Image Classification using Convolutional Neural Networks

JONATHAN BHASKAR

Motivation

Object Recognition using computers has a wide range of applications. It has been researched a lot over the decades, but recent advances in the field have achieved a breakthrough, with models achieving near human accuracy. This was done with deep learning, which is essentially a neural network with a complex structure and many hidden layers.

The specific paper I've chosen is titled "ImageNet Classification with Deep Convolutional Neural Networks" [1]. ImageNet is an annual competition in image recognition where researchers in the field pit their models against each other to achieve the highest classification accuracy on the same set of images. The model put forward in this paper, named AlexNet from it's main author, beat the second best model by over 10% accuracy in 2012, using deep learning techniques which were not popular in the field at that time. This led to a lot of interest and research in deep learning, not only for image recognition but also other tasks, and it has become one of the hottest topics in Computer Science today.

With state of the art results in computer vision, speech recognition and other tasks, deep learning has challenged the existing notions of what a computer is capable of. This excites me and I chose the topic to learn more about it.

Before talking about the CNN model in the paper, I will explain CNNs and its parts in general. As this is a comparatively new field, I was unable to find books on this topic. I studied the lecture notes from a course on this topic at Stanford [2]. This was my source for everything I've explained.

Concepts

In this section I will "build the ladder" from a traditional neural network and explain the building blocks of a CNN that are different from the traditional one.

Activation Functions [3]

We studied the sigmoid activation function in class. It scales the output of a hidden node between 0 and 1 [4]. Using an activation function like the sigmoid introduces non-linearity in the network. This leads to a neural network having the property that any continuous deterministic function can be approximated by it. The reason a sigmoid function is popular in particular is because it is easily differentiable (the derivative is used in back propagation of errors) [5]. The derivative of a sigmoid function is

$$\frac{dy}{dx} = y(1 - y)$$

However, the sigmoid function has a major problem. It relies on the range of input values a lot. If the input is not scaled properly, the output of the sigmoid would be saturated and close to 0 or 1. This leads to the derivative from the above equation to be almost 0. When this happens, the neural network does not learn from the values in back propagation. Therefore, a more common activation function found in CNNs is the ReLU or Rectified Linear Unit.

A ReLU takes the form $f(x) = \max(0, x)$. This does not depend on the range of inputs. It also requires almost no computation as it just outputs 0 or the input, and so it is a lot faster. A neural network using ReLU has also been shown to learn 25% faster than a tanh function [1]. An interesting point here with regard to the Kolmogorov's Universal Approximation theorem [3] is that a single layer in a neural network is enough to approximate any continuous function. But adding layers to a CNN has been empirically shown to improve accuracy rates. This is a subject of active research.

Another reason why more hidden layers are preferred is because they are easier to train. As neural networks are non-convex, it is hard to study the gradient descent. But when more number of layers are used, even though there are more local minima, these tend to better solutions. Getting stuck in a bad local minimum is more likely with fewer layers.

Data augmentation

Data augmentation was done in AlexNet to prevent overfitting. The input images were of dimensions 256*256 i.e. a length of 256 pixels and a height of 256 pixels. Random parts of the image with dimensions 224*224 were taken as separate observations for training. These were also horizontally flipped (the pixels were reversed in the horizontal direction). The reason for picking 224 as the dimensions is explained later. Even though the observations were similar, this led to a reduction in overfitting. When testing, again, five samples of 224*224 dimensions were taken from the image and the output classes were averaged.

Dropout [6]

This is another technique used to prevent overfitting. It "switches off" neurons in the hidden layers with a certain probability. A common value for this parameter is 0.5, which means for any iteration, only half of the hidden nodes in a layer are active, and the other half are not used. This works for two reasons:

1. It forces neurons in a layer to learn from a specific set of neurons in the previous layer, rather than the entire set of neurons. This makes the learning more robust, as a neuron is more sensitive to changes from this particular set of neurons from the previous layer.

2. The architecture of the network changes with each iteration. As there are a large number of neurons in each layer in deep learning techniques, this means that the network is almost never the same for two iterations. Therefore, the weights learnt by each neuron could be thought of as an average of these network architectures. It has an effect similar to model averaging, which improves model performance in general.

The output value of a neuron is lower when dropout is used because of the lower number of variables. Dropout is not used when predicting test data as the learning of all the neurons has to be incorporated in the prediction. So the test data has to be scaled accordingly to account for the lower output value in training, as this is the input to the next layer in the model.

