Status Report:

Lung Cancer detection using 3D Convolutional Neural Network

Pranali Pawar • Shreyansh Chajjer • Shubham Talbar

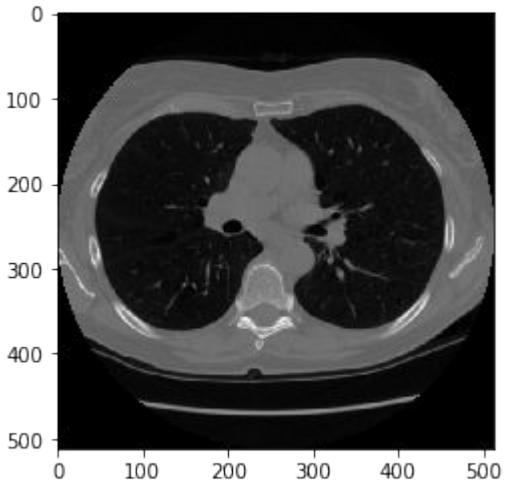
Problem statement

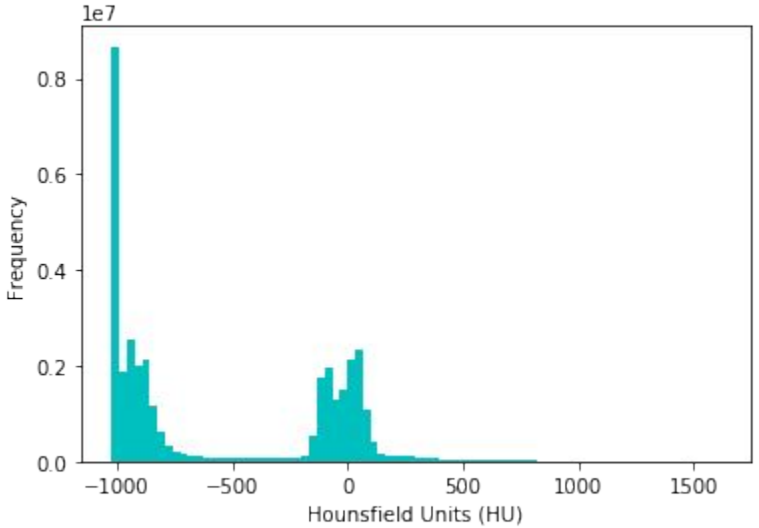
- The following problem is an actual competition previously hosted by Kaggle as “Data Science Bowl 2017”.
- The aim here is to take the sample data, consisting of low-dose CT scan information, and predict what the likelihood of a patient having lung cancer is.
- Accuracy is scored based on the log loss of predictions.
- The dataset is pretty large at ~140GB just in initial training data.
- As of now we have generated results on a sample dataset **~1GB** consisting CT scans of **20 patients** by implementing a **3D CNN model**.

Handling Data:

- The data consists of many 2D "slices," which, when combined, produce a 3-dimensional rendering of whatever was scanned. In this case, that's the chest cavity of the patient.
- We've got CT scans of total 1397 patients, and then we've got another file that contains the labels for this data. The labels indicate whether the patient was diagnosed with cancer within next 1 year or not.
- The scans are in "dicom" format.

