CAB420: Network Depth

AND IT’S ADVANTAGES AND PITFALLS

Convolutional Neural Networks

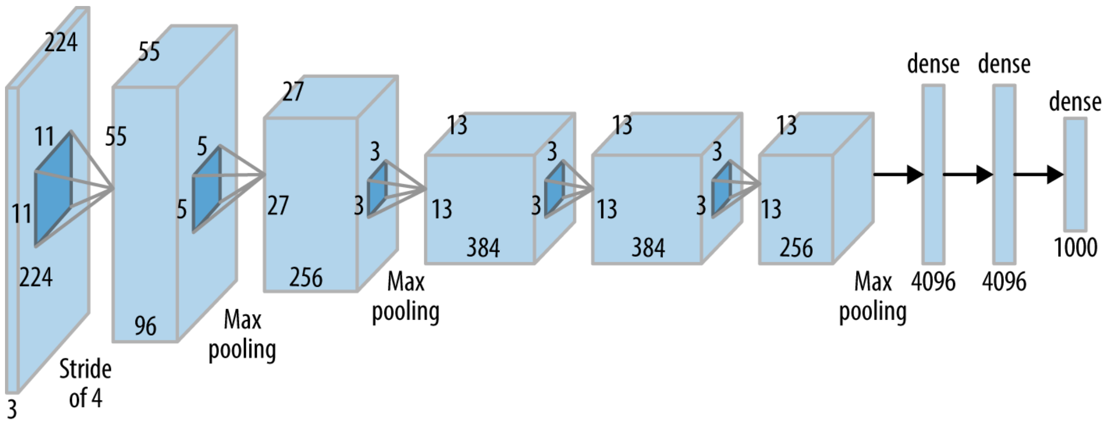
- Neural networks using convolutional filters
- Convolutional filters learn local spatial relationships
- We typically stack convolutional layers, allowing us to learn more complex patterns
- Deeper networks can learn more complex patterns
  - But get increasingly slow to train

Convolution Layers and Filters

- Convolutional layers learn filters that are applied to the input signal
- Filters are of a fixed size
  - Set during network design
  - Usually, odd numbered square size
- Size controls
  - How much of an input a filter operates over
    - Bigger filters see more of an input at a time
  - How many parameters are in the filter
    - Bigger filters mean more parameters

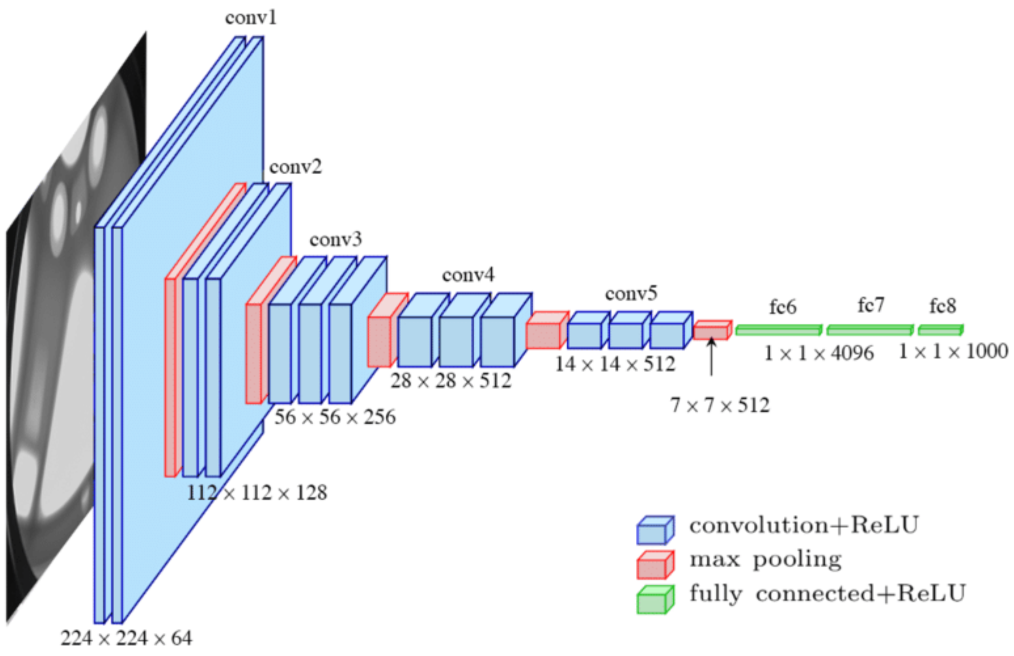
AlexNet

- AlexNet is where DCNNs really got started
  - 8 computational layers (really not that deep)
    - 5 convolutional layers
    - 3 fully connected layers
  - Large filters in early convolution layers
    - 11x11 filters in first layer (96 filters)
    - 5x5 filters in second layer (256 filters)



VGG Style Networks

- So far, we've played (mostly) with VGG-style networks
  - Two (or more) small (3x3) convolutions, followed by a max-pool
  - Rinse and repeat
  - End with FC layers

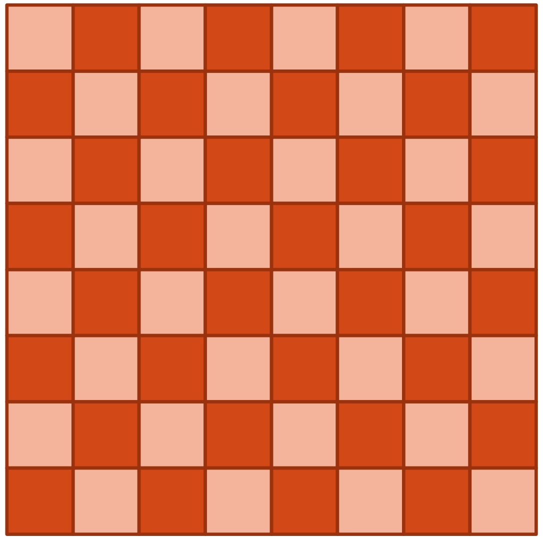


Why Small Convolutions?

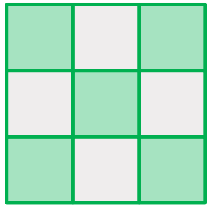
- Why do we use 3x3 convolutions rather than bigger filter sizes?
- Intuitively
  - Smaller filters see smaller patches of the image
  - They extract more localised features
  - They can’t extract or “see” large patches of the image, and thus should miss large features
- But
  - If we stack them deep enough, we overcome this
- And
  - Because they’re smaller, they have fewer parameters, and are easier to learn

Receptive Fields

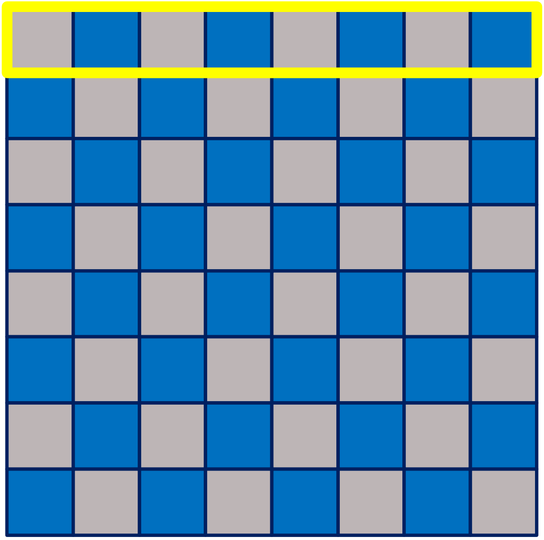
- How much of an image a filter can effectively see
- Consider the following
  - Input image, followed by
  - 3x3 Conv2D layer



Input Image



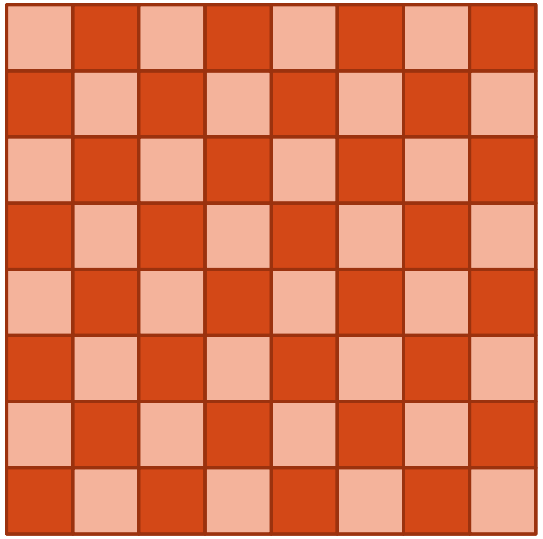
3x3 Conv Filter



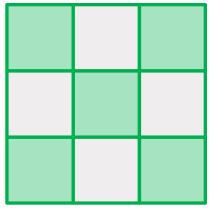
Output Features

Receptive Fields

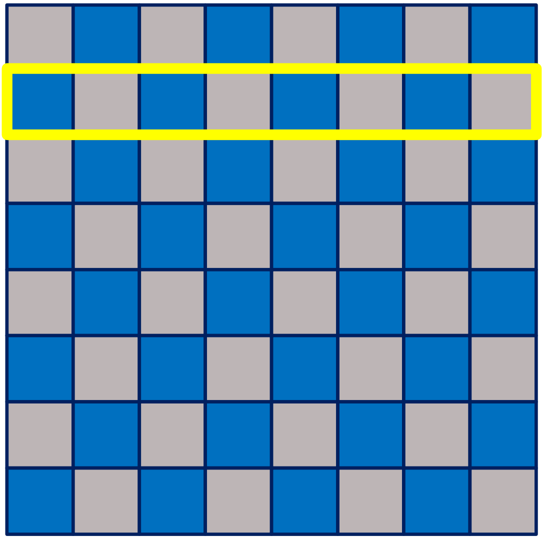
- How much of an image a filter can effectively see
- Consider the following
  - Input image, followed by
  - 3x3 Conv2D layer



Input Image



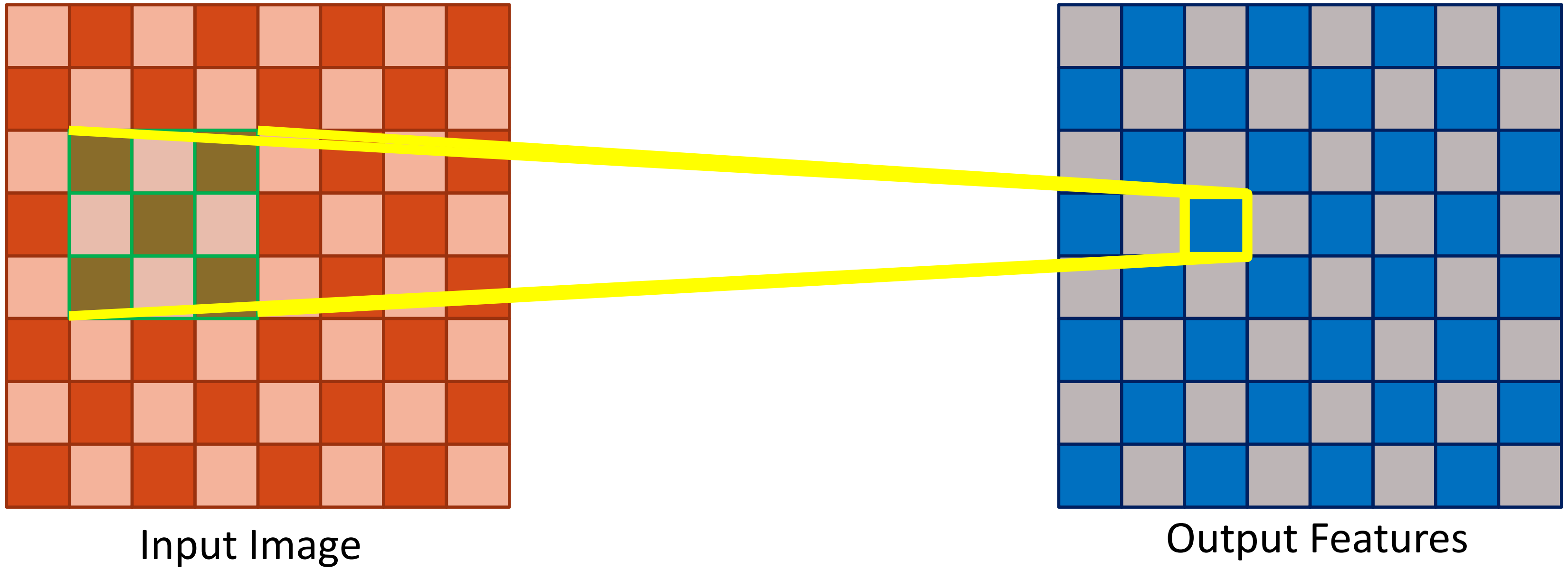
3x3 Conv Filter



Output Features

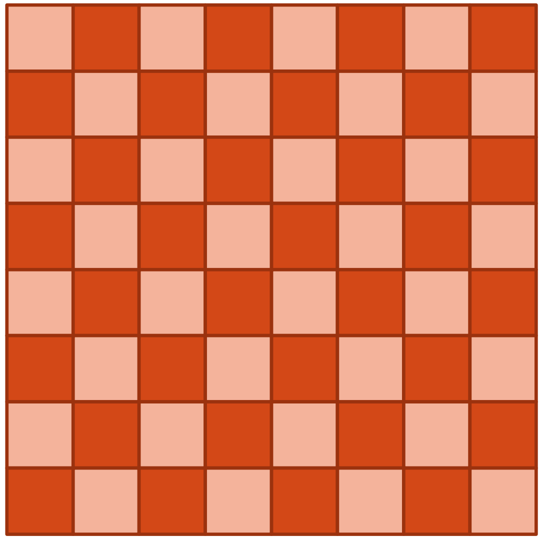
Receptive Fields

- How much of an image a filter can effectively see
- Consider the following
  - Input image, followed by
  - 3x3 Conv2D layer

