UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING

Introduction to ML
Lecture 5: Modern CNNs

Hamed Tabkhi

Department of Electrical and Computer Engineering,

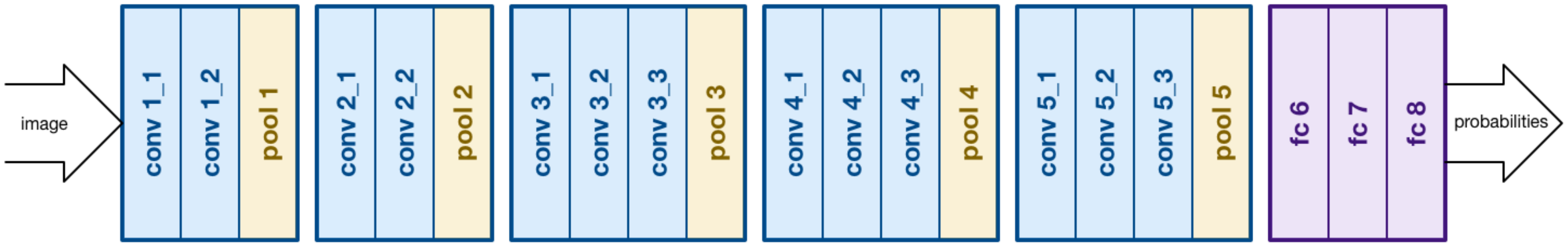
University of North Carolina Charlotte (UNCC)