Sample CT scan slice and its histogram



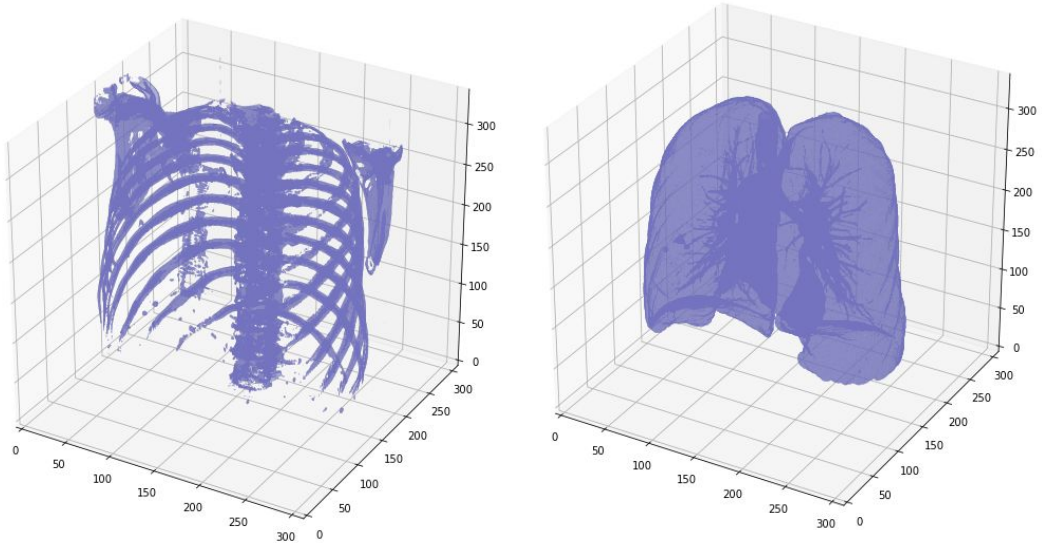


Sample label data

	cancer
id	
0015ceb851d7251b8f399e39779d1e7d	1
0030a160d58723ff36d73f41b170ec21	0
003f41c78e6acfa92430a057ac0b306e	0

3D plotting the scan

- For visualization it is useful to be able to show a 3D image of the scan. We have used **marching cubes** to create an approximate mesh for our 3D object, and plot this with matplotlib.



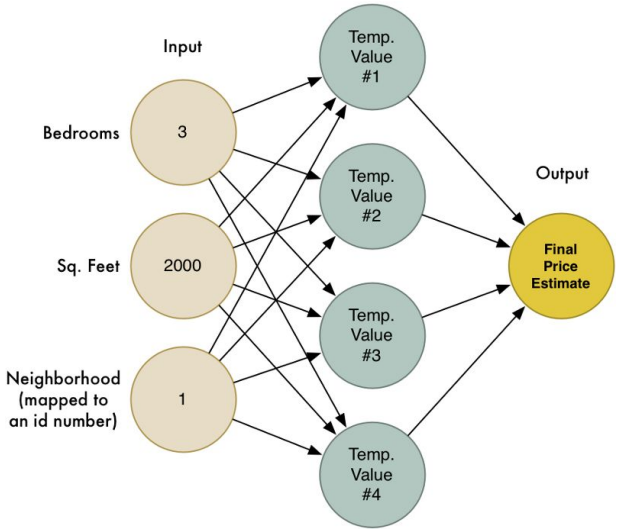
Why Convolutional Neural Networks?

Deep Learning

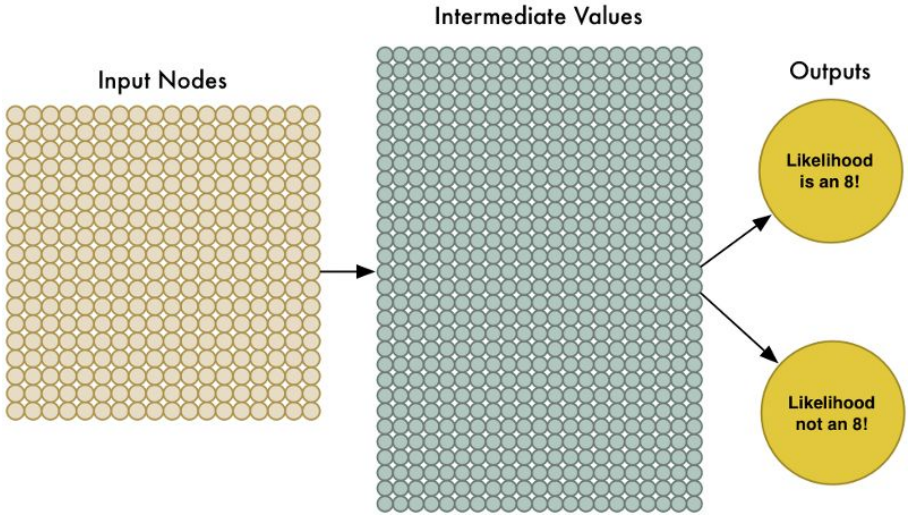
- The idea is to use deep learning to segment cancer nodules in CT images. In other words, we're going to explain the black magic that allows Google Photos to search your photos based on what is in the picture.

