7.1

Working With Images

Part 2: Image Data And Its Challenges

Sebastian Raschka and the Lightning AI Team

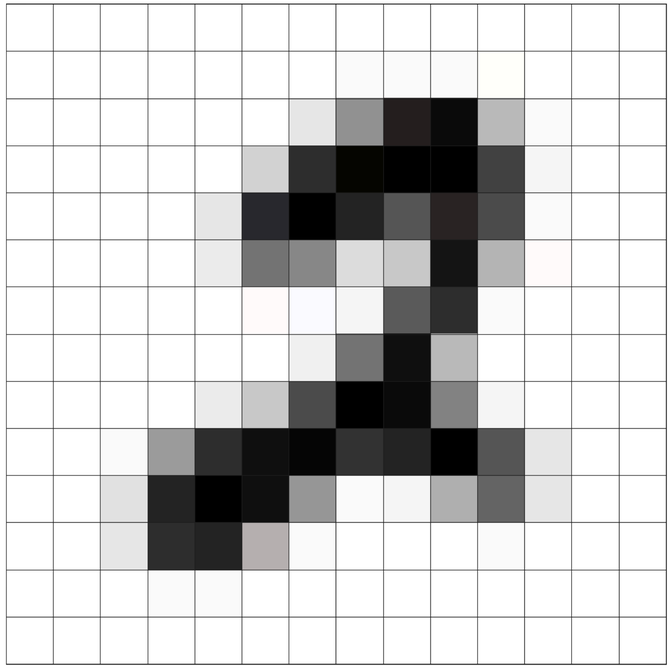
Let's start with a question!

Can you use a multilayer perceptron on image data?

Let's start with a question!

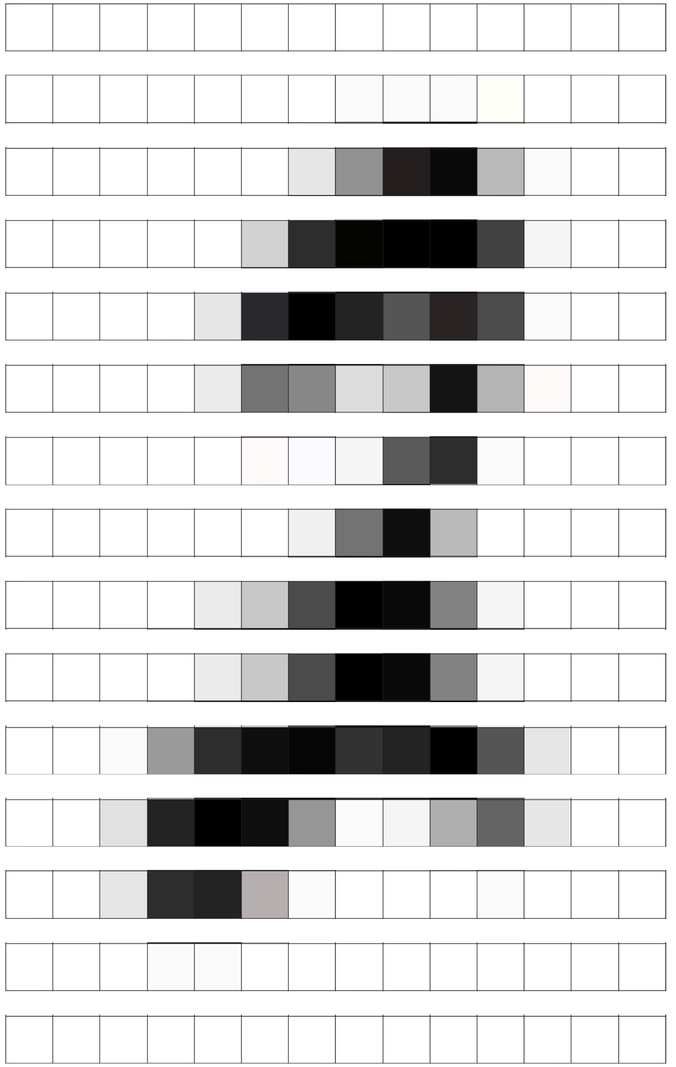
Can you use a multilayer perceptron on image data?