htabkhiv@uncc.edu

UNC CHARLOTTE

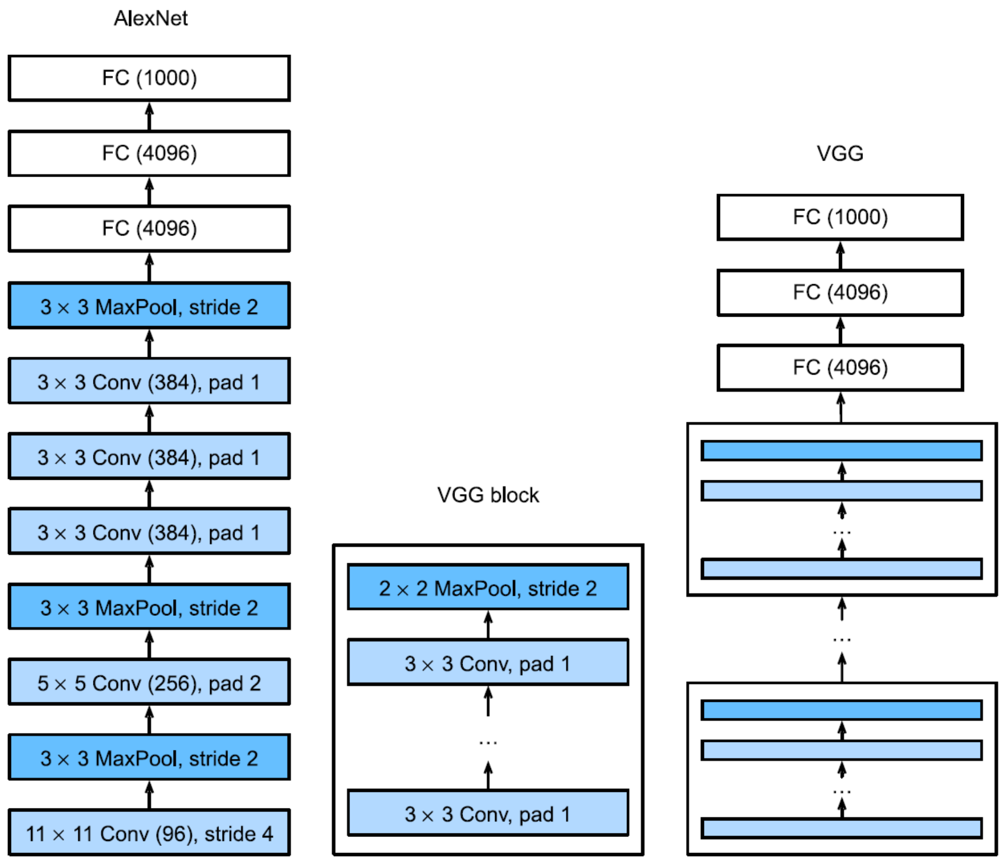
VGG Network

- The key idea of VGGNet was to use *multiple* convolutions in between downsampling via max-pooling in the form of a block.
- They were primarily interested in whether deep or wide networks perform better.
- For instance, the successive application of two 33 convolutions touches the same pixels as a single 55 convolution does.
- In a rather detailed analysis they showed that deep and narrow networks significantly outperform their shallow counterparts.
- This set deep learning on a quest for ever deeper networks with over 100 layers for typical applications.
- Stacking 33 convolutions has become a gold standard in later deep networks.



VGG Block

Back to VGG: a VGG block consists of a *sequence* of convolutions with 3*3 kernels with
padding of 1 (keeping height and width) followed by a 2*2 max-pooling layer with stride of 2 (halving height and width after each block). In the code below, we define a function called vgg_block to implement one VGG block.



VGG-11

- The original VGG network had 5 convolutional blocks, among which the first two have one convolutional layer each and the latter three contain two convolutional layers each.
- The first block has 64 output channels and each subsequent block doubles the number of output channels, until that number reaches 512.
- Since this network uses 8 convolutional layers and 3 fully connected layers, it is often called VGG-11.

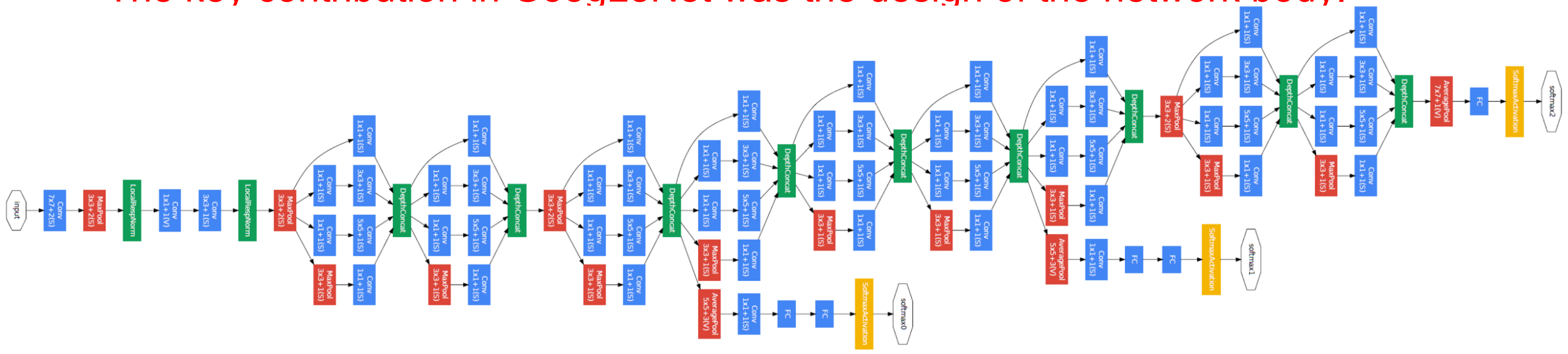
VGG Discussion

- One might argue that VGG is the first truly modern convolutional neural network.
- VGG arguably introduced key properties such as blocks of multiple convolutions and a preference for deep and narrow networks.
- It is also the first network that is actually an entire family of similarly parametrized models, giving the practitioner ample trade-off between complexity and speed.

