7.3

Convolutional Neural Network Architectures

Part 1: The Main Ideas

Sebastian Raschka and the Lightning AI Team

Summary of the main concepts behind CNNs

- Sparse-connectivity: A single element in the feature map is connected to only a small patch of pixels (versus fully-connected layers in MLPs)

Summary of the main concepts behind CNNs

- Sparse-connectivity: A single element in the feature map is connected to only a small patch of pixels (versus fully-connected layers in MLPs)

- Parameter-sharing: The same weights are used for different patches of the input image.

Summary of the main concepts behind CNNs

- **Sparse-connectivity:** A single element in the feature map is connected to only a small patch of pixels (versus fully-connected layers in MLPs)
- **Parameter-sharing:** The same weights are used for different patches of the input image.
- **Many layers:** Combining extracted local patterns to global patterns

- Sparse-connectivity → Neighboring pixels are related
- Parameter-sharing → Same filters work in different parts of the image

Summary of the main concepts behind CNNs

- Sparse-connectivity → Neighboring pixels are related
- Parameter-sharing → Same filters work in different parts of the image

"Inductive biases" that help CNNs learn more quickly
and generalize better
(compared to fully-connected networks)

- Sparse-connectivity
- Parameter-sharing

Nice side effect: convolutional layers are small!

```python
class PyTorchMLP(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(3*224*224, 10_000),
            torch.nn.ReLU(),

            # 2nd hidden layer
            torch.nn.Linear(10_000, 1_000),
            torch.nn.ReLU(),

            # 3rd hidden layer
            torch.nn.Linear(1_000, 100),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(100, 10),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

```python
class PyTorchMLP(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(3*224*224, 10_000),
            torch.nn.ReLU(),

            # 2nd hidden layer
            torch.nn.Linear(10_000, 1_000),
            torch.nn.ReLU(),

            # 3rd hidden layer
            torch.nn.Linear(1_000, 100),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(100, 10),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

$3*224*224 * 10{,}000 + 10{,}000 = 1{,}505{,}290{,}000$

```python
class PyTorchMLP(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(3*224*224, 10_000),
            torch.nn.ReLU(),

            # 2nd hidden layer
            torch.nn.Linear(10_000, 1_000),
            torch.nn.ReLU(),

            # 3rd hidden layer
            torch.nn.Linear(1_000, 100),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(100, 10),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

3*224*224 * 10,000 + 10,000 = 1,505,290,000

10,001,000

100,100

1,010

1,515,392,110 parameters!!!

```python
class PyTorchMLP(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(3*224*224, 10_000),
            torch.nn.ReLU(),

            # 2nd hidden layer
            torch.nn.Linear(10_000, 1_000),
            torch.nn.ReLU(),

            # 3rd hidden layer
            torch.nn.Linear(1_000, 100),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(100, 10),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

3*224*224 * 10,000 + 10,000 = 1,505,290,000

10,001,000

100,100

1,010

1,515,392,110 parameters!!!

```python
size = 0.
for name, param in mlp.named_parameters():
    size += sys.getsizeof(param.storage()) / 1_024**3
print(f"Model size: {size:.2f} GB")
```

```
Model size: 5.65 GB
```

```
Sequential(
24, 10_000),  →  3*224*224 * 10,000 + 10,000 = 1,505,290,000

 1_000),  →  10,001,000

100),  →  100,100

),  →  1,010

dim=1)
```

1,515,392,110 parameters!!!

```
ters():
rage()) / 1_024**3
```

```
class PyTorchCNN(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.cnn_layers = torch.nn.Sequential(
            torch.nn.Conv2d(3, 8, kernel_size=5, stride=2),  →  3*5*5*8 + 8 = 608
            torch.nn.ReLU(),

            torch.nn.Conv2d(8, 24, kernel_size=5, stride=2),  →  4,824
            torch.nn.ReLU(),

            torch.nn.Conv2d(24, 32, kernel_size=3, stride=2),  →  6,944
            torch.nn.ReLU(),

            torch.nn.Conv2d(32, 48, kernel_size=3, stride=2),  →  13,872
            torch.nn.ReLU(),
        )

        self.fc_layers = torch.nn.Sequential(
            torch.nn.Linear(48*12*12, 200),  →  1,382,600
            torch.nn.ReLU(),
            torch.nn.Linear(200, 10),  →  2,010
        )

    def forward(self, x):

        x = self.cnn_layers(x)
        x = torch.flatten(x, start_dim=1)
        logits = self.fc_layers(x)
        return logits
```

1,410,858 parameters

```
size = 0.
for name, param in cnn.named_parameters():
    size += sys.getsizeof(param.storage()) / 1_024**3
print(f"Model size: {size:.3f} GB")
```

```
Model size: 0.016 GB
```

Next: CNNs And Their Inception(s)