YES!

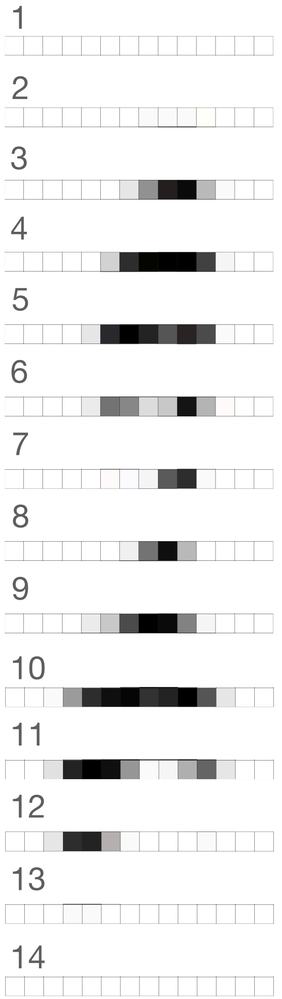


14x14-dimensional image

Consider the rows in this image

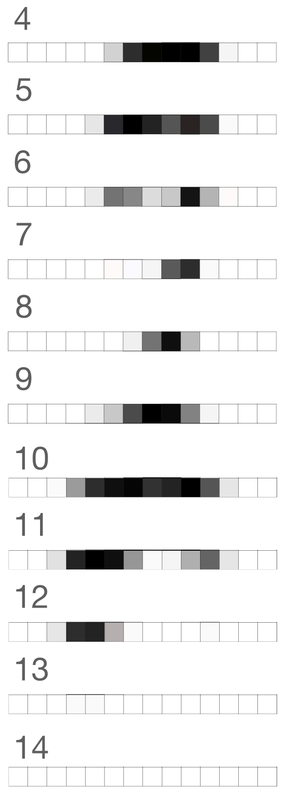


Let's shrink it to fit it better on the slide

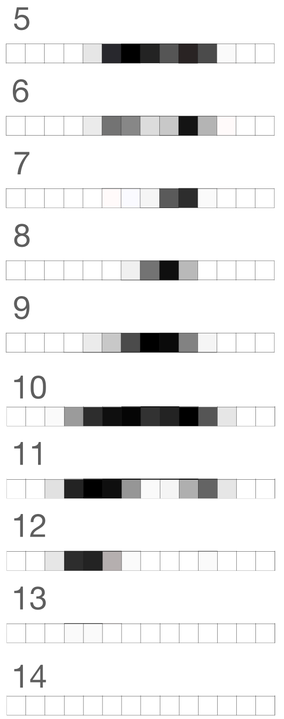


Concatenate the rows

Concatenate the rows



Concatenate the rows

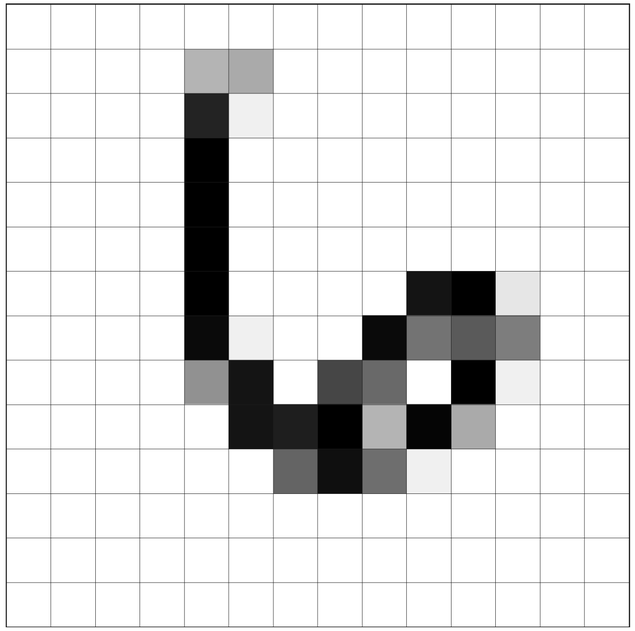


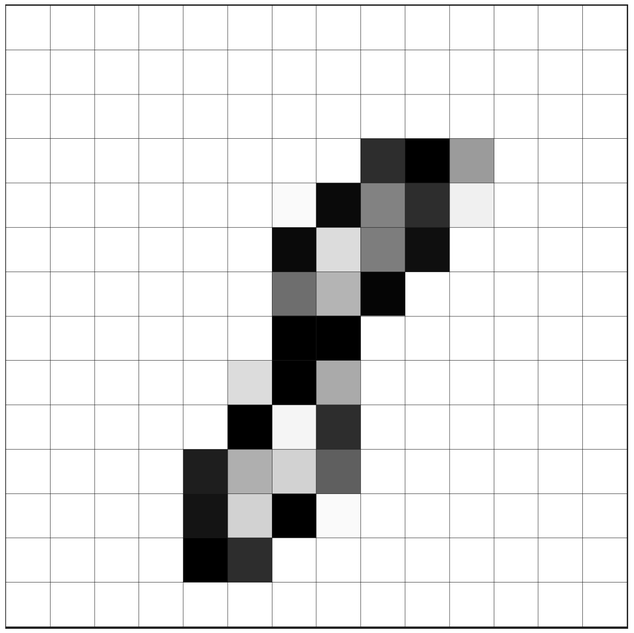
We get a 196-dimensional row vector