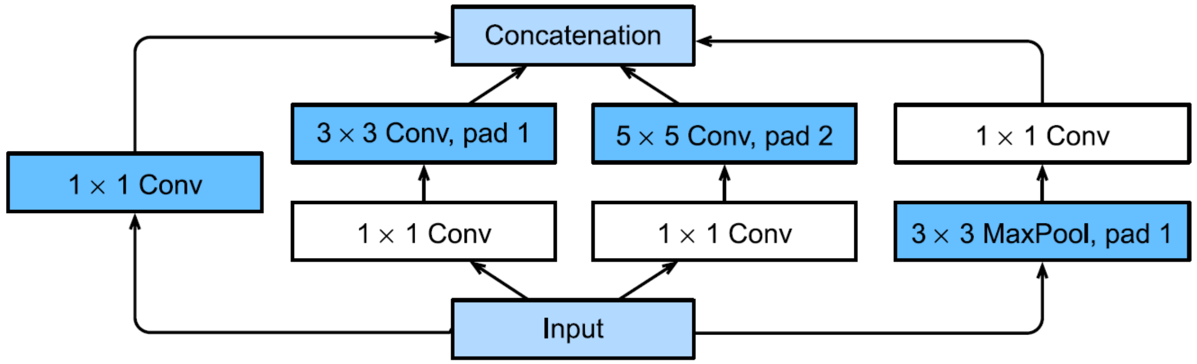
Multi-Branch Networks (GoogleNet)

- In 2014, *GoogLeNet* won the ImageNet Challenge (Szegedy *et al.*, 2015), using a hybrid structure.
- It is arguably also the first network that exhibits a clear distinction among the
- (1) stem (data ingest), (2) body (data processing), and (3) head (prediction) in a CNN.

- This design pattern has persisted ever since in the design of deep networks:
  1. the *stem* is given by the first 2–3 convolutions that operate on the image. They extract low-level features from the underlying images.
  2. This is followed by a *body* of convolutional blocks (Major convolutional blocks)
  3. Finally, the *head* maps the features obtained so far to the required classification, segmentation, detection, or tracking problem at hand.
- The key contribution in GoogLeNet was the design of the network body.