Neural Networks

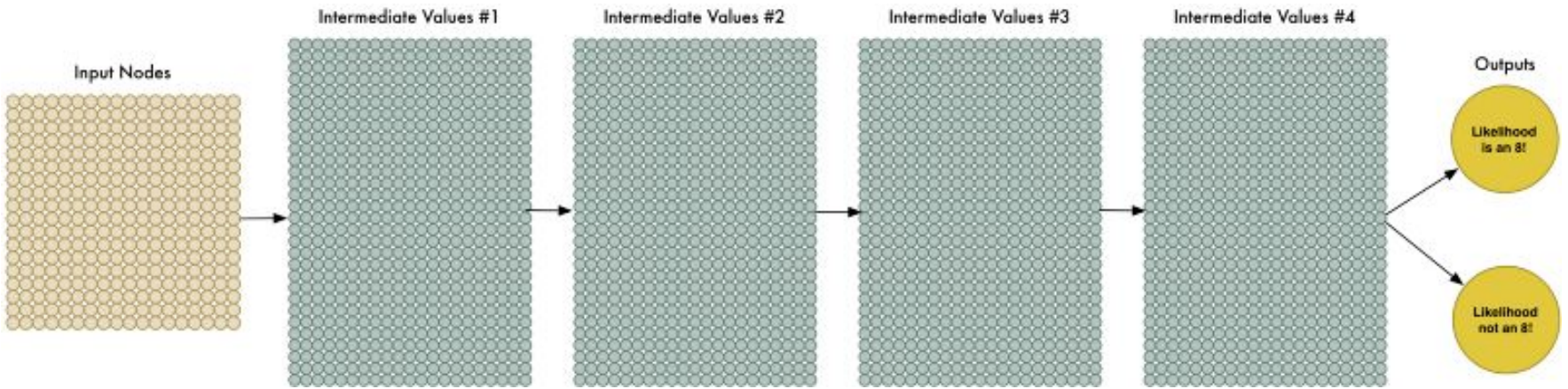
- Following is an example of a small neural network to estimate the price of a house based on **how many bedrooms it had, how big it was, and which neighborhood it was in**.



- Since to a computer, an image is really just a grid of numbers that represent how dark each pixel is. We can feed a neural network with numbers as input.
- Let's say the task is to identify whether the numeral **"8"** lies in an image or not, the neural network would look something like this:



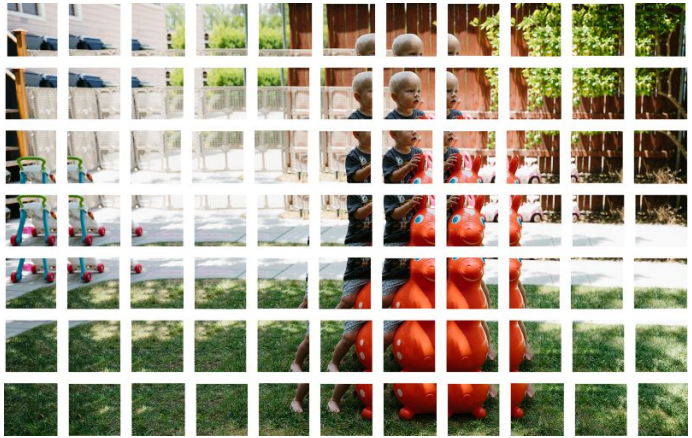
- For a machine learning algorithm to work well we need lots of data. More data makes the problem harder for our neural network to solve, but we can compensate for that by making our network bigger and thus able to learn more complicated patterns.
- To make the network bigger, we just stack up layer upon layer of nodes.
- We call this a **“deep neural network”** because it has more layers than a traditional neural network.


Input Nodes
Intermediate Values #1
Intermediate Values #2
Intermediate Values #3
Intermediate Values #4
Outputs
Likelihood is an 8!
Likelihood not an 8!