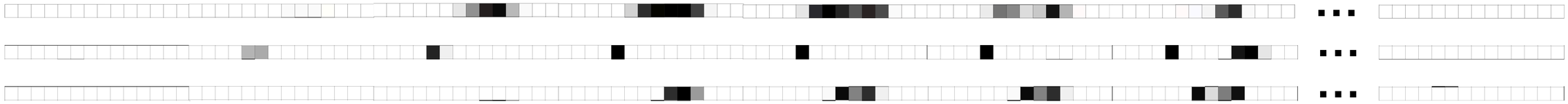
14×14 pixels $= 196$ pixels

We get a 196-dimensional row vector

This is a single training example

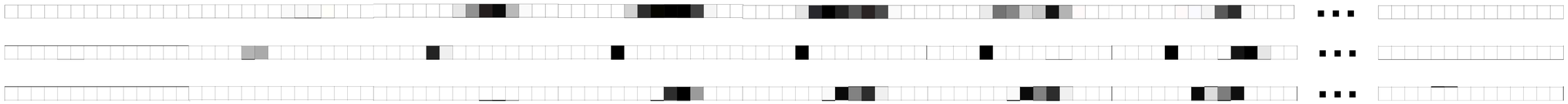




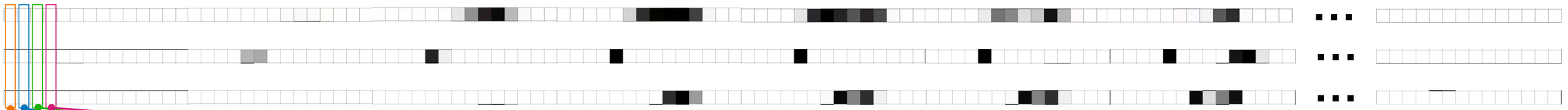


We now have 3 training examples

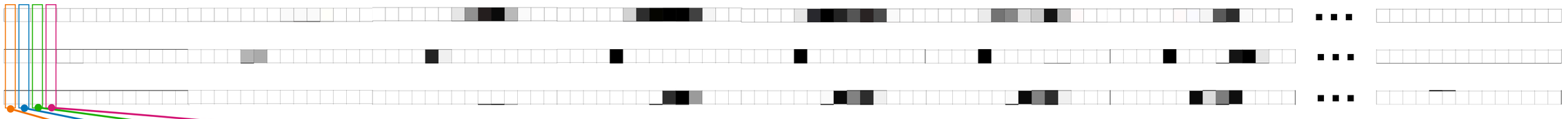
This is now a tabular dataset



Sepal length	Sepal width	Petal length	Petal length
5.1	3.5	1.4	0.3
4.9	3	1.4	0.2
5.9	3	5.1	1.8
...	...	...	...



