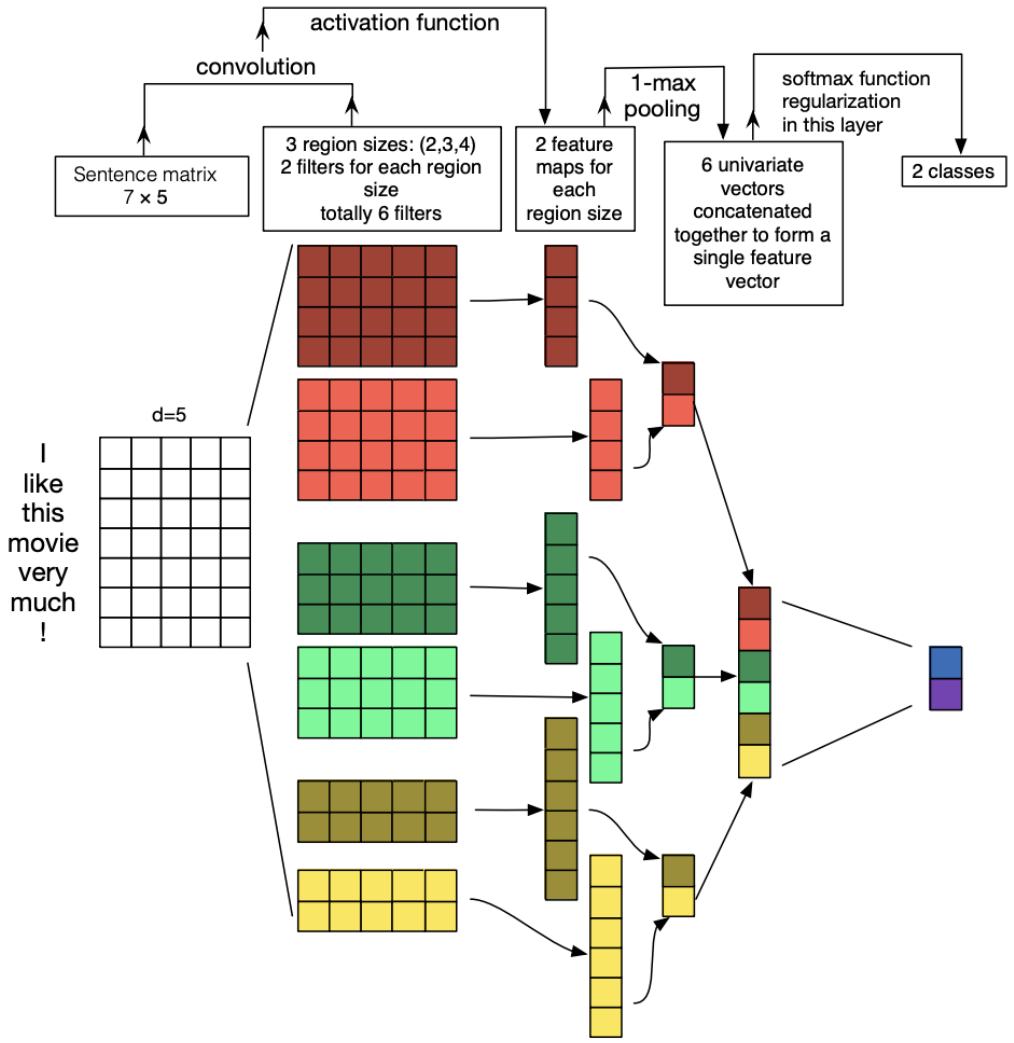


Text Classification

NTC-SCV Dataset

- TextCNN Model

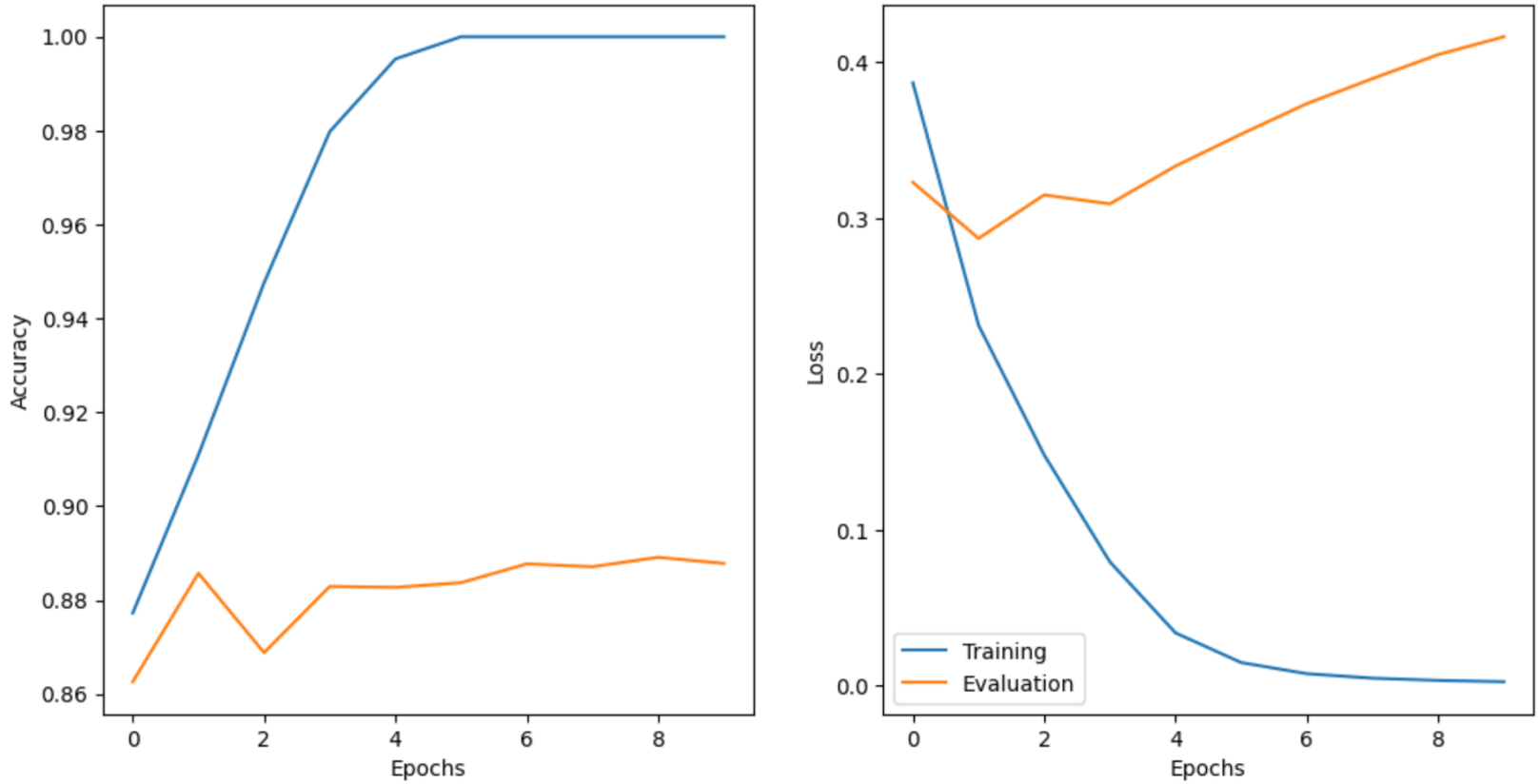
```
class TextCNN(nn.Module):
    def __init__(
        self,
        vocab_size, embedding_dim, kernel_sizes, num_filters, num_classes):
        super(TextCNN, self).__init__()

        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.kernel_sizes = kernel_sizes
        self.num_filters = num_filters
        self.num_classes = num_classes
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.conv = nn.ModuleList([
            nn.Conv1d(
                in_channels=embedding_dim,
                out_channels=num_filters,
                kernel_size=k,
                stride=1
            ) for k in kernel_sizes])
        self.fc = nn.Linear(len(kernel_sizes) * num_filters, num_classes)

    def forward(self, x):
        batch_size, sequence_length = x.shape
        x = self.embedding(x.T).transpose(1, 2)
        x = [F.relu(conv(x)) for conv in self.conv]
        x = [F.max_pool1d(c, c.size(-1)).squeeze(dim=-1) for c in x]
        x = torch.cat(x, dim=1)
        x = self.fc(x)
        return x
```



