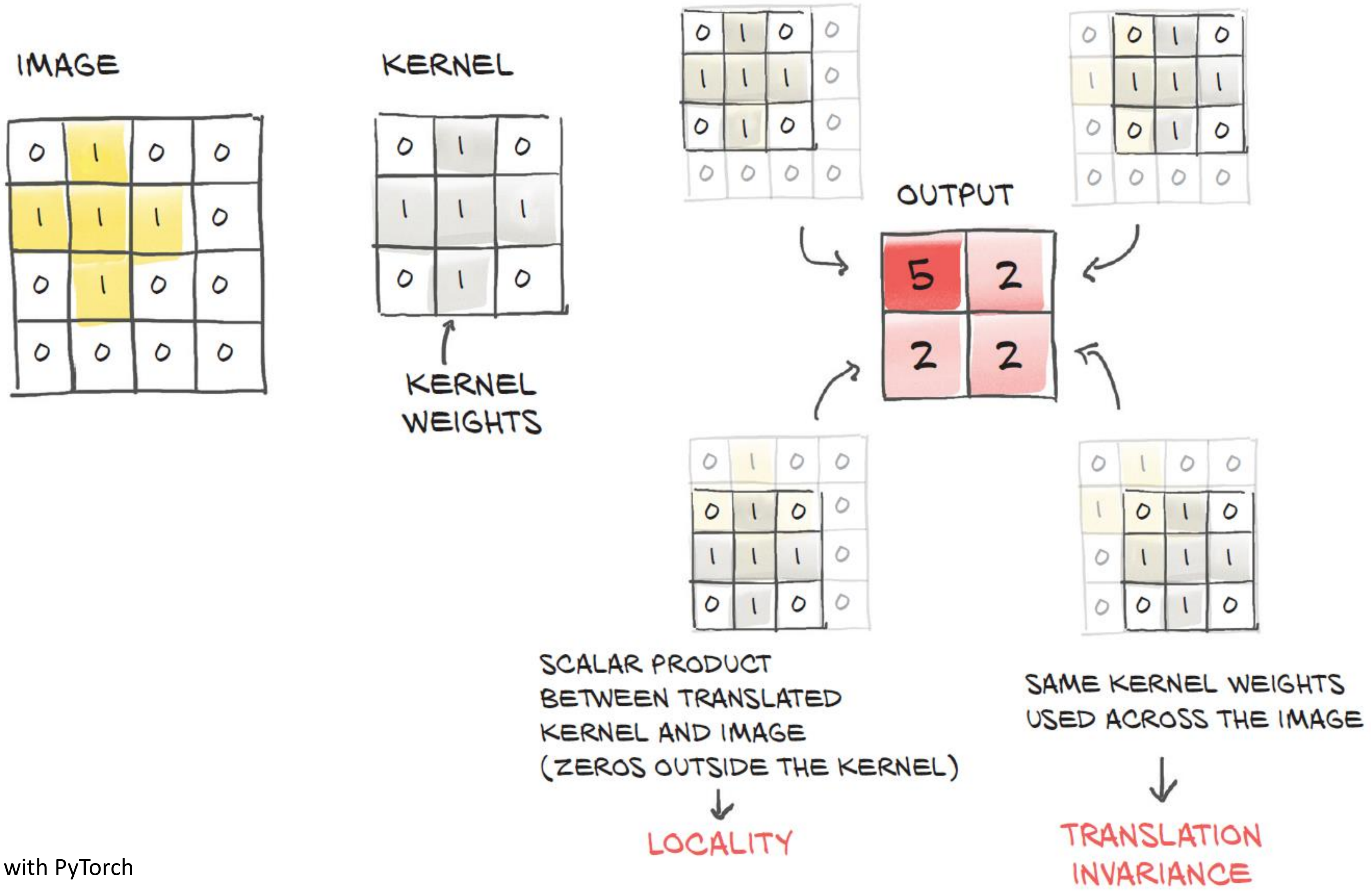