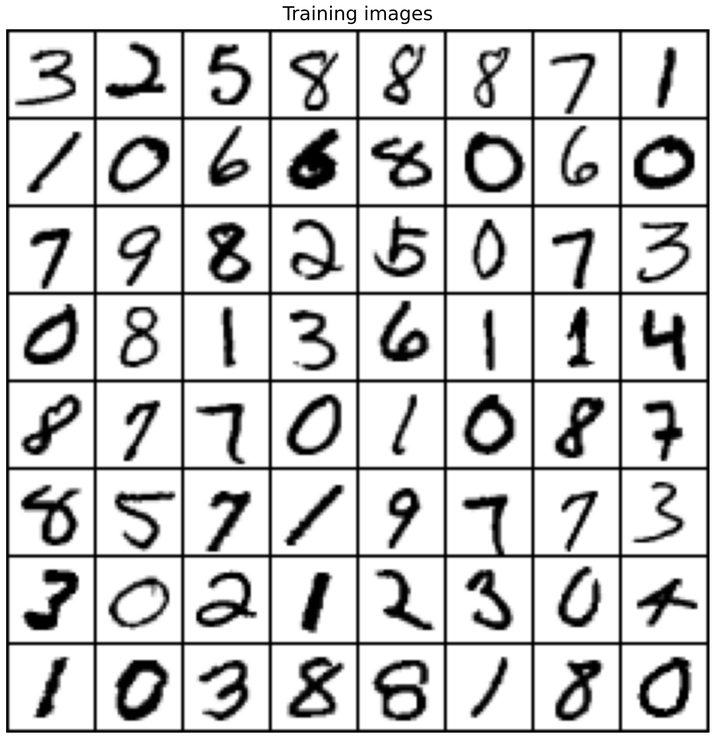
Position 1	Position 2	Position 3	Position 4	...	Position 196
0	0	0	0		0
0	0	0	0		0
0	0	0	0		0
...	...	...	...		...



Unnormalized pixel values range from 0 to 255

Position 1	Position 2	Position 3	Position 4	...	Position 196
0	0	0	0		0
0	0	0	0		0
0	0	0	0		0
...	...	...	...		...

We have already seen this in Unit 4!




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

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 50),
            torch.nn.ReLU(),
            # 2nd hidden layer
            torch.nn.Linear(50, 25),
            torch.nn.ReLU(),
            # output layer
            torch.nn.Linear(25, num_classes),
        )

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

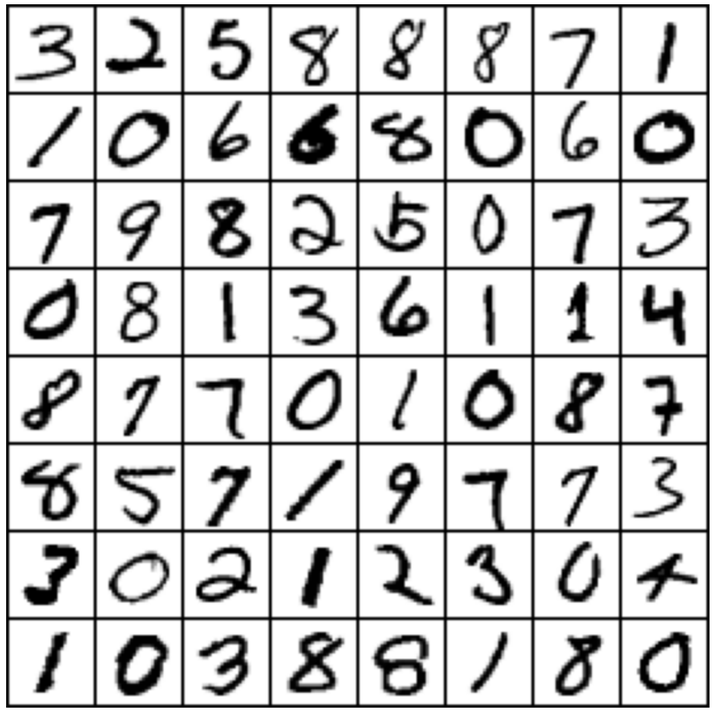
```python
train_acc = compute_accuracy(model, train_loader)
val_acc = compute_accuracy(model, val_loader)
test_acc = compute_accuracy(model, test_loader)

print(f"Train Acc {train_acc*100:.2f}%")
print(f"Val Acc {val_acc*100:.2f}%")
print(f"Test Acc {test_acc*100:.2f}%")
```

```
Train Acc 97.18%
Val Acc 96.00%
Test Acc 96.34%
```