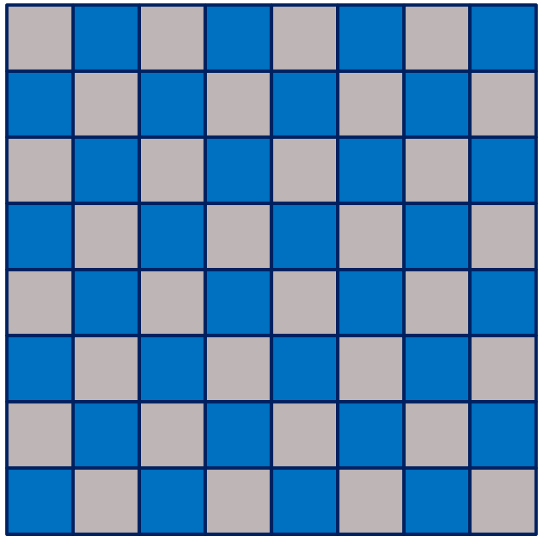


Receptive Fields

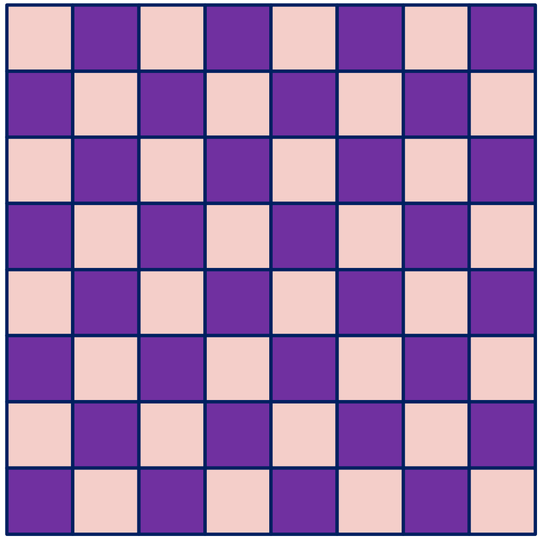
- Now consider
  - Input image, followed by
  - 3x3 Conv2D layer, followed by
  - 3x3 Conv2D layer



Input Image



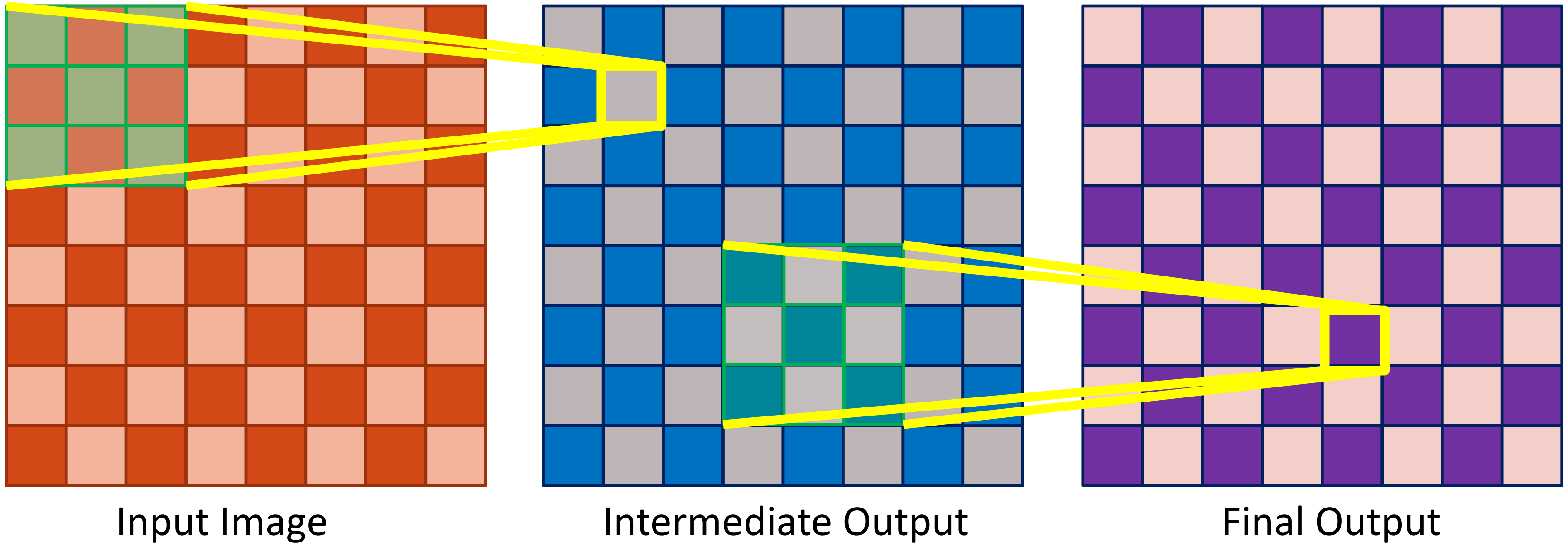
Intermediate Output



Final Output

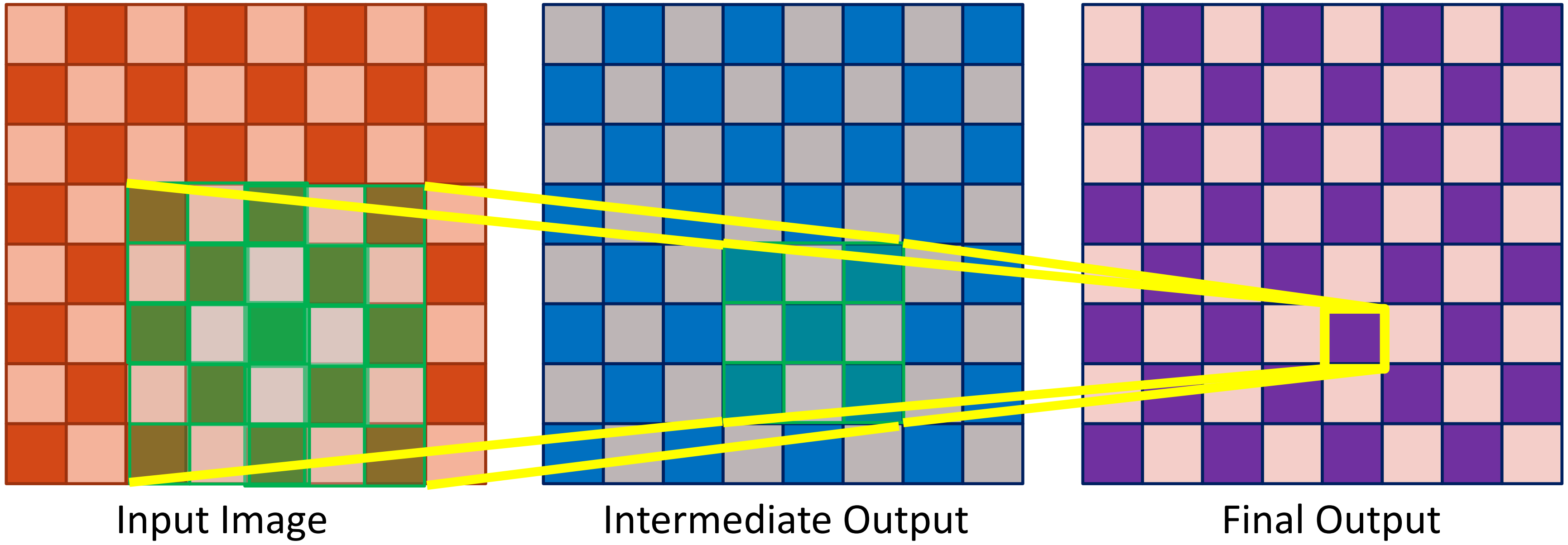
Receptive Fields

- Now consider
  - Input image, followed by
  - 3x3 Conv2D layer, followed by
  - 3x3 Conv2D layer



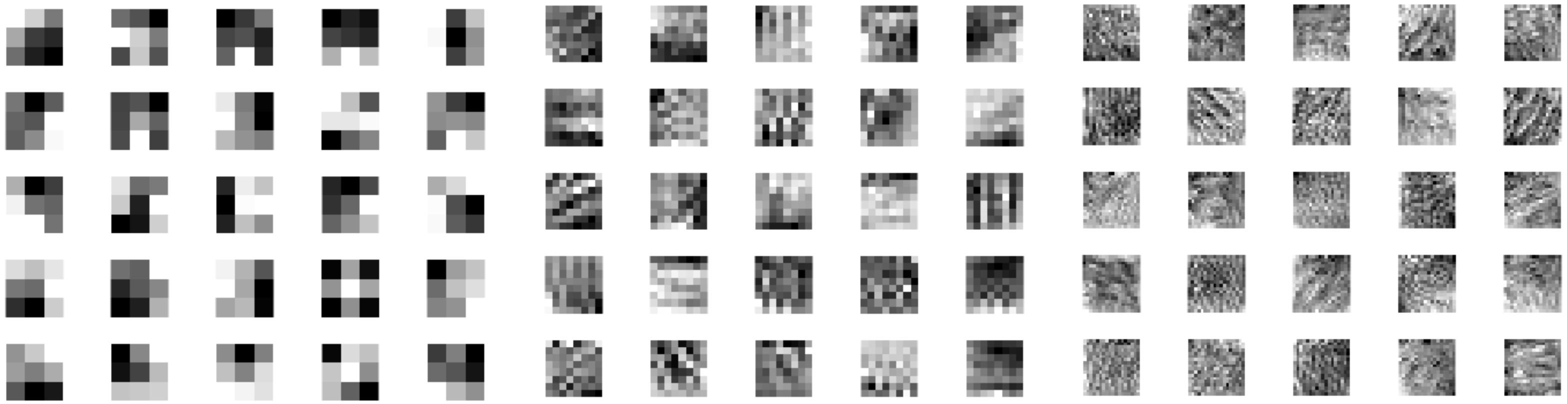
Receptive Fields

- Now consider
  - Input image, followed by
  - 3x3 Conv2D layer, followed by
  - 3x3 Conv2D layer