Inception Block

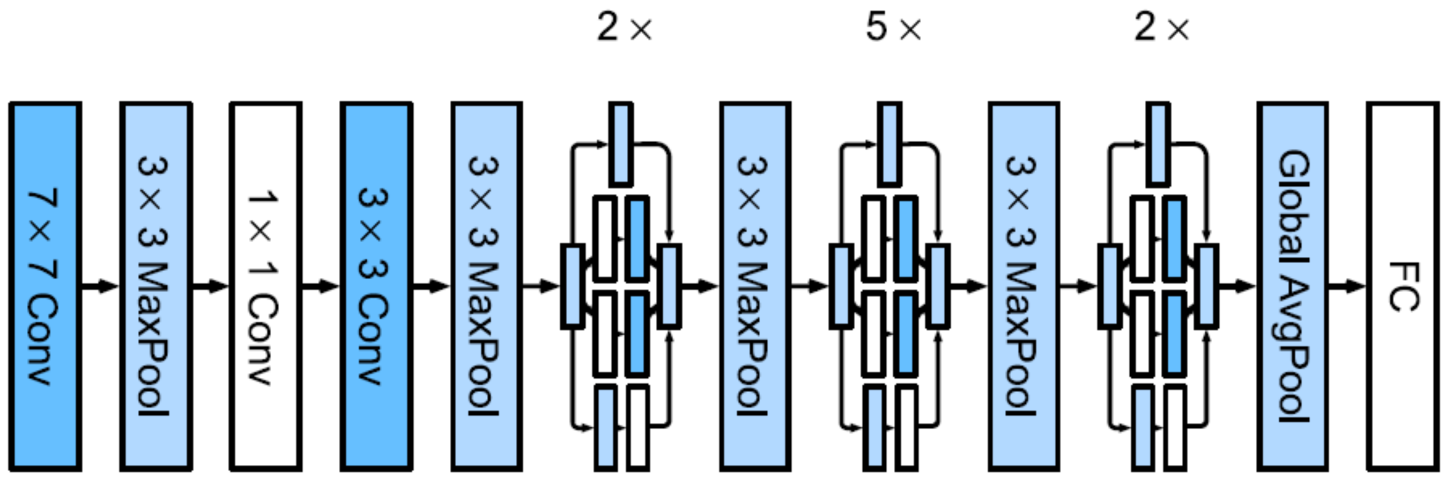


- The basic convolutional block in GoogLeNet is called an *Inception block*, stemming from the meme “we need to go deeper” of the movie *Inception*.
- The inception block consists of four parallel branches:
- The first three branches use convolutional layers with window sizes of 1*1, 3*3, and 5*5 to extract information from different spatial sizes.
- The middle two branches also add a 1 1 convolution of the input to reduce the number of channels, reducing the model’s complexity.
- The fourth branch uses a 3*3 max-pooling layer, followed by a 1*1 convolutional layer to change the number of channels.
- The four branches all use appropriate padding to give the input and output the same height and width.
- Finally, the outputs along each branch are concatenated along the channel dimension and comprise the block’s output.

UNC CHARLOTTE

GoogLeNet Model

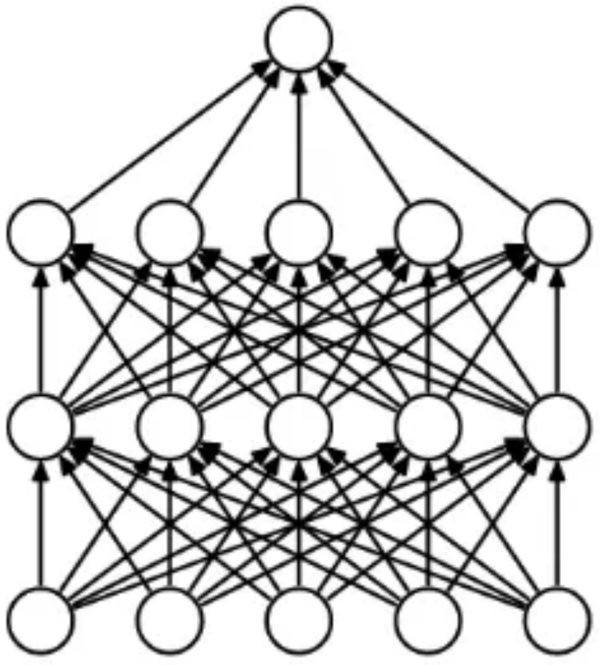
- GoogLeNet uses a stack of a total of 9 inception blocks, arranged into 3 groups with max-pooling in between, and global average pooling in its head to generate its estimates.
- Max-pooling between inception blocks reduces the dimensionality.
- At its stem, the first module is similar to AlexNet and LeNet.

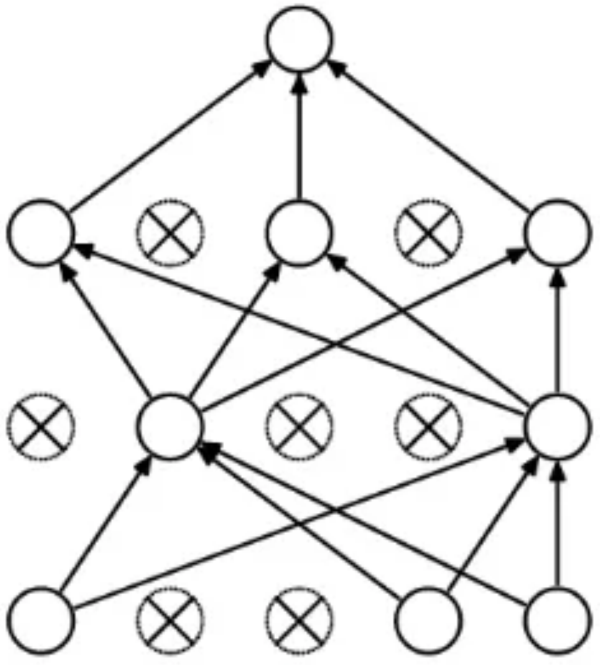


Regularization Techniques

- **Dropout!** → helps reduse overfitting
  → overfitting is more likely when network is too large for amount of data
  - we typically prefer larger networks, so this is where dropout comes in
- **Batch Normalization!**
  - Basically scales data again for each given batch
  - ensures a single channel doesn't take over training
  - normalize feature map per batch
  - Captures dependencies within batch
  - Put after neuron, but before activation