Classification over Regression [6]

A softmax function is used in AlexNet to predict the probabilities of classes in the final output layer. It is possible to do regression with CNNs using the MSE instead of the softmax function, but this is not recommended. In classification, when a model predicts the correct class for an observation, the loss function would start decreasing. But in regression, the exact value would have to be predicted and it is a lot harder for the model to get there. Also, as the softmax function outputs probabilities, the values are between 0 and 1. The back propagation errors are always in a predictable range. But for regression, there could be a huge difference in back propagation errors for different observations. This makes the learning unpredictable.

Momentum [7]

In traditional neural networks the change in weights for each iteration is equal to the gradient multiplied by the learning rate. In CNNs, another parameter called momentum is used in this calculation. The formula is:

$$change = momentum * change - learning_{rate} * gradient$$
$$weights = weights + change$$

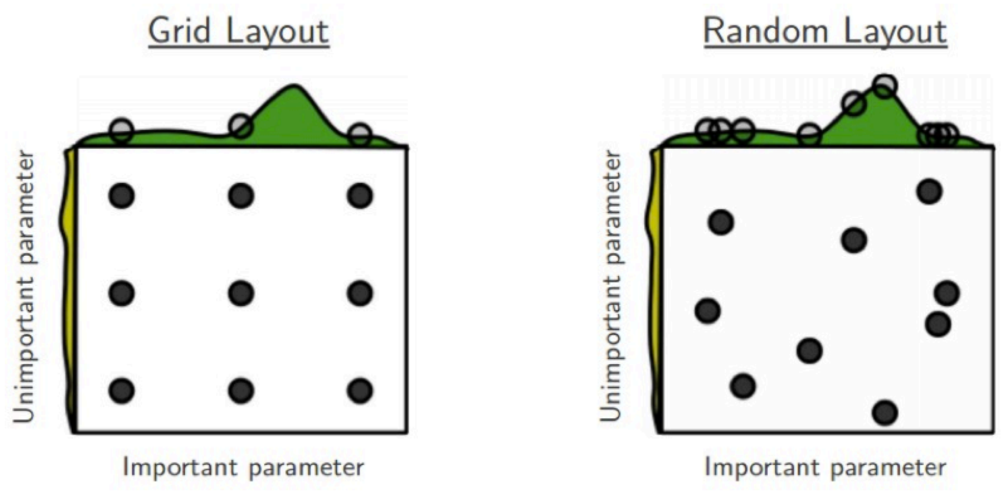
A common value for momentum is 0.9. As the change in weights depends on the change in the previous iteration, this stops the gradient from fluctuating too much. It puts the gradient on the same path it was already taking toward the local minima. This has empirically been shown to achieve better results.

Learning Decay [8]

Intuitively, when a model just starts learning, a higher learning rate is better as the gradient can move towards the minima faster. As it gets closer to the minima, lowering the learning rate would make the model more accurate as it can reach a lower point in the minima. So a common approach used is to lower the learning rate by a factor after a specific number of iterations. For example, we could start off with a learning rate of 0.1, decrease it to 0.01 (by a factor of 10) after 1000 iterations, decrease it to 0.001 after the next 1000 iterations, and so on.

Setting parameters [8]

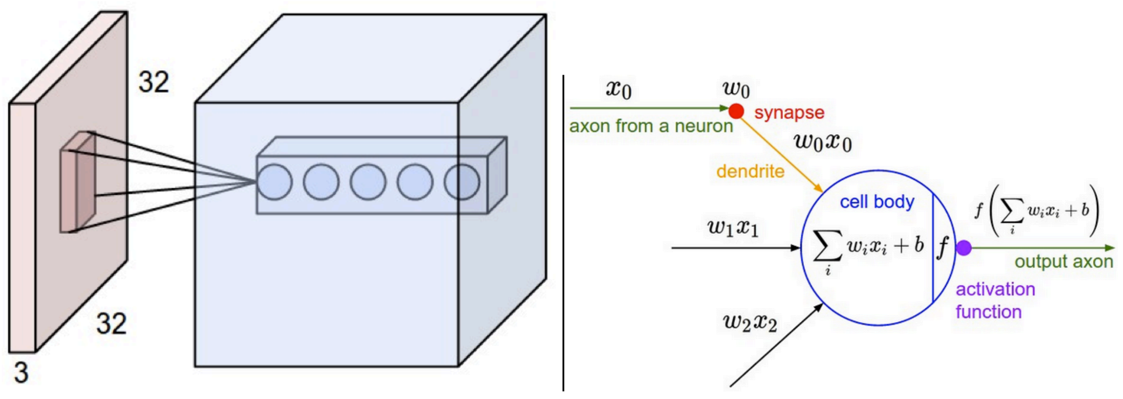
As seen in class, the parameters to be set for a neural network are the learning rate and the regularization. Here, the learning rate also includes the learning decay. Usually, because of the large input data in a neural network, instead of cross validation, just one validation set is used. Also, research shows that instead of using a grid to choose the values of parameters, doing a random search and optimizing based on results does better and is also faster, as shown in this picture.



