UNC CHARLOTTE
The WILLIAM STATES LEE COLLEGE of ENGINEERING

TECSAR
Transformative Computer Systems and Architecture Research

Introduction to ML
Lecture 18: CNNs Model Design

Hamed Tabkhi
Department of Electrical and Computer Engineering,
University of North Carolina Charlotte (UNCC)
htabkhiv@uncc.edu

CNN Design Principles

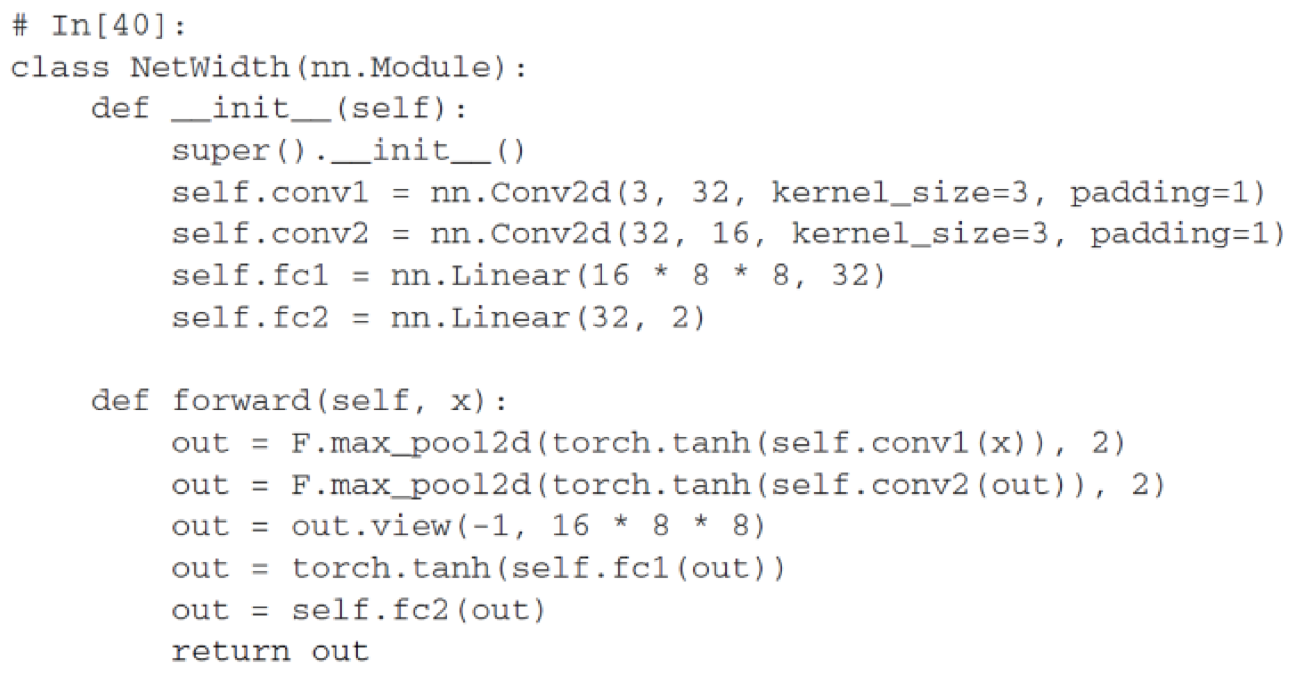
- The promise of neural networks is sufficient flexibility to solve problems on all these kinds of data given the proper architecture (that is, the interconnection of layers or modules) and the proper loss function.

- PyTorch ships with a very comprehensive collection of modules and loss functions to implement state-of-the-art architectures.
- Several models are available through PyTorch Hub or as part of `torchvision` and other vertical community efforts.
- Given our feed-forward architecture, there are a couple of dimensions we'd likely want to explore before getting into further complications.
- In this lecture, we look at three important aspects of CNN design:

(1) Width of Network, (2) Depth of Network, and (3) Regularization

Width of Network

- The first dimension is the *width* of the network. It defines the number of neurons per layer, or channels per convolution.
- We can make a model wider very easily in PyTorch, by just specifying a larger number of
- output channels in the first convolution and increase the subsequent layers accordingly.



```
# In[40]:
class NetWidth(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 16, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(16 * 8 * 8, 32)
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = out.view(-1, 16 * 8 * 8)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

Width of Network

- If we want to avoid hardcoding numbers in the definition of the model, we can easily pass a parameter to *init* and parameterize the width
- We also need to parameterize the call to `view` in the `forward` function.
- The numbers specifying channels and features for each layer are directly related to the number of parameters in a model;
- All other things being equal -> basically, we are increasing the *capacity* of the model

```
# In[42]:
class NetWidth(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.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        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 = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = out.view(-1, 8 * 8 * self.n_chans1 // 2)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

Width of Network

- As we did previously, we can look at how many parameters our model has now:

```
# In[44]:
sum(p.numel() for p in model.parameters())

# Out[44]:
38386
```

- The greater the capacity, the more variability in the inputs the model will be able to manage;
- At the same time, the more likely overfitting will be, since the model can use a greater number of parameters to memorize unessential aspects of the input.