DropOut

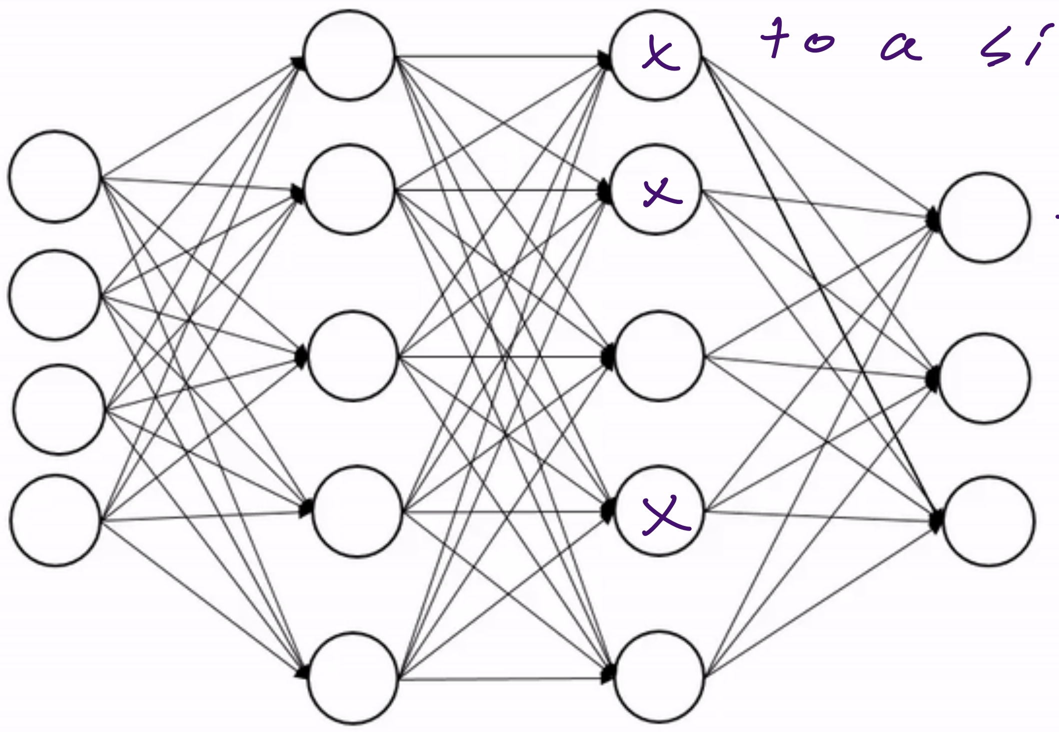

(a) Standard Neural Net


(b) After applying dropout.

- *The term "dropout" refers to dropping out the nodes (input and hidden layer) in a neural network.*
- *All the forward and backward connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network.*
- *The nodes are dropped by a dropout probability of p.*

UNC CHARLOTTE

DropOut



- we can apply dropout to a single layer

- Never apply dropout on input layer

Generally, for the input layers, the keep probability, i.e. 1- drop probability, is closer to 1, 0.8 being the best as suggested by the authors. For the hidden layers, the greater the drop probability more sparse the model, where 0.5 is the most optimised keep probability, which states dropping 50% of the nodes.

UNC CHARLOTTE

How does it solve the Overfitting problem?

- In the overfitting problem, the model learns the statistical noise. To be precise, the main motive of training is to decrease the loss function, given all the units (neurons).
- So, in overfitting, a unit may change in a way that fixes up the mistakes of the other units. This leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalize on the unseen dataset.
- Dropout prevents these units from fixing the mistakes of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable.
-
- By randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.
- This ensures that the model is getting generalized and hence reducing the overfitting problem.

Batch Normalization

- Dropout was all the rage when, in 2015, another seminal paper was published by Sergey Ioffe and Christian Szegedy from Google, entitled

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" (https://arxiv.org/abs/1502.03167).