We have already seen this in Unit 4!

Training images



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

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 50),
            torch.nn.ReLU(),
            # 2nd hidden layer
            torch.nn.Linear(50, 25),
            torch.nn.ReLU(),
            # output layer
            torch.nn.Linear(25, num_classes),
        )

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

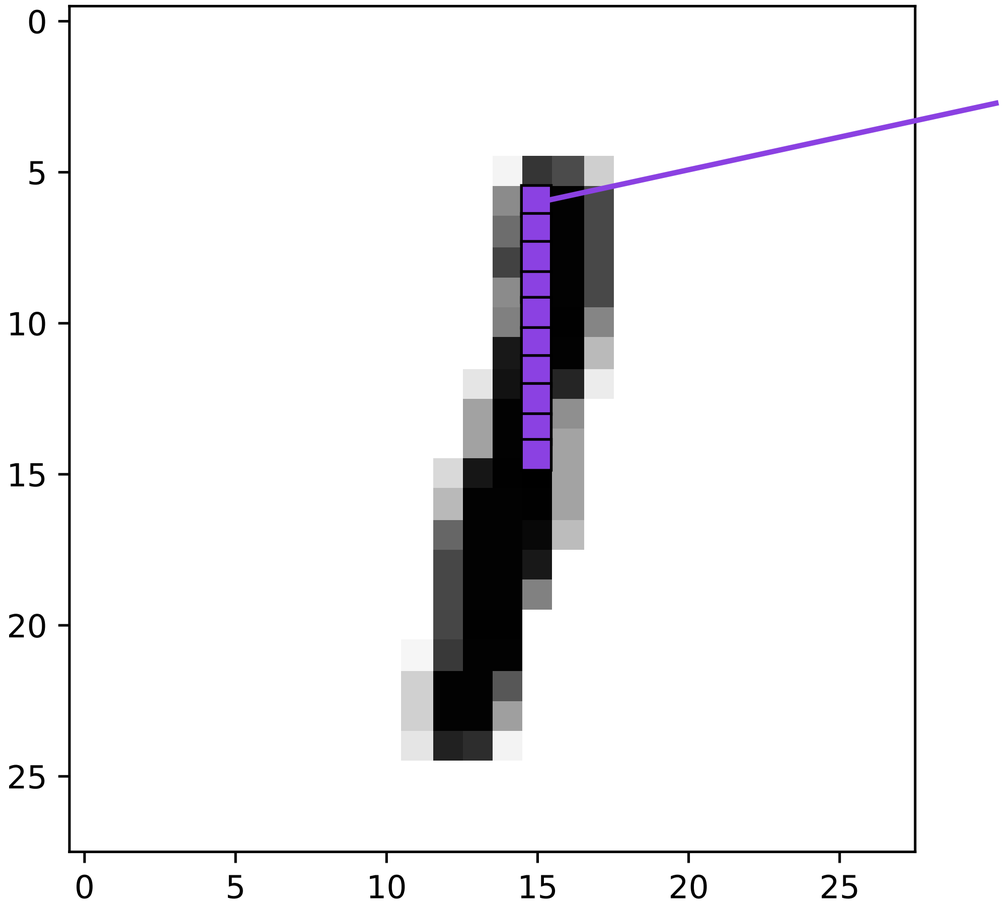
Reshapes the 28x28 images to 784-dimensional vectors

```python
train_acc = compute_accuracy(model, train_loader)
val_acc = compute_accuracy(model, val_loader)
test_acc = compute_accuracy(model, test_loader)

print(f"Train Acc {train_acc*100:.2f}%")
print(f"Val Acc {val_acc*100:.2f}%")
print(f"Test Acc {test_acc*100:.2f}%")
```

