Medical Image Segmentation to Diagnosis Alzheimer Disease using Neural Networks

R.Anitha 1  
Research Scholar  
Sri Padmavathi Mahila University,Tirupathi

S.Jyothi 2  
Professor  
Sri Padmavathi Mahila University,Tirupathi

P.Siva Krishna 3  
Asst.Professor  
PACE Institute of Tech&Sciences,Ongole

Abstract:  
This article provides an overview of the current state of the art deep learning architectures and optimization techniques, and uses the ADNI hippocampus MRI dataset as an example to compare the effectiveness and efficiency of different convolution architectures on the task of patch-based 3-dimensional hippocampus segmentation, which is important in the diagnosis of Alzheimer's Disease. We also examined the popular “tri-planar” approach, and found that it provides much better results than the 2D approaches. Finally, we evaluated a full 3D convolution architecture, and found that it provides marginally better results than the tri-planar approach, but at the cost of a very significant increase in computational power requirement.

Keywords: Alzheimer's disease, ADNI Hippo camps, MRI, 2D approaches etc.

I.INTRODUCTION

The most common cause of dementia occurring in patients over 45 yrs of age is Alzheimer's disease. People who get Alzheimer's disease are usually older, but the disease isn't a normal part of aging. It causes seem to come from two main types of nerve damage: Nerv cells get tangles, called neurofibrillary tangles and Protein deposits called beta-amyloidial plaques build up in the brain.

Deep learning techniques have been applied to a wide variety of problems in recent years - most prominently in computer vision and computational audio analysis. In many of these applications, algorithms based on deep machine learning have surpassed the previous state-of-art performance.

II.ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANN) are a machine learning technique inspired by and loosely based on biological neural networks (BNN). ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. While they are similar in the sense that they both use a large number of identical and linked simple computational units to achieve high performance on complex tasks, modern ANNs have been so heavily optimized for efficient implementation on electronic computers that they bear little resemblance to their biological counterpart.

In particular, time-dependent integrate-and-fire mechanism in BNNs have been replaced by steady state values representing frequency of ring, and most ANNs also have vastly simplified connection architectures that allow for efficient propagation.

A/Network Architectures

In a typical neural network, nodes are placed in layers, with the first layer being the input layer, and the last layer being the output layer. The input nodes are special in that their outputs are simply the value of the corresponding features in the input vector. For example, in a classification task that has a 3-dimensional input (x, y, z) and a binary output, one possible network design is to have 3 input nodes, and 1 output node. The input and output layers are usually considered fixed in network design.

Fig1: Structure of Alzheimer's Brain Cell

Fig2: Neural network with 3 nodes input and 1 node output
With only an input layer and an output layer, with all input nodes connected to all output nodes, the network essentially implements a matrix multiply, or a linear transformation. This type of networks can solve simple problems where the feature space is linearly-separable. However, for linearly-separable problems, simpler techniques such as linear regression or logistic regression can usually achieve similar performance, with the only difference being training methods.

Most modern applications of neural networks use one or more hidden layers, layers that sit between the input layer and the output layers, to allow the network to model non-linearity in the feature space. The number of hidden layers and the number of hidden nodes in each layer are hyper-parameters that are not always easy to determine. While some rules-of-thumb have been proposed, they are still, for the most part, determined by trial-and-error. The risk of using too-small a network is that it may not have enough representative power to model all useful patterns in the input (high bias), while the risk of using too-large a network is that it may over t the data, and start modeling noise in the training set (high variance). It is usually better to err on the side of larger networks, because many effective techniques exist to combat over fitting, as will be detailed in later sections of the report. Using network size to limit over fitting is error-prone, time-consuming, and not very effective.

However, these methods have not seen widespread adoption, due to the large increase in training time, and marginal benefits when over fitting is avoided using other methods than limiting network size.

It has been proven that a network with 1 hidden layer can approximate any continuous (in feature space) function to any accuracy, and a network with 2 hidden layers can approximate any function to any accuracy. Given in finite computational power, memory, and training set, there is theoretically no reason to go above 2 hidden layers.

B) Training Neural Networks

Training algorithms for neural networks fall into two major categories - gradient-based and non-gradient-based. This report focuses on gradient-based methods as it is much more commonly used in recent times, and usually converges much faster as well. The weight and bias of a node are the parameters of the node. If we concatenate the weightand biases of all nodes in a network into one vector, it completely different the behavior of a network (for a given set of hyper-parameters, ie. network architecture).

If the set of hyper-parameters (network architecture) is encoded into a function f(), we can define the output of the network ,where y and x are the output and input vectors respectively. The goal of the training process, therefore, is to find a θ, such that f(θ,X);approximates the function we are trying to model. In other words, given a set of inputs and their desired outputs, we are trying to find a θ that minimizes the difference between the desired outputs and network outputs, for all entries in the training set. For that, a measurement of error is needed.

\[ E(θ,T_s) = \frac{1}{2} \sum_{(x_i,y_i) \in T_s} (y_i - f(θ,x_i))^2 \]

One such error measure is mean-squared-error (MSE), and it is the most commonly used error measure. This is given in Equation 3, where Ts is the training set, xi and yi are the input and desired output of a training pair, N is the number of entries in the training set, and g(xi) is the network output.

Our goal is to minimize E(θ,Ts) given Ts. For very small networks, it may be feasible to do an exhaustive search to find the point in parameter space where the mean-squared-error is minimized, but for networks of reasonable sizes, an exhaustive search is not practical. Gradient descent is the most commonly used optimization algorithm for neural networks.

Another common variation is learning rate scheduling - changing the learning rate as training progresses. The goal is to get to somewhere close to a local minimum quickly, then slow down to avoid overshooting. This idea is taken further in resilient back-propagation (RPROP), where only the sign of the gradient is used. In RPROP, each weight has an independent learning rate, that is increased (usually multiplied by 1.2) if the sign of the gradient has not changed from the previous iteration, and reduced (usually by a factor of 0.5) if the gradient has changed signs [15]. This allows all weights to train at close to their optimal learning rate, and eliminates the learning rate parameter that must be manually tuned in other gradient descent variants. An initial learning rate still needs to be chosen, but it doesn’t significantly affect training time or result.
One popular way to combat over fitting is regularization - the idea of encouraging weights to have smaller values. The most common form of regularization is L2 regularization, where the L2 norm of (the parameter vector) θ is added to the error function.

C) Why Build Deep Neural Networks?

A neural network with 2 hidden layers is already theoretically a universal function approximate capable of approximating any function, continuous or not, to any arbitrary accuracy. In light of that, it may seem pointless to pursue networks with more hidden layers.

The main benefit of using deep networks is node efficiency - it is often possible to approximate complex functions to the same accuracy using a deeper network with much fewer total nodes compared to a 2-hidden-layer network with very large hidden layers. Besides computational benefit, a model with a smaller degree of freedom (number of parameters) requires a smaller dataset to train [16], and size of the training set is often a limiting factor in neural network training.

Intuitively, the reason for a smaller and deeper network to be more effective than an equally sized (in total nodes) shallower network is that a deep network reduces the amount of redundant work done. On the other hand, if we have a shallow network, the low level tasks would have to be performed multiple times, since there is little cross-feeding of intermediate results. This would result in a lot of redundant work done.

With very deep networks, it is possible to model functions that work with many layers of abstraction for example, classifying the gender of images of faces, or the breed of dogs. It is not practical to perform these tasks using shallow networks, because the redundant work done is exponential to the number of layers, and an equivalent shallow network would require exponentially more computational power, and exponentially larger training sets, neither of which are usually available.

Making Deep Nets Shallow Continuing on the theme of model compression, with the help of a high performance deep network, a shallow network can be trained to perform much better than a similar shallow network that is trained directly on the training set.

Their algorithm first trains a deep net (or an ensemble of deep nets) on the original training set, and use it to provide "extended labels" for all entries in the training set. In case of classification problems where the output layer is often softmax, the "extended labels" are inputs to the softmax layer. The inputs to the softmax layer are log probabilities of each class. Finally, a shallow network can be trained to predict the log probabilities, instead of the original single-class label. This makes training much easier for the shallow network, because multi-class log probabilities labels provide much more information than a single-class label. By modeling the log probabilities, the shallow network is also mimicking how the deep net (or ensemble) will generalize to unseen data, which mostly depend on the relative values of log probabilities for classes that are not the highest. The performances of these shallow networks are much higher than networks trained on the single-class labels.

This result is significant because it proves that the reason why shallow networks perform worse than deep networks is not entirely due to the increase in representational power and edibility of deep net-works. It is also due to our current training algorithms being sub-optimal for shallow networks, and if we can develop better training algorithms, we can potentially significantly improve the performance of shallow networks.

However, the performance of these mimic-ing shallow networks is still not quite as good as the deep networks or ensembles they are mimic-ing. Therefore, the option of creating a mimic-ing shallow network allows a tradeoff to be made between accuracy and speed.

Convolutional Neural Networks

Convolution neural networks are a neural network architecture that uses extensive weight-sharing to reduce the degrees of freedom of models that operate on features that are spatially correlated. This includes 2D and 3D images and 2D videos, which can be seen as 3D images), but it has also very recently been successfully applied to natural language processing.

Convolutional neural networks are inspired by the observation that for inputs likes images (with each pixel being an input dimension), many low level operations are local, and they are not position dependent. For example, one operation that is useful in many computer vision applications is edge detection. In a fully-connected deep network, the edge detector would have to be separately trained for each part of the image, even though they would most likely all arrive at similar results. It would be better if a kernel can be trained to do edge detection for the entire image at the same time. In its current iteration, convolution neural networks are composed of 3 different types of layers - Convolution, max-pooling, and fully-connected. One typical arrangement is alternating between convolution and max pooling layers, before finishing off with 2 fully-connected hidden layers.
Each convolution layer has the same dimensions as the input, but each pixel is only activated by a region of pixels centered around the pixels at the same location in the input images. The weights are also shared for each output pixel. In effect, each map in a convolution layer performs a convolution of the input images, with a learned kernel. Max-pooling layers perform down sampling on the images. One typical down sample factor is 2x2 (dividing both width and height by 2). While averaging can also be used, empirical results suggest that down sampling by taking the maximum in each sub-region gives the best performance in most cases [28]. Max-pooling is responsible for summarizing each sub-region, and it gives the network some translational and rotational invariance.

Fully-connected layers are often used as final layers to encode position-dependent information and more global patterns.

Most existing applications of convolution neural networks are on 2D images, but the idea can also be extended to 3D, with 3D images and 3D kernels. It can be used to process actual 3D images (eg. MR Images), or videos, using time as the third dimension. Max-pooling layers perform down sampling on the images. One typical down sample factor is 2x2 (dividing both width and height by 2). While averaging can also be used, empirical results suggest that down sampling by taking the maximum in each sub-region gives the best performance in most cases. Max-pooling is responsible for summarizing each sub-region, and it gives the network some translational and rotational invariance.

Fully-connected layers are often used as final layers to encode position-dependent information and more global patterns.

Most existing applications of convolution neural networks are on 2D images, but the idea can also be extended to 3D, with 3D images and 3D kernels. It can be used to process actual 3D images (eg. MR Images), or videos, using time as the third dimension.

D) Hippocampus Segmentation

![Fig4: Hippocampus of human brain](image)

The hippocampus is a component of human brains responsible for committing short-term episodic and declarative memory into long-term memory, as well as navigation. Hippocampus segmentation is important in the diagnosis of Alzheimer’s disease (AD), as it is one of the components first affected by the disease. A reduction in hippocampus volume can be used as a marker for AD diagnosis.

Humans have 2 hippocampus, shaped like seahorses, as shown in Figure 4. Our goal is to classify each voxel in an MR Image as non-hippocampus, left hippocampus, or right hippocampus. We are using this problem to evaluate different deep learning techniques for patch-based segmentation. All images are labeled by one human expert. Unfortunately, none of the images have been labeled by more than 1 human expert to determine variances in human labeling.

We explore 3 convolution neural network architectures for patch-based segmentation on the ADNI Alzheimer's MRI dataset. For all 3 cases, the pre-processing and post-processing done are identical. For all 3 cases, 60% (120) of the images are used as the training set, 20% (40) as the validation set, and 20% (40) as the testing set. Patches from the same image are always only used in one of the sets.

E) Pre-Processing

Before we begin labeling an image, we first crop it down to a rectangular bounding box, so we can perform masking in normalized coordinates. In case of the ADNI dataset, all images are already in the same orientation, so no rotation is needed.

From going through all images in the training set, we determined that the hippocampi are always in the region (0.42 < x < 0.81; 0.30 < y < 0.67; 0.22 < z < 0.80), relative to each dimension of the bounding box of their respective brains. We enlarged the region by 0.03 on each side, and use (0.39 < x < 0.84; 0.27 < y < 0.70; 0.19 < z < 0.83) as the mask. All voxels outside of the mask are automatically classified as non-hippocampus. All training patches are drawn from within the mask.

F) Sampling

It would be dangerous to draw training voxels uniformly randomly from within the mask, because even within the mask, the vast majority of voxels are non-hippocampus, and hence there would be very few positive samples. Another problem is that edge voxels (voxels at the edges between positive and negative voxels) would be severely under-represented, even though they will most likely be the most difficult voxels to classify. Therefore, we draw samples as follows -

For 50% of the samples, we keep drawing randomly until we get a voxel where the 5x5x5 bounding box around the voxel contains more than 1 class. For 25% of the samples, we keep drawing randomly until we...
get a positive voxel. For the remaining 25% of the samples, we keep drawing randomly until we get a negative voxel. This drawing scheme ensures that none of the important types of voxels are under-represented. The biggest downside of this scheme is that it distorts the prior probabilities of each class, possible solutions to which are discussed later in the report.

G) Stacked 2D Patches

The first method we tried is to use a stack of 2D patches around each voxel we want to sample. For example, for a patch size of 24 and a layer count of 3, we would extract three 24x24 patches - one around the voxel in question, one in parallel and above, and one in parallel and below. Each of the layers are given to a 2D convolution neural network as different channels. This method gives the network some 3D context around the voxel (in case of stack sizes greater than 1), at a relatively low space overhead. However, the network is not convolution in the third dimension. Network architecture is 20 5x5 kernels in first convolution layer, 50 5x5 kernels in the second convolutional layer, a 1000 nodes fully-connected layer, then finally a softmax layer for exponential normalization. No max-pooling is used, since network performs slightly worse with any max-pooling.

H) Tri-planar Patches

The second method we tried is the tri-planar method used in medical imaging applications. For each voxel, we extract 3 square patches around the voxel, perpendicular to each axis. For example, if we want a patch size of 24, we would extract a 24x24 patch on the x-y plane centered around the voxel in question, and a 24x24 patch on the x-z plane, and another 24x24 patch on the y-z plane.

Since the corresponding pixels from the 3 patches are not spatially correlated in this case, we use a network architecture that consists of 2 convolution layers (20 5x5 kernels and 50 5x5 kernels) for each of the 3 patches with no connections between them until the very end, where we feed all their outputs into a 1000 nodes fully-connected layer for final classification. No max-pooling is used. 3D Patches

This approach is an intuitive extension of the 2D approach into 3D. For each voxel we want to sample, we take a 3D patch with equal length on each side, around the voxel. For a patch size of 24, we would extract 24x24x24 patches.

Network architecture is 20 5x5x5 kernels in the first convolution layer, 50 5x5x5 kernels in the second convolution layer, then a 1000 nodes fully-connected layer as before. No max-pooling is used.

I) Image Labeling

After the network is trained, to label an image, patches are extracted for every voxel in the mask region (in the correct format for the network architecture in use), and the result is used to label the voxel. Any voxel outside of the mask region is automatically classified as negative.

K) Training

All network training are done with standard stochastic gradient descent with a batch size of 50 and a fixed learning rate of 0.01. At the beginning of training, termination iteration is set to 1 validation period. Validation is done after every pass through the training set (24,000 patches). Every time a validation score improves the current best validation score by more than 1% (in error, not classification rate), the terminating iteration is set to twice the current iteration count. This means training will only be terminated if there is no significant improvement for at least the second half of the elapsed time.

L) Post-Processing

Besides comparing raw output after labeling by convolution neural networks, we also want to see what kind of performance can we get. For each labeled image, we first calculate the centroid of all voxels labeled left-hippocampus, and the centroid of all voxels labeled right-hippocampus. For each labeled image, we first calculate the centroid of all voxels labeled left-hippocampus, and the centroid of all voxels labeled right-hippocampus. Then, we divide up the image into blobs (connected voxels with the same classification), and for each blob, we check their size. If a blob is smaller than a certain threshold (500 voxels in our case), and the labeling is negative (non-hippocampus), it is re-labeled to be the nearest hippocampus (based on centroid). The next
experiment is to test different patch sizes, also with the 2D architecture. As shown in Table 1, there are little benefits in going beyond 24x24. However, in this case, training becomes much slower for larger patch sizes (in time per iteration). Therefore, the optimal patch size seems to be 24x24.

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Best Val Perf</th>
<th>Test Perf</th>
<th>False Pos (vxl)</th>
<th>False Neg (vxl)</th>
<th>Iter</th>
<th>T (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>7.54%</td>
<td>8.66%</td>
<td>1690</td>
<td>117</td>
<td>12960</td>
<td>47.71</td>
</tr>
<tr>
<td>16</td>
<td>6.73%</td>
<td>8.05%</td>
<td>1799</td>
<td>108</td>
<td>7680</td>
<td>66.53</td>
</tr>
<tr>
<td>20</td>
<td>7.06%</td>
<td>7.50%</td>
<td>729</td>
<td>127</td>
<td>13920</td>
<td>194.08</td>
</tr>
</tbody>
</table>

Table 1: Performance using different 2D patch sizes

Finally, we look at patch sizes for the 3D architecture. Unfortunately, we are constrained by available GPU memory in this case, and can only use up to 20x20x20 patches. We nd that unlike the previous two architectures, the 3D architecture performs well even at a very small patch size of 12x12x12, and it is not clear whether it actually benefits from having larger patches, as shown in Table 3.

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Best Val Perf</th>
<th>Test Perf</th>
<th>False Pos (vxl)</th>
<th>False Neg (vxl)</th>
<th>Iter</th>
<th>T (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>11.33%</td>
<td>12.64%</td>
<td>2341</td>
<td>275</td>
<td>72960</td>
<td>10.93</td>
</tr>
<tr>
<td>16</td>
<td>10.20%</td>
<td>10.64%</td>
<td>674</td>
<td>234</td>
<td>47520</td>
<td>12.41</td>
</tr>
<tr>
<td>24</td>
<td>8.85%</td>
<td>10.00%</td>
<td>778</td>
<td>278</td>
<td>24480</td>
<td>11.72</td>
</tr>
<tr>
<td>32</td>
<td>8.98%</td>
<td>9.86%</td>
<td>647</td>
<td>283</td>
<td>19680</td>
<td>13.72</td>
</tr>
<tr>
<td>48</td>
<td>8.96%</td>
<td>9.80%</td>
<td>547</td>
<td>257</td>
<td>22560</td>
<td>27.96</td>
</tr>
</tbody>
</table>

Table 3: Performance using different 3D patch sizes

IV. CONCLUSION AND FUTURE WORK

In this article we investigated the use of three different convolution network architectures for patch-based segmentation of the hippocampus region in MRI images. We discovered that the popular tri-planar approach offers a good tradeoff between accuracy and training time. While the 3D approach performs marginally better at patch classification, it does not seem to perform as well at labeling an entire image. This is most likely due to the sampling method altering prior probabilities of the classes. Presented to the training algorithm, and if this problem is solved, the 3D approach should perform marginally better than the tri-planar approach in whole-image labeling as well, but with a much higher computational power requirement.

REFERENCES