Handling images with PyTorch

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

Clouds dataset

[1] https://www.kaggle.com/competitions/cloud-type-classification2/data

What is an image?

- Image consists of pixels ("picture elements")
- Each pixel contains color information
- Grayscale images: integer in 0 - 255
  - 30:

- Color images: three integers, one for each color channel (Red, Green, Blue)
  - RGB = (52, 171, 235):

Loading images to PyTorch

Desired directory structure:

```
clouds_train
  - cumulus
    - 75cbf18.jpg
    - ...
  - cumulonimbus
  - ...
clouds_test
  - cumulus
  - cumulonimbus
  - ...
```

- Main folders: `clouds_train` and `clouds_test`
- Inside each main folder: one folder per category
- Inside each class folder: image files

Loading images to PyTorch

```
from torchvision.datasets import ImageFolder
from torchvision import transforms

train_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((128, 128)),
])

dataset_train = ImageFolder(
    "data/clouds_train",
    transform=train_transforms,
)
```

- Define transformations:
  - Parse to tensor
  - Resize to 128×128
- Create dataset passing:
  - Path to data
  - Predefined transformations

Displaying images

```
dataloader_train = DataLoader(
    dataset_train,
    shuffle=True,
    batch_size=1,
)

image, label = next(iter(dataloader_train))
print(image.shape)
```

```
torch.Size([1, 3, 128, 128])
```

```
image = image.squeeze().permute(1, 2, 0)
print(image.shape)
```

```
torch.Size([128, 128, 3])
```

```
import matplotlib.pyplot as plt
plt.imshow(image)
plt.show()
```

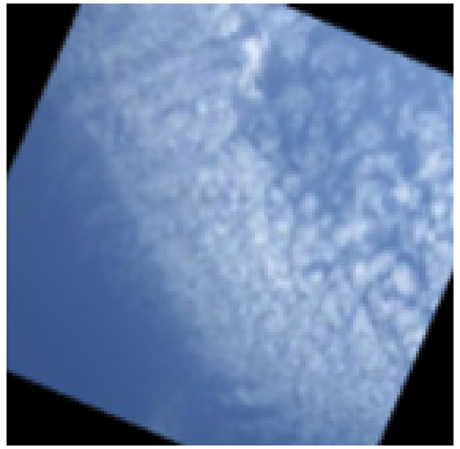
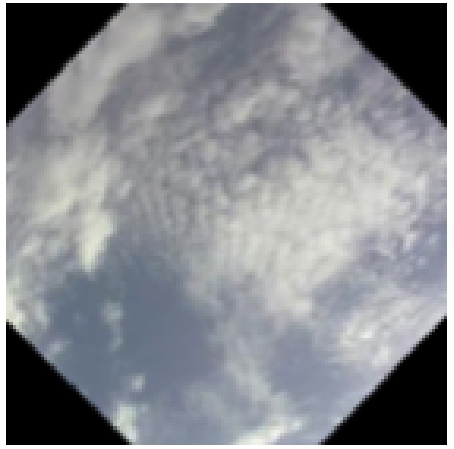
Data augmentation

```python
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(45),
    transforms.ToTensor(),
    transforms.Resize((128, 128)),
])

dataset_train = ImageFolder(
    "data/clouds/train",
    transform=train_transforms,
)
```

Data augmentation: Generating more data by applying random transformations to original images

- Increase the size and diversity of the training set
- Improve model robustness
- Reduce overfitting


Let's practice!

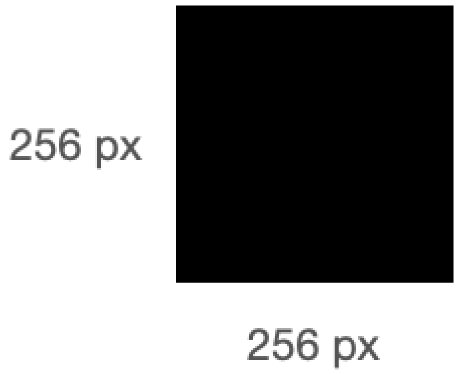
INTERMEDIATE DEEP LEARNING WITH PYTORCH

Convolutional Neural Networks

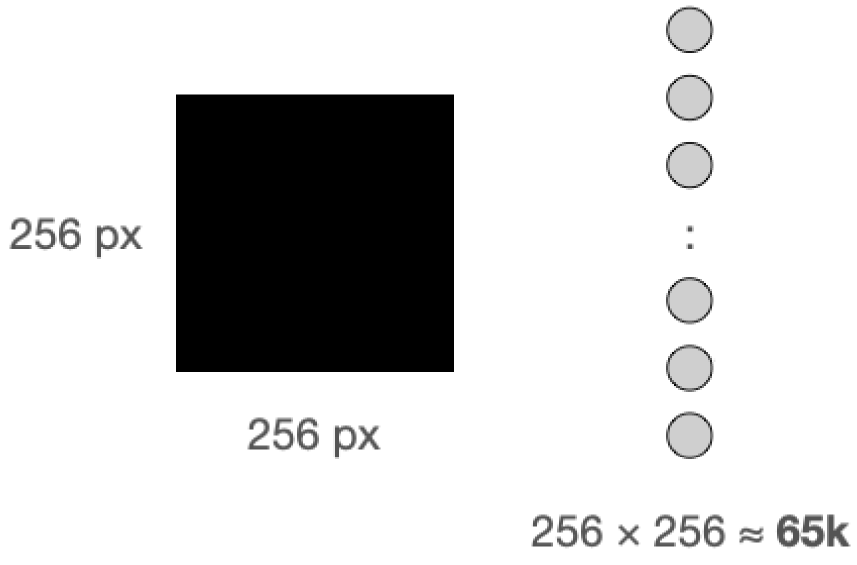
INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

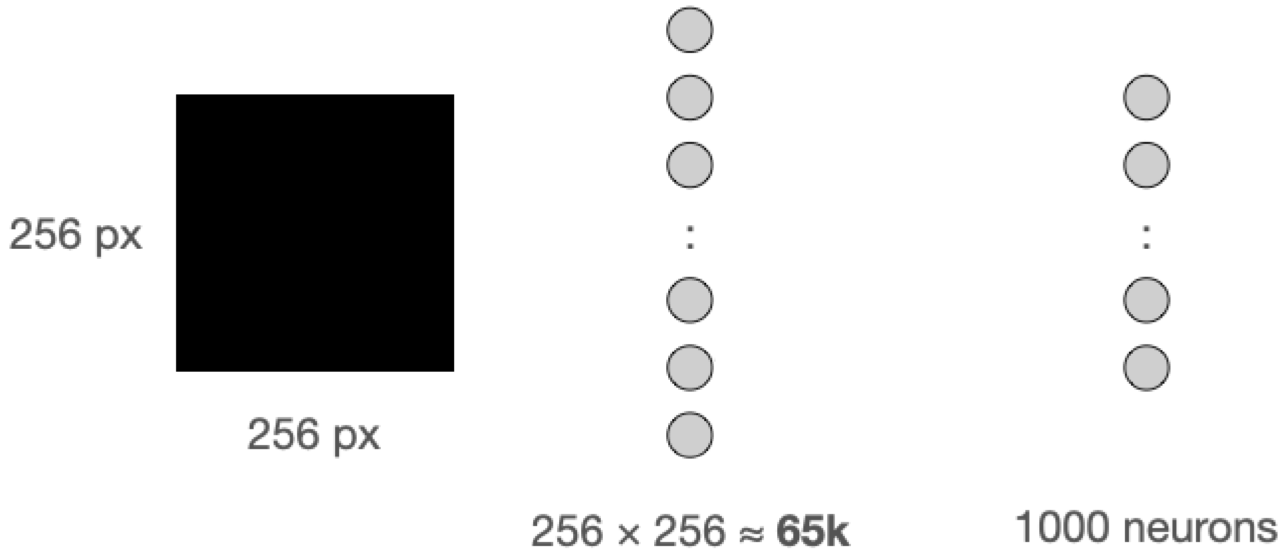
Why not use linear layers?