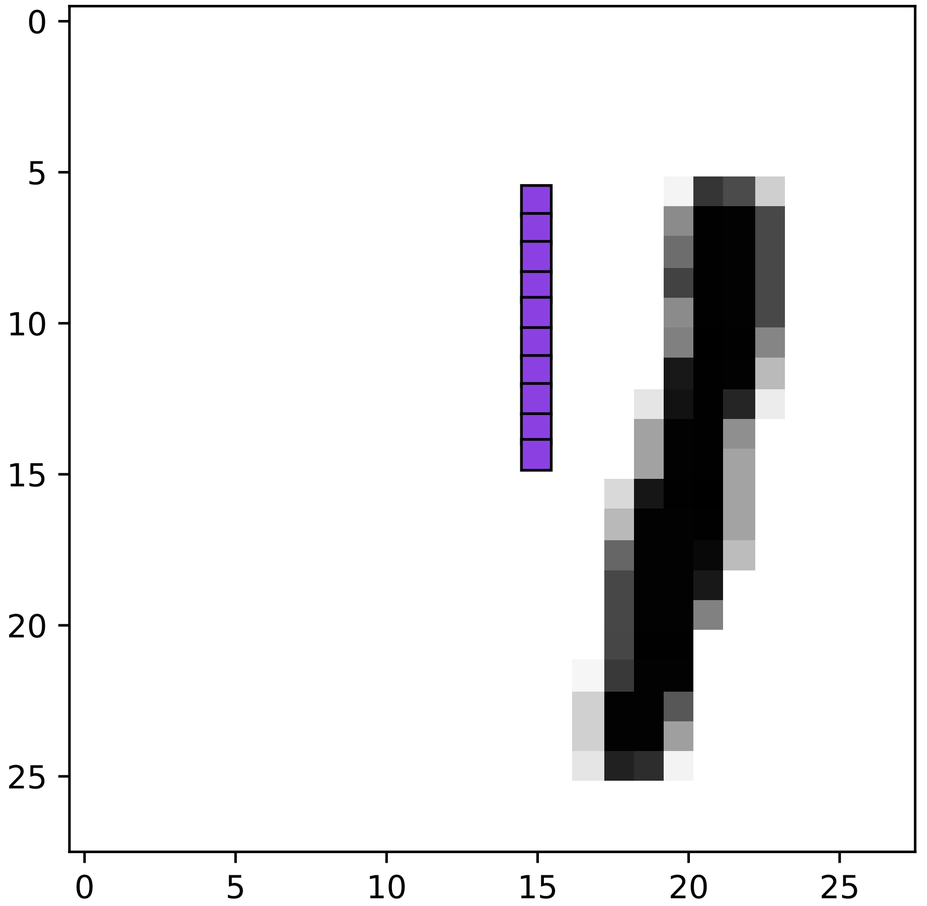
```
Train Acc 97.18%
Val Acc 96.00%
Test Acc 96.34%
```