Text Classification

NTC-SCV Dataset

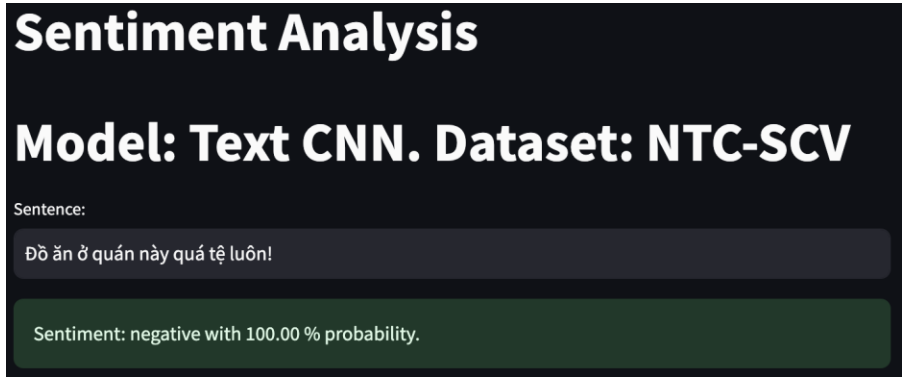
- Training (Test Set: 88%)



Text Classification

NTC-SCV Dataset

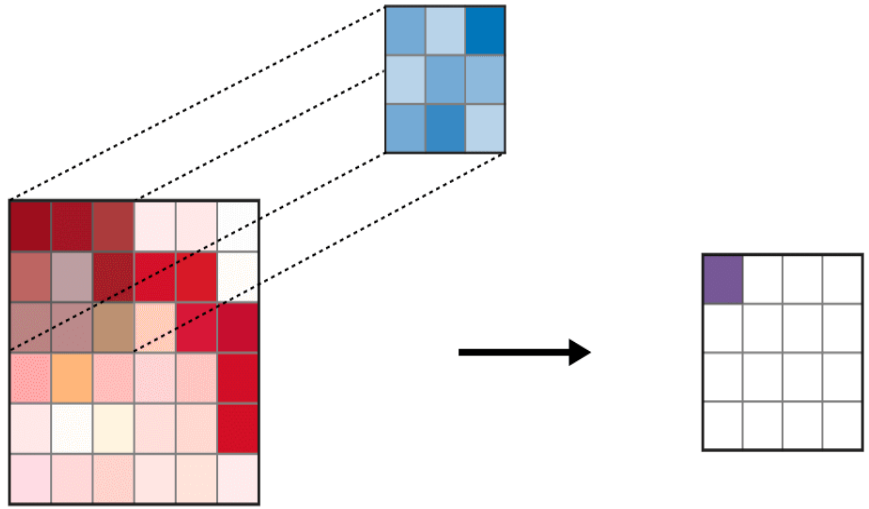
- Deployment (Streamlit)

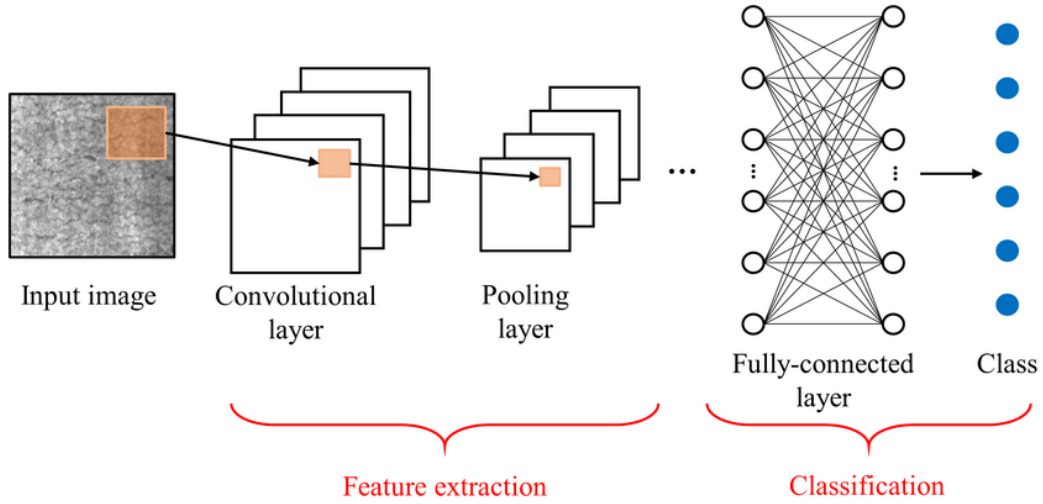


Objectives

CNN

- Convolution Layer, Pooling Layer
- Multiple Input – Output Channels
- LeNet Model





Classification

- Image Classification
- TextCNN Model
- Text Classification

Thanks!

Any questions?