- The paper described a technique that had multiple beneficial effects on training: allowing us to increase the learning rate and make training less dependent on initialization.
- It acts as a regularizer, thus representing an alternative to dropout.
- The main idea behind batch normalization is to rescale the inputs to the activations of the network so that minibatches have a certain desirable distribution.
- Recalling the mechanics of learning and the role of nonlinear activation functions, this helps avoid the inputs to activation functions being too far into the saturated portion of the function, thereby killing gradients and slowing training.

Regularization: Batch Normalization

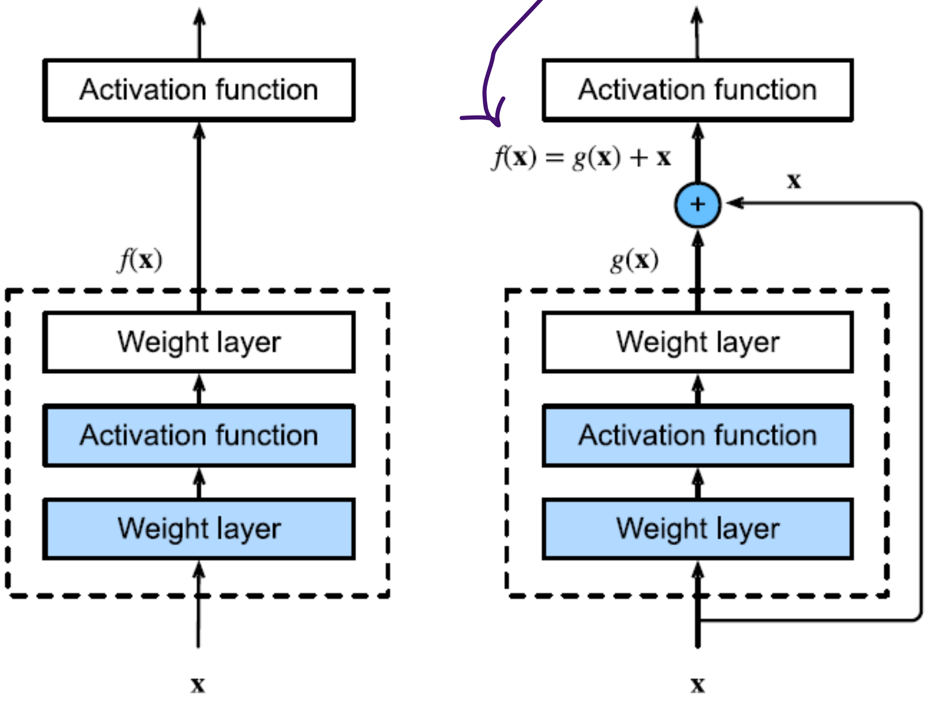
- In practical terms, batch normalization shifts and scales an intermediate input using the mean and standard deviation collected at that intermediate location over the samples of the minibatch.
- The regularization effect is a result of the fact that an individual sample and its downstream activations are always seen by the model as shifted and scaled, depending on the statistics across the randomly extracted minibatch.
- The authors of the paper suggest that using batch normalization eliminates or at least alleviates the need for dropout
- Since the aim for batch normalization is to rescale the inputs of the activations, the natural location is after the linear transformation (convolution, in this case).

Batch Normalization for Inference (Validation)

- Just as for dropout, batch normalization needs to behave differently during training and inference.
- In fact, at inference time, we want to avoid having the output for a specific input depend on the statistics of the other inputs we’re presenting to the model.
- As such, we need a way to normalize, but this time fixing the normalization parameters once and for all.
- As minibatches are processed, in addition to estimating the mean and standard deviation for the current minibatch, PyTorch also updates the running estimates for mean and standard deviation that are representative of the whole dataset, as an approximation..
- This way, when the user specifies and the model contains a batch normalization module, the running estimates are frozen and used for normalization during the Inference (validation)

Residual Blocks

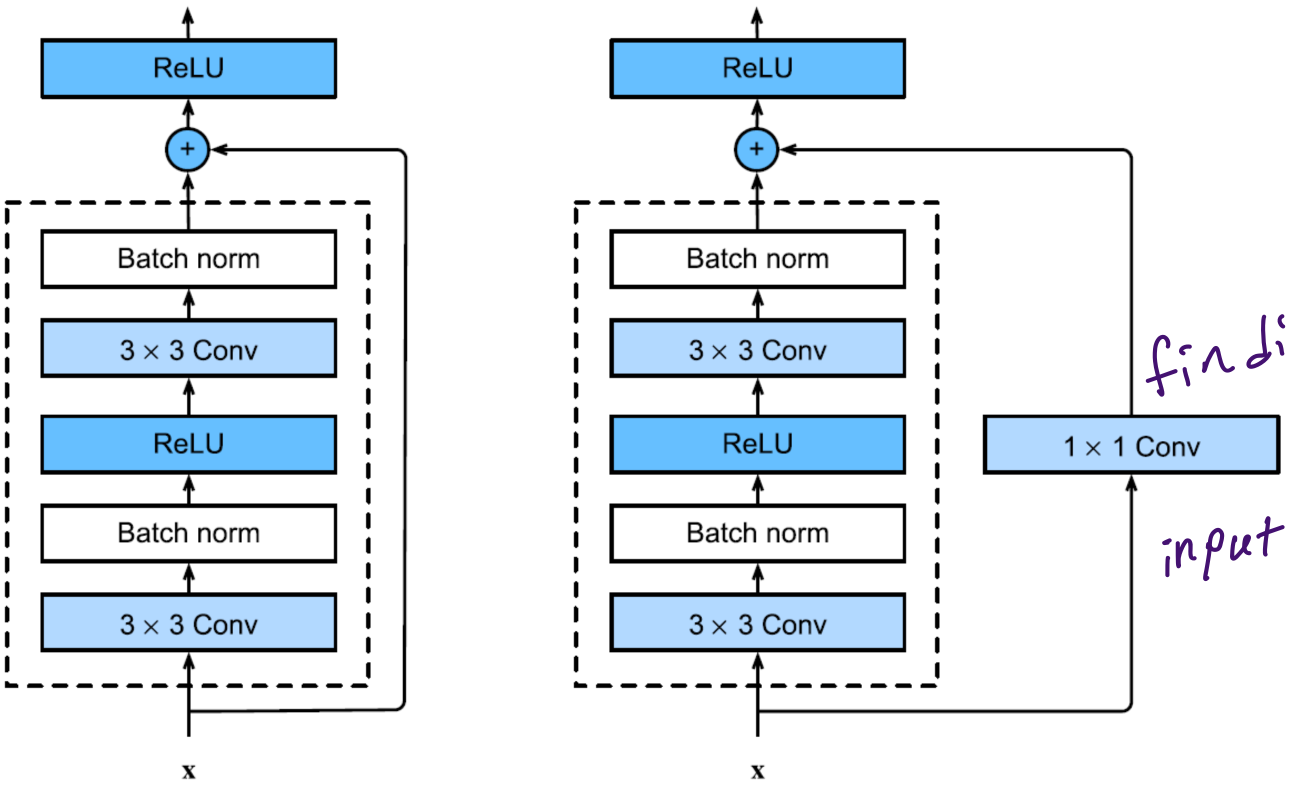
- In a regular block (left), the portion within the dotted-line box must directly learn the mapping f(**x**).
- In a residual block (right), the portion within the dotted-line box needs to learn the residual mapping ~~g(**x**) = f(**x**) + **x**~~, making the identity mapping f(**x**) = **x** easier to learn.



UNC CHARLOTTE

Residual Blocks

ResNet block with and without 1*1 convolution, which transforms the input into the desired shape for the addition operation.



finding correlations between inputs and outputs

UNC CHARLOTTE

Residual Blocks

- ResNet follows VGG's full 3*3 convolutional layer design.
- The residual block has two 3*3 convolutional layers with the same number of output channels.
- Each convolutional layer is followed by a batch normalization layer and a ReLU activation function.
- Then, we skip these two convolution operations and add the input directly before the final ReLU activation function.
- This kind of design requires that the output of the two convolutional layers has to be of the same shape as the input, so that they can be added together.
- If we want to change the number of channels, we need to introduce an additional 1*1 convolutional layer to transform the input into the desired shape for the addition operation.