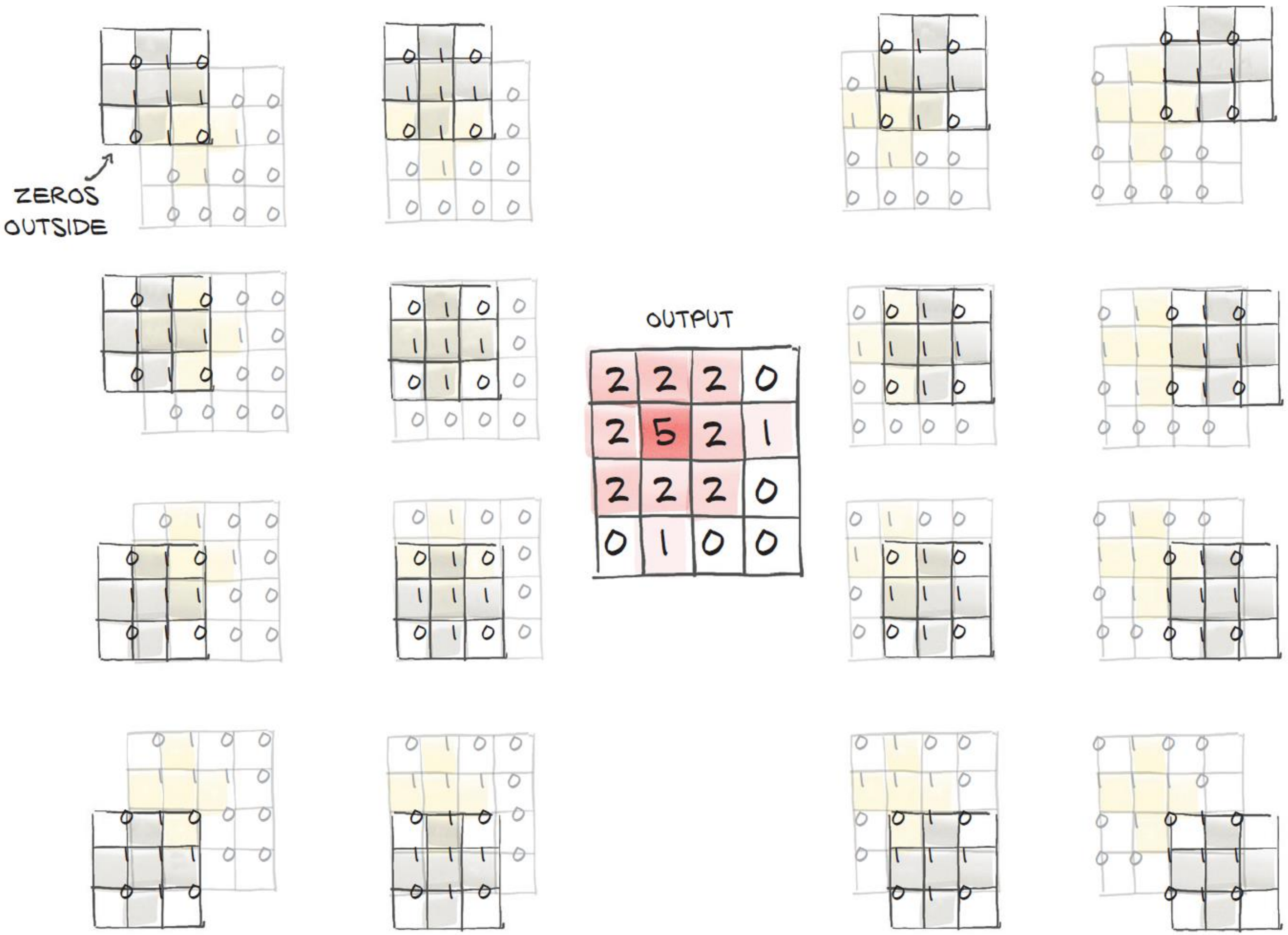
Vorlesung 8

Convolutional Kernel

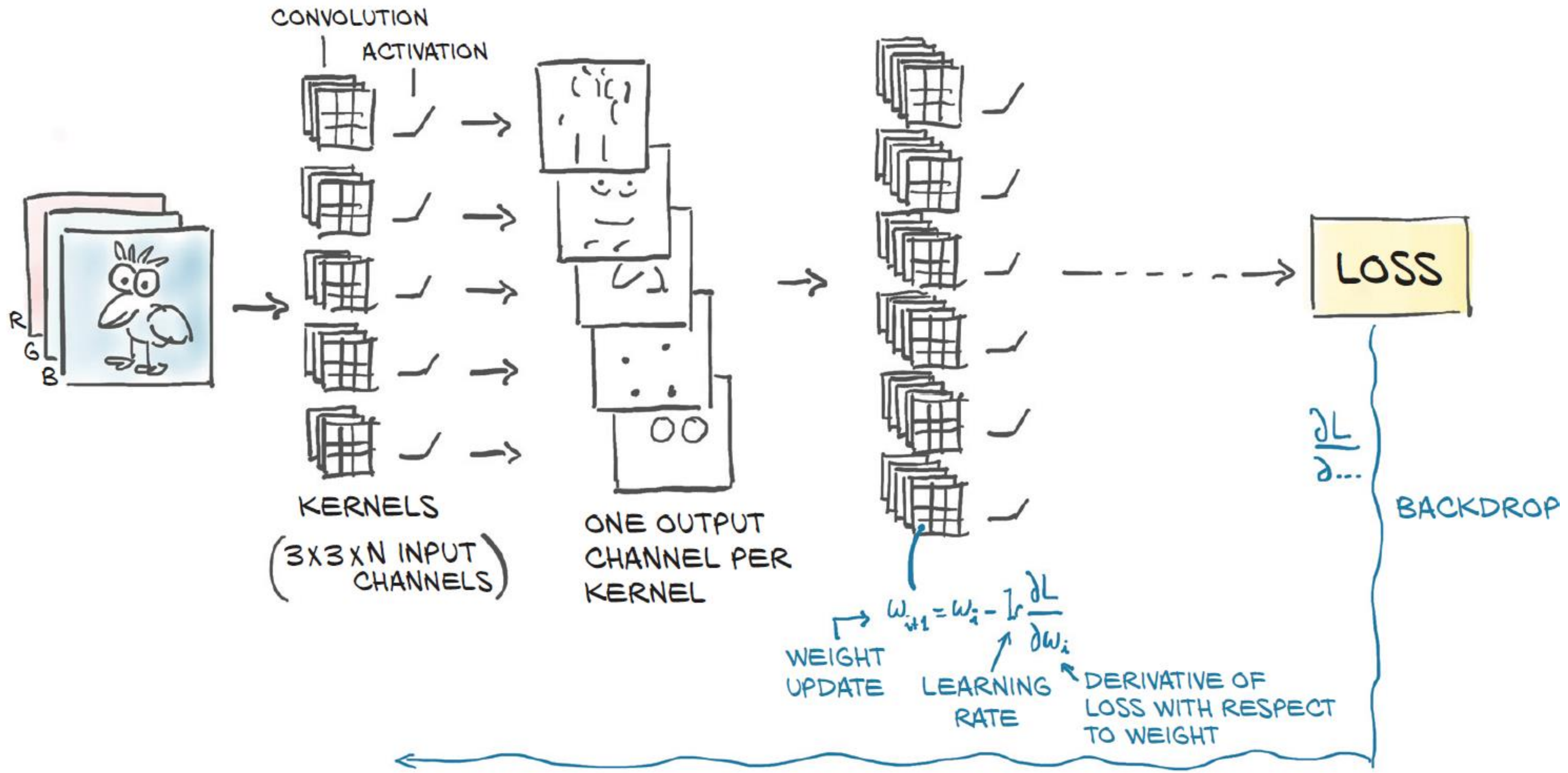


Quelle: Deep Learning with PyTorch



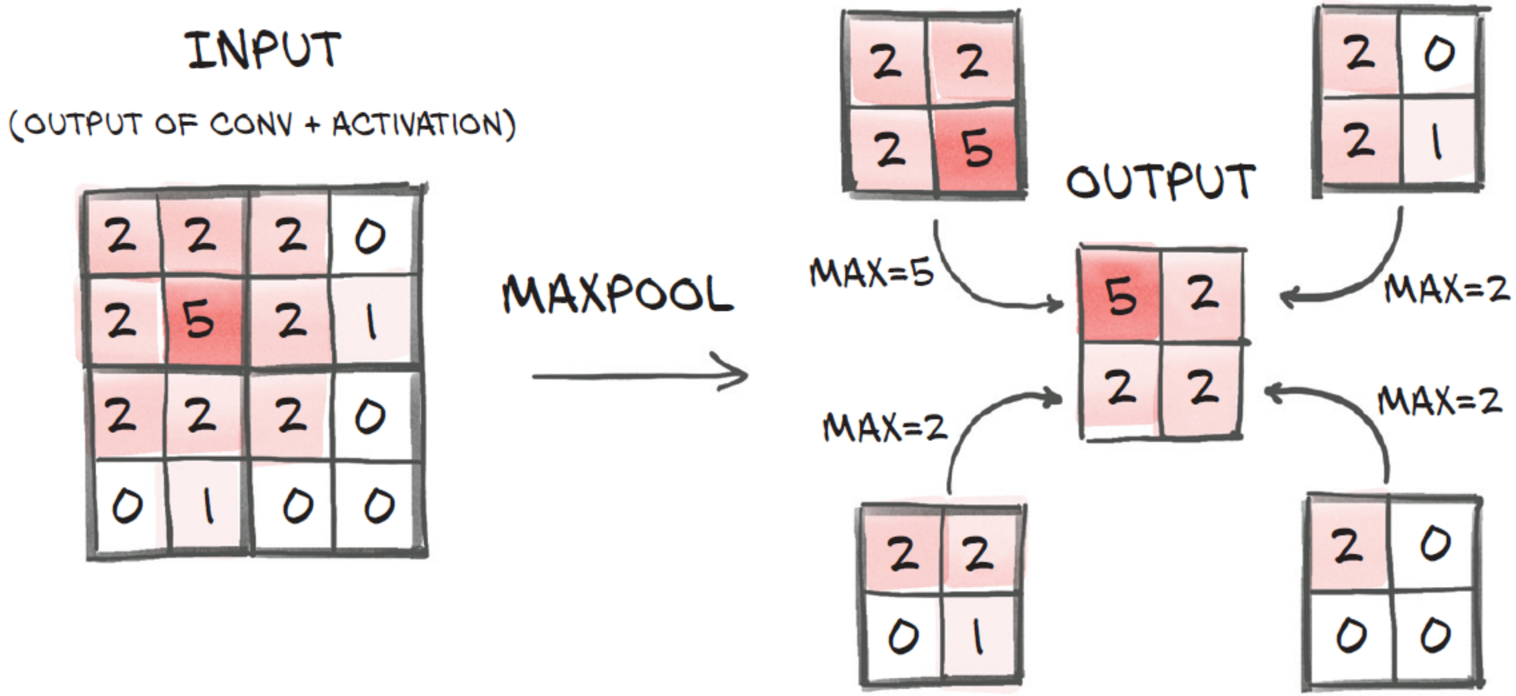
Quelle: Deep Learning with PyTorch

Generation of Kernels by Training



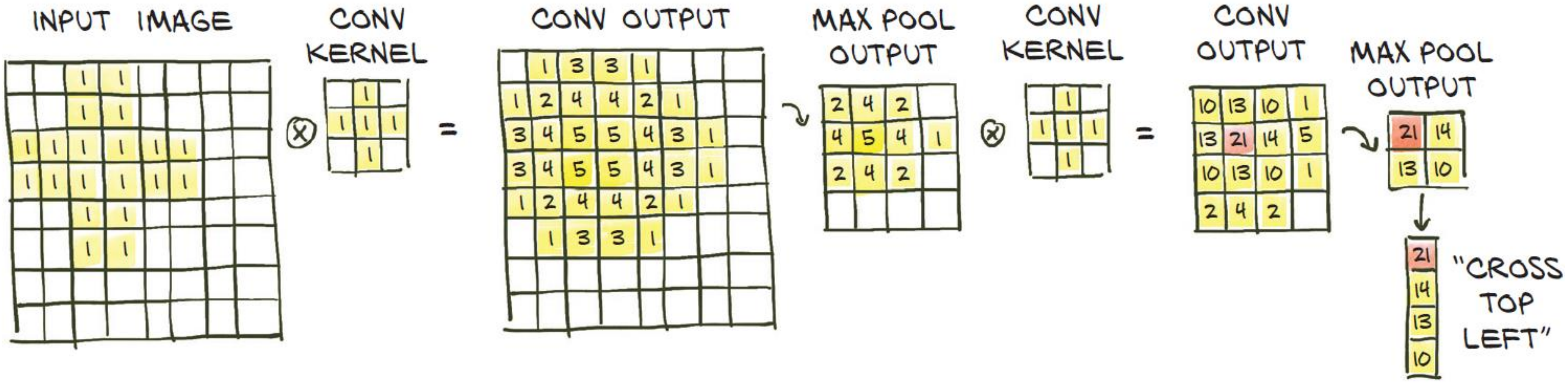
Quelle: Deep Learning with PyTorch

Downsampling using Maxpooling



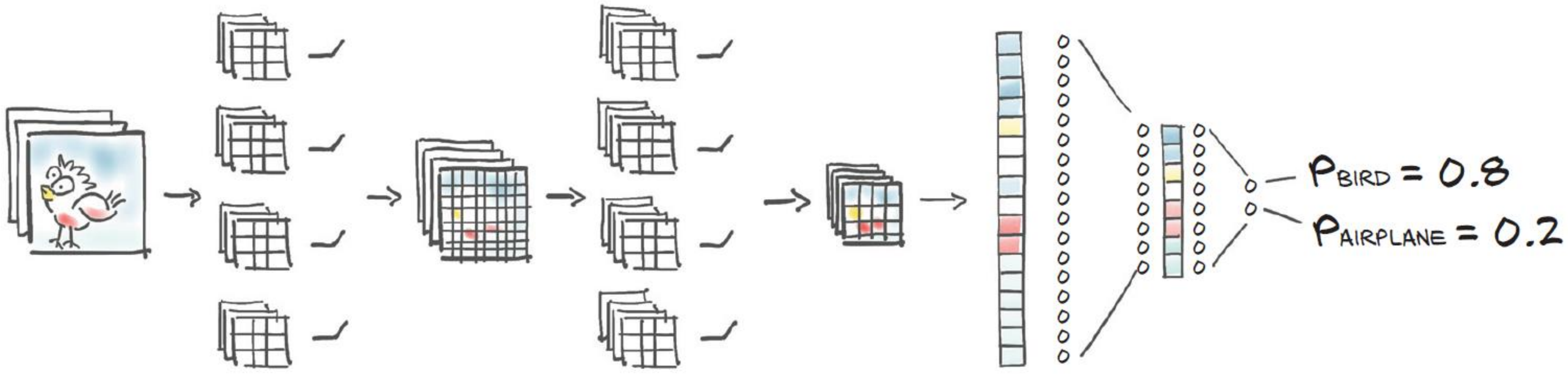
Quelle: Deep Learning with PyTorch

Combining Convolution and Maxpooling



Quelle: Deep Learning with PyTorch

Applying Convolution and Maxpooling to CIFAR



Quelle: Deep Learning with PyTorch

Weight Penalties

```
# In[45]:
def training_loop_l2reg(n_epochs, optimizer, model, loss_fn,
                        train_loader):
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        for imgs, labels in train_loader:
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)
            outputs = model(imgs)
            loss = loss_fn(outputs, labels)

            l2_lambda = 0.001
            l2_norm = sum(p.pow(2.0).sum()
                          for p in model.parameters())
            loss = loss + l2_lambda * l2_norm

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_train += loss.item()
        if epoch == 1 or epoch % 10 == 0:
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch,
                loss_train / len(train_loader)))
```

Replaces pow(2.0) with abs() for L1 regularization

Quelle: Deep Learning with PyTorch

Dropout

```
# In[47]:
class NetDropout(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_dropout = nn.Dropout2d(p=0.4)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        self.conv2_dropout = nn.Dropout2d(p=0.4)
        self.fc1 = nn.Linear(8 * 8 * n_chans1 // 2, 32)
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = self.conv1_dropout(out)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = self.conv2_dropout(out)
        out = out.view(-1, 8 * 8 * self.n_chans1 // 2)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

Quelle: Deep Learning with PyTorch

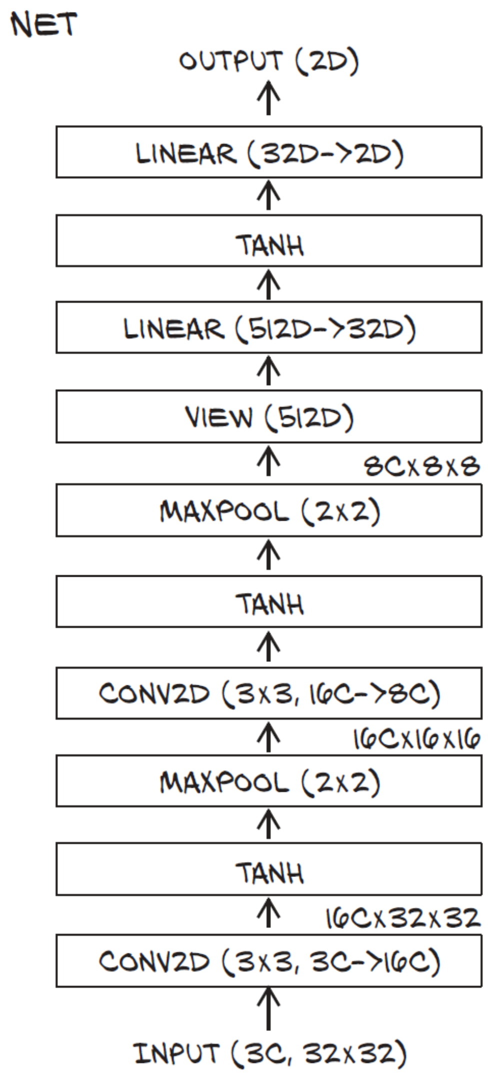
Batch Normalization

```
# In[49]:
class NetBatchNorm(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_batchnorm = nn.BatchNorm2d(num_features=n_chans1)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        self.conv2_batchnorm = nn.BatchNorm2d(num_features=n_chans1 // 2)
        self.fc1 = nn.Linear(8 * 8 * n_chans1 // 2, 32)
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = self.conv1_batchnorm(self.conv1(x))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = self.conv2_batchnorm(self.conv2(out))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = out.view(-1, 8 * 8 * self.n_chans1 // 2)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