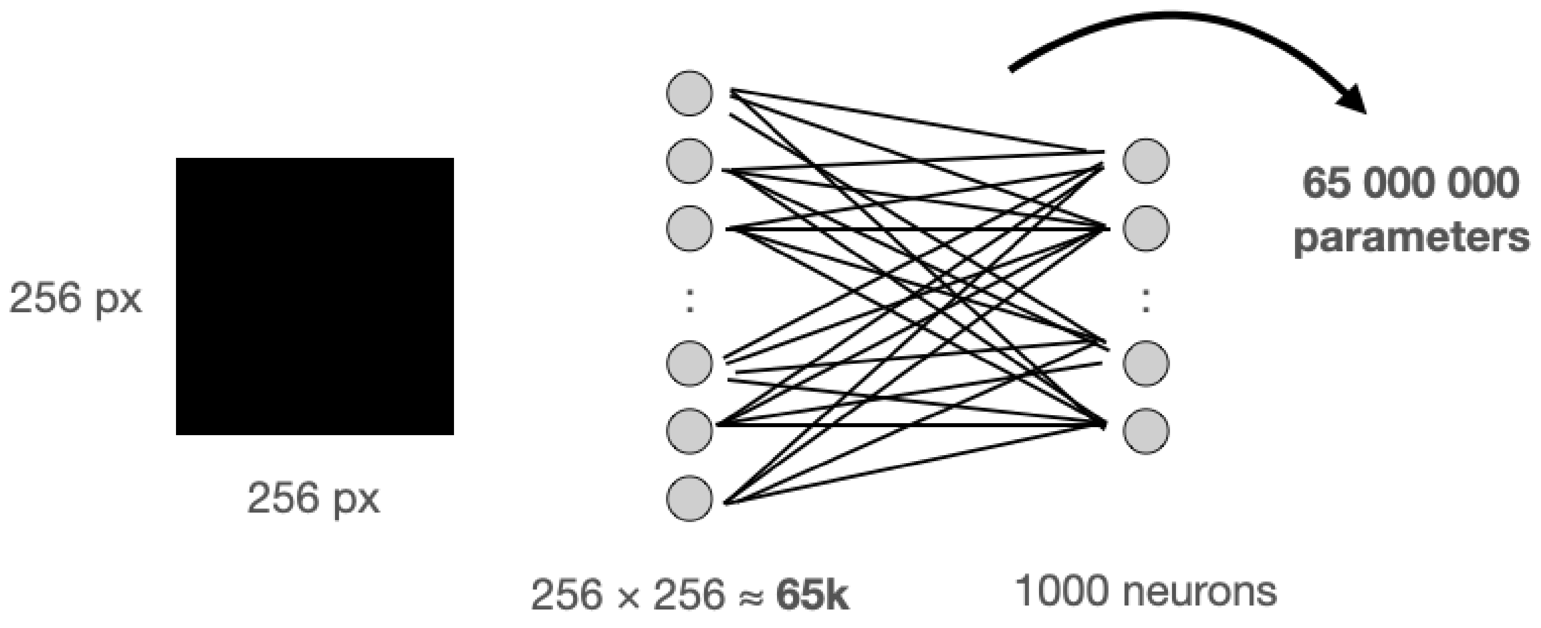
Why not use linear layers?



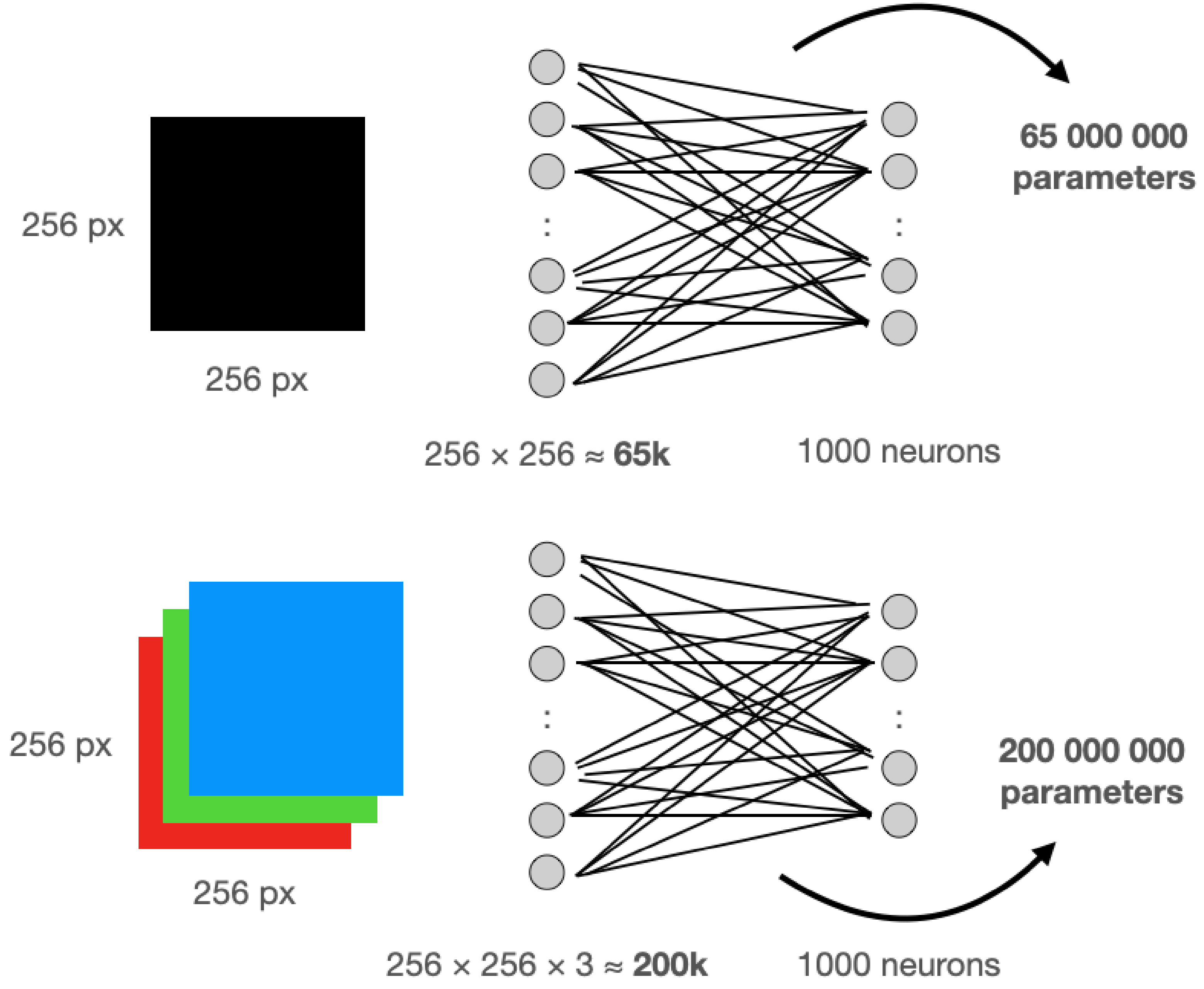
Why not use linear layers?



Why not use linear layers?

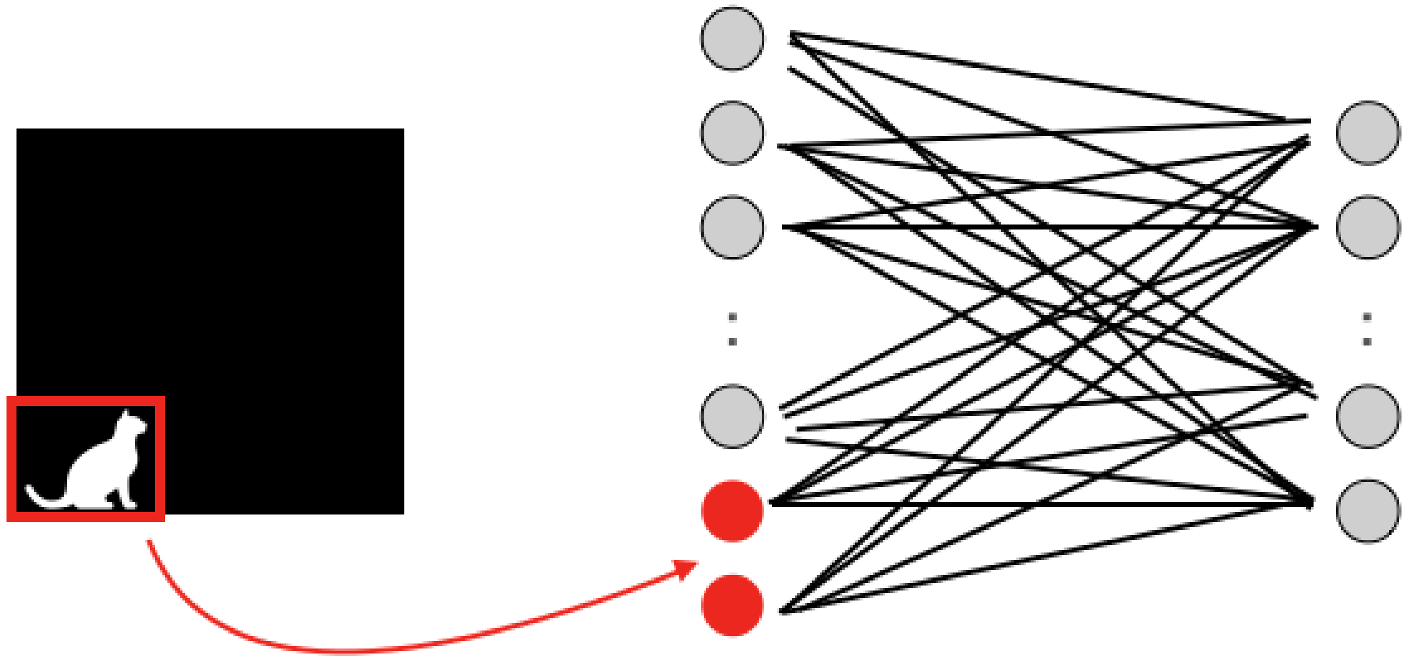


Why not use linear layers?