More Complex Structures: Depth of Network

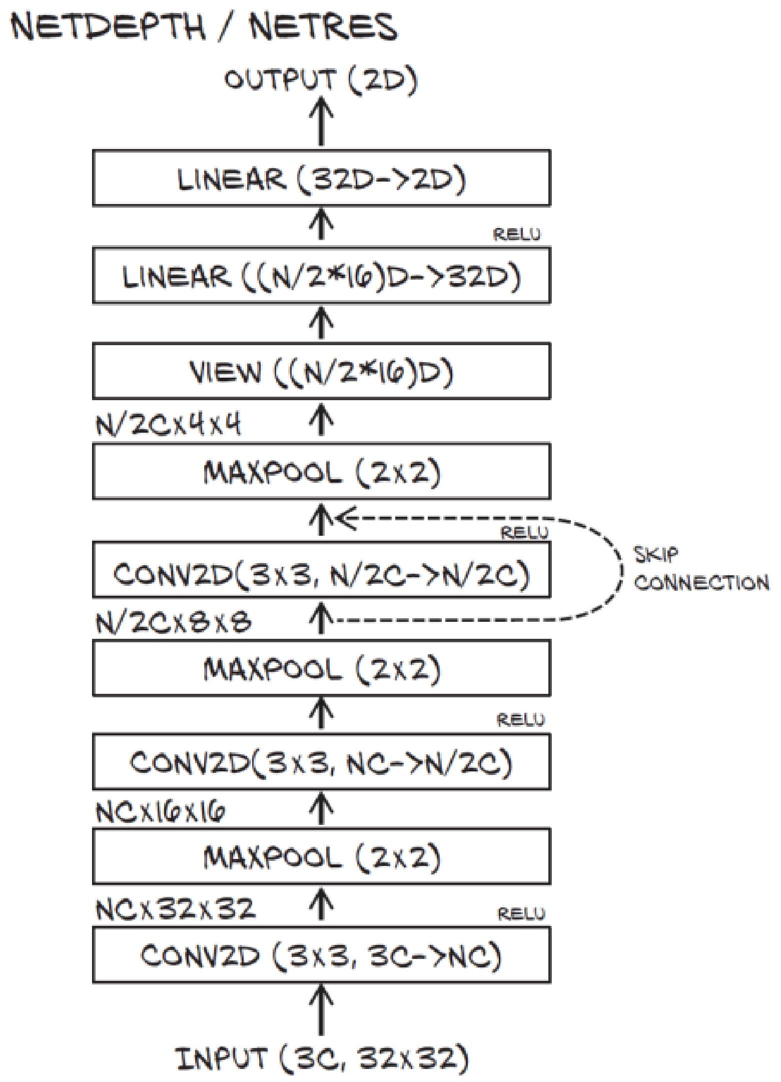
- The second fundamental dimension is obviously, *depth*.
- After all, deeper models are always better than shallow ones.
- With depth, the complexity of the function the network is able to approximate generally increases.
- In regard to computer vision, a shallower network could identify a person’s shape in a photo, whereas a deeper network could identify the person, the face on their top half, and the mouth within the face.
- Depth allows a model to deal with hierarchical information when we need to understand the context in order to say something about some input.

Depth of Network: Skip Connections

- Depth comes with some additional challenges, which prevented deep learning models from reaching 20 or more layers until late 2015.
- Adding depth to a model generally makes training harder to converge.
- For very deep networks, the bottom line is that a long chain of multiplications will tend to make the contribution of the parameter to the gradient *vanish*
- This leads to ineffective training of that layer since that parameter and others like it won’t be properly updated.
- In December 2015, Kaiming He and coauthors presented *Residual Networks* (ResNets), an architecture that uses a simple trick to allow very deep networks to be successfully trained

  https://arxiv.org/abs/1512.03385
- That work opened the door to networks ranging from tens of layers to 100 layers in depth, surpassing the then state of the art in computer vision benchmark problems.

Depth of Network: Skip Connections



- Using a *skip connection* to short-circuit blocks of layers
- **A skip connection is nothing but the addition of the input to the output of a block of layers.**

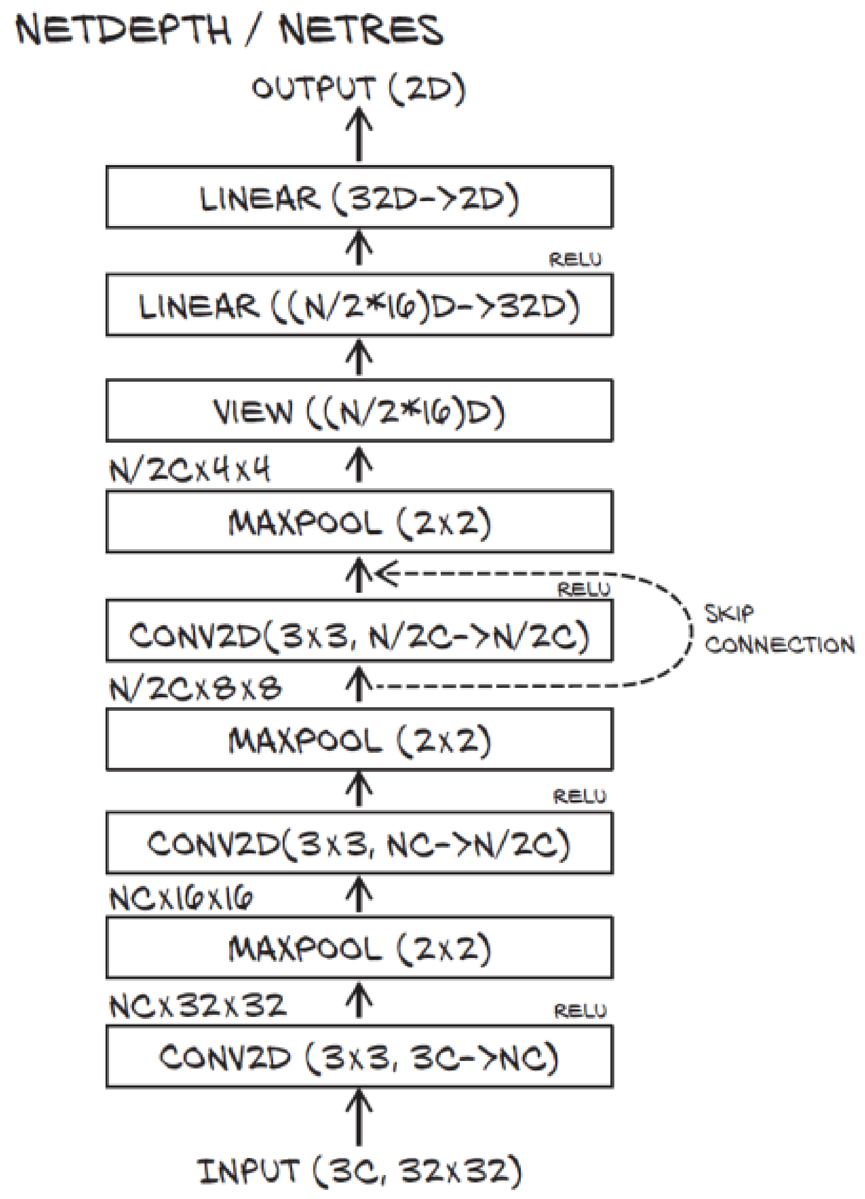
- Adding a skip connection is adding the output of the first layer in the `forward` function to the input of the third layer.
- In other words, we're using the output of the first activations as inputs to the last, in addition to the standard feed-forward path.
- This is also referred to as *identity mapping*.

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

Depth of Network: Skip Connections

```
# In[53]:
class NetRes(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.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        self.conv3 = nn.Conv2d(n_chans1 // 2, n_chans1 // 2,
                               kernel_size=3, padding=1)
        self.fc1 = nn.Linear(4 * 4 * n_chans1 // 2, 32)
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = F.max_pool2d(torch.relu(self.conv2(out)), 2)
        out1 = out
        out = F.max_pool2d(torch.relu(self.conv3(out)) + out1, 2)
        out = out.view(-1, 4 * 4 * self.n_chans1 // 2)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```



Depth of Network: Skip Connections

- Thinking about backpropagation, we can appreciate that a skip connection, or a sequence of skip connections in a deep network, creates a direct path from the deeper parameters to the loss.
- This makes their contribution to the gradient of the loss more direct, as partial derivatives of the loss with respect to those parameters have a chance not to be multiplied by a long chain of other operations.
- It has been observed that skip connections have a beneficial effect on convergence especially in the initial phases of training.
- Also, the loss landscape of deep residual networks is a lot smoother than feed-forward networks of the same depth and width.
- Since the advent of **ResNets**, other architectures have taken skip connections to the next level.
- One in particular, **DenseNet**, proposed to connect each layer with several other layers downstream through skip connections, achieving state-of-the-art results with fewer parameters.