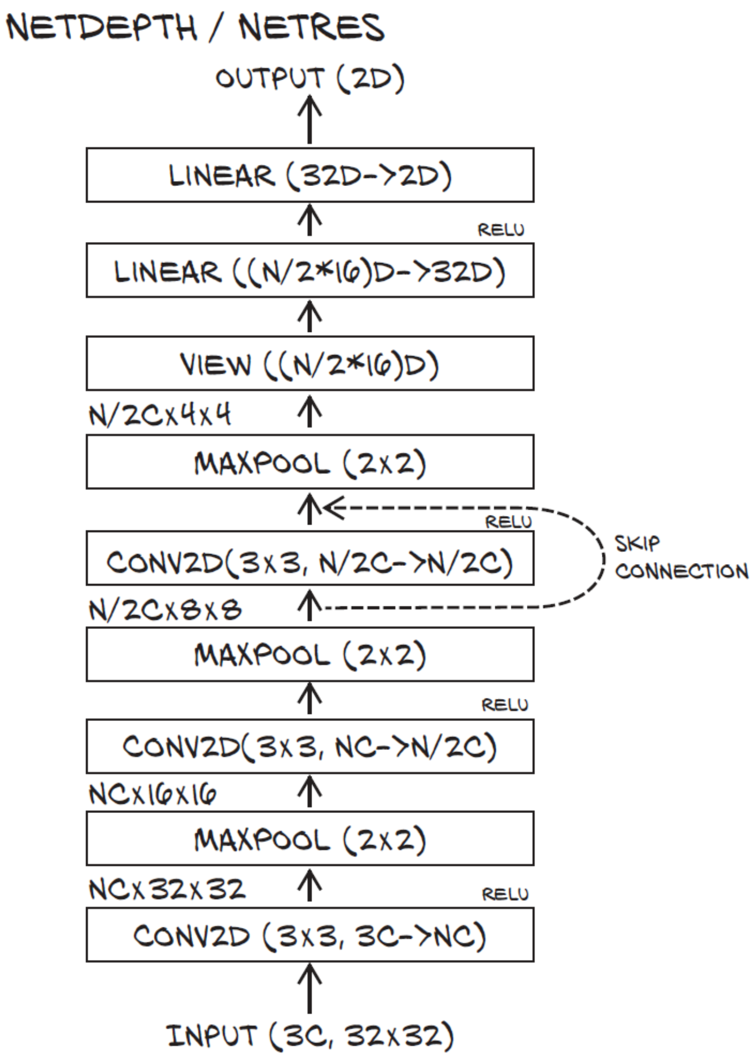
Quelle: Deep Learning with PyTorch

Net Architecture

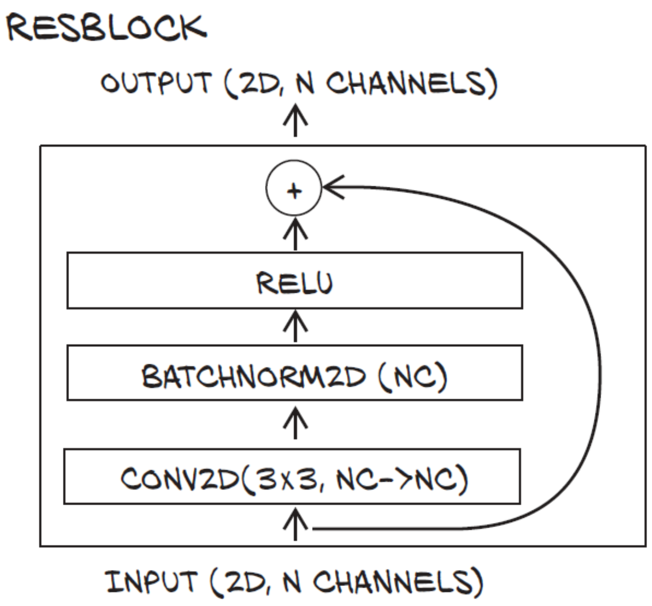


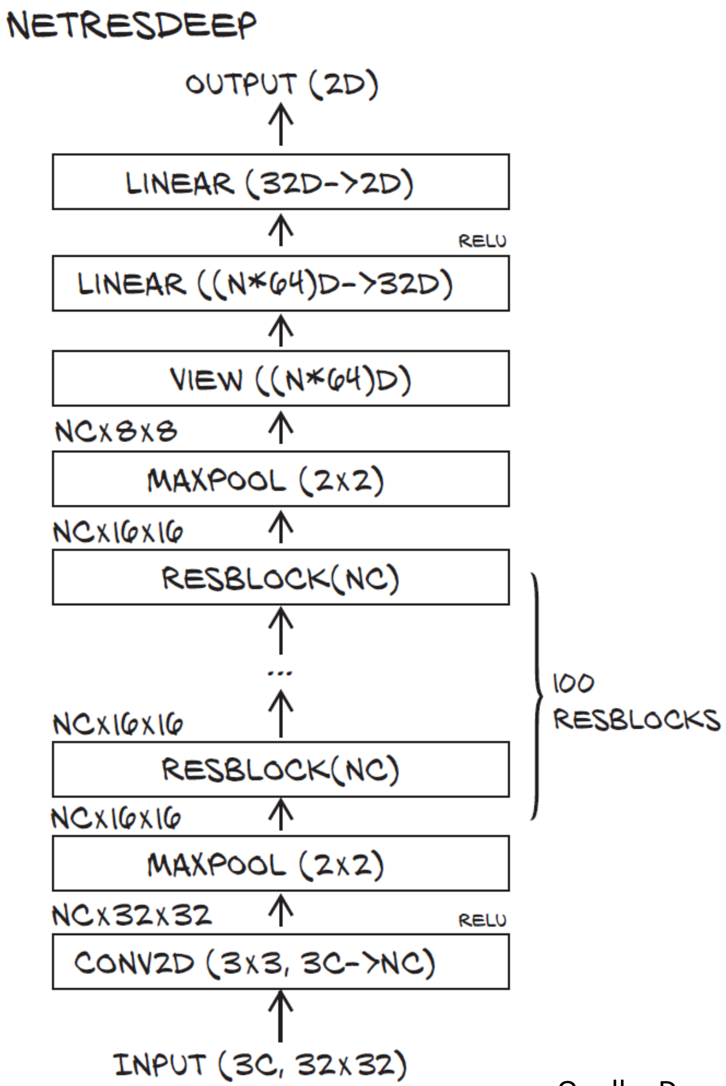
Quelle: Deep Learning with PyTorch

Skip Connections



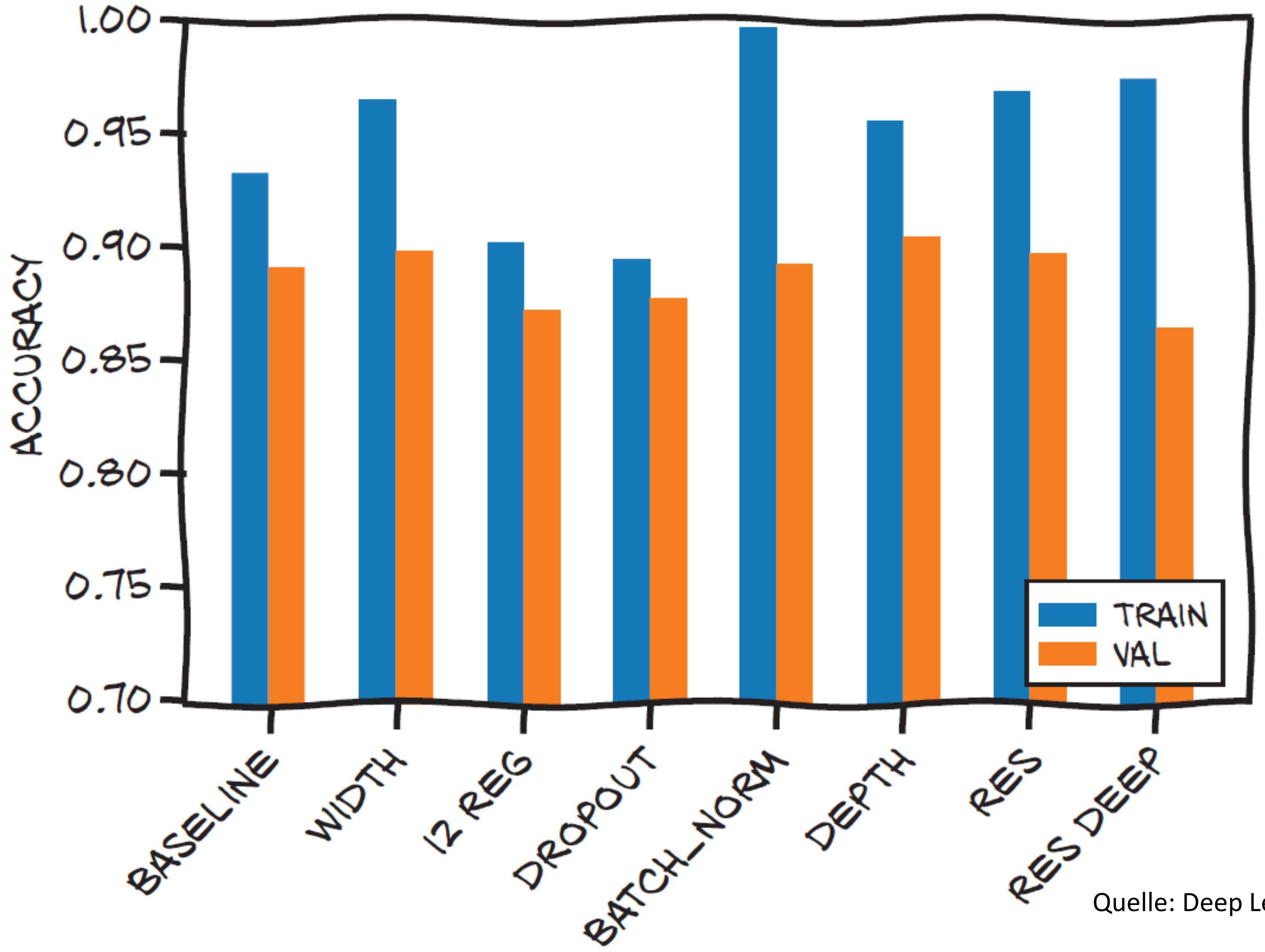
Building Deep Nets





Quelle: Deep Learning with PyTorch

Performance Comparison



Quelle: Deep Learning with PyTorch

Aufgaben

- Modifizieren Sie das Modell so, dass es einen 5x5 Kernel verwendet
  - Wie verändert sich dann die Anzahl der freien Parameter im Modell?
  - Verbessert oder verschlechtert sich Overfitting?
- Können Sie ein Bild finden, welches weder Vogel noch Flugzeug enthält, von dem Netz aber mit mindestens 95% wahrscheinlichkeit als eines der beiden Objekte erkannt wird?

Abgabe per Github bis zum 06.06.2023 23:59 Uhr