CNN layers [9]

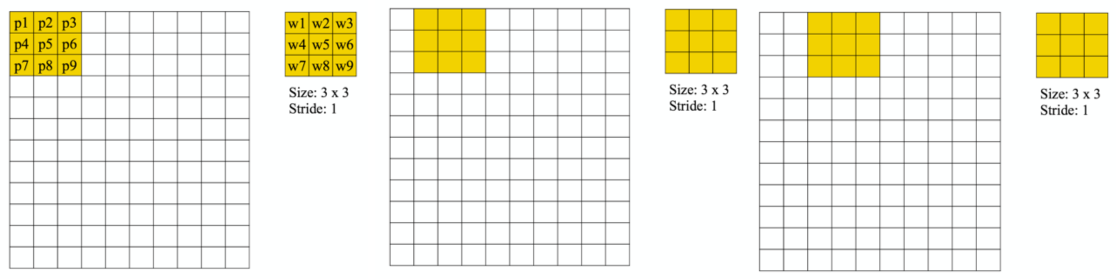
A convolutional neural network has layers specifically designed for image data. Consider an image with dimensions 256*256 pixels. This means that there are 65536 pixels in the image. Each pixel has three channels – red, green and blue. Each of these channels store a value between 0 and 256. So this image has 65536 * 3 numbers, each ranging between 0 and 256. If a traditional network was used to classify images, each neuron in the first hidden layer would have to learn 196608 weights. This does not scale. A CNN has convolutional layers, which make the model scalable.

Convolutional Layer



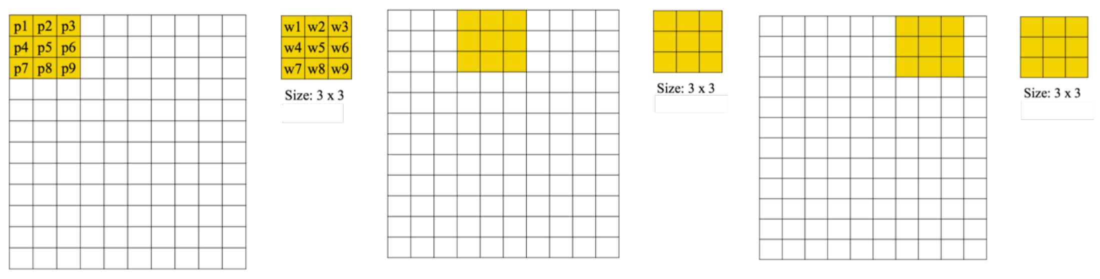
Consider an image with dimensions 32*32 pixels. As it has three channels, the dimensions are actually 32*32*3. As seen in the image, more than one convolutional layer is stacked up together. This is referred to as the depth of the output volume. The example in the image has a depth of 5.

Consider the neurons in the first convolutional layer in the output volume. Instead of being connected to all the pixels or inputs in the image, it is connected to a few pixels in a specific part of the image. If the size of these connected pixels is set to 3*3, each neuron in the first layer would be connected to 9 pixels. This is known as the field size.



[10]

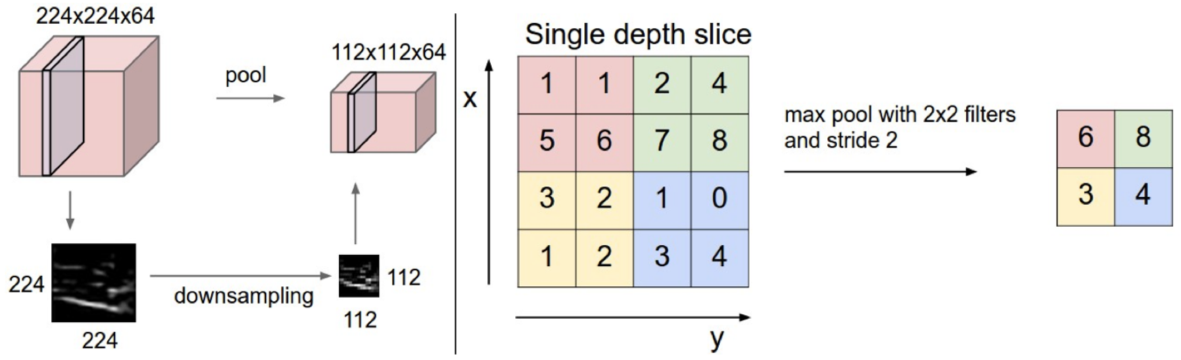
The first neuron in the first convolutional layer would be connected to the first 3*3 pixels in the image, the next would be connected to the next 3*3, and so on. Another parameter here is the stride. It defines how much the inputs should overlap for each neuron. In the example above the stride is set to 1, and so the second neuron's inputs are only one pixel away from the first neuron's inputs. If the stride is set to 3 instead, the difference would be 3 pixels as follows:



Intuitively, stride and the field size should be set in such a way that they exactly cover the entire input image. This is not true in the above case, as after the third input there is still one pixel left on the right border. In practice, the input image is padded with zero values at the boundaries to achieve this. This is known as the zero padding, and its value depends on the size of the input image, the field size, and the stride value.

Convolutional layers overcome the scaling problem mentioned earlier by sharing parameters or weights. Each layer in the output volume (the depth mentioned earlier) share the same weights. Consider the first convolutional layer. Each neuron has a field size of 3*3 and because the image has 3 channels, this means that each neuron has 3*3*3=27 weights. Every neuron in the first convolutional layer, even though they are connected to different regions in the input image, share the same 27 weights. Intuitively, this is because the weights are set to learn a particular characteristic in the image, maybe the color, or presence of edges. If the weights for a layer are set to learn the colors in the image, it makes sense for every neuron in the layer to learn colors for different parts of the image. This is where the depth of the volume comes into play. A neuron in the second layer of the volume which is exactly behind a neuron in the first layer, would learn from the same part of the image, but they would have a different set of weights (which is shared among all neurons in the second layer). So if the common weights of the first layer help the neurons in the first layer learn the colors for different parts of the image, the common weights for the second layer of neurons would maybe learn some other characteristic (like the presence of edges) in different parts of the image. The total number of weights for the volume would be 27*5 (27 for each layer, and 5 is the depth).

Pooling Layer



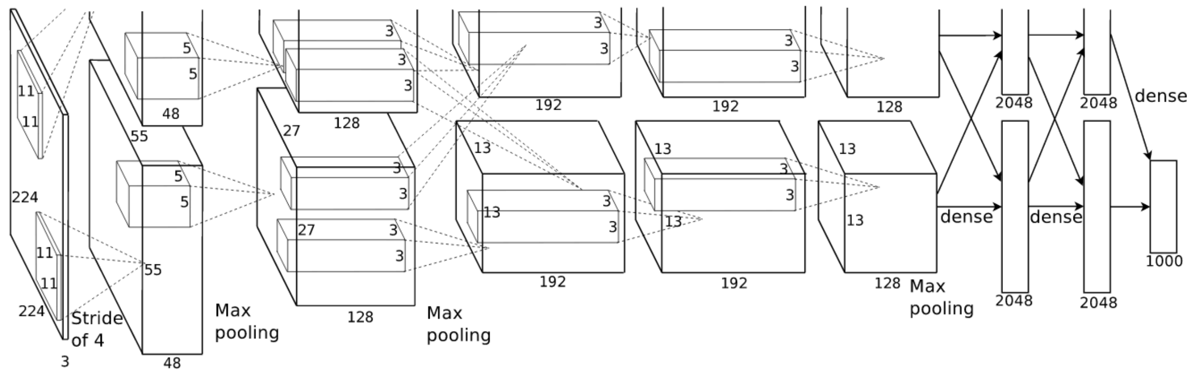
The pooling layer has a convolutional layer as input. It is used to prevent overfitting. In the image above, the pooling layer has a field size of 2*2, and a Max function. The Max function is commonly used in practice. This pooling layer takes the Max of every 2*2 grid in the input. This reduces the height by a factor of 2, and the width by a factor of 2 which results in a 75% reduction. So in the image, the input volume is of size 224*224 and has a depth of 64 (it has 64 layers). The output reduces each layer to 112*112. A field size of 2*2 is common in practice and so the input size is usually set to be divisible by 2 many times, so the model can have many pooling layers, each reducing the dimensions by a factor of 2. This is why input images of dimensions 256*256 are resized to 224*224 (this was mentioned earlier).

ReLU and Fully connected layers

It is common to have ReLU layers in between convolutional layers to introduce non-linearity. Fully connected layers are present at the end of the model, these are just traditional neural network layers.