ResNet Model: First two layers

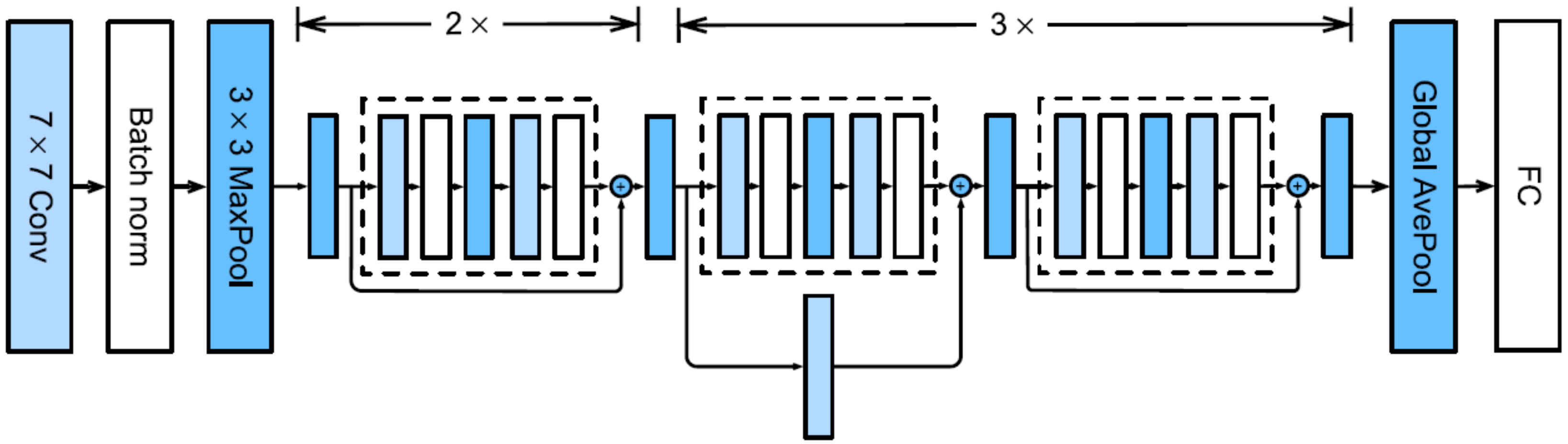
- The first two layers of ResNet are the same as those of the GoogLeNet we described before:
- The 7*7 convolutional layer with 64 output channels and a stride of 2 is followed by the 3 max-pooling layer with a stride of 2.
- The difference is the batch normalization layer added after each convolutional layer in ResNet.

- ResNet's body is composed of multiple modules which each has multiple residual blocks
- The number of channels in the first module is the same as the number of input channels.
- In the first residual block for each of the subsequent modules, the number of channels is doubled compared with that of the previous module, and the height and width are halved.

- Then, we add all the modules to ResNet.
- Lastly, just like GoogLeNet, we add a global average pooling layer, followed by the fully connected layer output.

ResNet Architecture

- There are 4 convolutional layers in each module (excluding the 1*1 convolutional layer).
- Together with the first 7*7 convolutional layer and the final fully connected layer, there are 18 layers in total! Therefore, this model is commonly known as ResNet-18!
- By configuring different numbers of channels and residual blocks in the module, we can create different ResNet models, such as the deeper 152-layer ResNet-152.
- Although the main architecture of ResNet is similar to that of GoogLeNet, ResNet's structure is simpler and easier to modify.
- All these factors have resulted in the rapid and widespread use of ResNet.



ResNet Training

- We train ResNet on the Fashion-MNIST dataset, just like before.
- ResNet is quite a powerful and flexible architecture.
- The plot capturing training and validation loss illustrates a significant gap between both graphs, with the training loss being significantly lower.
- For a network of this flexibility, more training data would offer significant benefit in closing the gap and improving accuracy.