1. What is convolution?

* A feature transformation on the image
* What makes one convolution (blur) different from another (edge detection)?
* The filter
* Mechanics of convolution
* Input length = N, kernel length = K -> Output length = N - K + 1
* Images are not usually square, kernels are
* Pseudocode:

A white screen with black text

Description automatically generated

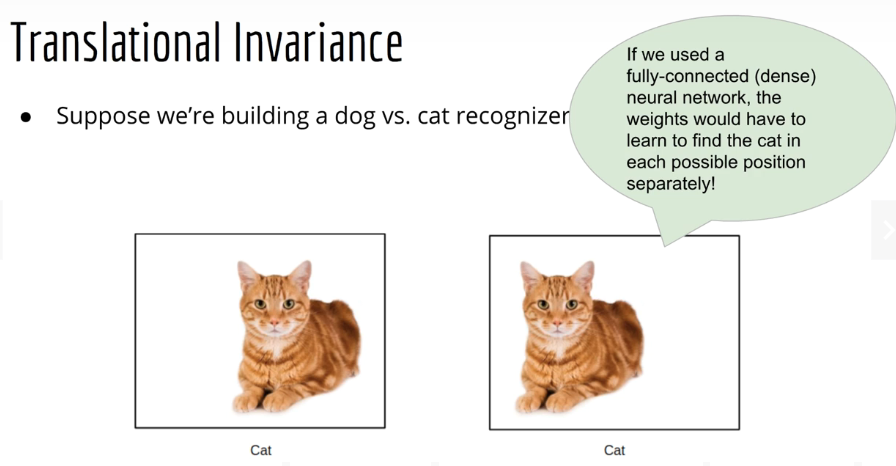
* Convolution equation (what we’re really doing is cross-correlation)
* Valid mode: Output is always smaller than input
* Padding (same mode): what if we want output to be the same size as input?
* Imaginary 0 around the input
* Full mode (more padding)
* We could extend the filter and still get non-zero outputs
* Input length = N
* Kernel length = K
* Output length = N + K – 1

1. More on CNN

* Vectorization:
* Dot product – cosine similarity [-1, 1] -> measure of correlation
* Pearson correlation – almost the same, just with mean subtraction

1. Alternative perspectives on CNN

* How to view convolution as matrix multiplication
* 1D convolution
* Same as 2D convolution, just without 2nd index
* Matrix multiplication
* Same operation using matrix multiplication – repeat operation along each row and shift it
* By repeating the same filter again and again -> convolution without actually doing convolution
* Problem: takes up too much space
* Convolution:
* Instead of a full matrix multiplication -> replace with convolution
* Only use the same 2 weights over and over?
* Less params, use up less RAM, more efficient computation
* Why do this?
* Convolution = pattern finder
* We want the same filter to look at all locations in the image
* Translational invariance
* Translational invariance
* E.g.: Dog vs. cat recognizer



1. Convolution on color images

* Images – 3d objects: HxWxC

A diagram of a box

Description automatically generated

* 3D dot product = color pattern finder
* Input image: HxWx3
* Kernel: KxKx3
* Output image (2D): (H-K+1)x(W-K+1)
* Neural networks have repeating structures
* E.g., for Dense layer, input is 1D vector, output is also 1D vector
* Can be fed into another Dense layer
* But if output is 2D, how can we do another convolution later?
* Multiple features – We should have more than 1 filter per image, because each filter is looking for sth different
* A.shape = HxWx3
* If we use same mode, then B1.shape = HxW, B2.shape = HxW
* If we stack B1 and B2, we get B.shape=HxWx2
* We can add any number of features
* Depth of output can be much more than 3 (same as number of filters used)
* Convolution in a DNN
* Summary so far
* Convolution for 2D images with 2D filters
* Extended into color images by saying the filter should just have the same depth=3 -> dot along all 3 axes
* Breaks uniformity, because output is 2D
* Would not be able to stack multiple convolutions sequentially
* Extended further by noting that each layer should find multiple features (using multiple filters)
* Multiple 2D outputs which can be stacked upon each other to get a 3D image once again (depth = n\_filters)
* Color:
* Input to the neural network is a true color image
* But after subsequent convolutions, we just have arbitrary depth
* Our terminology must be more general -> feature maps
* Size of the final dimension: number of maps/features
* Convolution layer
* What does a convolution look like as a neural network layer?
* Convolution is like a shared-weight version of matrix multiplication (e.g., in a Dense layer)
* In a Dense layer, we also have a bias term and activation function
* So does a convolution layer
* Shape of bias term
* In a dense layer, if is a vector of size M, b is also a vector of size M
* In a Conv layer, b does not have the same shape as (3-D image)
* Technically, this is not allowed by rules of matrix arithmetic
* But the rules of broadcasting (in numpy) allow it
* If has the shape HxWxC2, then b is a vector of size C2 (one scalar per feature map)
* How much do we save?
* Savings from doing convolution instead of matrix multiplication
* See video for explanation
* How are convolution filters found?
* See video for explanation

1. CNN architecture
2. Data augmentation

* Why data is important:
* For tabular data: what you have is just what you have
* Can’t invent new data – that would be contaminated with my own biases
* Images are an important class of data: I can see images
* With images, it makes sense to invent new data
* Problem: data takes up space
* More invented data, more space
* There are endless number of ways to invent new data
* This can be done automatically with Tensorflow’s Keras API instead
* Generators / iterators:
* Loop from 0, …, 10
  + Python 2: range(10) -> a list

Use xrange(10) -> does NOT create a list

* + Python 3: range(10) -> yields range(0,10)
* Create your own generator
* Can you write your own function to do something like this?
* Using yield command

A close-up of a black text

Description automatically generated

* No list ever created
* All values do not need to be stored in memory simultaneously
* Apply this to data augmentation:
* Generate augmented data on the fly

A close up of a text

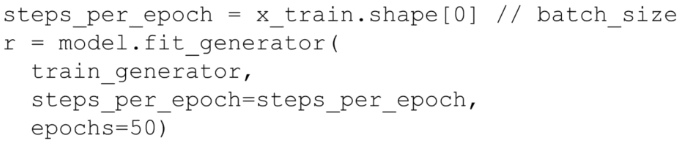
Description automatically generated

* How does it work in tf.keras?

A screenshot of a computer program

Description automatically generated





1. Batch normalization

* Normalization / standardization:
* Early on, we noted that its important to normalize/standardize data before passing it into algorithms like linear/logistic regression
* Problem: because this operation is done only on input data -> only the first layer sees normalized data (after being transformed by Dense layer it’s no longer normalized)
* Batch normalization:
* What if we had a layer that would look at each batch, calculate the mean and standard deviation on the fly, and standardize based on that?

A diagram of a mathematical equation

Description automatically generated

* An improvement:
* How do we know normalization is good? We don’t
* What if a different location / scale are better? Let’s learn this automatically
* Batch norm as regularization:
* Can help with overfitting
* Since every batch is slightly different, you’ll get a slightly different mean and standard deviation
* Not the true mean/std of the whole dataset
* Essentially noise, and using noise during training makes the neural network impervious to noise
* Where is batch norm used?
* We discussed batch norm in terms of dense layer, but they are usually more commonly used after convolution layers
* Check out architecture of popular CNNs: VGG, ResNet, Inception, etc.

A close-up of a pink and black square

Description automatically generated