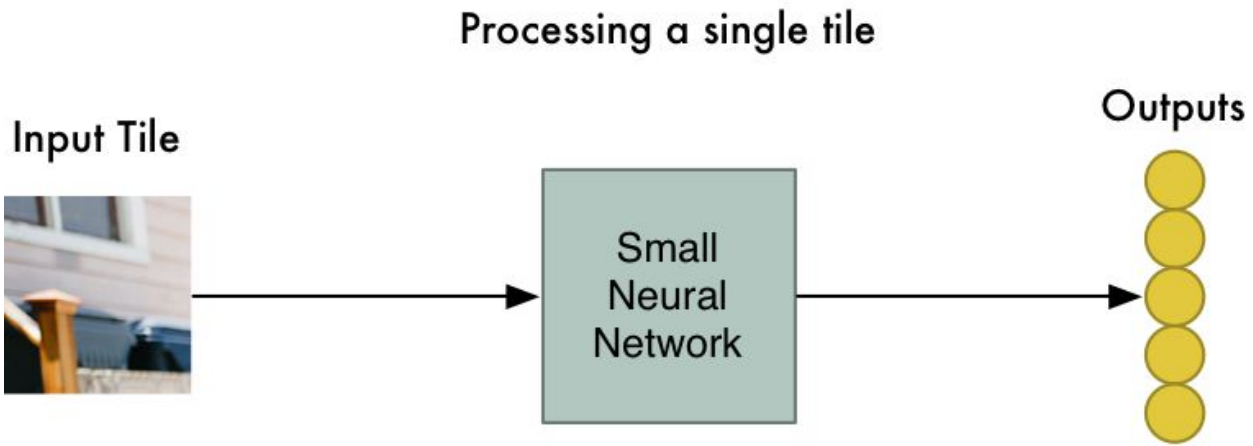
- It doesn’t make sense to train a network to recognize an “8” at the top of a picture separately from training it to recognize an “8” at the bottom of a picture as if those were two totally different objects.
- The solution to this problem is **Convolution**.
- Instead of feeding entire images into our neural network as one grid of numbers, we’re going to do something a lot smarter that takes advantage of the idea that an object is the same no matter where it appears in a picture.
- The idea is to break an image into overlapping image files.

- Input Image

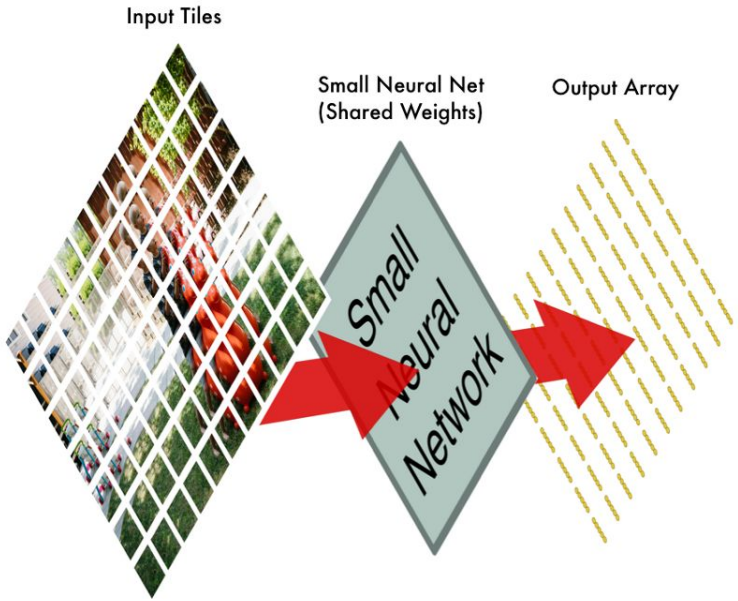
- Convolved Image



- Processing one single tile of input image:

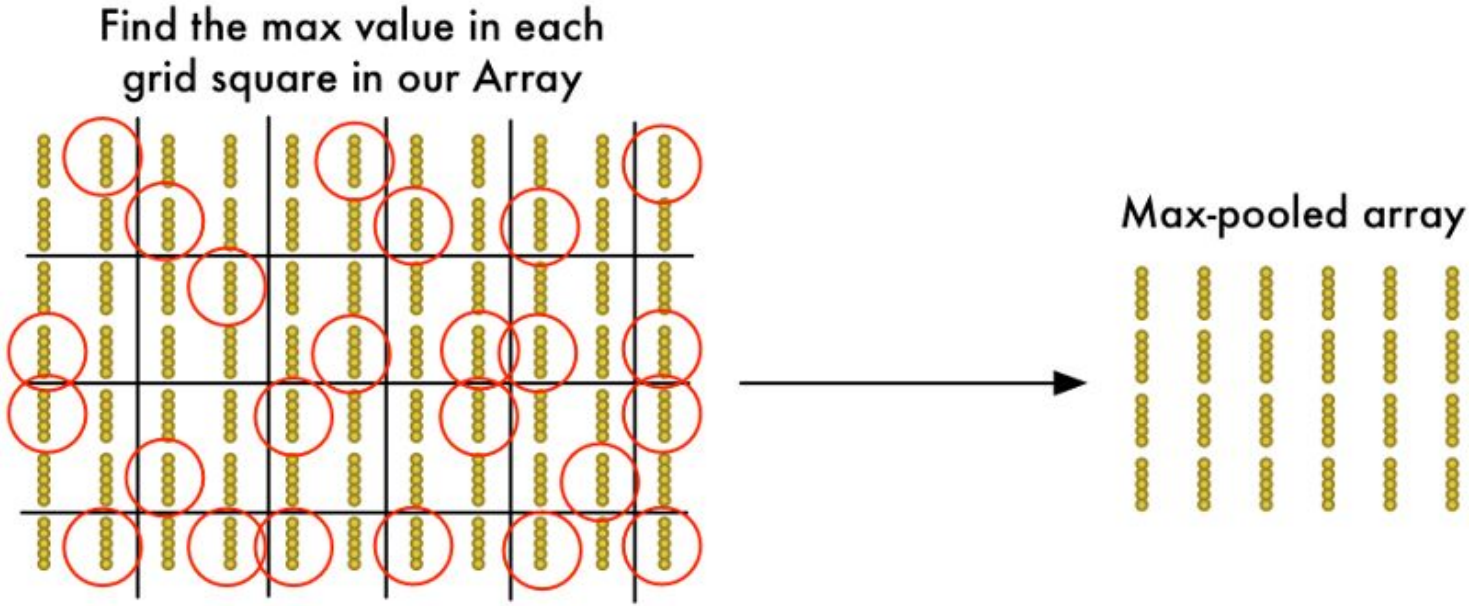


- Save the results from each tile into a new array:



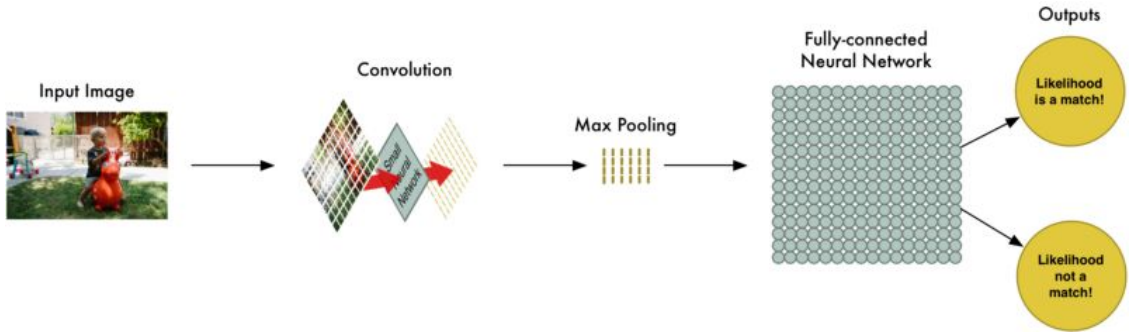
- **Downsampling:** The result of previous step was an array that maps out which parts of the original image are the most interesting. But that array is still pretty big.
- To reduce the size of the array, we *downsample* it using an algorithm called max pooling.
- We'll just look at each **2x2 square** of the array and keep the biggest number.
- The idea here is that if we found something interesting in any of the four input tiles that makes up each 2x2 grid square, we'll just keep the most interesting bit. **This reduces the size of our array while keeping the most important bits.**

- Max pooling:


Find the max value in each grid square in our Array
Max-pooled array

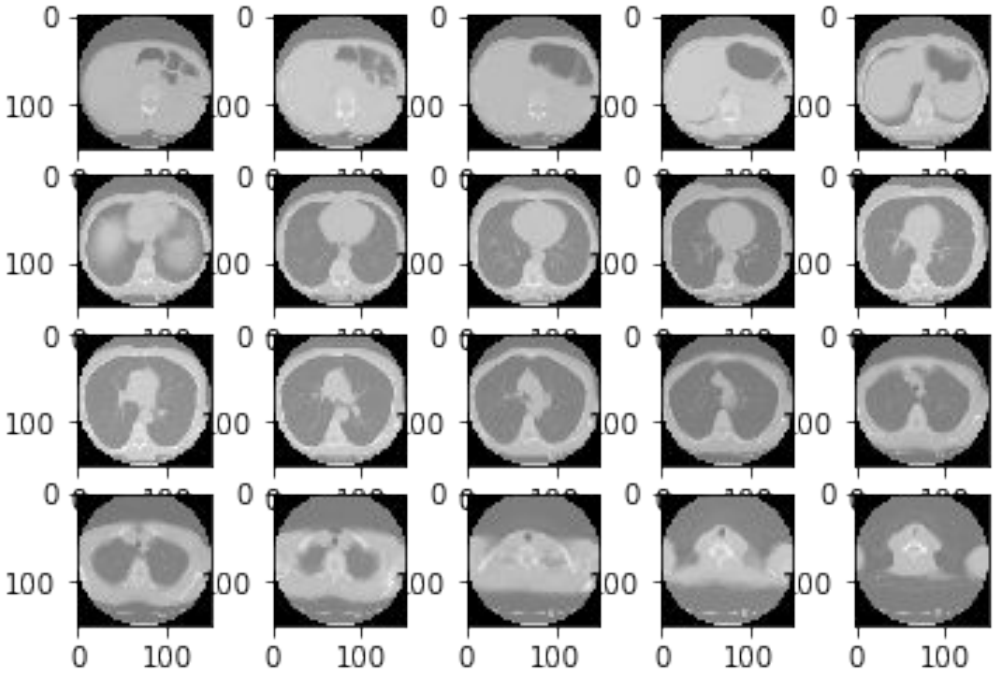
Final Step: Make a prediction

- That array is just a bunch of numbers, so we can use that small array as input into *another neural network*. This final neural network will decide if the image is or isn't a match.
- To differentiate it from the convolution step, we call it a **"fully connected"** network.



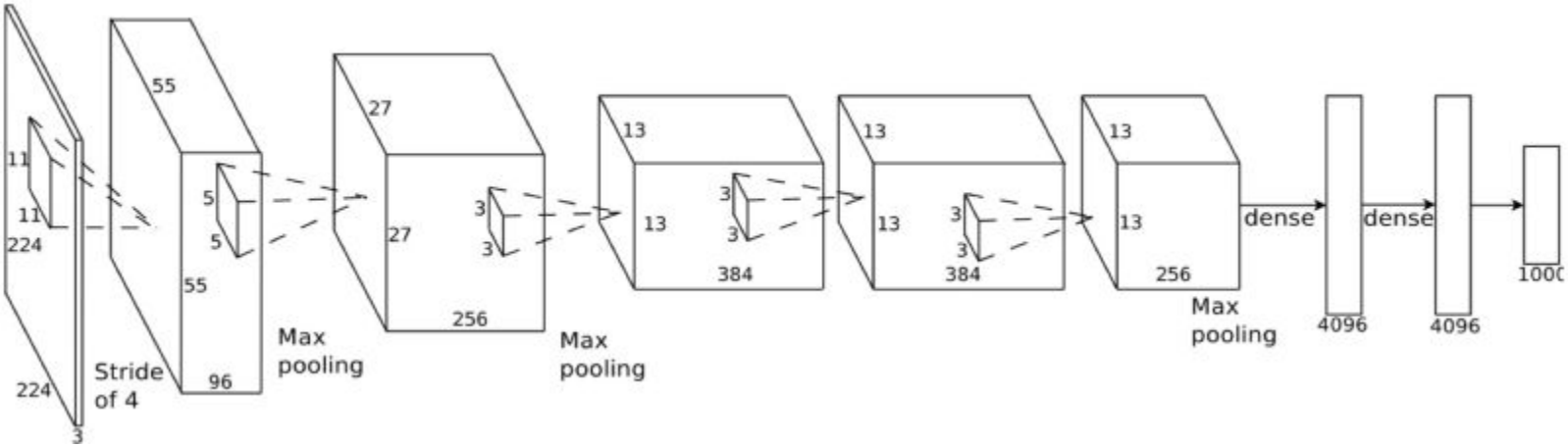
Preprocessing CT scans

- Since the dimension of each 2D slice was 512x512 it was important to reduce the dimension of each patient slice. Moreover the number of slices in each patient's CT scan varied from patient to patient. There were instances with a CT scan having 150 slices while also other ones with 250 slices.
- In order to reduce data and bring uniformity, the number of slices in all patients were mapped to just 20. The slice dimension was also reduced to 50x50.



- Preprocessed CT scan of a patient with initially 147 slices.
- The new dimension of each slice is 50x50 while total number of slices is 20.

Our CNN model: Note the multiple max pooling and fully connection steps.



Results

- Due to unavailability of sufficient computational power, as of now we have used a sample dataset of just **20 patients ~1GB**.
- Out of 20, 18 CT scans were used for training and 2 for testing, we did 10-fold cross validation over the dataset.
- The accuracy achieved was **67%** with **10 iterations**.
- With sufficient computational power, we speculate that increasing the number of iterations and training our CNN model over a larger dataset can further improve results.

References

- https://www.kaggle.com/c/data-science-bowl-2017
- https://www.kaggle.com/sentdex/first-pass-through-data-w-3d-convnet
- https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721
- https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial