Spencer Plavoukos
Numerical Linear Algebra
Spring 2023

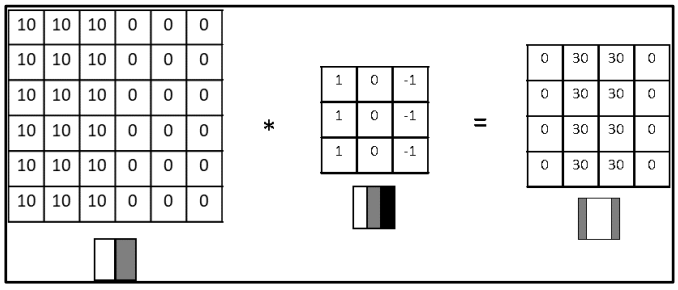
This paper walks through the components of a convolutional neural network and how it learns. It's broken down into three parts: feature extraction, neural network, and medical imaging instabilities. The first two are the two main segments of a convolutional neural network, and the third explores how these models are used in the medical imaging field and the possible instabilities that can occur as a result.

I - Feature Extraction

Kernels

The first important component of the feature extraction process is the kernel. A kernel is a small matrix, usually 3x3,5x5, and up to 11x11, that is applied to an input matrix and outputs a feature map. Kernels are a vital part of feature extraction because they can be manually changed – i.e., size and number of kernels - to better support the learning process of the network. These parts of the network that can be manually changed are called hyperparameters, and we will see more of these as we get further along the network. Kernels perform a convolution operation on the input matrix by doing element-wise multiplication to calculate the output. These kernels can be thought of as filters being applied to the input image, although the real terminology has a slightly different definition (more on that later).

Example: A 3x3 kernel is applied to a 6x6 input image. The kernel first performs a convolution on the 3x3 submatrix of the input starting in the top left corner and moving across every subsection until it reaches the bottom right. The result is a 4x4 matrix representing a mapping of the particular features that the kernel is made to identify. To identify vertical edges in an input, for example, the kernel would look like this:



Vertical Edge Detection Kernel

The feature matrix has zeros where there were no vertical edges detected and high values that indicate where the vertical edges are. A similar kernel can be constructed for horizontal edges, rounded corners, colors, and other basic features that can be extracted from an image. The beauty of a neural network is that the values of these kernels are learned through many training iterations, meaning the important features of some inputs don't have to be explicitly told to the network. This makes a network like this very adaptable and possible to use for many different applications. The values in the kernel are also considered to be weights. The strength of each weight is learned for the task that the network is made for, and is conceptually equivalent to weights in a classical neural network.

Filters

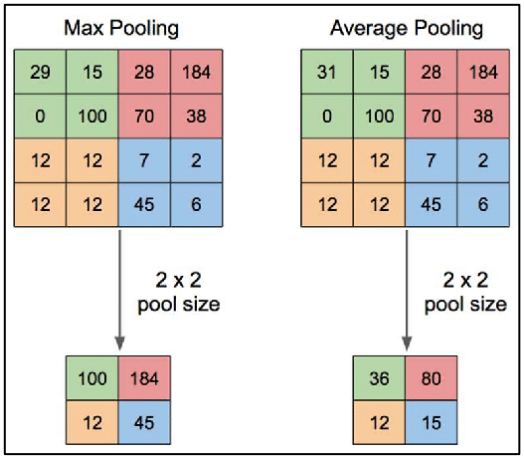
Although a kernel can be thought of as a filter being applied to an image, the term 'filter' actually refers to a concatenation of multiple kernels into a tensor. Multiple kernels are needed to carry out these convolution operations because images usually have multiple channels. These channels could be simple, for example a color image might have three channels, one for red,

green, and blue (RGB). A kernel would be needed for each of these channels. If there are three 3x3 kernels, they would be concatenated into a 3x3x3 tensor to be applied to the input with RGB channels. Tensors are also important for computational efficiency as they take advantage of parallel computing, where a graphics processing unit (GPU) uses many cores to handle computations for multiple channels at once. This reduces compute cost and training time.

Pooling Layers

Another type of hyperparameter found during feature extraction is the pooling layer, which comes after a convolution layer. It functions in a similar way as the convolution layer with respect to how it moves over the input image. Recall that the convolution kernel starts at the top left of the input matrix and slides over the image, row by row, until it reaches the bottom right. The pooling layer uses a kernel that moves in a similar fashion. However, instead of taking a convolution with the input, it performs one of two operations: taking the average or maximum of the values of the submatrix it is on.

In the example to the right, the input is a 4x4 matrix and the pooling kernel is 2x2. The pooling kernel starts in the top left and takes either the maximum or average of the values of the 2x2 submatrix it is looking at. It then does this for the rest of the matrix until there is a condensed output. This pooling layer is also a hyperparameter because the size of the kernels and whether they take the maximum or average are determined by the architects of the network.



Pooling layers have two main functions: dimensionality reduction and translation invariance. As evident in the example, the output matrix is smaller than the input matrix. This feature of the pooling layer is very important, especially in practice, for reducing the dimensionality of the convolution layers, which allows for less expensive computation and faster processing times while training the network. The other important aspect of this layer is translation invariance. Translation variance refers to objects of interest in input images being in different locations across training samples. For example, a coffee cup could be in the bottom left corner of a training image, but if that is the only type of image the network is trained on, the network will only learn to identify coffee cups in the bottom left corner. This explains the need for variation within training sets, which subsequently shows the need for the network to be indifferent to these types of variations. Hence, the pooling layer proves to be a necessary tool for dealing with movements (translations) within the inputs.

II– Neural Network

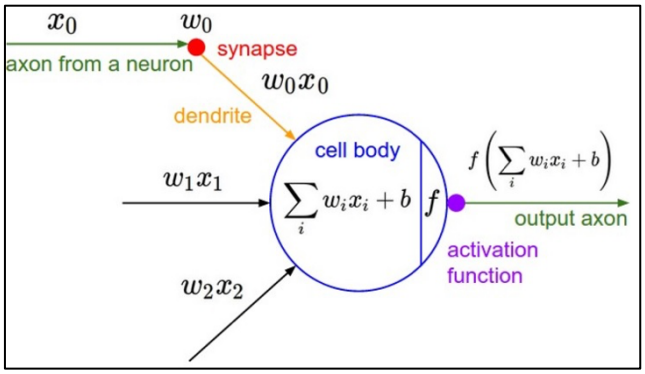
Nodes

The second part of a convolutional neural network that comes after feature extraction is the actual neural network. Neural networks on their own are used in a myriad of ways for machine learning, and in this case, it is augmented with a feature extraction segment.

Nonetheless, this part of the network still functions the same as the classical neural network. To investigate how it works, we'll start with the most basic components, then work our way up to the functions comprising the whole network.

It is called a neural network because it is roughly modeled after just that – the network of neurons in our brain that allow us to learn and adapt to new things and environments. The most fundamental component is analogous to a single neuron, the 'messenger' cells in the brain that pass electrical signals to one another by taking inputs and deciding which outputs to pass on. These neurons work together to create a network complex enough and dense enough to do all the amazing things we do with our brains. Thus, a perfect thing to model a machine learning model after. The digital equivalent to a neuron is a node, which also takes inputs and produces outputs that it passes on to other nodes.

A node holds a value, called an activation, that is calculated from the inputs of previous neurons. These inputs are called weights and biases and they're the parameters that the network learns to get better at making predictions.



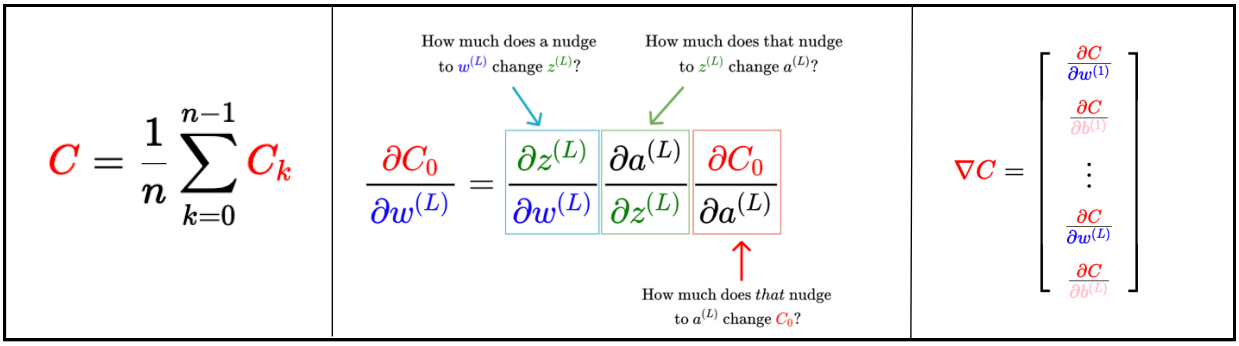
In the figure to the right, you can see there are multiple inputs to the 'cell body', a.k.a. the node. $x_0 \ldots x_i$ are the activations from previous nodes and $w_0 \ldots w_i$ are the weights associated with those nodes. the activation of the current node is calculated by taking the linear combination of the previous activations and corresponding weights. A bias, *b*, is then added to the linear combination. The resulting value is then normalized by an activation function, *f*. The most common activation functions are ReLU (Rectified Linear Unit) and SoftMax. These activation functions introduce nonlinearity into the network, allowing the network to learn nonlinear behavior. This increases the functionality and subsequent performance of the network that would not be possible with simple linear regression.

The process of taking linear combinations of activations and weights, adding biases and normalizing is a vital process in neural nets because the activation of a node determines how important a node is in a network. If a node has an activation close to zero, the network is essentially not taking into account the features that the node represents. These activations improve in accuracy over time as the network is trained and goes through the learning process. More on that later.

Forward/Backpropagation

Combining multiple nodes makes a layer of nodes, and multiple layers make a network. In the first part of a neural network's learning process, it makes a prediction based on the input it is given by starting from the beginning of the network (the input matrix) and taking linear combinations of the activations and weights, along with the other parameters discussed. When it gets to the end of the network, it makes a prediction of, in the case of a convolutional neural network, what object was in the image (and other predictions, see part III). This process of making a prediction is called forward propagation. The network then learns by a process called back propagation. It first calculates a cost function *C* that represents how wrong it was in its prediction. The goal of the network is to update its parameters to make it more accurate. This can

be done by seeing how each parameter affects the cost function, then updating each one to minimize the cost. To do this, the network calculates a gradient vector ∇C that holds the partial derivatives of C with respect to every parameter. To calculate each partial derivative to a parameter, it chains together the partial derivatives of all the other parameters that the parameter affected up to the cost function. This means that as long as a function is differentiable, it can be backpropagated through and therefore updated. For linear combinations, derivatives are calculated for every addition and multiplication operation. The same goes for convolution operations, which are just element-wise multiplication.



In the example above, the cost function C is calculated by taking the average cost of each training sample C_k. Partial derivatives of each parameter are then calculated by chaining together subsequent derivatives from C to the parameter. In the example, $w^{(L)}$ is a weight in layer L, meaning a weight in the last layer for a network with L layers. These derivatives are then stored in the gradient vector ∇C. Only after every parameter's derivative is calculated are the gradients applied to those parameter values. Performing forward propagation to predict then backpropagation many times for many samples allows the network to continuously update and minimize its cost function, and thus make better predictions about the inputs it is given.

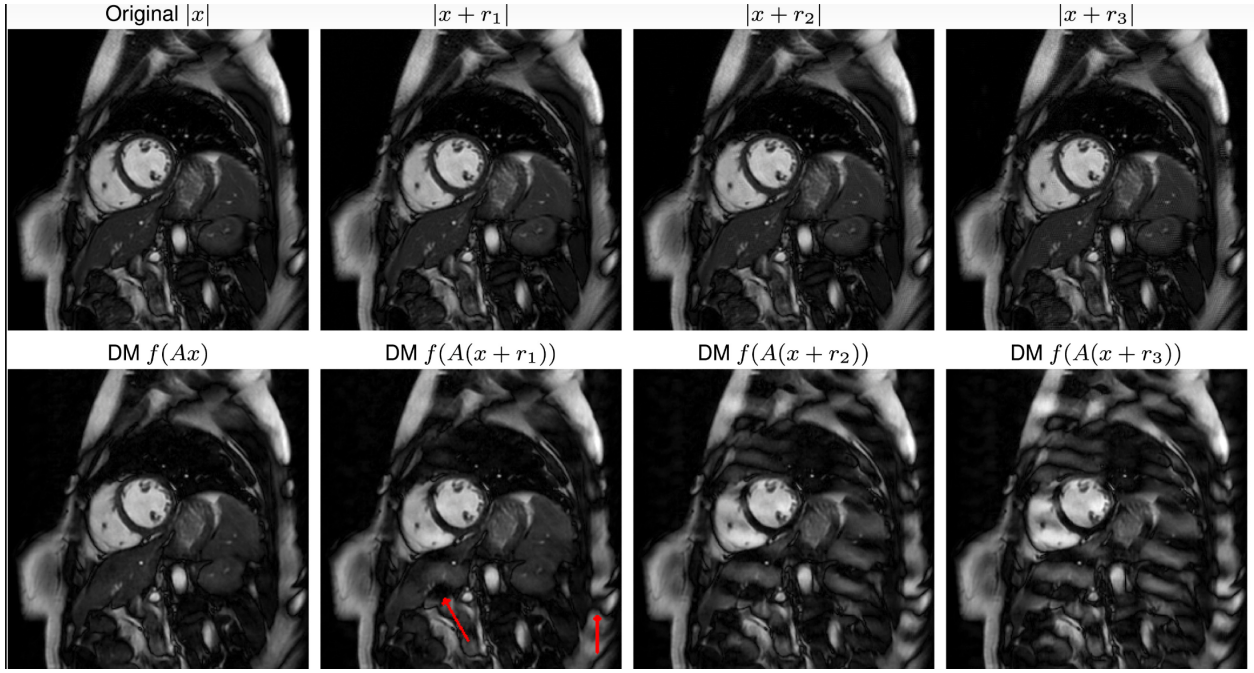
III – Medical Imaging & Instability

Instability in Reconstruction

Another common application of CNNs is image reconstruction. In the case of an MRI, a strong magnetic field is created around a patient, radio waves are sent to the body, and signals are sent back (an over simplified explanation, but it will do for now). These signals are then used by a computer to reconstruct the image it took by undersampling the data and exploiting its redundancies. Current methods for this process, however, are slow and problem-specific rather than generally applicable. CNNs can be used for this reconstruction process for faster and, sometimes, more accurate results. The 2019 paper *"On Instabilities of Deep Learning in Image Reconstruction and the Potential Costs of AI"* (Antun et al, 2019) explored how small perturbations in input images from MRIs can have drastically negative effects on the reconstruction. I will be referencing this paper and the work they did for the remainder of the paper.

Perturbations from medical imaging can be caused by a variety of things including, but not limited to, slight movements from a patient during scanning, very small tumors, and machine operator error. The authors of the paper modeled these small, worst-case perturbations to demonstrate the negative effects they can have while reconstructing images. The model goes as follows: Given a flattened input image $x \in \mathbb{R}^N$, calculate a worst case perturbation

$r \in \mathbb{R}^N$ defined as $\| f(y + Ar) - f(y) \|$ is large, while $\| r \|$ is small, and where $|r_1| < |r_2| < |r_3|$. Then the reconstructed image is $\hat{x} = f\big(A(x + r)\big)$, where $A \in \mathbb{R}^{m \times N}$ represents the sampling modality, a.k.a. the reconstruction method. Finally, f is the neural network being trained, defined as $f: \mathbb{R}^n \longrightarrow \mathbb{R}^N$. Reconstructing these increasingly bad but tiny perturbed inputs returns the following results:



As you can see, very small, almost unnoticeable perturbations in the inputs return extremely perturbed outputs, displaying a high degree of instability during reconstruction. The authors tested a variety of networks with this same instability test and found many different types of artefacts due to instability. Although varying in representation depending on the network type, the instability phenomenon was a common theme across all types of networks. These 'bad' perturbations were specifically constructed, but the authors of the paper also came to the conclusion that there is a nonzero probability that these bad perturbations come up in practice, and that the probability is nontrivial to find. This is because the perturbations are the sum of two variables, as stated in the paper, "where one variable comes from generic noise and one highly nongeneric variable is due to patient movements, anatomic differences, apparatus malfunctions, etc." The latter, as previously mentioned, comes from the actual patient-doctor-machine interactions and is harder to account for and predict, hence the probability of these bad perturbations being hard to find.

Finally, the authors concluded that this phenomenon is not an easy problem to solve because of its ubiquity across very different types of networks. The ubiquitous instability across network types is an important problem to solve because of the importance of advancing medical imaging technology and the significance that something like an MRI holds for a patient and doctor to examine. With the rapid evolution of machine learning techniques and increasing attention being brought to the field, one can hope a solution to this problem can be discovered soon.

Works cited

Antun, et al. (2020, May 11). *On instabilities of deep learning in image reconstruction and the potential costs of AI*. PNAS. Retrieved from https://www.pnas.org/doi/full/10.1073/pnas.1907377117

Jeremy Jordan. (2018, January 26). *Neural networks: Representation.* Jeremy Jordan. Retrieved May 1, 2023, from https://www.jeremyjordan.me/intro-to-neural-networks/

Solai, P. (2018, April 18). *Convolutions and backpropagations*. Medium. Retrieved May 1, 2023, from https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

Backpropagation calculus. 3Blue1Brown. (n.d.). Retrieved May 1, 2023, from https://www.3blue1brown.com/lessons/backpropagation-calculus