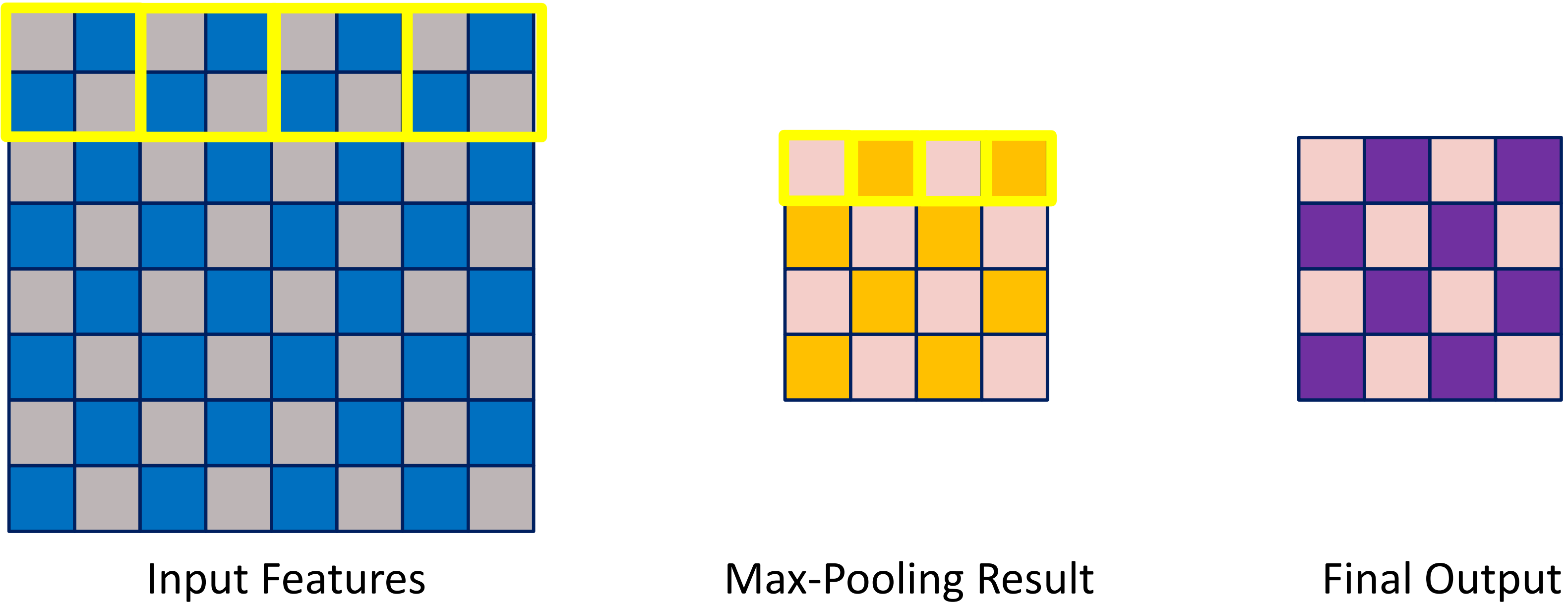
Stacking Convolutions

- As we stack convolutions, we get bigger receptive fields
- But, why use two 3x3 layers over one 5x5 layer?
  - With multiple filters we learn more complex representations
  - Filters “build” on each other
  - Below (left to right)
    - conv1, conv3 and conv5 from a simple VGG trained on Fashion MNIST



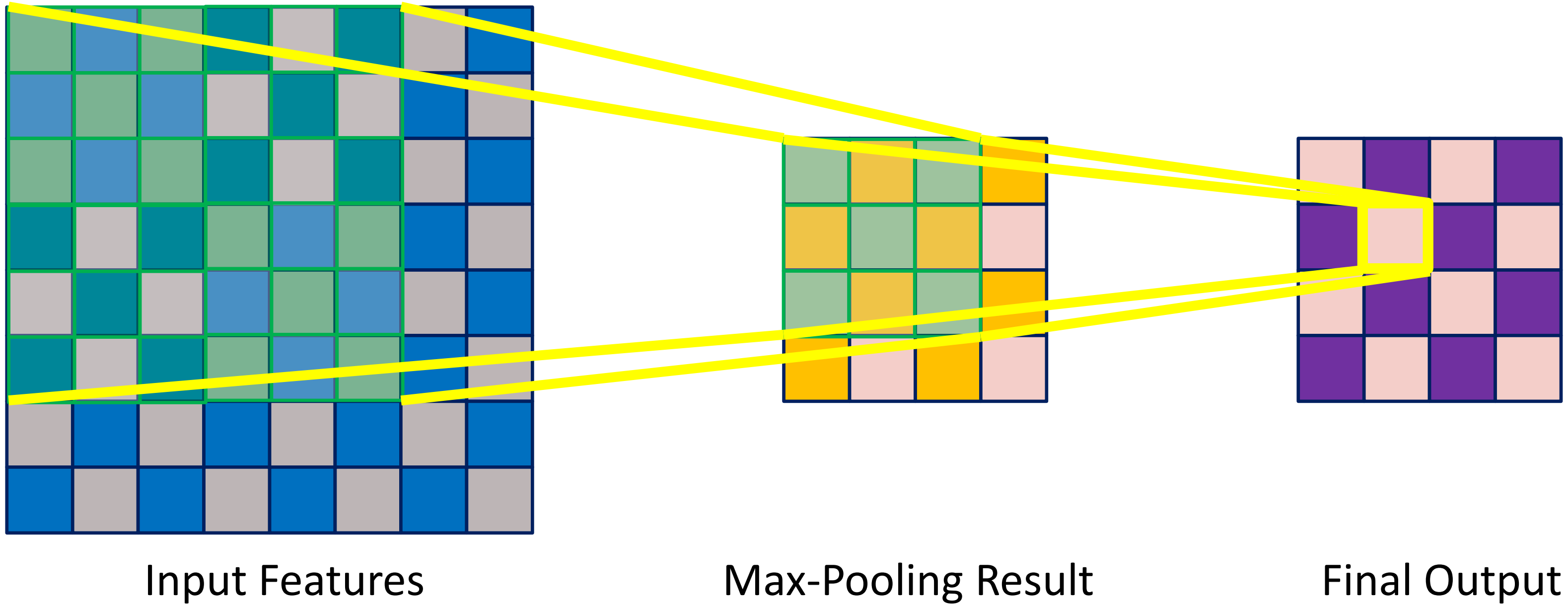
Convolutions with Max-Pooling

- Now consider
  - 3x3 Convolution Layer
  - 2x2 Max-Pooling
  - 3x3 Convolution Layer



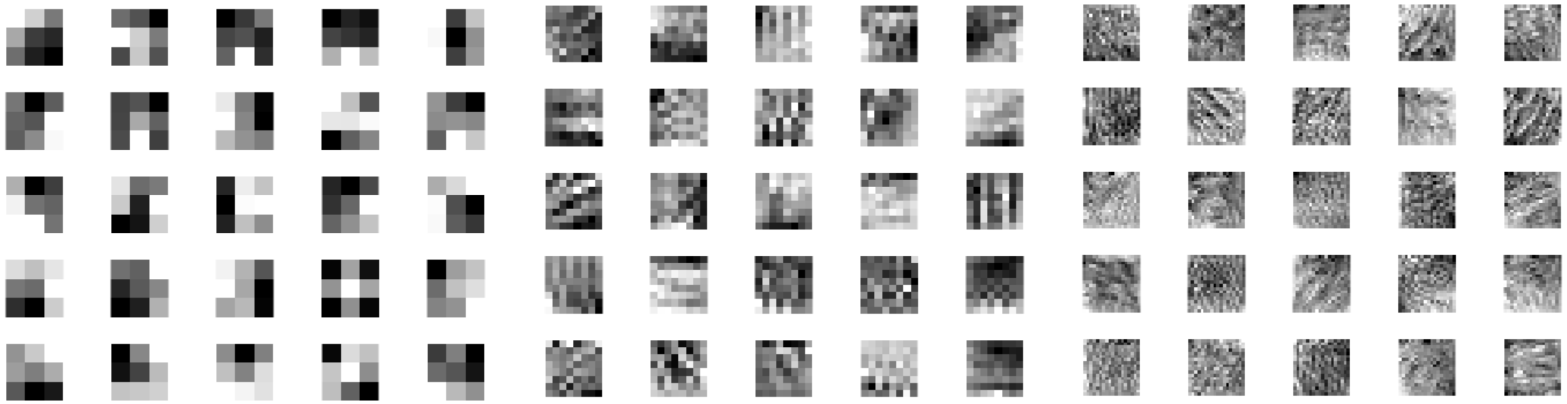
Convolutions with Max-Pooling

- Now consider
  - 3x3 Convolution Layer
  - 2x2 Max-Pooling
  - 3x3 Convolution Layer



3x3 Convolutions For Life?

- Mostly…
  - In, general, multiple layers of smaller filters is better
- But, sometimes…
  - You don't have data to train that many layers
    - Few layers of larger filters may be better given the constraints
  - Your data may have no fine/small details
    - Initial filters at least should be bigger to capture relevant details
  - You may have particular considerations around filter stride, padding, output size, etc.
    - Filter parameters become a function of other constraints



Problems with VGG-Style Nets

- We want more depth to learn higher level representations
  - Extract more complex concepts as we go deeper

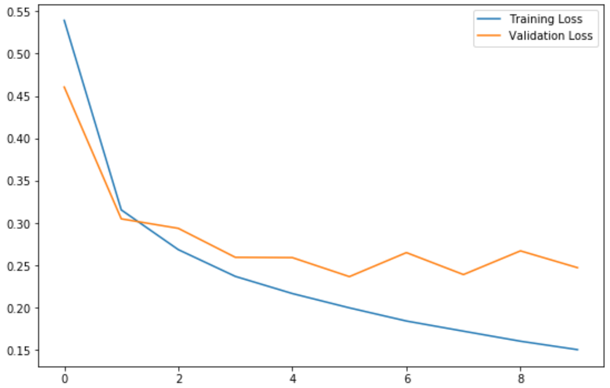
- But as we go deeper, it's harder to train
  - Data has to go through a lot of layers
  - Changes in early layers impact later layers
  - Training can collapse
  - BatchNorm can help, but won't totally solve the problem
  - Performance tends to max out (and go backwards) somewhere between 20-50 layers
    - Exact point depends on data, etc.

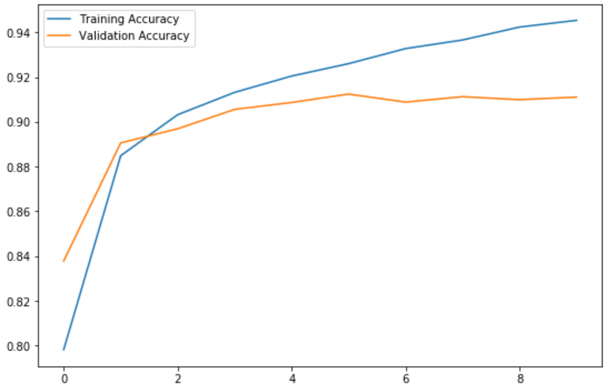
Breaking VGG

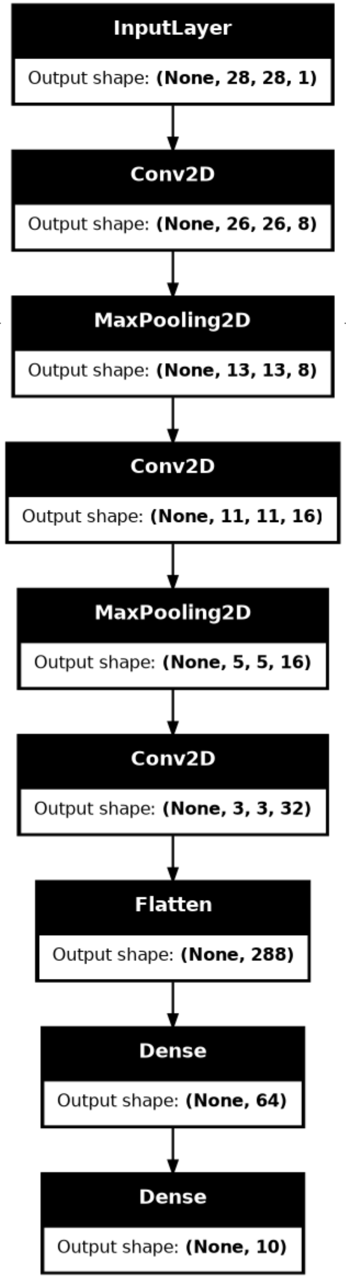
- See ***CAB420_DCNNs_Additional_Example_6_Breaking_VGG***
- Data
  - Fashion MNIST, of course
- Task
  - Standard classification, though with increasingly deep networks

A Simple Network

- 3 Convolutions
  - Max pooling after each
- Model trains and works as expected

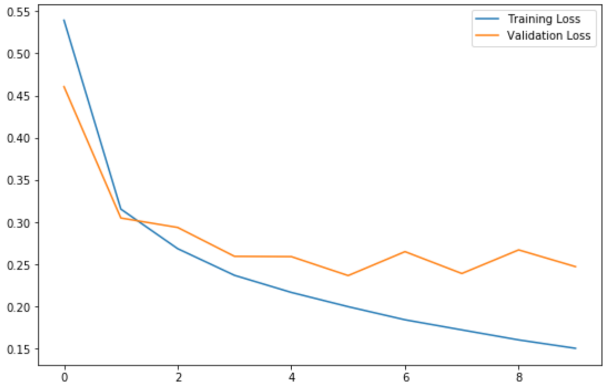


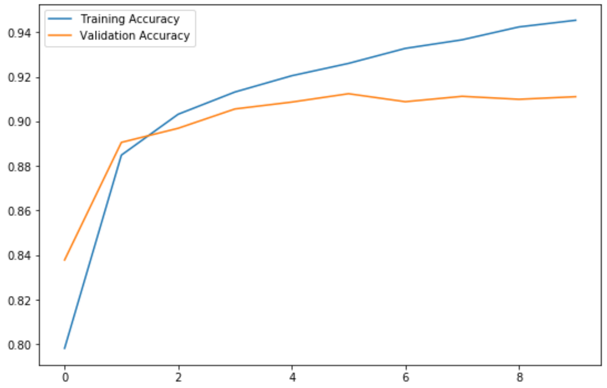


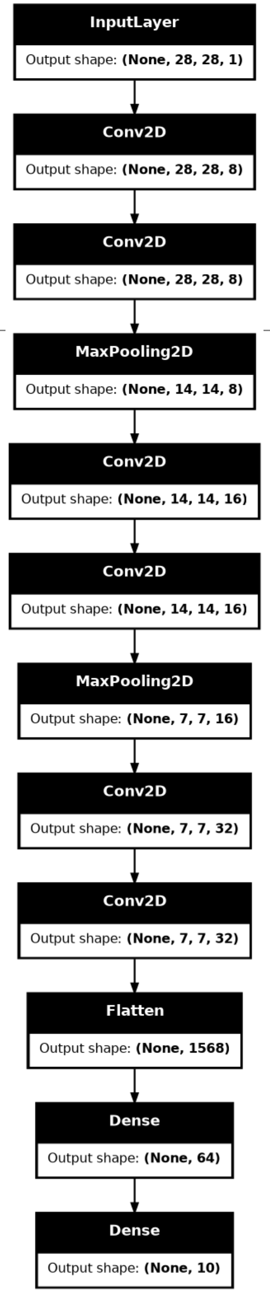


VGG Style Network

- 3 pairs of convolutions
  - Max pooling after each pair
- Model trains and works as expected
  - Better than our simple network

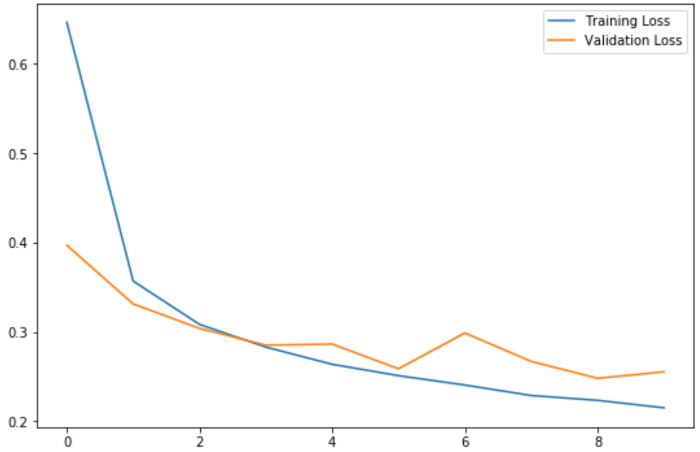


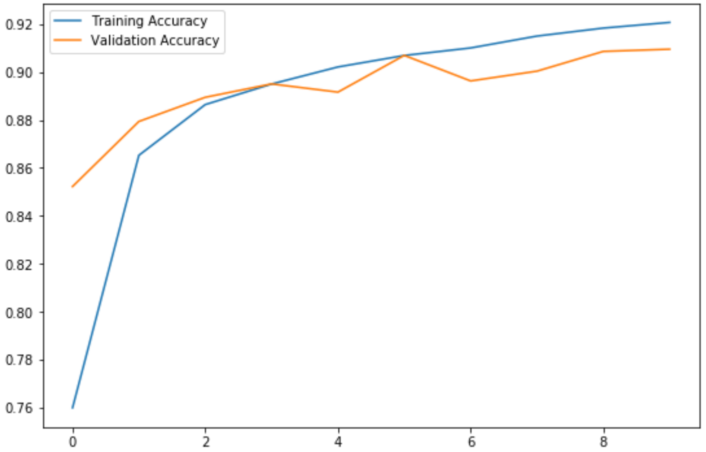


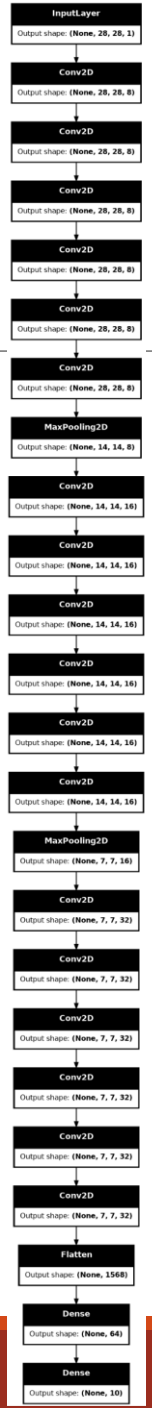


Getting Silly

- 3 sets of 6 stacked convolution layers
  - Max pooling after each set
- Still works
  - Training takes a while though
  - Not really any better

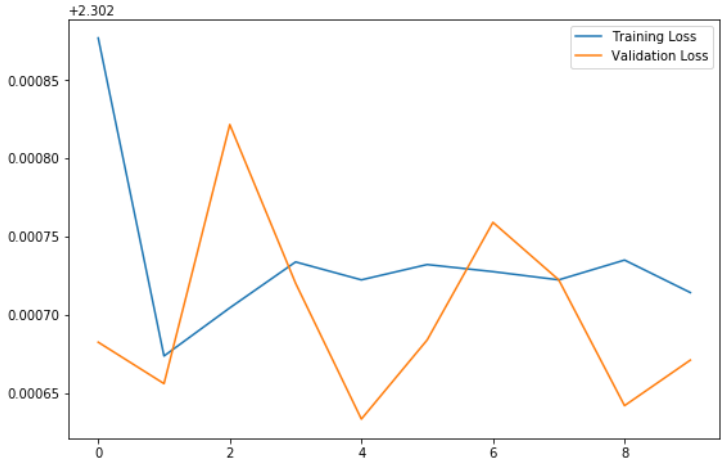


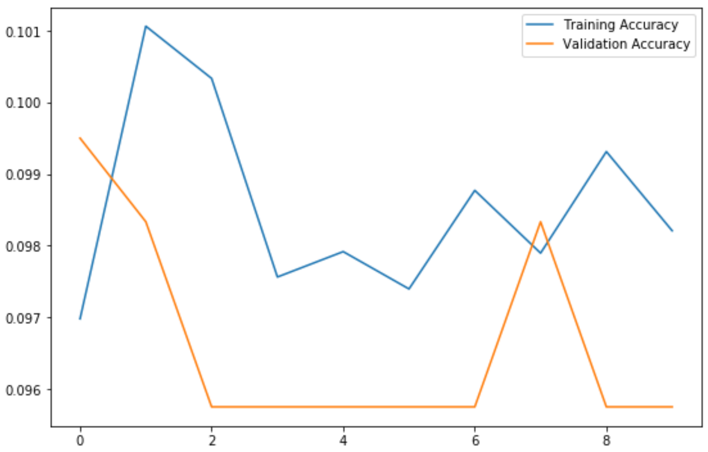


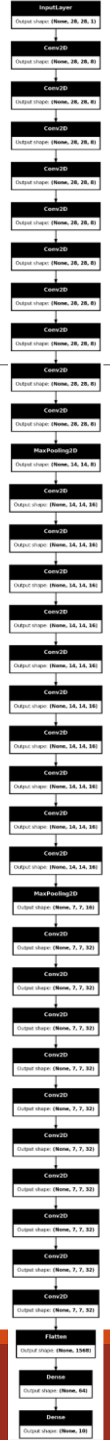


Getting Very Silly

- 3 sets of 10 convolution layers
  - Max pooling after each set
- Training fails
  - Classifier never gets better than chance







Stop that, it's silly

- Networks train via back-propagation
  - The error at the output is propagated back towards the input
  - If our network gets too deep, the gradients we use to adapt the network vanish
    - i.e. there is not enough information to update any parameters, and thus the network stops learning
- We can increase the viable depth slightly though batch normalisation, but this won't get us too much deeper
  - We need a way to get a more direct connection between the network output and input

CAB420: ResNet

LESS SILLY, MORE DEEP

ResNet

- Residual Networks
  - Introduces the idea of skip connections
  - Makes it easier to push data though the network, in particular for deep networks earlier in training
    - Skip connection may need to adjust the dimensionality in the identity block for the addition to be valid

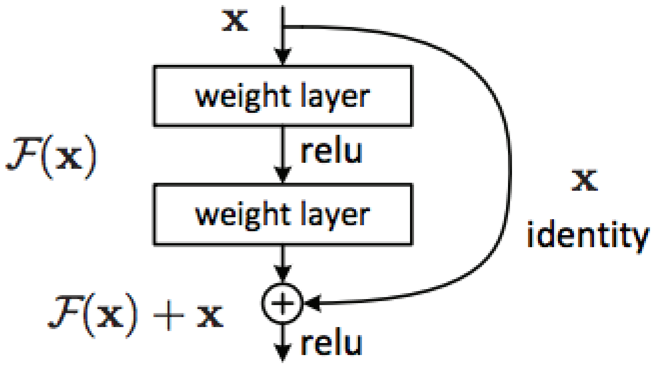
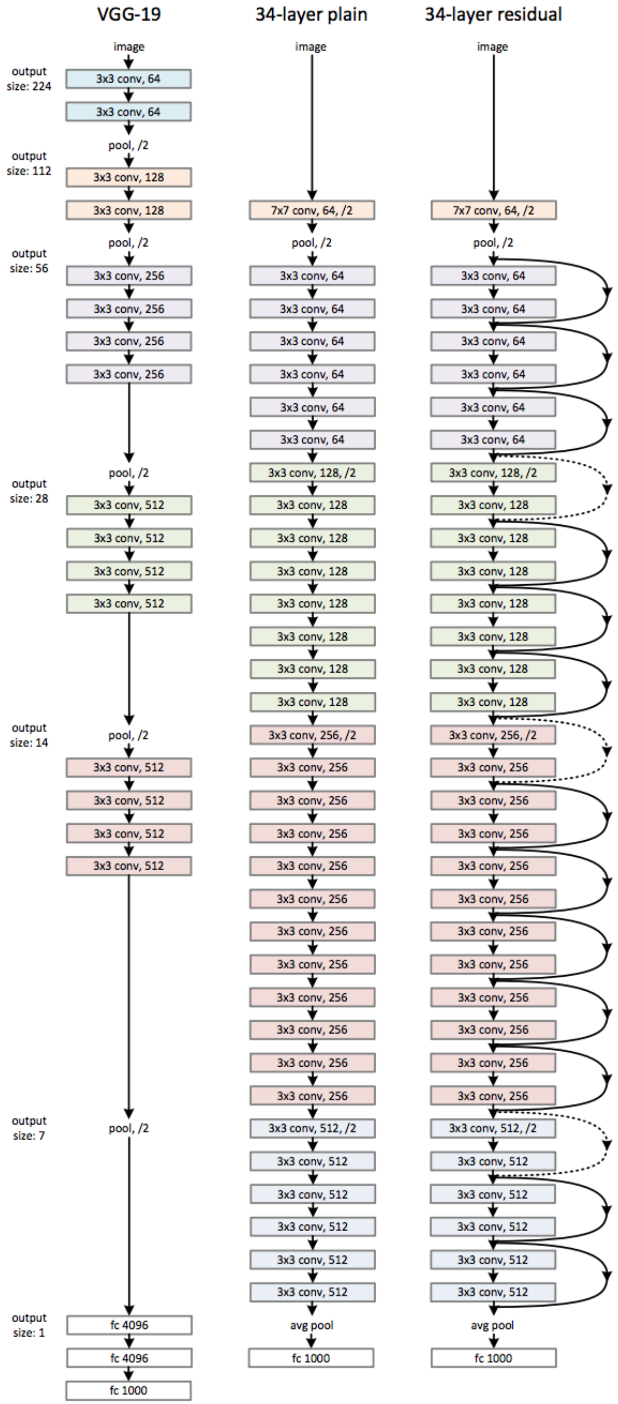


Figure 2. Residual learning: a building block.

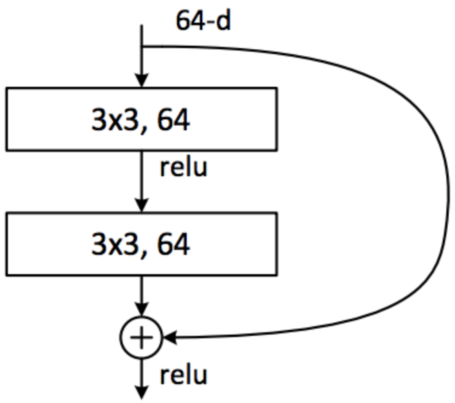
ResNet vs VGG

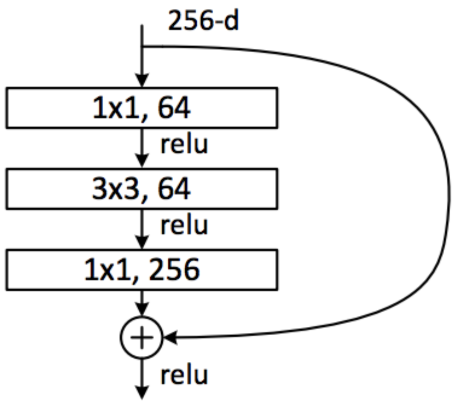
- ResNet has many more convolution layers
- Skip connections give a “direct“ path back to the input
  - Gradients propagate on two paths, the “main” path and the “skip” connections
- Even early in training, early layers can receive meaningful feedback
  - Leads to faster and more stable training



Bottleneck Blocks

- ResNet lets us get some very deep networks
  - But having huge numbers of 3x3 convolutions becomes expensive in terms of memory
- Bottleneck blocks help overcome this by
  - Down sampling via a 1x1 convolution
  - Computing a 3x3 convolution
  - Up sampling with a 1x1 convolution





1x1 Convolutions

- We normally think of convolutions as spatial filters
  - Look for patterns in a local region
- Remember, filters operate across channels
  - For example, we have a [14 x 14 x 8] representation and a [3 x 3] convolution layer
  - Each filter will have 3 x 3 x 8 parameters
    - Each filter will operate over the whole 8 channel volume
    - i.e. patterns across the channels are considered

1x1 Convolution

- Considers patterns across the channels only
  - No consideration of spatial information
- Allows us to
  - Increase the number of channels
  - Decrease the number of channels
- Effectively learns a set of channels that are weighted combinations of existing channels
- Programmatically
  - The same as our regular 2D convolution
    - Just with a kernel of size [1, 1]

1x1 Convolution Visualisation

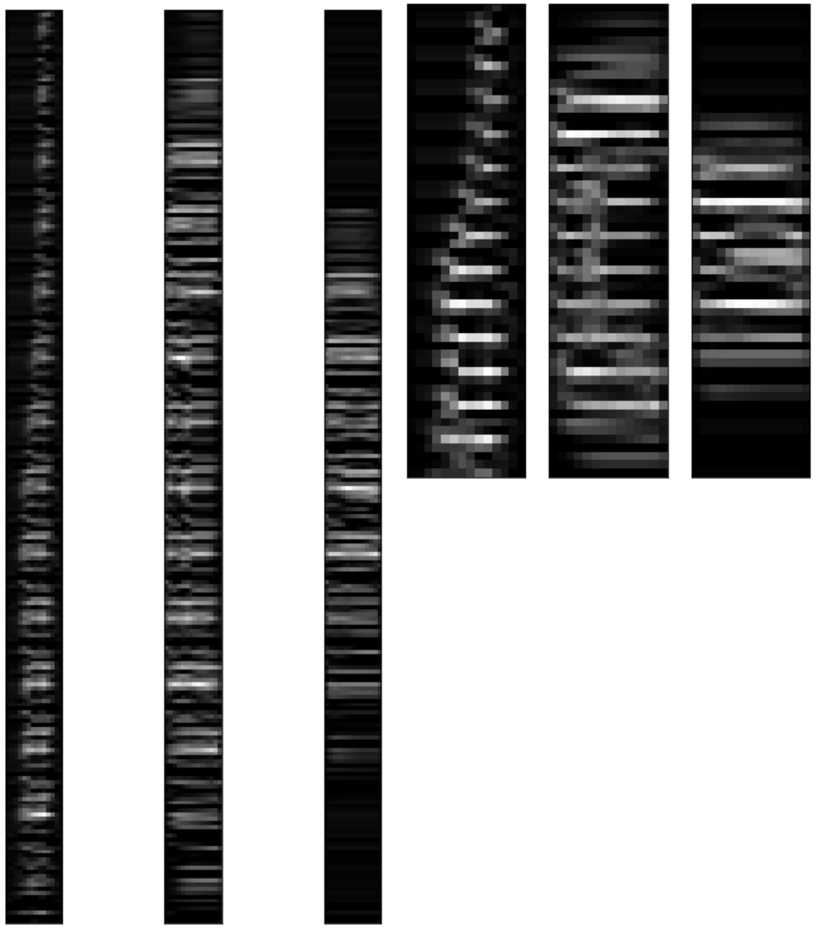
- See ***CAB420_DCNNs_Additional_Example_8_1x1_Convolutions.ipynb***
- We have the network to the right trained on Fashion MNIST
- 1x1 Convolutions:
  - To decrease channels from 16 to 4 (before1 -> after1)
  - To increase channels from 16 to 32 (before2 -> after2)

Note: This example is just to illustrate 1x1 convolutions. This is not a great network design

```
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 8)         80
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 8)         0
_________________________________________________________________
before1 (Conv2D)             (None, 14, 14, 16)        1168
_________________________________________________________________
after1 (Conv2D)              (None, 14, 14, 4)         68
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 4)           0
_________________________________________________________________
before2 (Conv2D)             (None, 7, 7, 16)          592
_________________________________________________________________
after2 (Conv2D)              (None, 7, 7, 32)          544
_________________________________________________________________
flatten_4 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                100416
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 103,518
Trainable params: 103,518
Non-trainable params: 0
```

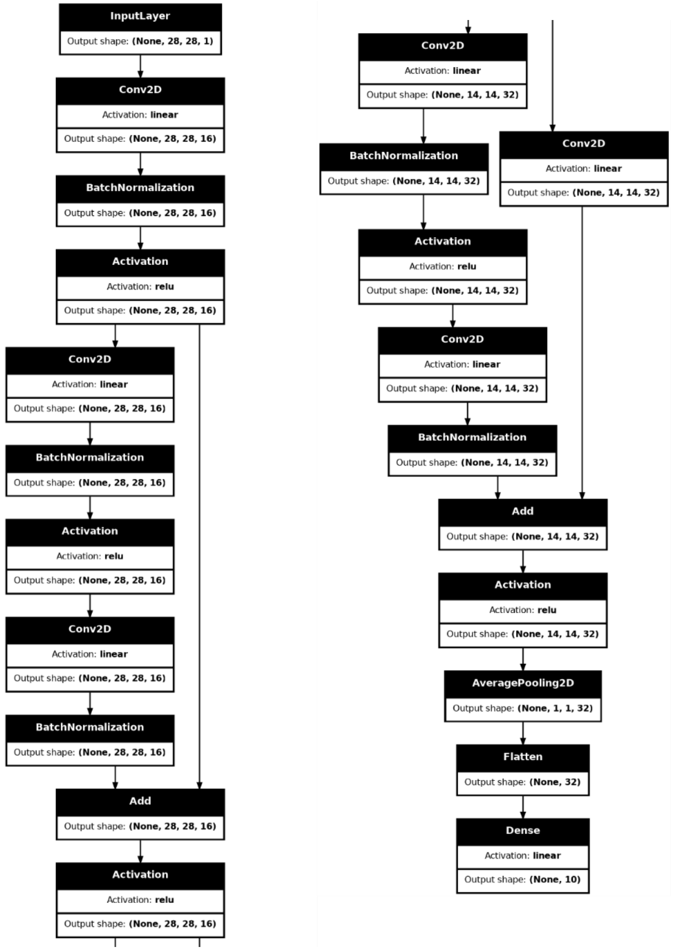
1x1 Convolution Visualisation

- Each column is the unrolled activations from all channels for a single image
  - Left: before 1x1
  - Right: after 1x1
- Right images are compressed versions of the left
  - Same patterns are present, just more compact
- Can use the same approach to upsample
  - See ***CAB420_DCNNs_Additional_Example_8_1x1_Convolutions.ipynb*** for an example



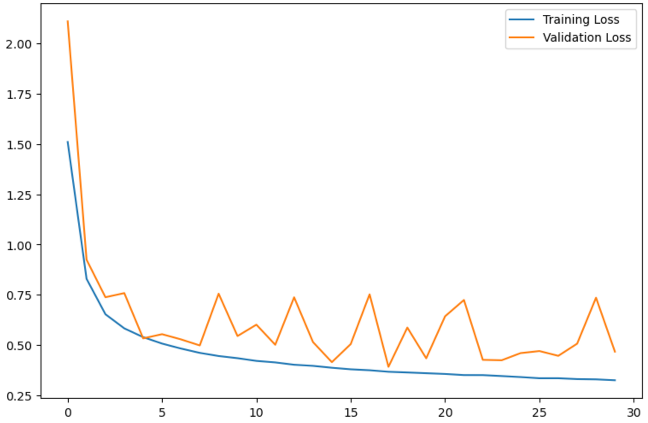
ResNets in Action

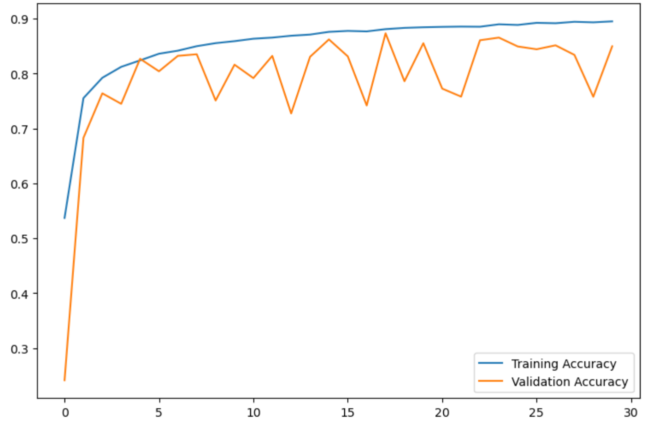
- See ***CAB420_DCNNs_Example_3_ResNet.ipynb***
  - Example uses function to build ResNets of varying sizes
- Simple ResNet
  - No bottleneck layers
  - ~20,000 params
    - Average pooling greatly reduces size of dense layer

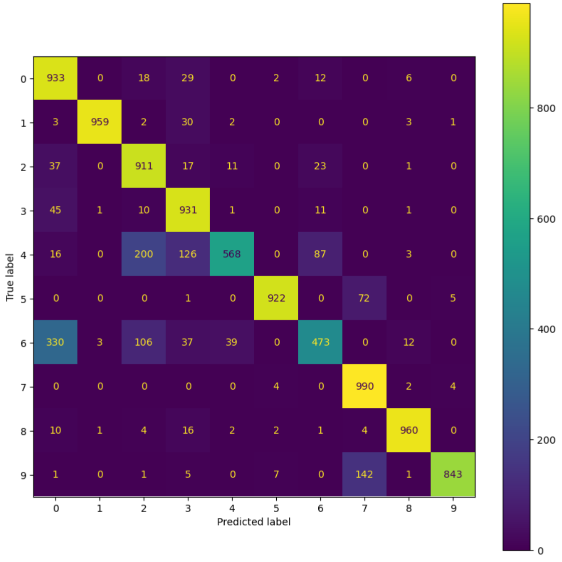


Simple ResNet Performance

- It's not suddenly magically better
  - ~85% test accuracy on FashionMNIST
  - Could perhaps train a little longer though
- The residual structure helps most with network depth
  - Allows us to go much deeper
  - For complex problems, this is important
    - FashionMNIST is not that complex

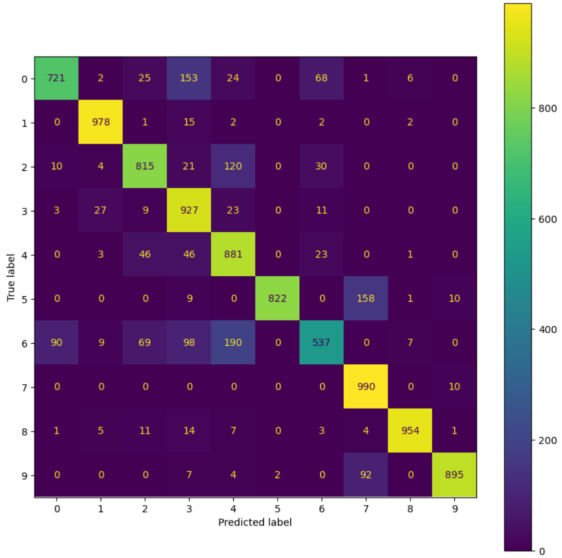


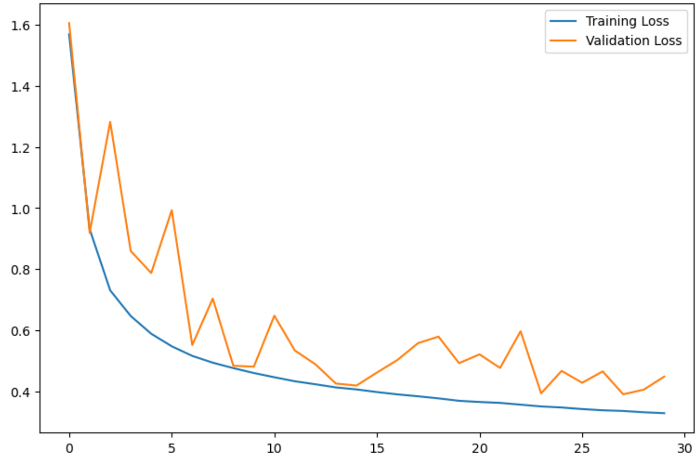


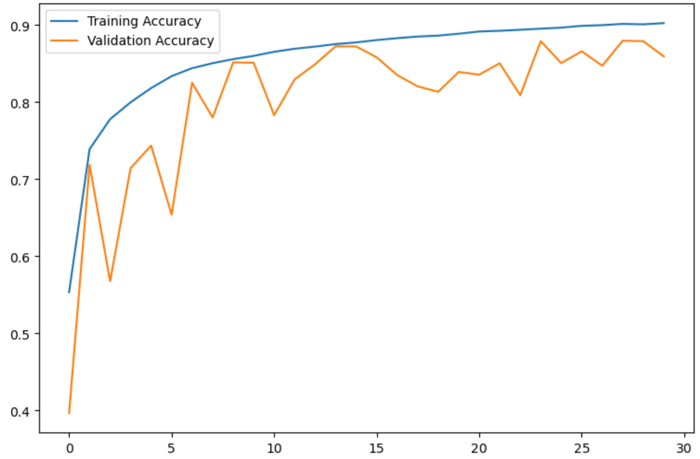


ResNet with Bottleneck

- This time with bottleneck layers
  - Slightly deeper than our previous network
  - ~24,000 params
  - Performance very similar
    - ~85% test accuracy

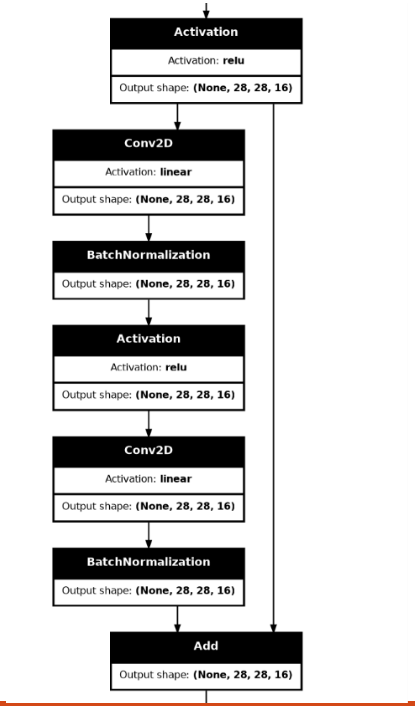




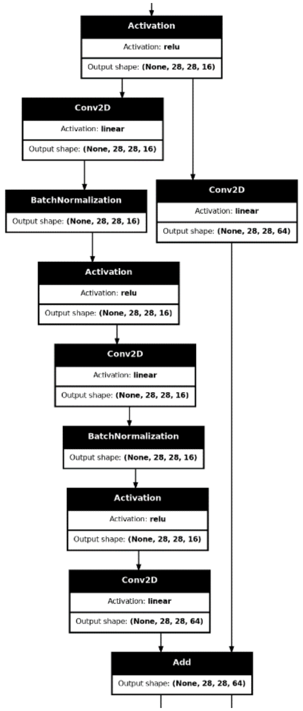


Residual Component (pt 1)

- V1 (no bottleneck)
  - Two convolutions in "main" branch
  - Nothing in the skip branch

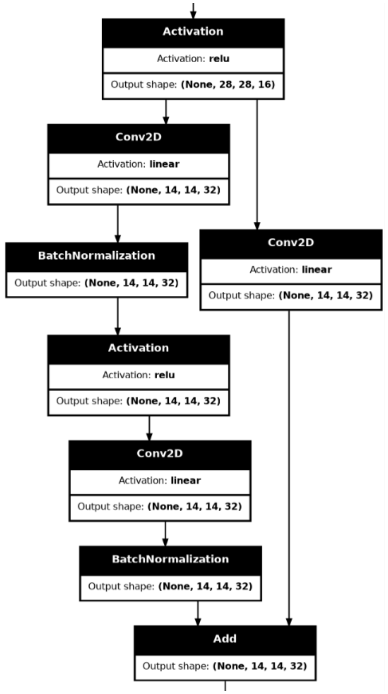


- V2 (with bottleneck)
  - Three convolutions in "main" branch
    - Last one is 1x1, to upsample channels
  - One 1x1 convolution in "skip" branch

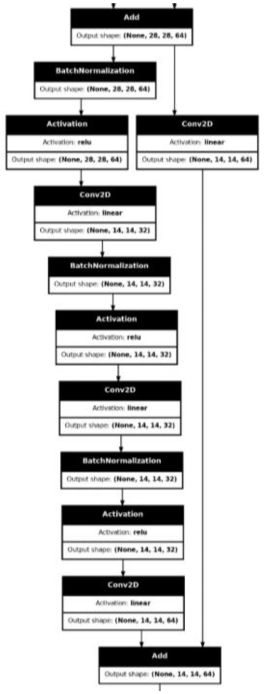


Residual Component (pt 2)

- V1 (no bottleneck)
  - Two convolutions in "main" branch
  - One convolution in "skip" branch
  - Same number of filters (32) in all



- V2 (with bottleneck)
  - Different number of filters in both branches
  - Larger numbers in filters in places than the V1 network
  - 1x1 convolutions at the start and end of the main branch



ResNet V1 vs V2

- V2 networks use bottleneck layers to keep internal representations small
  - Allows deeper networks while keeping memory use manageable
  - Need to have much larger networks to see the full benefit

- Both work well
  - Both improve over VGG and address the issue of vanishing gradients
  - Either one is a good choice
  - Parameters of the network (number of filters, number of residual blocks, depth, etc) more important for performance than which of V1 or V2 you choose

Beyond ResNet

- Several other architectures have been proposed since ResNet
  - ResNeXt
  - DenseNet
  - ….
- All of these retain the idea of skip connections
  - And often use a lot more of them
- I encourage you to explore these in your own time
  - See https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035 as a starting point

CAB420: DCNNs with Less Data

BECAUSE SOMETIMES YOU DON’T HAVE THE TIME TO COLLECT MILLIONS OF SAMPLES

DCNNs and Data

- So far, DCNNs have worked well. But
  - We've had large datasets (50,000 samples)
  - We've had small images (28x28)
- What if we have samples with more dimensions (i.e. bigger images), and fewer samples?
  - Overfitting
  - Training instability
- We need a way to work with smaller datasets. We'll look at two (which can be used together):
  - Fine tuning
  - Data augmentation

Fine Tuning

RECYCLING MODELS

Why?

- So far, we've always trained from scratch
  - i.e. start from random weights, learn everything
- Ideally, we'd always do this, but
  - It requires a lot of data
  - For large models, it requires a lot of time
    - ImageNet models can take weeks to train
- For many tasks, networks learn similar things
- Consider an image recognition task
  - Edges and primitive shapes will always be of interest
  - This is what the early layers learn
  - Why not re-use this?

How?

- Usually when we train a network, we
  - Start from a set of random weights and biases
  - Progressively update those over time

- Instead we can
  - Start from a set of weights learnt on a related task
  - Progressively update those over time
  - We may change some layers
    - Different output type
  - We may set some layers to be fixed and never change
    - i.e. leave early convolution layers as they are

Fine Tuning

- See ***CAB420_DCNNs_Example_4_Fine_Tuning_and_Data_Augmentation.ipynb***

- Basic approach is
  - Load a model from a related domain
  - Change output and other layers if needed. We may do this if:
    - If we're changing from a 10-class classification problem to a 40-class classification problem
    - We're changing from a classification problem to a regression problem
  - Change the input size if needed
    - This may require the removal and re-creation of all dense layers
    - Generally, if you can avoid doing this, it’s best to
  - Decide if some layers should have their weights frozen
    - Most likely early convolutional layers
  - Compile and train the network as you normally would

Advantages of Fine-Tuning

- Can train with much less data
  - Does require some commonality between the tasks
    - Usually, “image based” is enough commonality
- Can train much quicker
  - Given we already have a good initialisation, convergence is much faster
- Can use much more complex than otherwise possible
  - It’s very hard to train ResNet-152 yourself
    - But you can fine-tune it

Data Augmentation

MAKING THINGS UP, SORT OF

Data and DCNNs

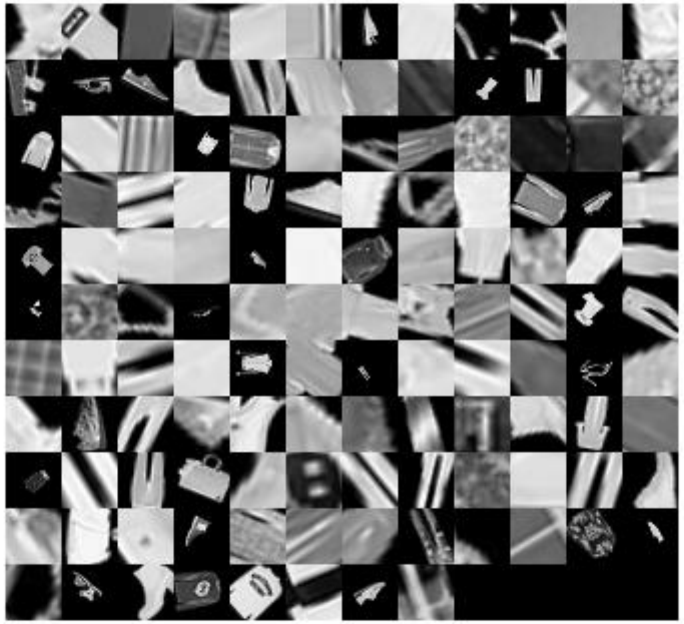
- DCNNs need lots of data. Sometimes this is hard
  - Data capture can be expensive
  - Annotation is also expensive
    - Or very boring
- Data variety is key to avoiding overfitting
- Yet often
  - Different data samples can appear very similar

Data Augmentation

- Creates a “new” dataset by applying simple transforms to what data you have
- Simple transforms may include:
  - Scale changes
  - Rotations
  - Horizontal and/or vertical flips
  - Adding noise or small colour shifts
  - Translations

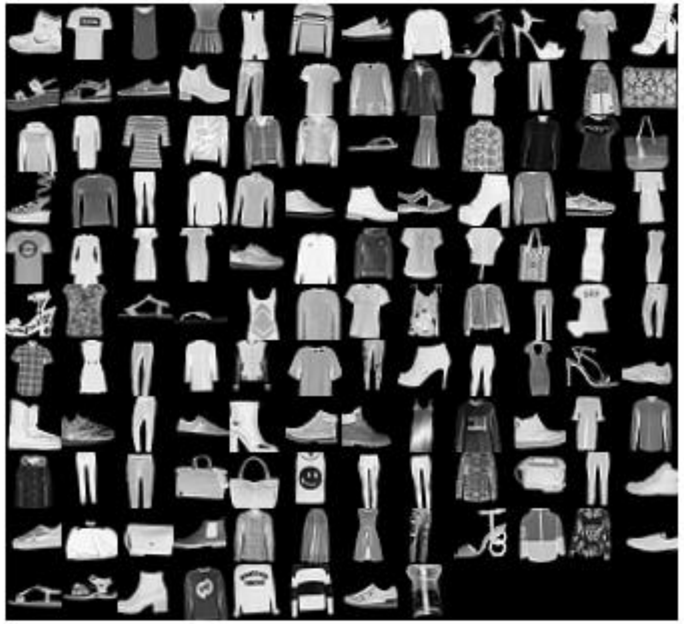
Using Data Augmentation

- Consider your data and what changes make sense
- On the right, we have augmented Fashion MNIST with
  - Vertical and Horizontal flips
  - Large rotations
  - Large scale changes
- This has gone too far



Using Data Augmentation

- Fashion MNIST, take 2:
  - Horizontal flips
  - Small rotations
  - Small translations
- Data looks subtly different from the original
  - Still recognisable as being from the same domain



Using Data Augmentation

- Less can be more
  - Augmentation should not change the meaning of an image
  - Characters for example may change meaning if flipped
    - Consider n vs u
- Inspect the augmentation results before proceeding
  - If they make sense to you, that’s a good start
  - If they've been changed so much that you can't tell things apart, you've gone to far

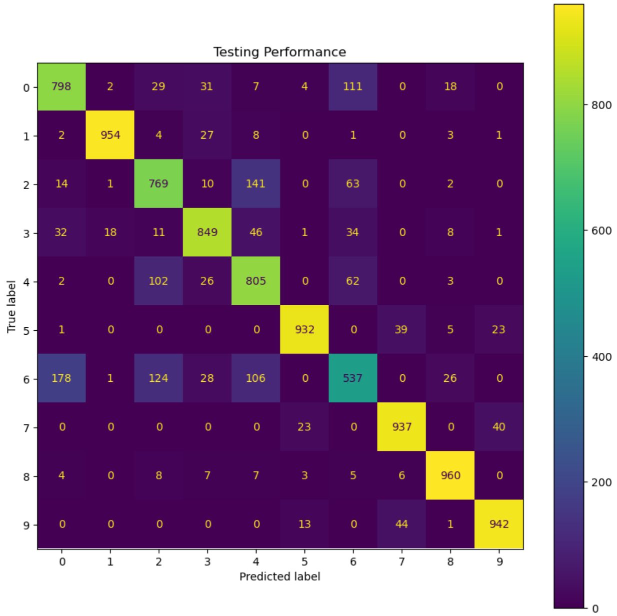
An Example

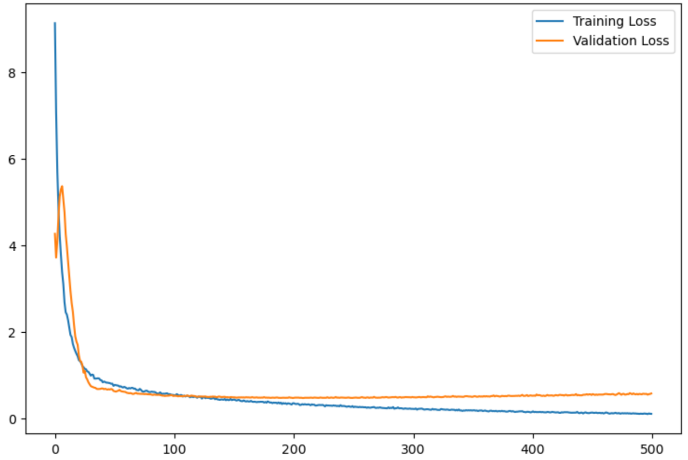
- From ***CAB420_DCNNs_Example_4_Fine_Tuning_and_Data_Augmentation.ipynb***

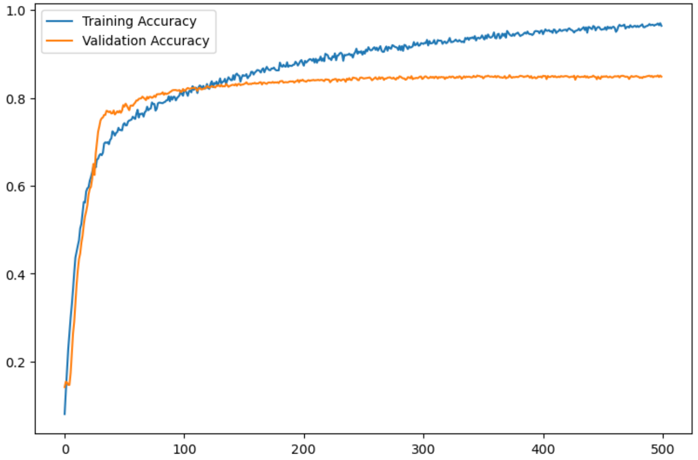
- Our task
  - Fine tune a simple VGG network, trained on MNIST, on FashionMNIST
  - All layers trained, no layers removed/changed
    - Datasets are the same size images, same number of classes, etc.
- The catch
  - We're going to remove 95% of the data
  - 3000 samples, total
- Compare
  - Fine-tuned network with no data augmentation
  - Fine-tuned network with augmentation

No Augmentation

- Overfit central
  - Very quickly we see validation performance flatline and the model begins to overfit
- Trained for 500 epochs
  - In part because we have such a small dataset
  - In part to watch it overfit

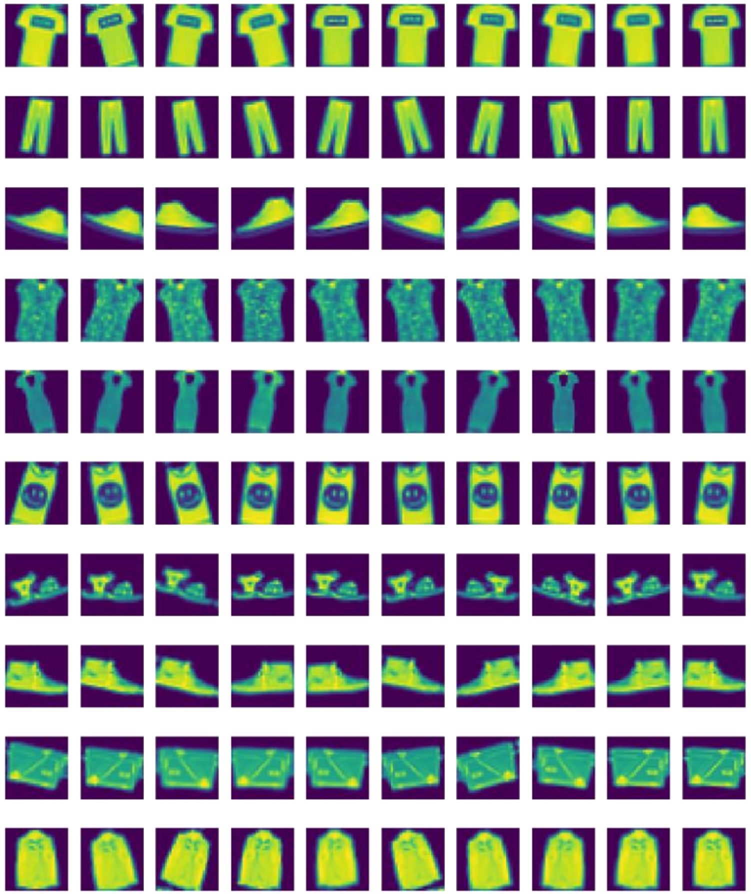






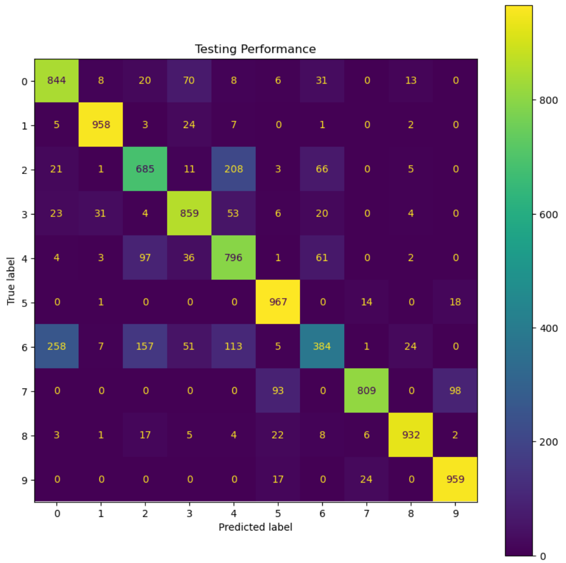
With Augmentation

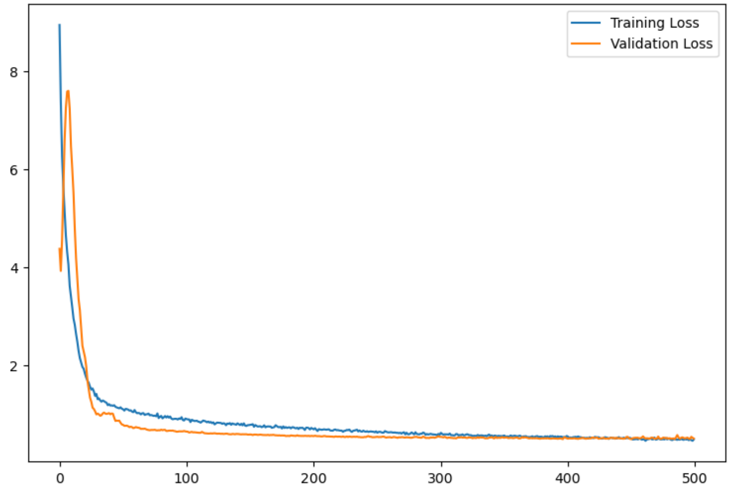
- Augmentation using
  - Rotations
  - Zoom
  - Translation
  - Horizontal Flip
- All samples still recognisable as their true class

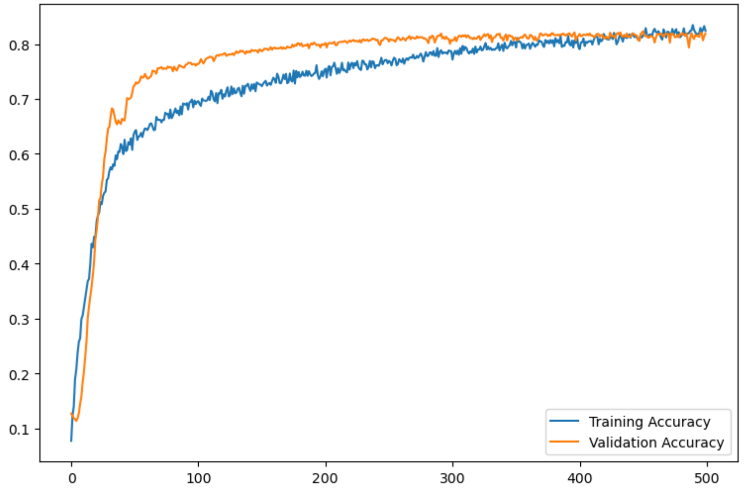


With Augmentation

- Solid performance
- Validation data is not augmented
  - Thus, we see validation data working better for a long time
- Could keep training for longer still at this point
  - Model not fully converged, though close







Things to Tune

SOME MODELS

Models for Fine-Tuning

- Two scripts are on canvas for python
  - ***CAB420_DCNN_Models_Additional_Script_Lots_of_ResNet_Models.ipynb***
  - ***CAB420_DCNN_Models_Additional_Script_Lots_of_VGG_Like_Models .ipynb***
  - These train a bunch of models of different sizes on
    - MNIST
    - FashionMNIST
    - CIFAR-10
  - Exported models are on canvas too
    - So you don't have to train them yourselves

- You can use these models as a basis for fine-tuning
  - Select a model based on size, source data, etc
  - Play with different models as a base, see what happens

Models for Fine-Tuning

- Keras Applications
  - https://keras.io/api/applications/
  - Lots of networks of various shapes and sizes for you to play with
- But what if I want something else to fine-tune?
  - Send an email to cab420query@qut.edu.au
  - If I can, (and I think the request is reasonable) I'll train it and post it

CAB420: DCNNs vs Everything Else

FIGHT!

DCNNs vs Other Classifiers

WHO WINS?

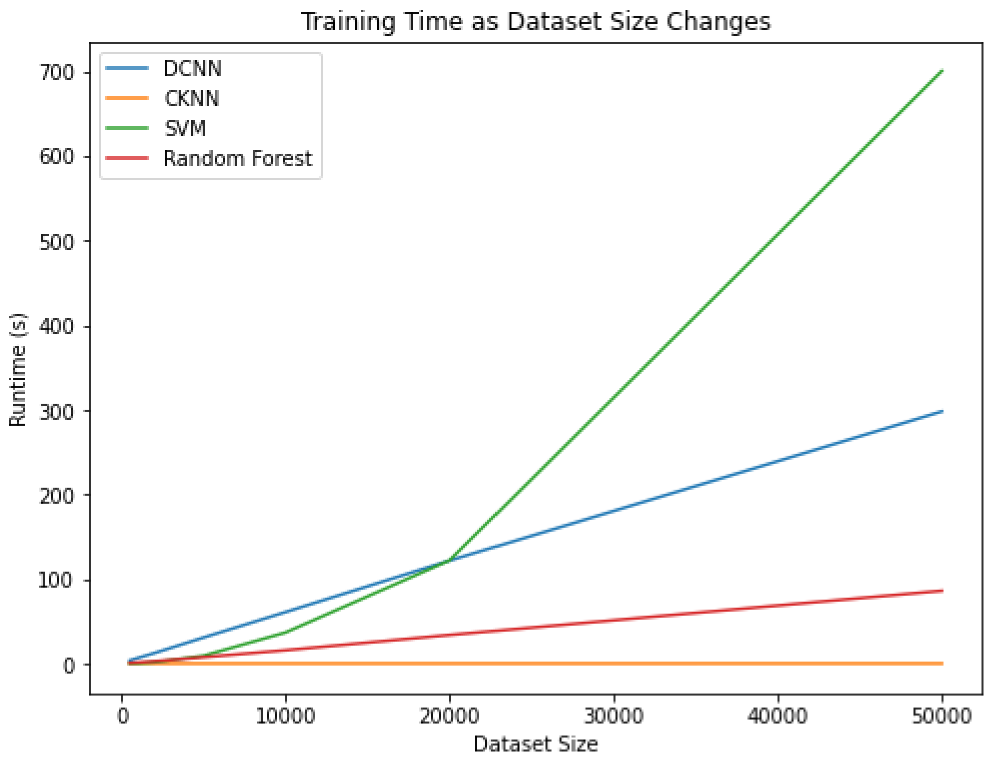
Comparing Classifiers

- DCNNs work best with specific data types
  - Images, Audio, other signals
  - Not well suited to tabular data
- DCNNs also need large amounts of data
  - Though this can be reduced through fine-tuning and augmentation

- We'll compare SVMs, CKNNs, Random Forests and DCNNs using Fashion MNIST
  - Vectorise data for SVMs, CKNNs and Random Forest
  - Compare performance, training time, and evaluation time using different sized datasets
    - We're using non-optimal feature extraction for non-DL methods
    - Our DCNN is very simple, and also not optimal
    - Evaluation will still show broad trends with respect to classifiers however
  - See ***CAB420_DCNNs_Additional_Example_9_Runtimes.ipynb***

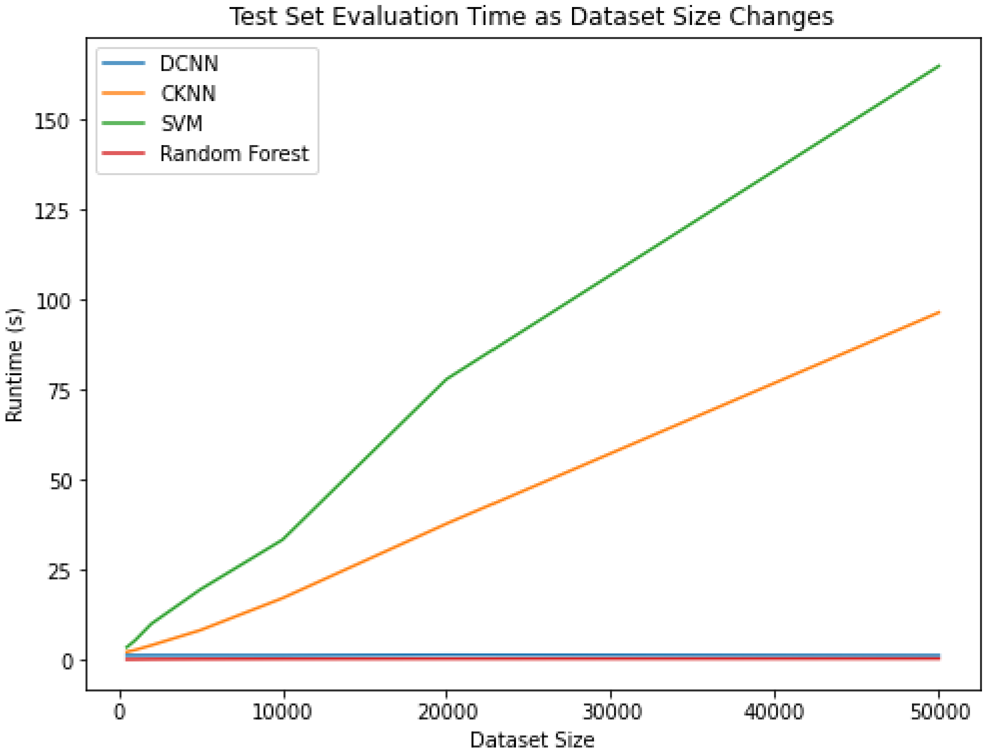
Training Runtime

- DCNN and Random Forest increase linearly with training dataset size
  - DCNN training for a fixed number of epochs
- SVM increase exponentially
  - Does not scale well to large dataset sizes
- CKNN very efficient
  - No real learning performed



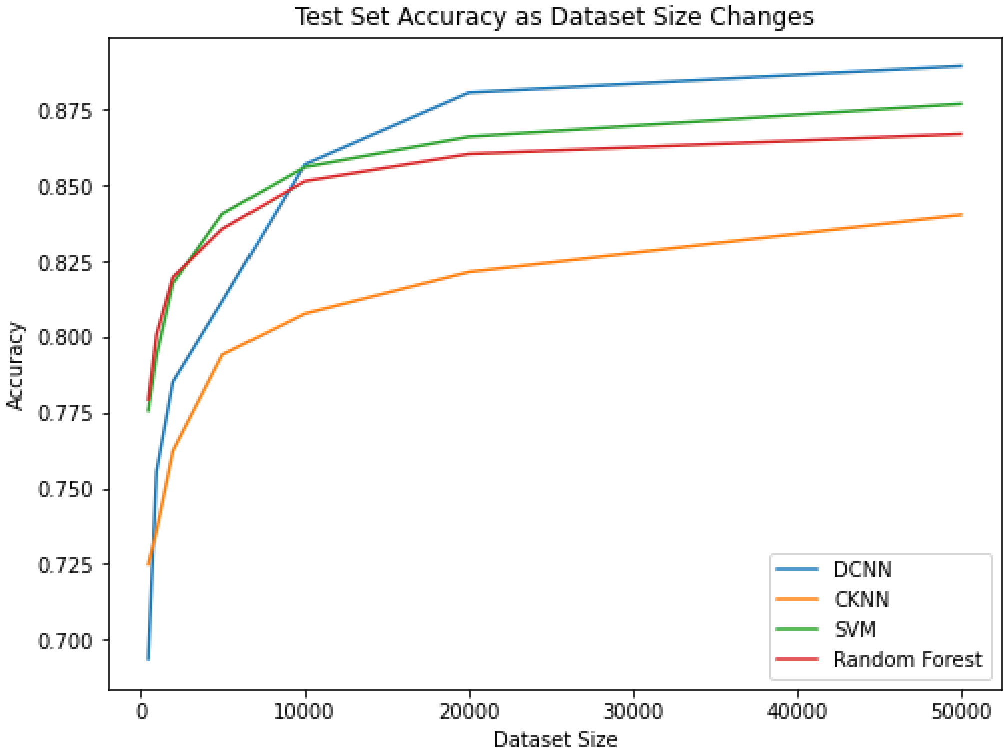
Testing Runtime

- Time to evaluate 10,000 test samples
- DCNN and Random Forest don't change
  - Evaluation time dependant on network complexity (DCNN); number and depth of trees (Random Forest)
- CKNN and SVM get more expensive as training set size increases
  - More points to search/compare to



Accuracy

- Once datasets size reaches 10,000 samples, DCNN wins
- For all classifiers, performance gains slow with increasing dataset size



DCNNs vs the Human Brain

BRAAAAINS!

Biological Motivations

- Neural networks as a whole are directly inspired by the brain, and various components also take inspiration from nature
  - Convolutional networks are inspired by the visual cortex
  - Mechanisms have been developed to model attention and memory
- While these are inspired by nature, they are not mimicking nature
  - We don’t know enough about the brain (human or otherwise) to simulate one
  - Often we seek to mimic the output of a process (as we understand it), rather than the underlying mechanism itself

Neural Network Properties

- Neural networks have become increasingly large in recent years
  - Recent networks for image processing can have 100’s of millions of parameters
  - GPT-3 (a natural language model) has about 175 billion parameters
  - GPT-4 is estimated at 2 Trillion parameters
- Neural networks generally have simple topologies (particularly within a family of models)
  - Networks may have simple branches, skip connections, etc.
  - Networks can use recurrent structures to simulate loops, feedback processes, etc.
- Networks propagate data through a known path
  - Data arrives at the input, propagates through the network, arrives at the output
- Limited fault tolerance, or ability to extrapolate beyond training bounds
- Networks learnt through gradient descent and back propagation
  - Typically trained once, and then used for inference with no on-going updates
    - Though there is work in this area to allow continuous learning

Properties of the Brain

- The human brain is estimated to have about 86 billion neurons
  - But over 100 trillion synapse (connections)
  - High level of interconnectedness, complex topologies
  - Neurons can fire asynchronously
- Biological networks are (to a point) fault tolerant
  - Both through redundancy, and an ability to heal
- Learning is not well understood
  - Learning is continuous
  - Neural plasticity allows connections to change over time
  - Learning is robust to rare or unseen examples

Summary

BECAUSE I TEND TO RAMBLE

Neural Networks

- Now state of the art for most machine learning tasks, but
  - Require lots of data
    - Or lots of tricks to work with limited data
  - Can be very resource intensive to train
- In general
  - Deep networks lead to greater representitive power
  - But at very high depth training becomes difficult and architectural tweaks are needed

Learning with Neural Networks

- Classification
  - Achieved via a “softmax” output
  - Attempts to create a “one-hot vector”, i.e. a vector where one element is 1 and the rest are 0
- Regression
  - Very much like classification
  - Just change the output and the loss function
- Network architectures are flexible
  - Almost identical networks were used for
    - Classification of pieces of clothing
    - Estimation of how much a digit had been rotated by
  - But the networks will have learned very different thing

Making Networks Better

- Can use various layers to improve model fitting
  - Dropout
    - Though be careful when applying to convolutional layers
  - BatchNorm
    - Makes training easier by ensuring that values in the middle of the network are in a known range
  - Weight Regularisation
    - Implementation varies according to platform
  - Explore demo example
    - ***CAB420_DCNNs_Additional_Example_5_Layer_Order_and_Overfitting.ipynb***

ResNet

- Simple feed forward models can only get us so far
  - More depth makes things harder to train
  - Ultimately, adding layers makes things worse
- ResNet uses skip connections to overcome this
  - Allows multiple paths through a network
  - Helps gradients propagate to the early layers
  - Improves training speed
- Skip connections are a standard features of many current architectures
  - DenseNet
  - U-Net
  - And many, many, more

Making do with Less Data

- Fine Tuning
  - Take an existing network, and adapt it
  - Can be seen as a type of transfer learning
- Data Augmentation
  - Create additional data by applying simple transforms to what you have
  - Don’t go overboard
  - Look at what your augmentations do to the data
- Both techniques can make use of DCNNs with small datasets possible