AlexNet

The entire model used in the paper is shown below:



The input image is resized to 224*224. The field size of the first convolutional volume is 11*11, and it has a stride of 4. This results in 55*55 neurons in each layer of the first convolutional volume. The first convolutional volume has a depth of 48.

Max pooling is done after the first convolutional volume. Instead of the 2*2 pooling discussed earlier, this model uses a field size of 3*3 and a stride of 2 (this introduces overlapping in pooling). This was shown to have reduced the classification error by 0.4% when compared to the traditional non overlapping pooling.

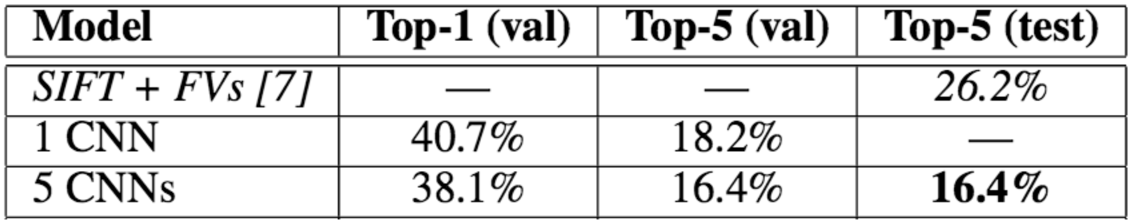
The reason each layer is shown as two sets in the image is because they were trained with two GPUs in parallel (for faster training). After the five convolutional volumes seen in the image, there are two fully connected layers with 4096 neurons each (2048 in each GPU). The final layer is a softmax function with 1000 class probabilities as output.

Each convolutional layer and fully connected layer is followed by a ReLU layer which computes the activation function for each neuron.

L2 regularization was done, and a single weight was used for all the neurons in the network for simplicity.

The other parameters used in the network are as follows:
Stochastic Gradient Descent Batch Size: 128
Momentum: 0.9
Dropout: 0.5
L2 Regularization Weight: 0.0005
Learning rate: Started with 0.01, reduced by a factor of 10 three times before termination

Results



Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**

As mentioned previously, the output has 1000 classes and the softmax function in the final layer gives the probability of all 1000 classes. There were two accuracies calculated, one was if the model predicted the right class, the other was if the right class was in the top 5 classes predicted by the model (the five classes with the highest probability). The misclassification rates are shown in the table. 5 CNNs is the average of five predictions on the same observation. The first row in the table is the accuracy of the second best model in the competition. AlexNet improves the misclassification rate by around 10%.

Transfer Learning [11]

A CNN needs to be trained on a lot of input data to perform well. The ImageNet competition has 1.2 million images. In practice, training a CNN from scratch is not feasible without the proper computational resources. CNN software, like Caffe, overcome this by letting us download and run predictions on pre-trained models. Also, as the convolutional layers share weights to learn specific characteristics of the image, it is common for the outputs of the early volumes to be similar for any image data. They learn the basic characteristics like the color and the presence of edges. The later volumes learn more specific characteristics depending on the input data, like the shapes of objects. There are two common ways a pre-trained model can be applied to our data.

1. Feature extraction: If the last layer (the softmax function which predicts probabilities) is removed from the model and predictions are done on our dataset, we get the output of the final fully connected layer. This is a vector with length 4096, which are essentially

features extracted from the image. This vector could be extracted for each observation in our data, and a different classification model could be trained with these features as inputs.

2. Fine tuning: The pre-trained model could be retrained with back propagation on a new set of input data. CNN software like Caffe allow us to do this.

Implementation

I used Caffe to run a pre-trained CNN model. I ran it on AWS (rented computation resources, free for students). Instead of the AlexNet model, I ran a newer model called VGGNet. This has a 5% misclassification rate, comparable to humans. Caffe has a convenient web interface which I have implemented. It lets us upload any image and get the classification predictions. You can find the implementation at http://54.175.110.234:5000/

References

[1] http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf
[2] http://cs231n.github.io/
[3] http://cs231n.github.io/neural-networks-1/
[4] Tom's Neural Network Lecture notes
[5] http://stackoverflow.com/questions/11677508/why-do-sigmoid-functions-work-in-neural-nets
[6] http://cs231n.github.io/neural-networks-2/
[7] http://stats.stackexchange.com/questions/70101/neural-networks-weight-change-momentum-and-weight-decay
[8] http://cs231n.github.io/neural-networks-3/#anneal
[9] http://cs231n.github.io/convolutional-networks/#overview
[10] https://courses.cs.sfu.ca/2015fa-cmpt-733-g1/pages/W3_CNN/view
[11] http://cs231n.github.io/transfer-learning/