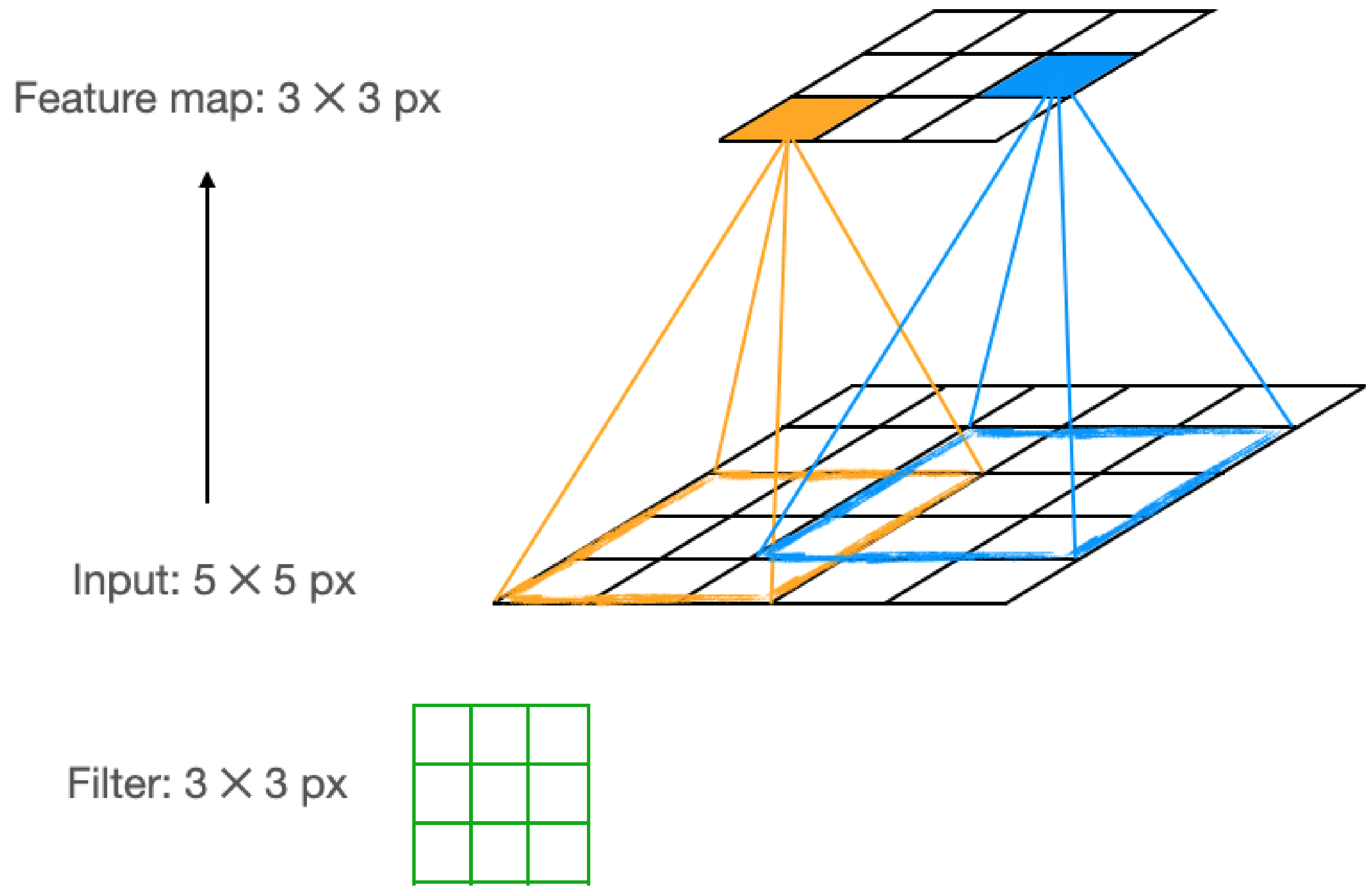


Why not use linear layers?

- Linear layers:
  - Slow training
  - Overfitting
  - Don't recognize spatial patterns
- A better alternative: convolutional layers!

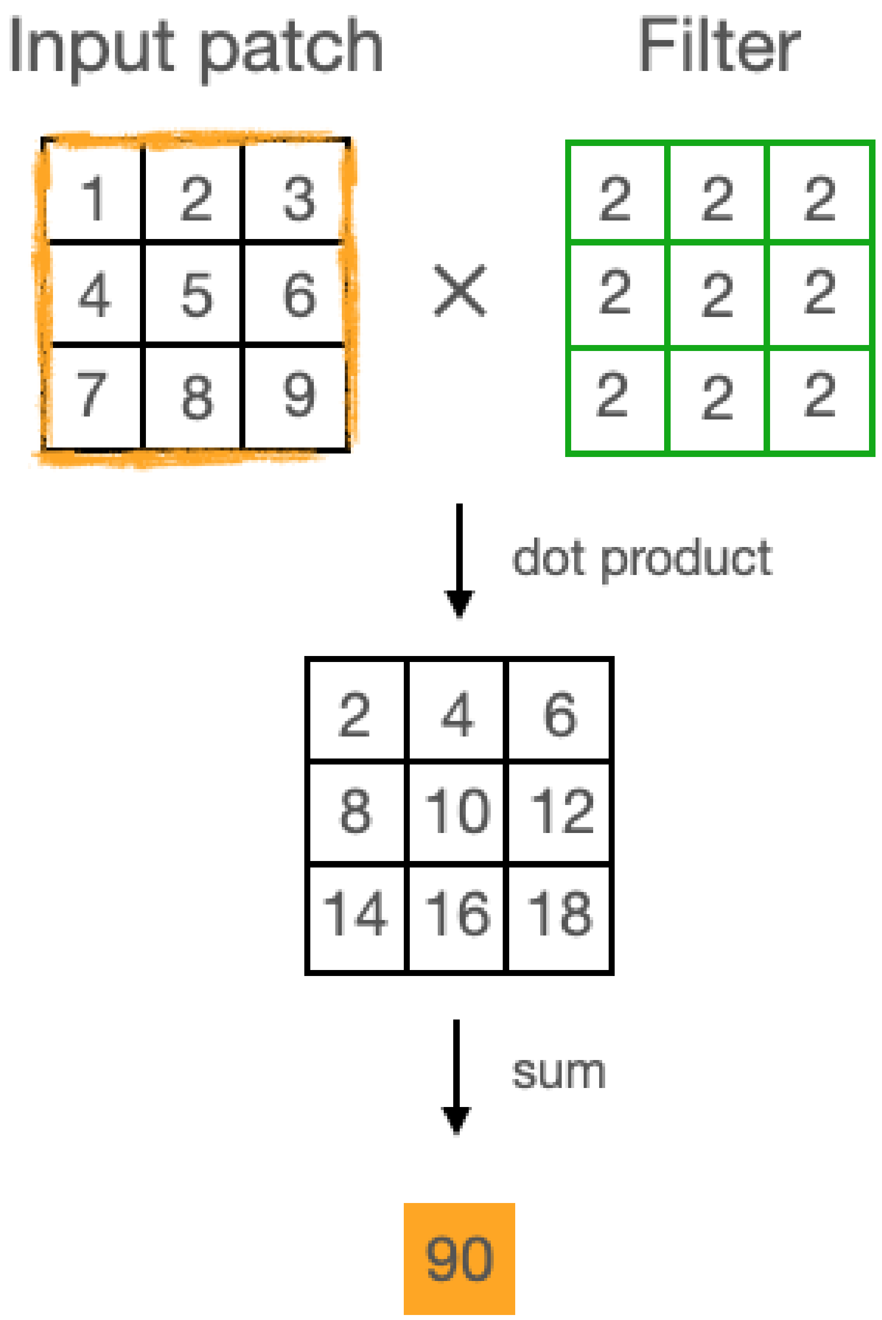


Convolutional layer



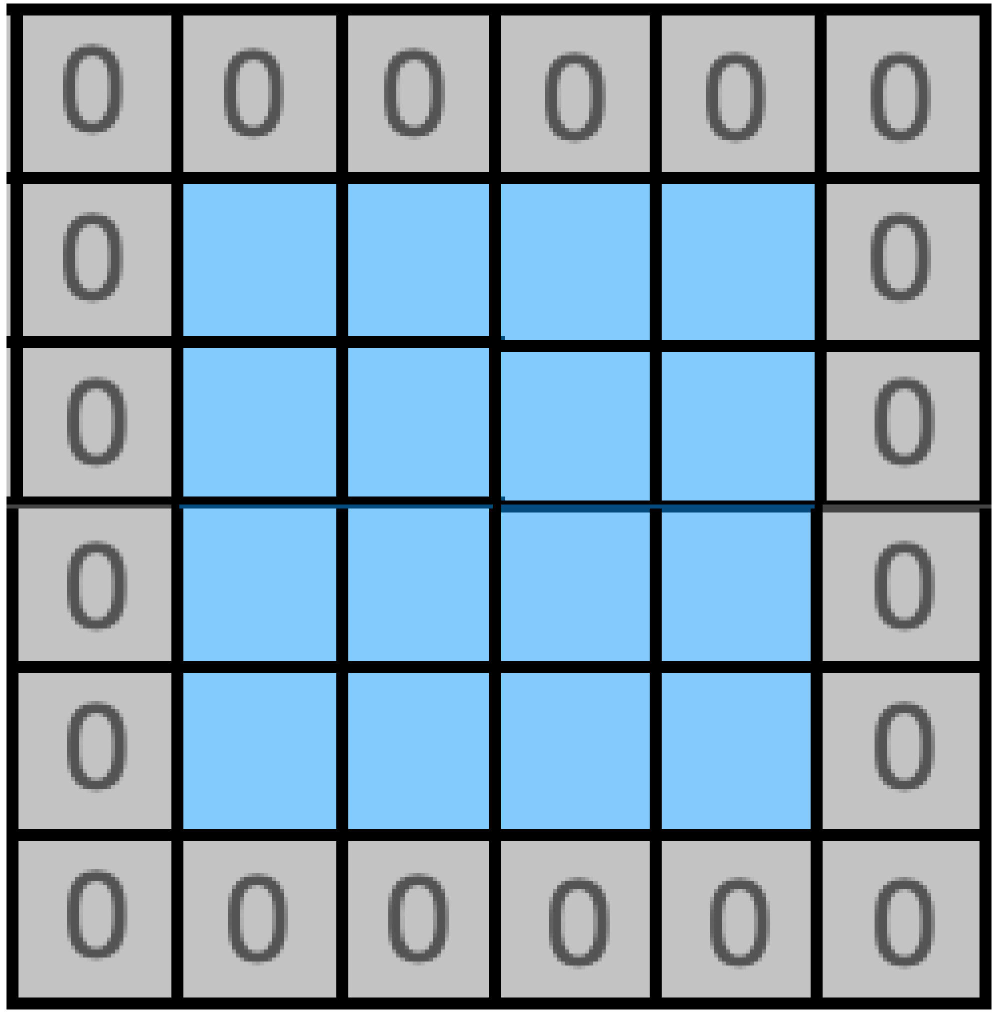
- Slide filter(s) of parameters over the input
- At each position, perform convolution
- Resulting feature map:
  - Preservers spatial patterns from input
  - Uses fewer parameters than linear layer
- One filter = one feature map
- Apply activations to feature maps
- All feature maps combined form the output
- `nn.Conv2d(3, 32, kernel_size=3)`

Convolution



1. Compute dot product of input patch and filter
   - Top-left field: $2 \times 1 = 2$
2. Sum the result

Zero-padding

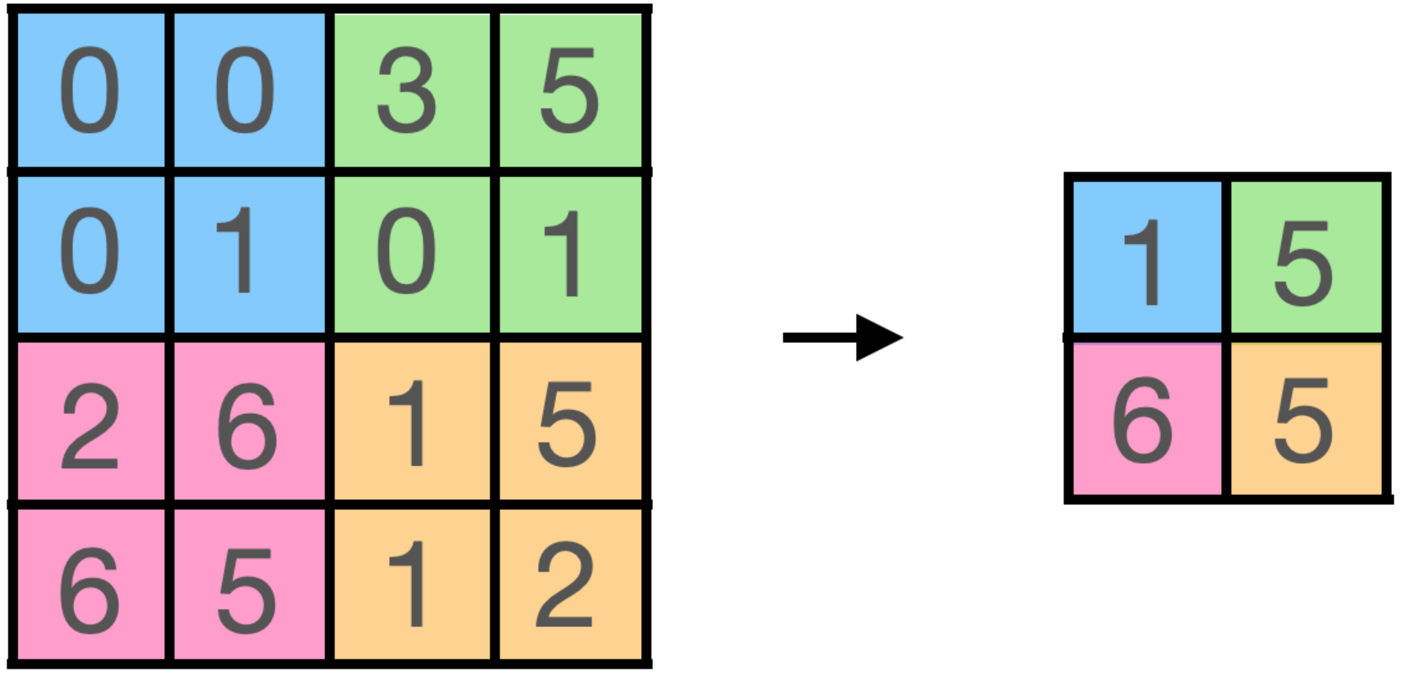


- Add a frames of zeros to convolutional layer's input

```
nn.Conv2d(
  3, 32, kernel_size=3, padding=1
)
```

- Maintains spatial dimensions of the input and output tensors
- Ensures border pixels are treated equally to others

Max Pooling



- Slide non-overlapping window over input
- At each position, retain only the maximum value
- Used after convolutional layers to reduce spatial dimensions
- `nn.MaxPool2d(kernel_size=2)`

Convolutional Neural Network

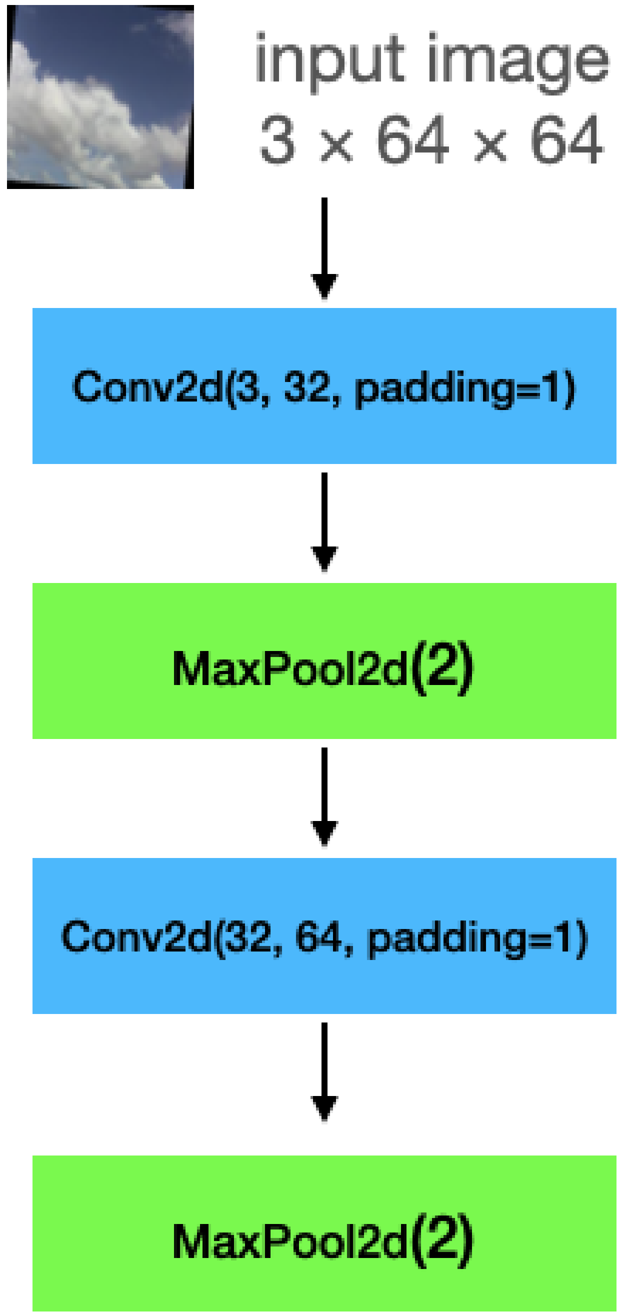
```
class Net(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Flatten(),
        )
        self.classifier = nn.Linear(64*16*16, num_classes)

    def forward(self, x):
        x = self.feature_extractor(x)
        x = self.classifier(x)
        return x
```

- `feature_extractor` : (convolution, activation, pooling), repeated twice and flattened
- `classifier` : single linear layer
- `forward()` : pass input image through feature extractor and classifier

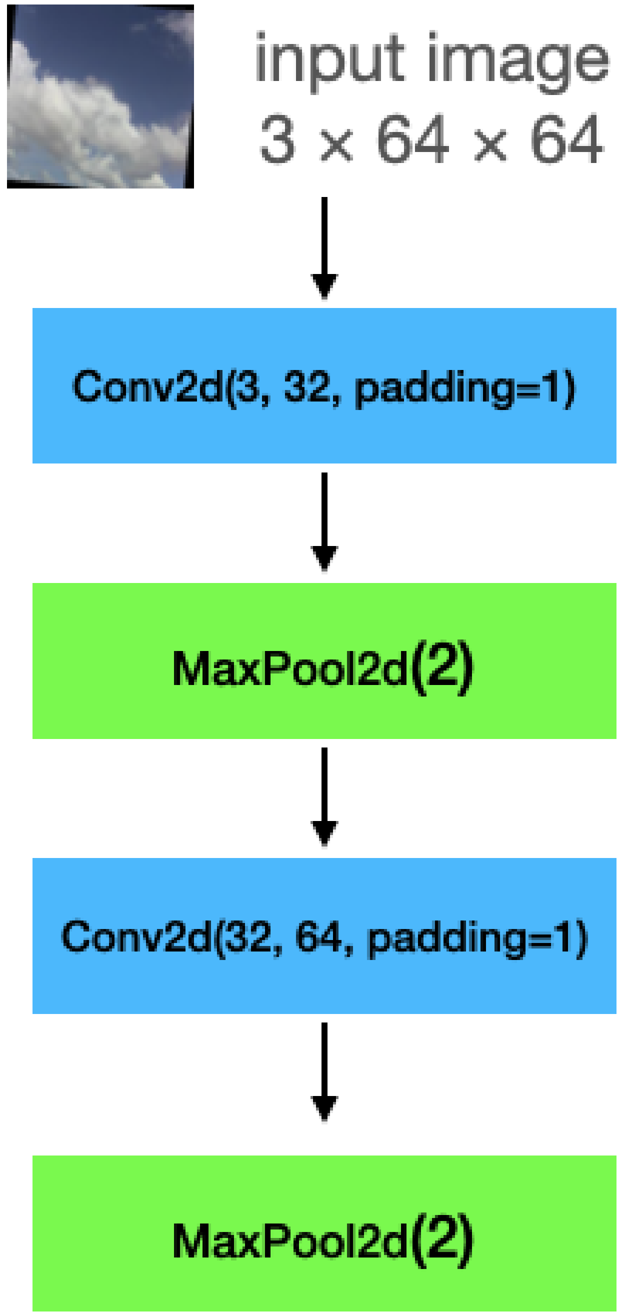
Feature extractor output size

```
self.feature_extractor = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Conv2d(32, 64, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Flatten(),
)
self.classifier = nn.Linear(64*16*16, num_classes)
`
```



Feature extractor output size

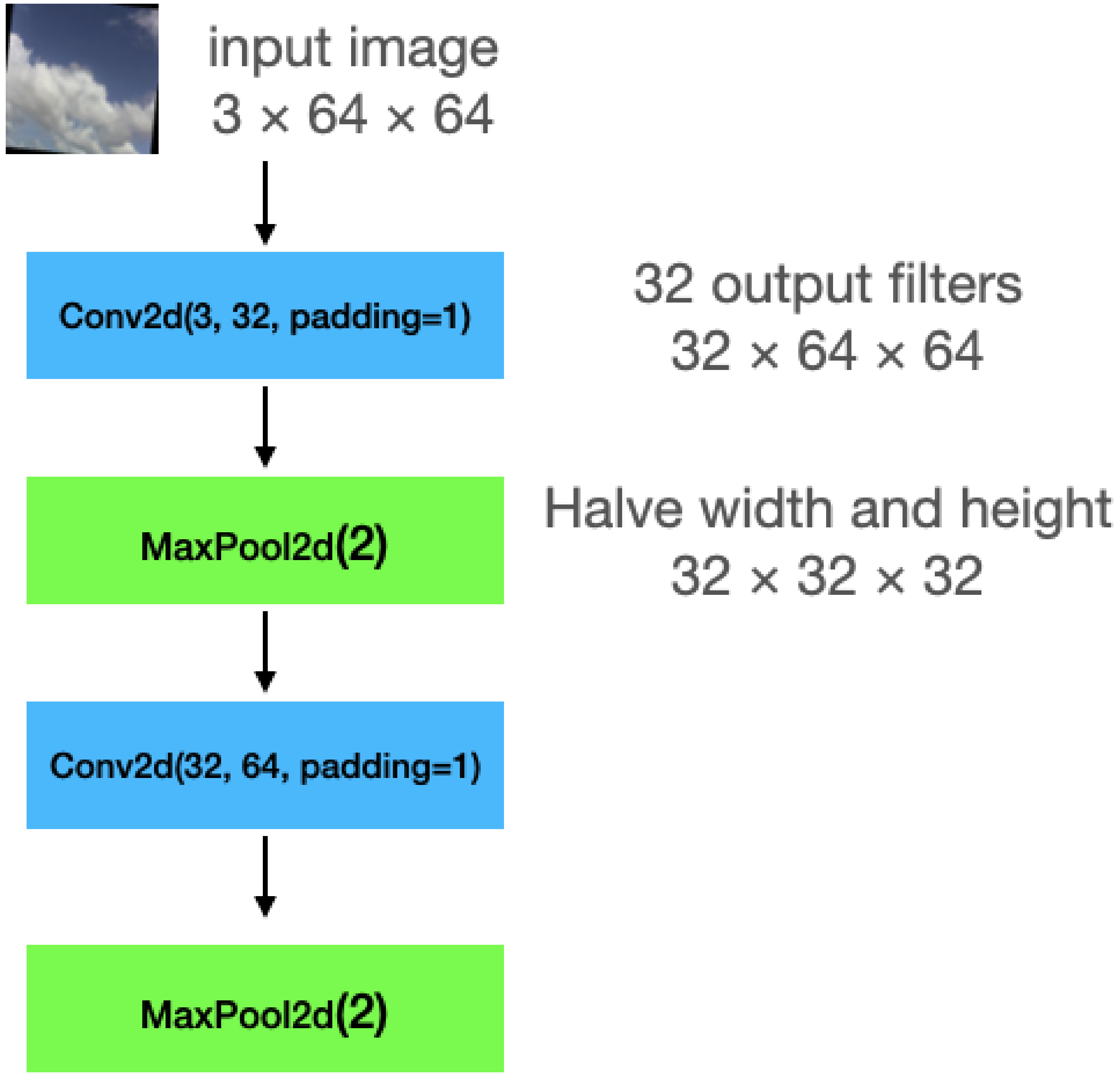
```
self.feature_extractor = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Conv2d(32, 64, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Flatten(),
)
self.classifier = nn.Linear(64*16*16, num_classes)
`
```



32 output filters
32 × 64 × 64

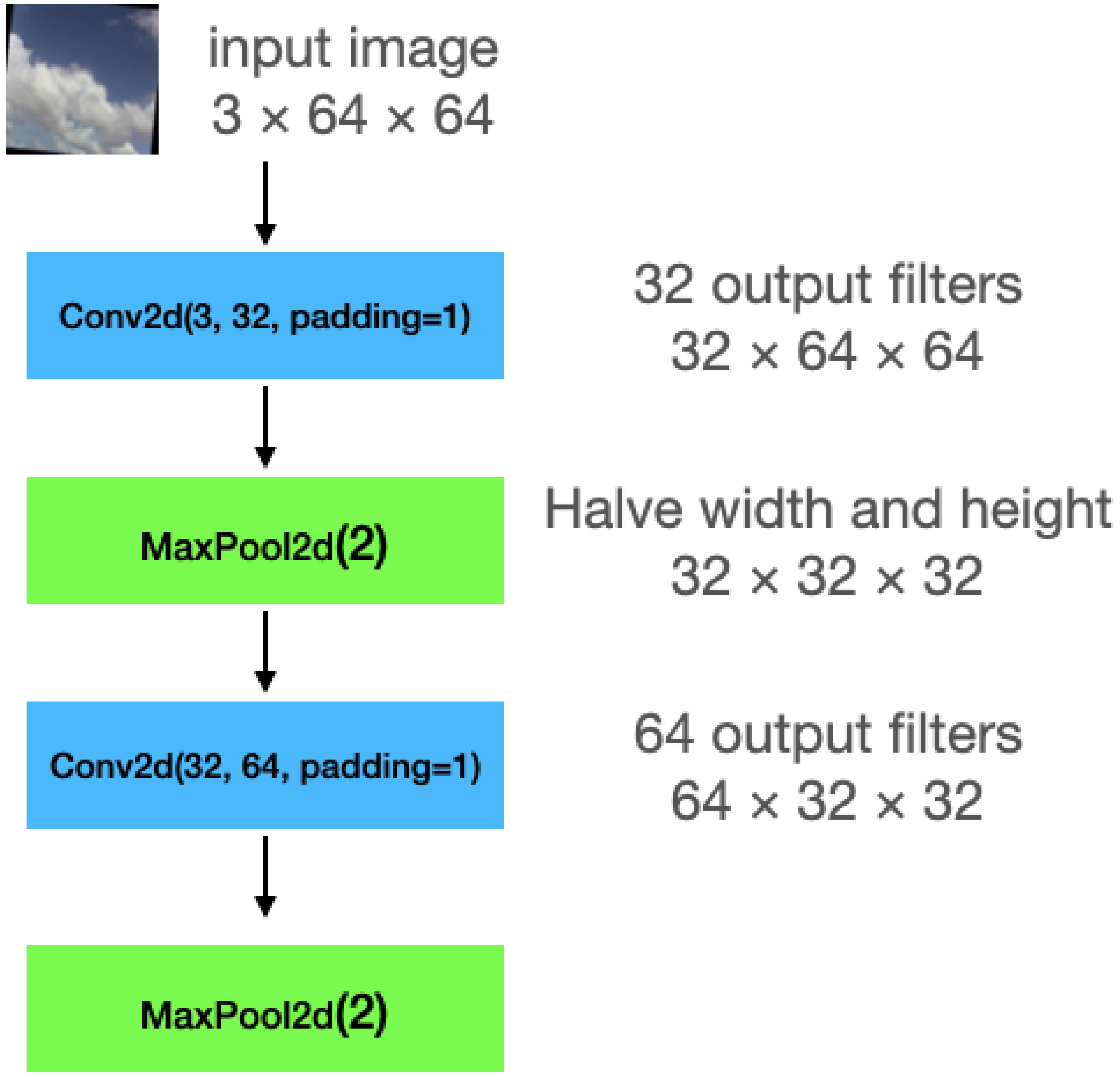
Feature extractor output size

```
self.feature_extractor = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Conv2d(32, 64, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Flatten(),
)
self.classifier = nn.Linear(64*16*16, num_classes)
`
```



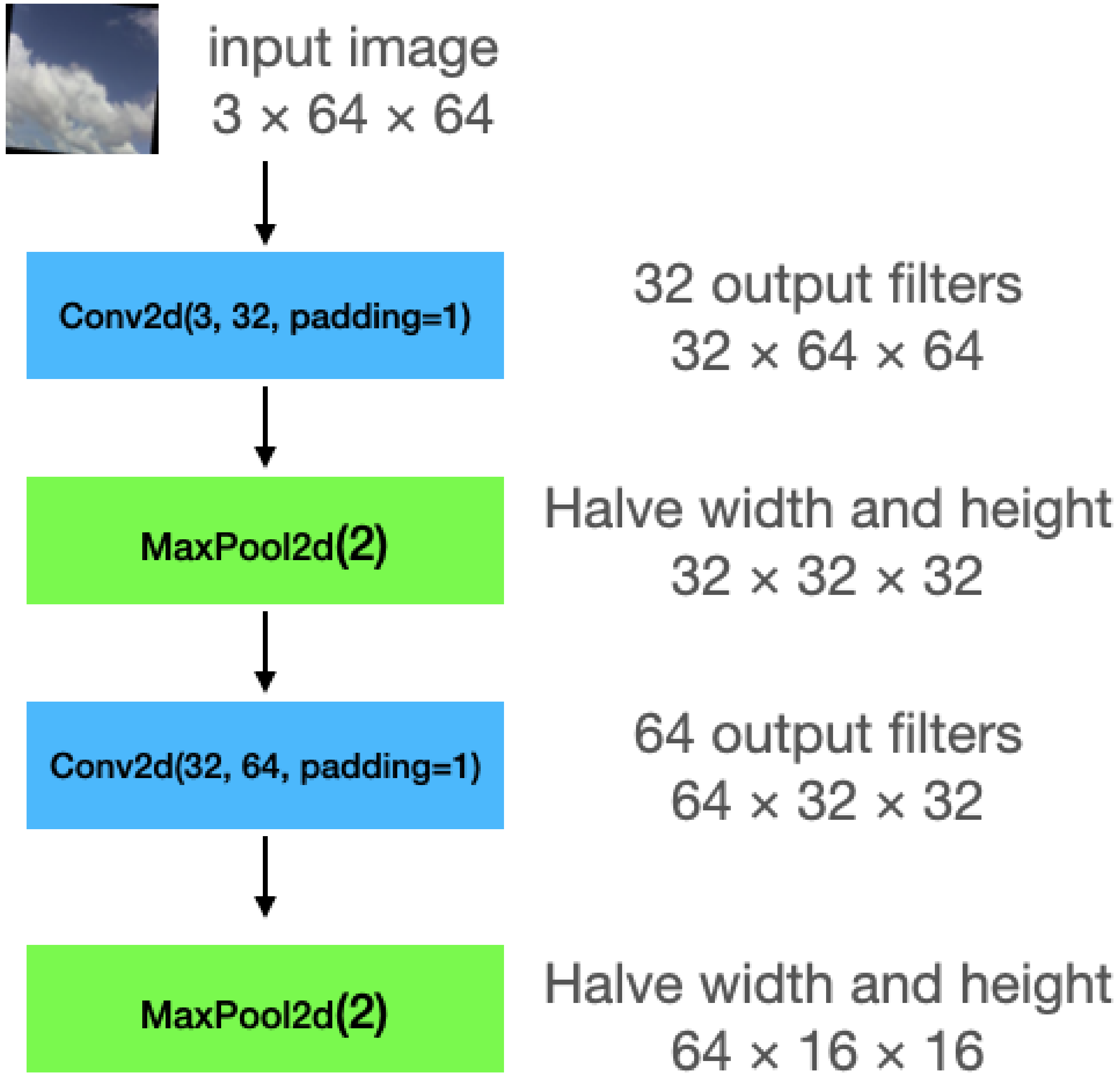
Feature extractor output size

```
self.feature_extractor = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Conv2d(32, 64, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Flatten(),
)
self.classifier = nn.Linear(64*16*16, num_classes)
`
```



Feature extractor output size

```
self.feature_extractor = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Conv2d(32, 64, kernel_size=3, padding=1),
  nn.ELU(),
  nn.MaxPool2d(kernel_size=2),
  nn.Flatten(),
)
self.classifier = nn.Linear(64*16*16, num_classes)
`
```



Let's practice!

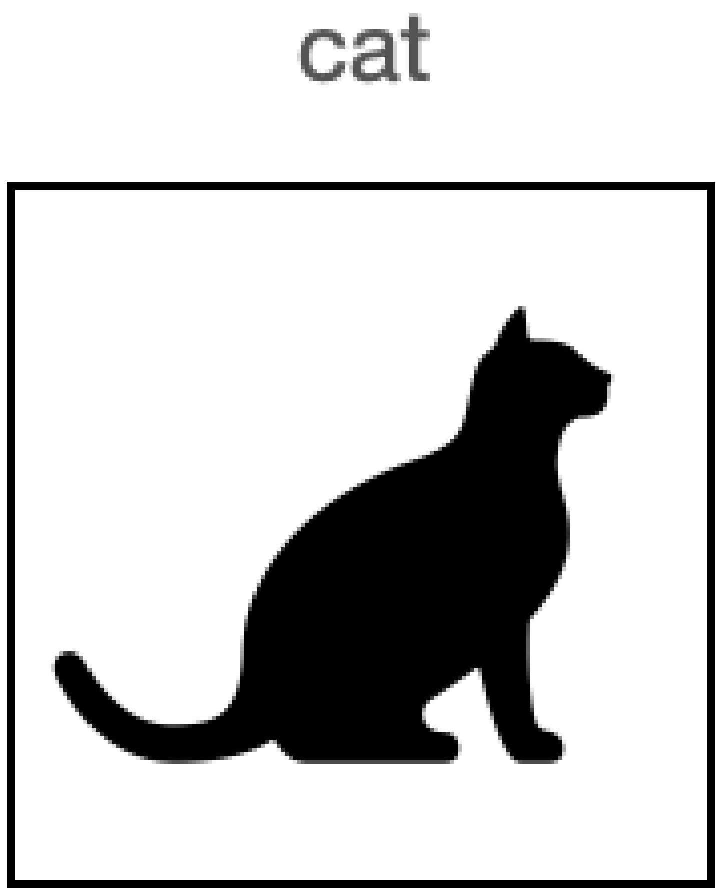
INTERMEDIATE DEEP LEARNING WITH PYTORCH

Training image classifiers

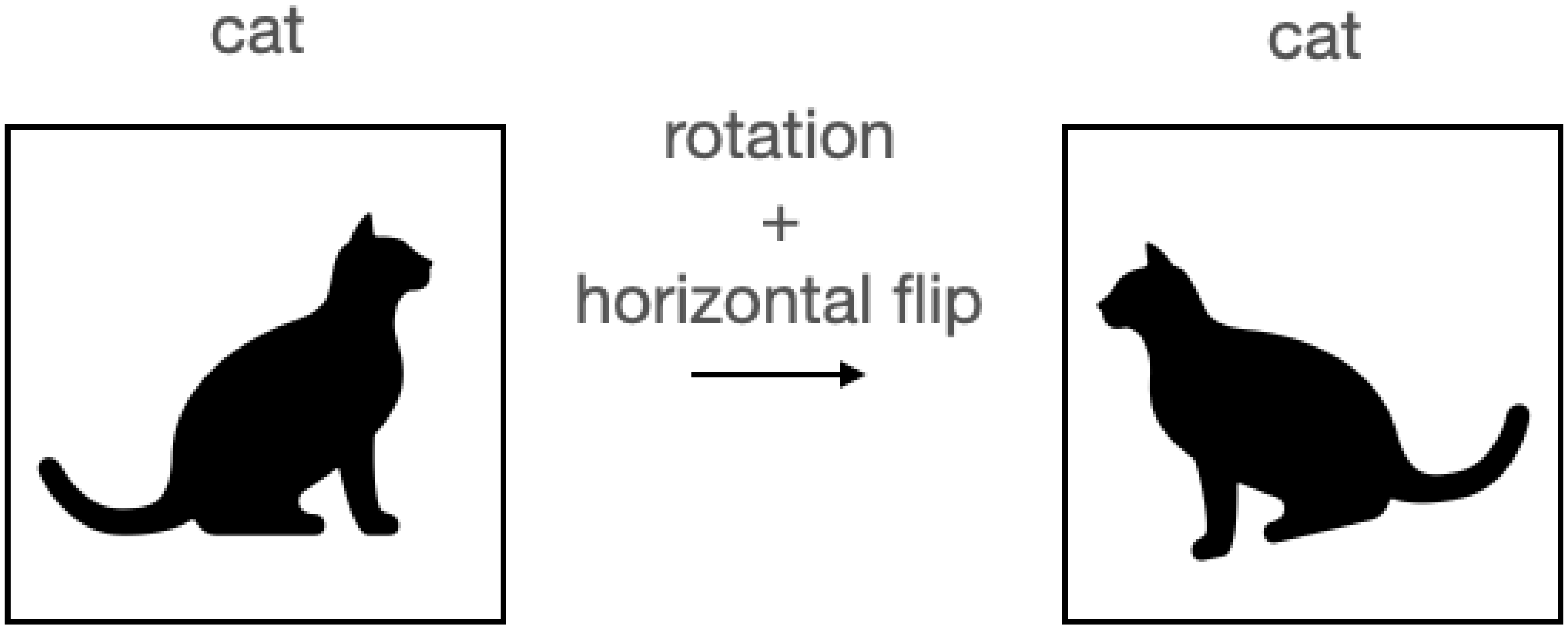
INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

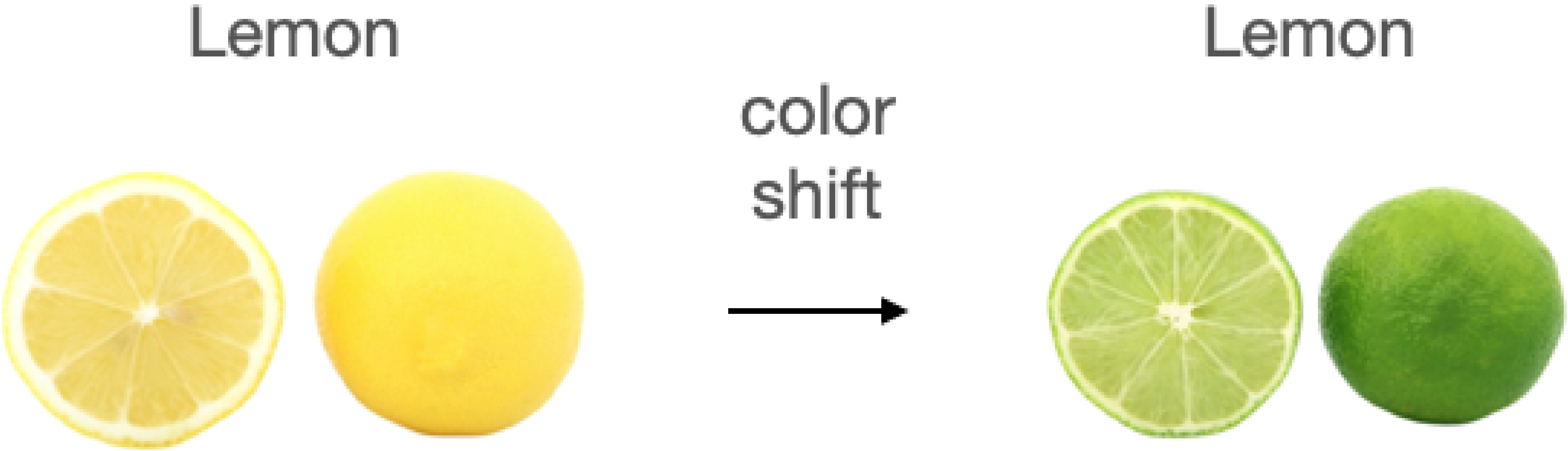
Data augmentation revisited



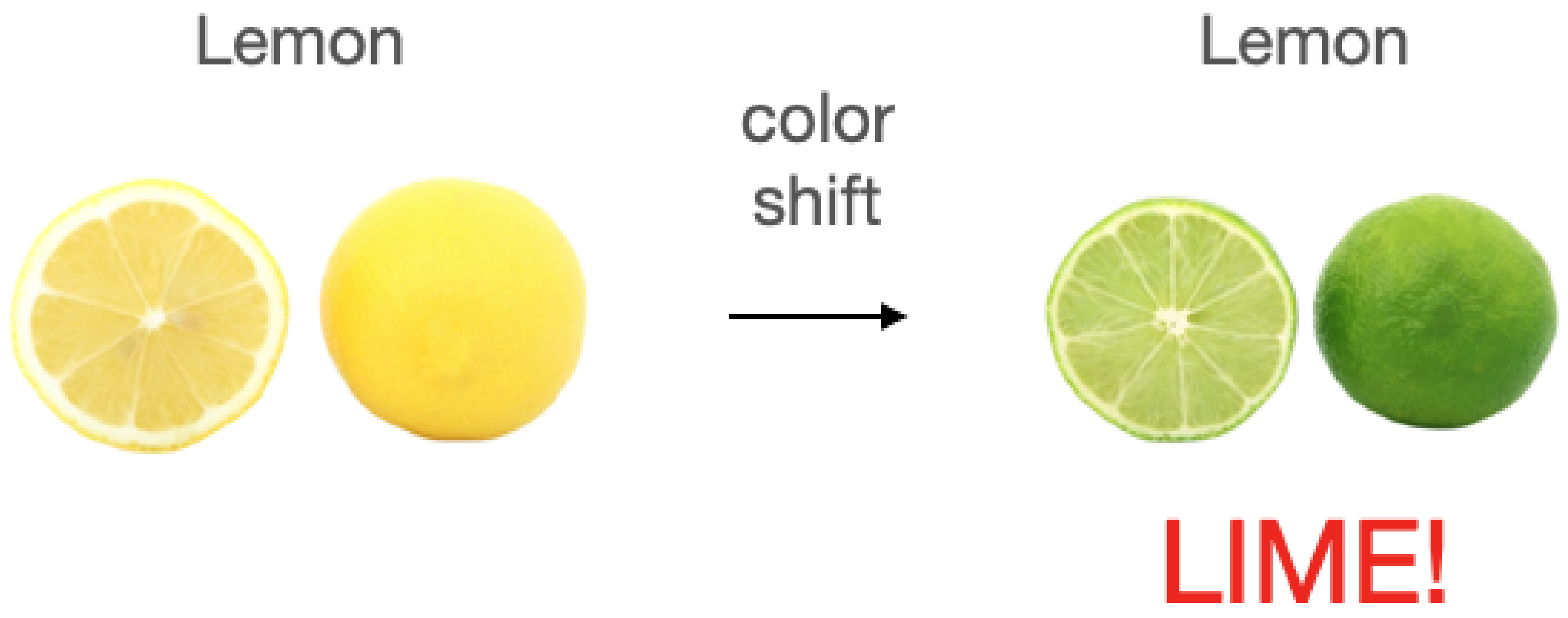
Data augmentation revisited



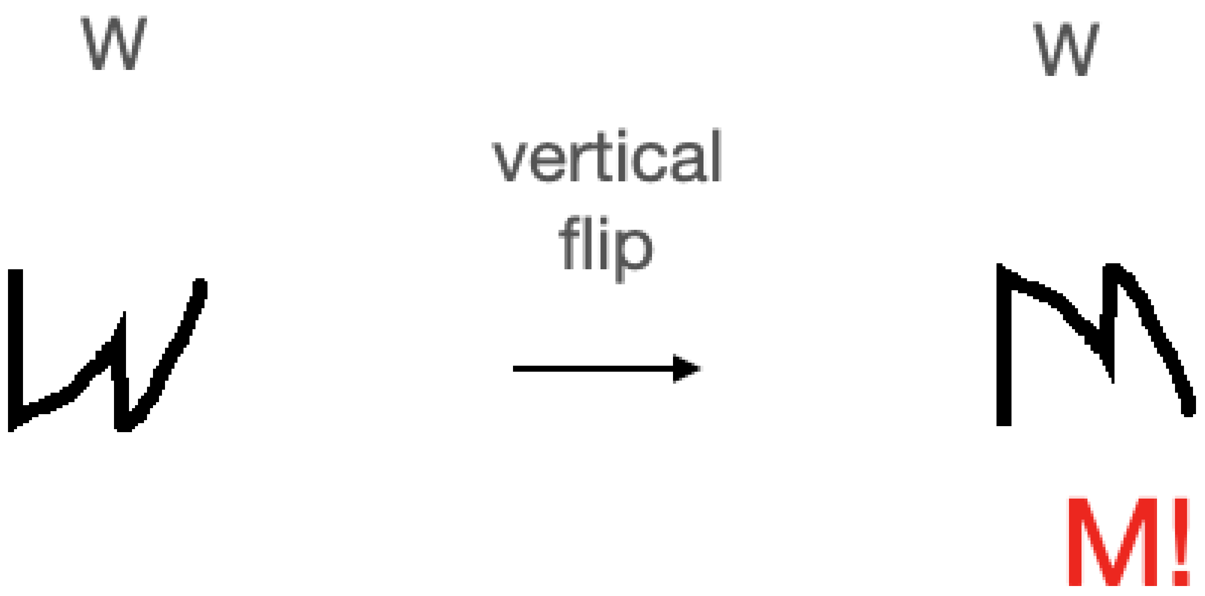
What should not be augmented



What should not be augmented



What should not be augmented



- Augmentations can impact the label
- Whether this is confusing depends on the task
- Always choose augmentations with the data and task in mind!

Augmentations for cloud classification




- **Random rotation:** expose model to different angles of cloud formations
- **Horizontal flip:** simulate different viewpoints of the sky
- **Auto contrast adjustment:** simulate different lighting conditions

```
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(45),
    transforms.RandomAutocontrast(),
    transforms.ToTensor(),
    transforms.Resize((128, 128))
])
```

Cross-Entropy loss

- Binary classification: binary cross-entropy (BCE) loss
- Multi-class classification: cross-entropy loss
- `criterion = nn.CrossEntropyLoss()`

Image classifier training loop

```
net = Net(num_classes=7)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

for epoch in range(10):
    for images, labels in dataloader_train:
        optimizer.zero_grad()
        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Let's practice!

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Evaluating image classifiers

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

Data augmentation at test time

Data augmentation for training data:

```
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(45),
    transforms.RandomAutocontrast(),
    transforms.ToTensor(),
    transforms.Resize((64, 64)),
])

dataset_train = ImageFolder(
  "clouds_train",
  transform=train_transforms,
)
```

Data augmentation for test data:

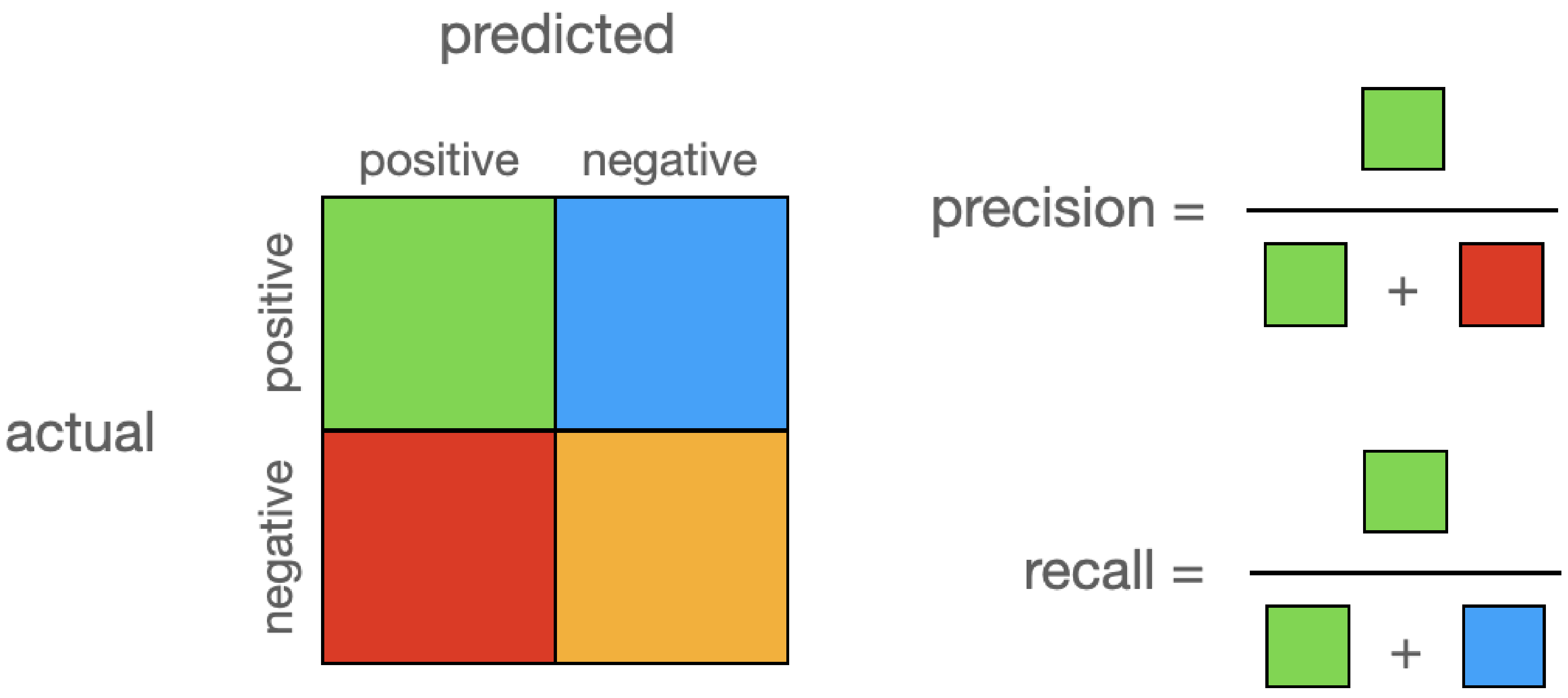
```
test_transforms = transforms.Compose([
    #
    # NO DATA AUGMENTATION AT TEST TIME
    #
    transforms.ToTensor(),
    transforms.Resize((64, 64)),
])

dataset_test = ImageFolder(
  "clouds_test",
  transform=test_transforms,
)
```

Precision & Recall: binary classification

In binary classification:

- **Precision:** Fraction of correct positive predictions
- **Recall:** Fraction of all positive examples correctly predicted



Precision & Recall: multi-class classification

In multi-class classification: separate precision and recall for each class

- **Precision:** Fraction of cumulus-predictions that were correct
- **Recall:** Fraction of all cumulus examples correctly predicted

Averaging multi-class metrics

- With 7 classes, we have 7 precision and 7 recall scores
- We can analyze them per-class, or aggregate:
  - **Micro average**: global calculation
  - **Macro average**: mean of per-class metrics
  - **Weighted average**: weighted mean of per-class metrics

Averaging multi-class metrics

```
from torchmetrics import Recall

recall_per_class = Recall(task="multiclass", num_classes=7, average=None)
recall_micro = Recall(task="multiclass", num_classes=7, average="micro")
recall_macro = Recall(task="multiclass", num_classes=7, average="macro")
recall_weighted = Recall(task="multiclass", num_classes=7, average="weighted")
```

When to use each:

- Micro: Imbalanced datasets
- Macro: Care about performance on small classes
- Weighted: Consider errors in larger classes as more important

Evaluation loop

```
from torchmetrics import Precision, Recall

metric_precision = Precision(
  task="multiclass", num_classes=7, average="macro"
)
metric_recall = Recall(
  task="multiclass", num_classes=7, average="macro"
)
net.eval()
with torch.no_grad():
    for images, labels in dataloader_test:
        outputs = net(images)
        _, preds = torch.max(outputs, 1)
        metric_precision(preds, labels)
        metric_recall(preds, labels)
precision = metric_precision.compute()
recall = metric_recall.compute()
```

- Import and define precision and recall metrics
- Iterate over test examples with no gradient
- For each test batch, get model outputs, take most likely class, and pass it to metric functions along with the labels
- Compute the metrics

```
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Precision: 0.7284010648727417
Recall: 0.763038694858551
```

Analyzing performance per class

```
metric_recall = Recall(
  task="multiclass", num_classes=7, average=None
)
net.eval()
with torch.no_grad():
    for images, labels in dataloader_test:
        outputs = net(images)
        _, preds = torch.max(outputs, 1)
        metric_recall(preds, labels)
recall = metric_recall.compute()
```

```
print(recall)
```

```
tensor([0.6364, 1.0000, 0.9091, 0.7917,
        0.5049, 0.9500, 0.5493],
       dtype=torch.float32)
```

- Compute metric with `average=None`
- This gives one score per class
- `Dataset` 's `.class_to_idx` attribute maps class names to indices

```
dataset_test.class_to_idx
```

```
{'cirriform clouds': 0,
 'clear sky': 1,
 'cumulonimbus clouds': 2,
 'cumulus clouds': 3,
 'high cumuliform clouds': 4,
 'stratiform clouds': 5,
 'stratocumulus clouds': 6}
```

Analyzing performance per class

```
{
  k: recall[v].item()
  for k, v
  in dataset_test.class_to_idx.items()
}
```

```
{'cirriform clouds': 0.6363636255264282,
 'clear sky': 1.0,
 'cumulonimbus clouds': 0.9090909361839294,
 'cumulus clouds': 0.7916666865348816,
 'high cumuliform clouds': 0.5048543810844421,
 'stratiform clouds': 0.949999988079071,
 'stratocumulus clouds': 0.5492957830429077}
```

- `k` = class name, e.g. `cirriform clouds`
- `v` = class index, e.g. `0`
- `recall[v]` = `tensor(0.6364, dtype=torch.float32)`
- `recall[v].item()` = `0.6364`

Let's practice!

INTERMEDIATE DEEP LEARNING WITH PYTORCH