A potential problem with this approach



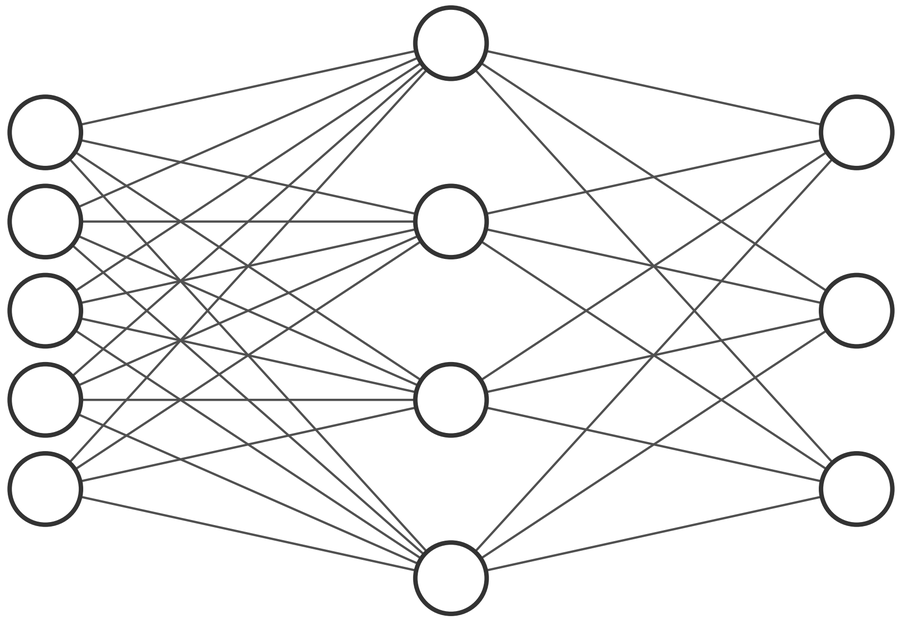
Suppose the MLP neural network relies on these pixels

A potential problem with this approach

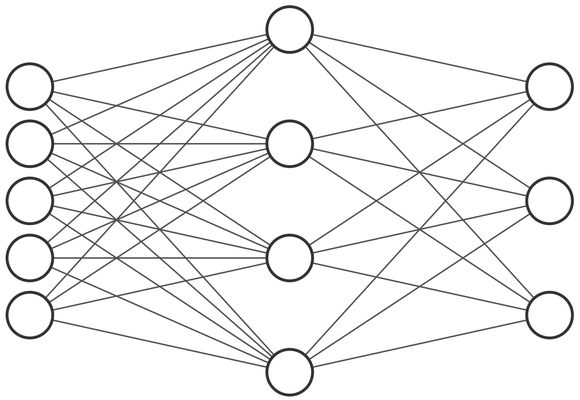


An MLP will not recognize the digit if it is in slightly different position

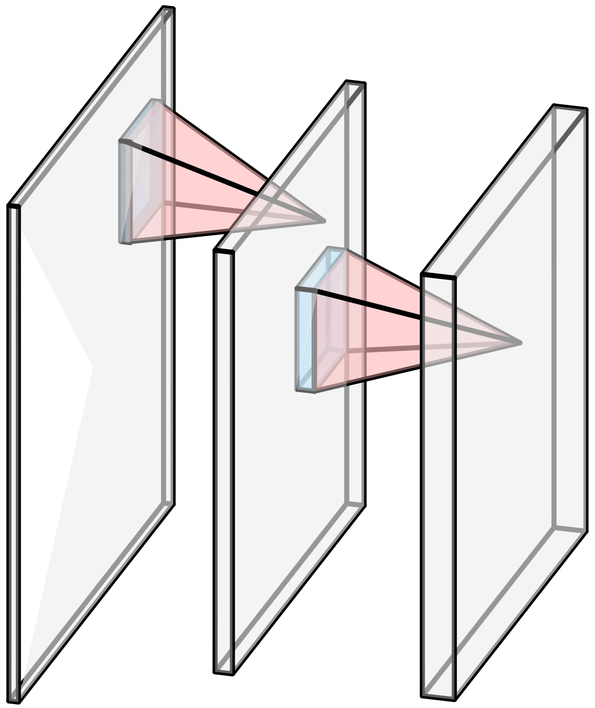
Multilayer perceptrons: Independence among features



Multilayer perceptrons:
Independence among features



Convolutional networks:
Take locality into account



Next: The Convolutional Network Architecture