Performance of Convolutional Neural Networks against Capsule Networks in Medical Imaging Data.

Computer Vision

University of Energy and Natural Resources

AUGUST 2020.

SUPERVISOR

Dr. Dr. Benjamin Asubam Weyori

REPRESENTATIVE

Botwe Asamoah Emmanuel	**UE20001016**
Adu Bruce Emmanuel	**UE20000316**
Joseph Adomako Junior	**UE20000216**

Performance of Convolutional neural networks against capsule networks in medical Imaging Data

A Thesis Presented to the University of Energy and Natural Resources in Partial Fulfillment of the Requirements for the Bachelor's Degree in Computer Science January 2020.

Authors Declaration

We understand that copyright in our thesis is transferred to the University of Energy and Natural Resources.

Botwe Asamoah Emmanuel UE20001016
Adu Bruce Emmanuel UE20000316
Joseph Adomako Junior UE20000216

Project Supervisor:

[Dr. Dr. Benjamin Asubam. Weyori [DATE]

Abstract

Contributing to the success of deep learning is the availability of massive amounts of training data. Building and annotating large datasets for solving medical image classification problems is today a bottleneck for many applications. Recently, capsule networks were proposed to deal with shortcomings of Convolutional Neural Networks (ConvNets) models. In this work, we provide an important benchmark on the behavior of the new state of the art capsule networks against Convolutional Neural Networks under datasets constraints of medical image analysis, like , small amounts of labelled data and class-imbalance. We evaluate our experiments on Breast Histopathology Images and brain MRI Images for Brain Tumor Detection, publicly available datasets. Our results suggest that capsule networks can be trained with less amount of data for the same or better performance and are more robust to an imbalanced class distribution, which makes our approach very promising for the medical imaging community.

Keywords: capsule networks, small datasets, class imbalance, Breast cancer, Classification, Convolutional Neural Networks

Table of Content

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 1 Introduction

Background

Computer vision is a rising field with still more to be done. The rise of the internet and digital media has helped computers understand how we can train computers to see. And one of the areas we have seen strong applications of in Computer vision is in Medical Imaging.With the rise of **Convolutional Networks,** Deep learning algorithms have been very successful with massive amounts of training data.

Convolutional Neural Network (CNN) is a deep learning architecture which is inspired by the structure of a visual system very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. **SVM/Softmax**) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply. Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the **"output layer"** and in classification settings it represents the **class scores.**

There are still many challenging problems to solve in computer vision. Nevertheless, deep learning methods are achieving state-of-the-art results on some specific problems.
It is not just the performance of deep learning models on benchmark problems that is most interesting; it is the fact that a single model can learn meaning from images and perform vision tasks, obviating the need for a pipeline of specialized and hand-crafted methods. In this paper we focus on solving Object CLassification or Image Recognition which is the task of assigning an input image one label from a fixed set of categories. This is

one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications.

Application Areas of Convolutional Neural Networks

In 1962, Hubel and Wiesel in their classic work on the cat's primary visual cortex found that cells in the visual cortex are sensitive to small sub-regions of the visual field called receptive field. These cells are responsible for detecting light in the receptive fields. Neocognitron proposed by Fukushima (Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, 1980) was the first model which was simulated on a computer and was inspired from the works of Hubel and Wiesel. This network is widely considered as a predecessor of CNN and it was based on the hierarchical organization between neurons for the transformation of image. LeCun et al. established the framework of CNNs by developing a multi-layer artificial neural network called LeNet-5.

LeNet-5 was used to classify handwritten digits and could be trained with the backpropagation algorithm whi;ch made it possible to recognize patterns directly from raw pixels thus eliminating a separate feature extraction mechanism. But even with all these advantages, due to the lack of large training data and computational power at that time, LeNet-5 failed to perform well on complex problems such as video classification. Since the advent of GPGPUs and their use in machine learning (Strigl, Daniel & Kofler, Klaus & Podlipnig, Stefan. (2010)), the field of CNN has gone through a renaissance phase. Several publications have established more efficient ways to train convolutional neural networks using GPU computing. Krizhevsky et al. proposed a deep CNN architecture called AlexNet, which demonstrated significant improvement in image classification tasks. AlexNet is very similar to the classic LeNet-5 albeit a deeper structure. Following the success of AlexNet several publications such as GoogleNet (Christian Szegedy et al, Going Deeper with Convolutions, 2015).

Simple applications of CNNs which we can see in everyday life are obvious choices, like facial recognition software, image classification, speech recognition programs, etc. These are terms which we, as laymen, are familiar with, and comprise a major part of our everyday life, especially with image-savvy social media networks like Instagram. Some of the key applications of CNN are

1. **Computer Vision**

Computer vision is the field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity.

Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labeling objects.
One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better.

2. **Natural Language Processing**

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language.
The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.Most NLP techniques rely on machine learning to derive meaning from human languages. Where NLP outperforms humans is in the amount of language and data it's able to process. Therefore, its potential uses go beyond the examples above and make possible tasks that would've otherwise taken employees months or years to accomplish.

Problem Statement

Medical images nowadays play an essential role in detection and diagnosis of numerous diseases. Ranging from anatomical information, functional activities, to the molecular and cellular expressions, medical imaging provides direct visualization means to see through the human bodies and observe the minute anatomical changes and biological processes characterized by different physical and biological parameters. Informative as they are, medical imaging usually requires experienced medical doctors to best interpret the information revealed in the images. However, because of various subjective factors as well as limited analysis time and tools, it is quite common that different medical doctors may come up with diverse interpretations, leading to different diagnoses. Moreover, for the same set of medical imaging, a medical doctor may make different diagnosis results at different times.

To attain a more reliable and accurate diagnosis, recently, varieties of computer-aided detection **(CAD)** and diagnosis (CADx) approaches have been developed to assist interpretation of the medical images. At least four types, denoted as Types I–IV, of efforts may be identified among these CAD and CADx approaches. Type I is to assist visual detection, qualitative analysis, and interactive quantitative analysis of the objects of interest in the medical images by either enhancing the salient features of the objects or suppressing the background noises. Type II is to assist feature extraction of the objects of interest for further quantitative analyses by such techniques as boundary delineation, tree-structure reconstruction, fiber tracking, texture analysis, and so on. Type III is to automatically detect and classify the objects of interest by integrating the data mining, medical image analysis, and signal processing technologies. Type IV is to estimate the anatomical and functional tissue properties not explicitly revealed in the medical images based on mathematical modeling, for example, physiology, biomechanics, heat transfer, and so forth.

Type III the problem of identifying regions in feature space where normal and abnormal candidates are located is solved by training a classifier . This requires a training set consisting of correctly classified candidates. Such a training set is usually created by having a human provide a reference standard with correct classifications, for example by indicating lesions on chest radiographs with prior knowledge of their location on a computed tomography (CT) scan obtained in the same patient.

State of the art Convolutional Neural Networks achieve the best results in image classification, but are rooted in problems - they are actually pretty bad in detecting objects in different poses.

This is because CNNs are not equivariant but invariant to pose changes. Equivariance means that when the input changes, the output changes the same way. Invariance in contrast means that when the input changes, the output doesn't change. Because CNNs are made to be invariant, this makes them robust against translation changes. But other pose changes (like rotation, orientation) are not handled well.

To achieve good performance a lot of training data is needed, and techniques like data augmentation are used. But the imbalance in medical data will always cause a problem.
Hinton et al. proposed a new architecture, **Capsule Networks (CapsNet)** which he described as a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part. In his paper he described capsules as performing better under constraints and limited datasets and being able to achieve better Accuracy is the most important measure in medical imaging.

In our experiments on Breast Histopathology Images, publicly available datasets andBrain MRI Images for Brain Tumor Detection are also publicly available. We test the performance of capsule networks trained with less amount of data for the same against state of the art CNN models fine tuned to fit out datasets and make a comparative study if capsules better fit as the architecture for CAD systems.

General Objectives of the Study

General Objectives

The study of computer vision scopes around detection and classification. The general objectives of the study is to investigate the problems rooted in CNN and how these problems can be tackled with capsule networks in medical imaging.

Specific Objectives

The specific objectives of the study is

1. To investigate the uses and applications of Convolutional neural network
2. To investigate the general performance of cnn
3. To investigate the performance and application areas suitable for implementation of such networks
4. To investigate the performance of CNN on Medical Images
5. To investigate the performance of Capsule Networks on Medical Images
6. To investigate how capsules can better produce results in medical images.

Scope of Study

The wider application areas of computer vision in medical imaging makes it harder to generalize the behaviour or performance of one algorithm with respect to another. With this project to reduce the level of complexity in using various datasets, Breast cancer dataset & Brain MRI Images for Brain Tumor Detection was chosen. Owing to the fact that there are also different methods in diagnosing breast cancer like the scintigraphy, mammography, we choose histology images relating to Invasive Ductal Carcinoma (IDC) which is the most common form of cancer in women, and invasive ductal carcinoma (IDC) is the most common form of breast cancer. Accurately identifying and categorizing breast cancer subtypes is an important clinical task, and automated methods can be used to save time and reduce error.

Invasive Ductal Carcinoma (IDC) is the most common subtype of all breast cancers. To assign an aggressiveness grade to a whole mount sample, pathologists typically focus on the regions which contain the IDC. As a result, one of the common pre-processing steps for automatic aggressiveness grading is to delineate the exact regions of IDC inside of a whole mount slide.

Brain tumor segmentation is one of the most important and difficult tasks in many medical-image applications because it usually involves a huge amount of data. Artifacts due to the patient's motion, limited acquisition time, and soft tissue boundaries are usually not well defined. There are large classes of tumor types which have a variety of shapes and sizes. They may appear indifferent sizes and types with different image intensities. Some of them may also affect the surrounding structures that change the image intensities around the tumor.

So The main purpose of this project was to build a CNN & a capsule model that would classify if a subject has a tumor or not based on MRI scan. Accuracy as a metric to justify the model performance.

Organisation of the Project

The project report is organized in five chapters. Chapter one is on the Introduction of the project, which gives a brief description of the project, the problem statement, the objectives of the project and the scope of the whole project. Chapter two is on Literature Review. Here, work related to the project is reviewed. Chapter three is on Project Methodology. The methodology used in developing the project is briefly explained here and how data was collected is clearly explained here. Chapter four constitutes the Design and Development, which describes the implementation of the project and how the various modules of the system were tested and evaluated and Chapter Five is on recommendations and Conclusion of the project. Recommendations regarding the project are described here, and a summary of the whole project discussed.

Chapter 2 Literature Review

Introduction to ANN

It is possible to introduce neural networks without appealing to brain analogies. In linear classification scores are computed for different visual categories given the image using the formula $s = Wx$ where $W$was a matrix and$x$ was an input column vector containing all pixel data of the image. In the case of CIFAR-10, **x** is a **[3072x1**] column vector, and $W$is a **[10x3072]** matrix, so that the output scores is a vector of **10** class scores.

An example neural network would instead compute $s = W_2\, max(0,\ W_1 x)$ **.** Here, W_1could be, for example, a [100x3072] matrix transforming the image into a 100-dimensional intermediate vector. The function $max(0,\ -)$ is a non-linearity that is applied elementwise. There are several choices we could make for the non-linearity (which we'll study below), but this one is a common choice and simply thresholds all activations that are below zero to zero. Finally, the matrix W_2 would then be of size **[10x100]**, so that we again get 10 numbers out that we interpret as the class scores. Notice that the non-linearity is critical computationally - if we left it out, the two matrices could be collapsed to a single matrix, and therefore the predicted class scores would again be a linear function of the input. The non-linearity is where we get the wiggle. The parameters W_2, W_1 are learned with **stochastic gradient descent**, and their gradients are derived with chain rule (and computed with **backpropagation**).

A three-layer neural network could analogously look like $s = W_3\, max(\,0,\ W_2\, max(0,\ W_1 x))$, where all of $W_1 W_2\ W_3$ are parameters to be learned. The sizes of the intermediate hidden vectors are **hyperparameters** of the network.

The area of Neural Networks has originally been primarily inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in **Machine Learning** tasks. A very brief and high-level description of the biological system that a large portion of this area has been inspired by can help make it better to understand.

Biological motivation and connections

The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately $10^{14} - 10^{15}$ synapses. The diagram below shows a cartoon drawing of a biological neuron (left) and a common mathematical model (right). Each neuron receives input signals from its dendrites and produces output signals along its **(single) axon**. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. x_0) interact multiplicatively (e.g.$w_0 x_0$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an **activation function** f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function σ, since it takes a real-valued input (the signal strength after the sum) and **squashes** it to range between 0 and 1..

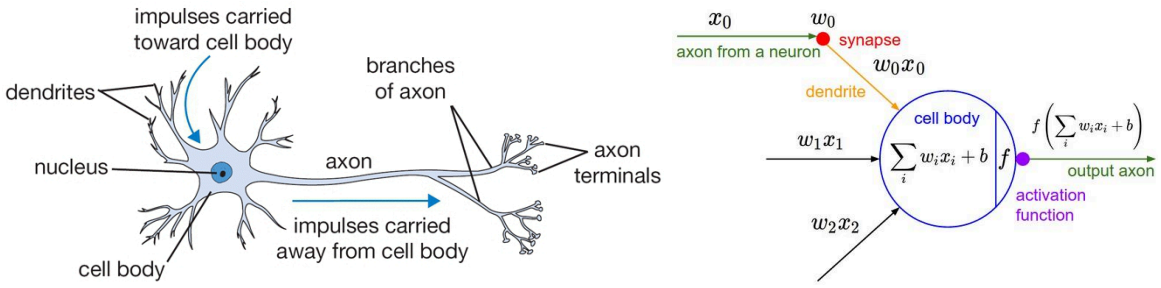


Fig 2.0.1 A cartoon drawing of a biological neuron (left) and its mathematical model (right)

In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function), in this case the sigmoid $\sigma(x) = 1/(1 + e - x)$. It's important to stress that this model of a biological neuron

is very coarse: For example, there are many different types of neurons, each with different properties. The dendrites in biological neurons perform complex nonlinear computations. The synapses are not just a single weight, they're a complex non-linear dynamical system. The exact timing of the output spikes in many systems is known to be important, suggesting that the rate code approximation may not hold.

Single neuron as a linear classifier

The mathematical form of the model Neuron's forward computation might look familiar to you. As we saw with linear classifiers, a neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space. Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

Binary Softmax classifier. For example, we can interpret $\sigma(\sum_i w_i x_i + b)$to be the probability of one of the classes $P(y_i = 1 \mid x_i; w)$. The probability of the other class would be $P(y_i = 0 \mid x_i; w) = 1 P(y_i = 1 \mid x_i; w)$, since they must sum to one. With this interpretation, we can formulate the cross-entropy loss, and optimizing it would lead to a binary Softmax classifier (also known as **logistic regression**). Since the sigmoid function is restricted to be between **0-1**, the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

Commonly used activation functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may

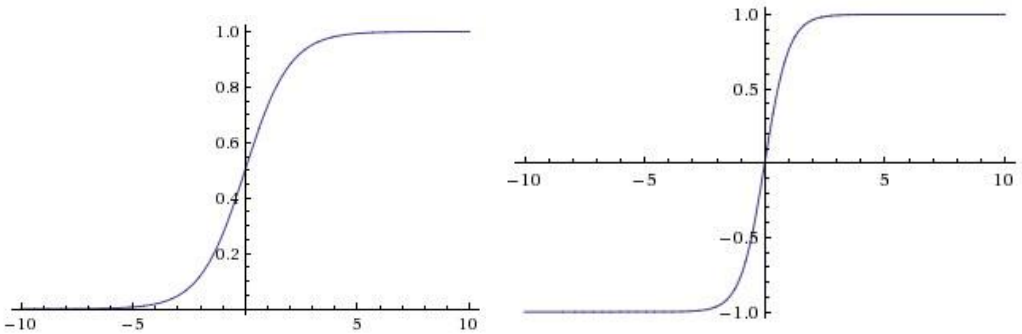


Fig 2.0.2 Left: Sigmoid nonlinearity squashes real numbers to range between [0,1] **Right:** The tanh nonlinearity squashes real numbers to range between [-1,1].

The sigmoid nonlinearity has the mathematical form σ(x)=1/(1+e−x) and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and "squashes" it into a range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid nonlinearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

1. Sigmoids saturate and kill gradients. A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero.
2. Sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. $x > 0$ elementwise in $f = wTx + b$), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f).

The tanh nonlinearity is shown on the image above on the right. It squashes a real-valued number to the range [-1, 1]. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh nonlinearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron, in particular the following holds: $tanh(x) = 2\sigma(2x) - 1$

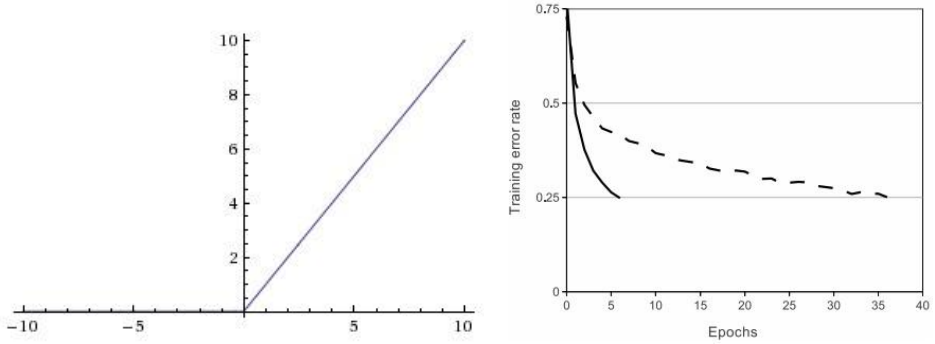


Fig 2.0.3 Left: Rectified Linear Unit (ReLU) activation function, which is zero when x < 0 and then linear with slope 1 when x > 0. **Right:** A plot from Krizhevsky et al. [10] paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.

ReLU. The Rectified Linear Unit has become very popular in the last few years. It computes the function f(x)=max(0,x). In other words, the activation is simply thresholded at zero (see image above on the left). There are several pros and cons to using the ReLUs.

- greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) [10] the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

Neural Networks are modeled as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. Cycles are not allowed since that would imply an infinite loop in the forward pass of a network. Instead of amorphous blobs of connected neurons, Neural Network models are often organized into distinct layers of neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections. Below are two example Neural Network topologies that use a stack of fully-connected layers:

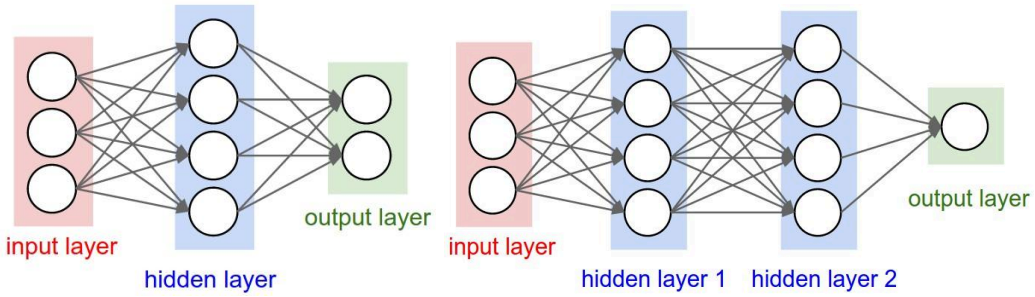


Fig 2.0.4 N Layer neural network

when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In that sense, you can sometimes hear people say that logistic regression or SVMs are simply a special case of single-layer Neural Networks. You may also hear these networks interchangeably referred to as ***"Artificial Neural Networks"*** **(ANN)** or ***"Multi-Layer Perceptrons"*** **(MLP)**. Many people do not like the analogies between Neural Networks and real brains and prefer to refer to neurons as *units*.

Review of CNN Methods

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problems. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It's like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time.

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a

non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

A convolution is a weighted sum of the pixel values of the image, as the window slides across the whole image. Turns out, this convolution process throughout an image with a weight matrix produces another image (of the same size, depending on the convention). Convolving is the process of applying a convolution.

The sliding-window shenanigans happen in the convolution layer of the neural network. A typical CNN has multiple convolution layers. Each convolutional layer typically generates many alternate convolutions, so the weight matrix is a tensor of 5 × 5 × n, where n is the number of convolutions.
The beauty of the CNN is that the number of parameters is independent of the size of the original image. You can run the same CNN on a 300 × 300 image, and the number of parameters won't change in the convolution layer.

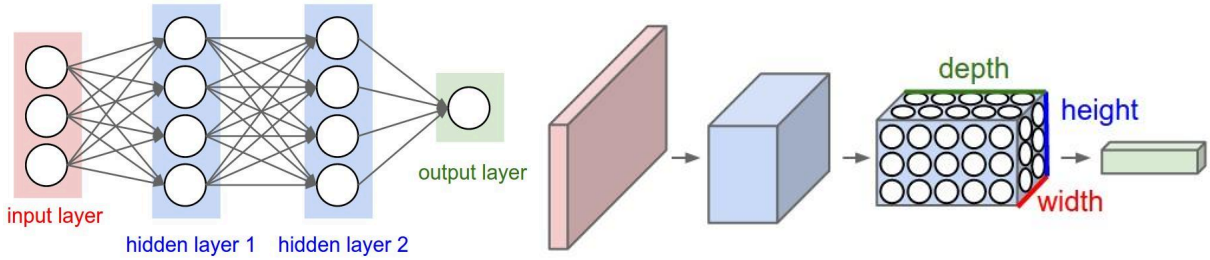


Fig 2.0.5 **Left**: A regular 3-layer Neural Network. **Right**: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer (exactly as seen in regular Neural Networks)**. We will stack these layers to form a full ConvNet architecture.

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- The CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- The RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in a volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Shortcomings of the CNN Method

To date, convolutional neural networks (CNN) have been used for computer vision tasks. Although CNNs have managed to achieve far greater accuracy, they still have some shortcomings.

CNNs were built at first to classify images; they do so by using successive layers of convolutions and pooling. The pooling layer in a convolutional block is used to reduce the data dimension and to achieve something called spatial invariance, which means regardless of where the object is placed in the image, it identifies the object and classifies it. While this is a powerful concept it has some drawbacks. One of them is that while performing pooling it tends to lose a lot of information, information particularly useful while performing tasks such as image segmentation and object detection.

When the pooling layer loses the required spatial information about the rotation, location, scale, and different positional attributes of the object, the process of object detection and segmentation becomes very difficult. While modern CNN architecture has managed to

reconstruct the positional information using various advanced techniques, they are not 100 percent accurate, and reconstruction itself is a tedious process. Another drawback of the pooling layer is that if the position of the object is slightly changed the activation doesn't seem to change with its proportion, which leads to good accuracy in terms of image classification but poor in performance, if you want to locate exactly where the object is in the image.

This is because CNNs are not equivariant but invariant to pose changes. Equivariance means that when the input changes, the output changes the same way. Invariance in contrast means that when the input changes, the output doesn't change. Because CNNs are made to be invariant, this makes them robust against translation changes. But other pose changes (like rotation, orientation) are not handled well.
This is because CNNs use pooling (e.g. max-pooling) to convey information from one layer to the next. CNNs detect certain features in an image, but through using a pooling layer valuable information gets lost.

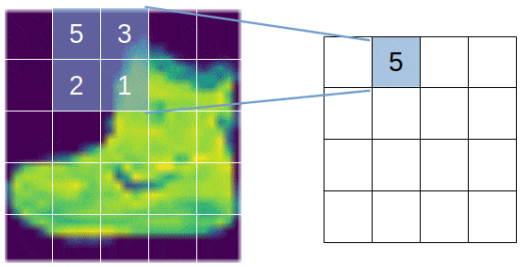


Fig 2.0.6 CNN striding

Let us consider a very simple and non-technical example. Imagine a face. What are the components? We have the face oval, two eyes, a nose and a mouth. For a CNN, a mere presence of these objects can be a very strong indicator to consider that there is a face in the image. Orientational and relative spatial relationships between these components are not very important to a CNN.

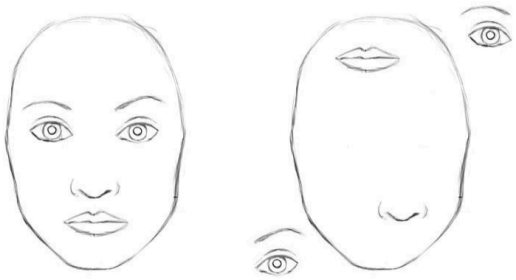


Fig 2.0.7 Simple face

To CNN, both pictures are similar, since they both contain similar elements. The main component of a CNN is a convolutional layer. Its job is to detect important features in the image pixels. Layers that are deeper (closer to the input) will learn to detect simple features such as edges and color gradients, whereas higher layers will combine simple features into more complex features. Finally, dense layers at the top of the network will combine very high level features and produce classification predictions.

Hinton himself stated that the fact that max pooling is working so well is a big mistake and a disaster.
Hinton: “The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.”
Of course, you can do away with max pooling and still get good results with traditional CNNs, but they still do not solve the key problem. Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects.
In the example above, a mere presence of 2 eyes, a mouth and a nose in a picture does not mean there is a face, we also need to know how these objects are oriented relative to each other.

To achieve good performance a lot of training data is needed, and techniques like data augmentation are used.

And this is a key idea in Medical Imaging. Accuracy is the most important measure in medical imaging as

Strategies to Overcome the Shortcomings

Owing to the fact that Convolutional Neural Network has been a pillar algorithm of deep learning, been one of the most influential innovations in the field of computer vision and they have performed a lot better than traditional computer vision algorithms is still deeply rooted in problems as described in the shortcomings section. A few strategies to improve the performance which might not necessarily overcome such problems are :

1. **Tune Parameters**

To improve CNN model performance, we can tune parameters like epochs, learning rate etc.. Number of epochs definitely affects the performance. For a large number of epochs , there is improvement in performance. But need to do certain experimentation for deciding epochs, learning rate. We can see after certain epochs there is not any reduction is training loss and improvement in training accuracy. Accordingly we can decide the number of epochs. Also we can use the dropout layer in the CNN model. As per the application need , decide the proper optimizer during compilation of model. We can use various optimizers e.g SGD,rmsprop etc. There is a need to tune models with various optimizers . All these things affect the performance of CNN.

2. **Data Augmentation**

CNN models are only relevant when you have a huge amount of data". It's not wrong. CNN requires the ability to learn features automatically from the data, which is generally only possible when lots of training data is available. If we have less training data available.. what to do? Solution here is to use Image Augmentation.

Image augmentation parameters that are generally used to increase the data sample count are zoom, shear, rotation, preprocessing function and so on. Usage of these parameters results in generation of images having these attributes during training of Deep Learning models. Image samples generated using image augmentation, in general existing data samples increased by the rate of nearly 3x to 4x times.

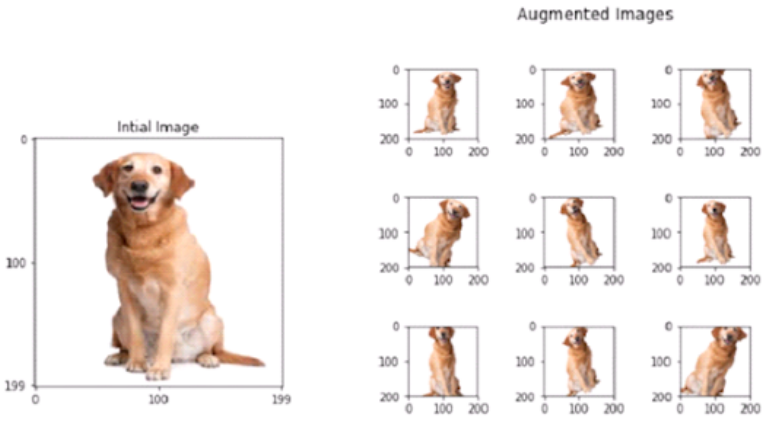


Fig 2.0. 8 Data Augmentation (source: wikipedia)

One more advantage of data augmentation is as we know CNN is not rotation invariant, using augmentation we can add the images in the dataset by considering rotation. Definitely it will increase the accuracy of the system

3. Deeper Network Topology

A wide neural network is possible to train with every possible input value. Hence, these networks are very good at memorization, but not so good at generalization. There are, however, a few difficulties with using an extremely wide, shallow network. Though, wide neural network is able to accept every possible input value, in the practical application we won't have every possible value for training.

Deeper networks capture the natural "hierarchy" that is present everywhere in nature. See a convnet for example, it captures low level features in the first layer, a little better but still low level features in the next layer and at higher layers object parts and simple structures are captured. The advantage of multiple layers is that they can learn features at various levels of abstraction.

So that explains why you might use a deep network rather than a very wide but shallow network. But why not a very deep, very wide network? The answer is we want our network to be as small as possible to produce good results. The wider network will take longer time to train. Deep networks are very computationally expensive to train. Hence, make them wide and deep enough that they work well, but no wider and deeper.

4. **Handel Overfitting and Underfitting problem**

Overfitting refers to a model that models the training data too well. What is the meaning of it? Lets simplify… In the overfitting your model gives very nice accuracy on trained data but very less accuracy on test data. The meaning of this overfitting model is having good memorization ability but less generalization ability. Our model doesn't generalize well from our training data to unseen data.

Underfitting refers to a model which works well on the testing data. It's very dangerous..isn't it? Model is having good accuracy on test data, but less accuracy on training data.

In the technical terms a model that overfits has low bias and high variance. A model that underfits has high bias and less variance. In any modeling, there will always be a tradeoff between bias and variance and when we build models, we try to achieve the best balance.

Now what is bias and variance? Bias is an error with respect to training set. Variance is how much a model changes in response to the training data. The meaning of variance is that the model doesn't give good accuracy on test data.

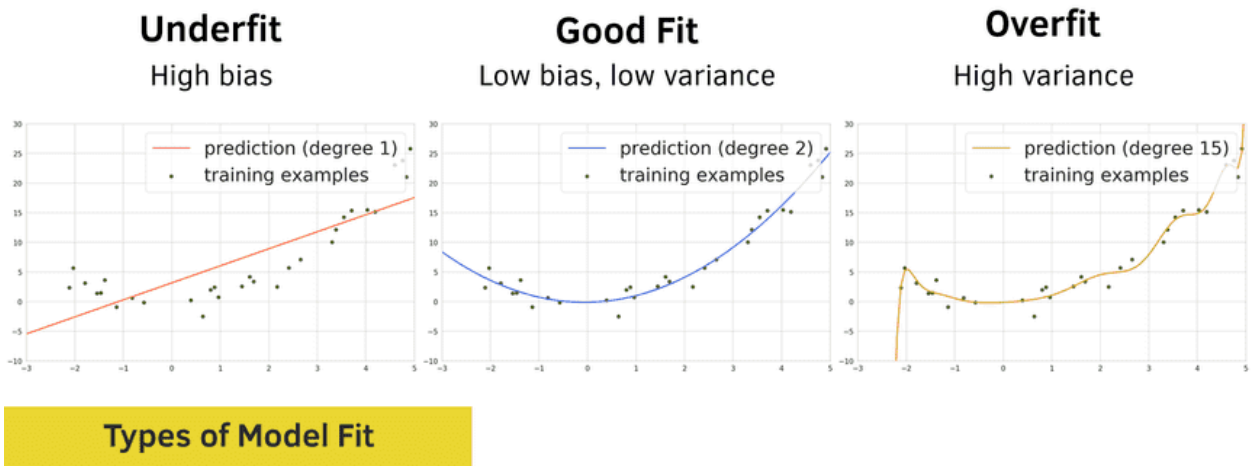


Fig 2.0.9 Underfitting Vs. Overfitting (Source: curiousily.com/)

Underfitting occurs when a model is too simple — informed by too few features or regularized too much — which makes it inflexible in learning from the dataset.

Focusing on the level of deepness of the model can be a solution. You may need to add layers.. as it will give you more detailed features. As we discussed above you need to tune parameters to avoid Underfitting.

Overfitting is a common problem in machine learning. There are certain solutions to avoid overfitting. Training with more data helps to increase accuracy of mode. Large training data may avoid the overfitting problem. In CNN we can use **data augmentation** to increase the size of the training set. System is getting trained with a number of iterations. Model is improved through each new iteration .. But wait.. after a certain number of iterations the model starts to overfit the training data. Hence, the model's generalization ability can be weakened. So do the Early stopping. **Early stopping** refers to stopping the training process before the learner passes that point.

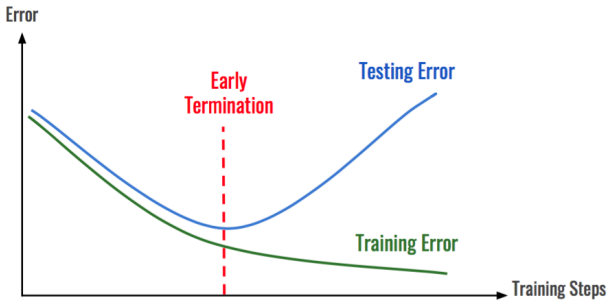


Fig.2.1.0 Early Stopping (Source: Wikipedia)

Cross validation is also a nice technique to avoid overfitting problems. Let's start with k-fold cross validation. (where k is any integer number)Partition the original training data set into k equal subsets. Each subset is called a fold. Let the folds be named as f1, f2, …, fk .

1. · For i = 1 to i = k
2. · Keep the fold fi as Validation set and keep all the remaining k-1 folds in the Cross validation training set.
3. · Train your machine learning model using the cross validation training set and calculate the accuracy of your model by validating the predicted results against the validation set.
4. · Estimate the accuracy of your machine learning model by averaging the accuracies derived in all the k cases of cross validation.

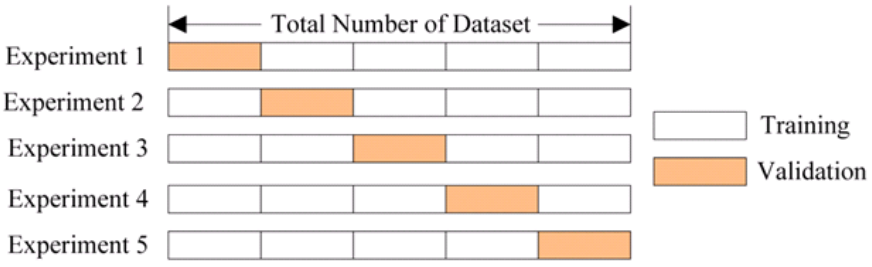


Fig 2.1.1-fold Cross Validation(Source: Wikipedia)

Cross-validation is definitely helpful to reduce overfitting problems.

Chapter 3 Methodology

Mode of Operation of CNN Method

To understand what the CONV layer computes without brain/neuron analogies.
CNN image classifications take an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers see an input image as an array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

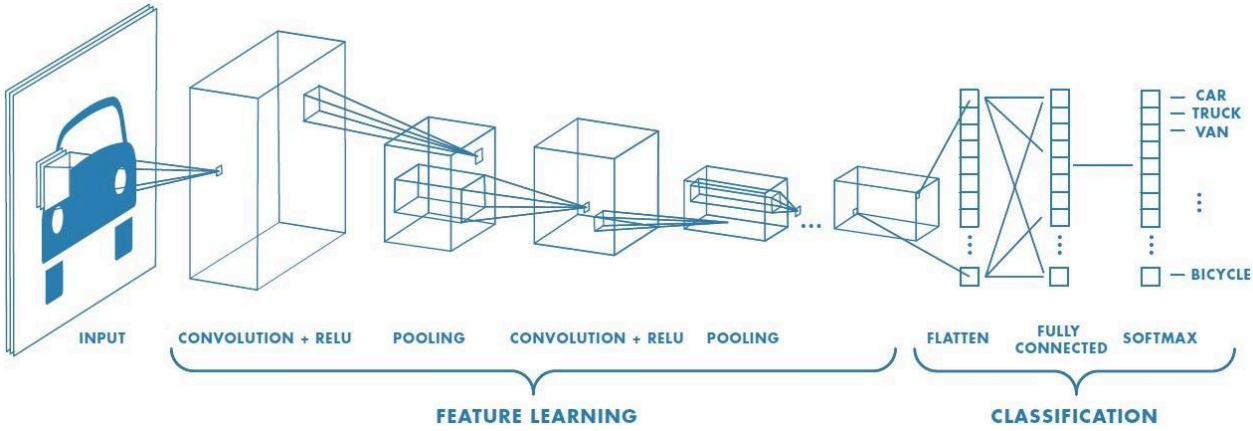


Figure 3.0.0 : Neural network with many convolutional layers

Example Architecture : Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture **[INPUT - CONV - RELU - POOL - FC].** In more detail:

INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
FC (i.e. fully-connected) layer will compute the class scores, resulting in a volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.
In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and others don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

In summary:

1. A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
2. There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
3. Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
4. Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
5. Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

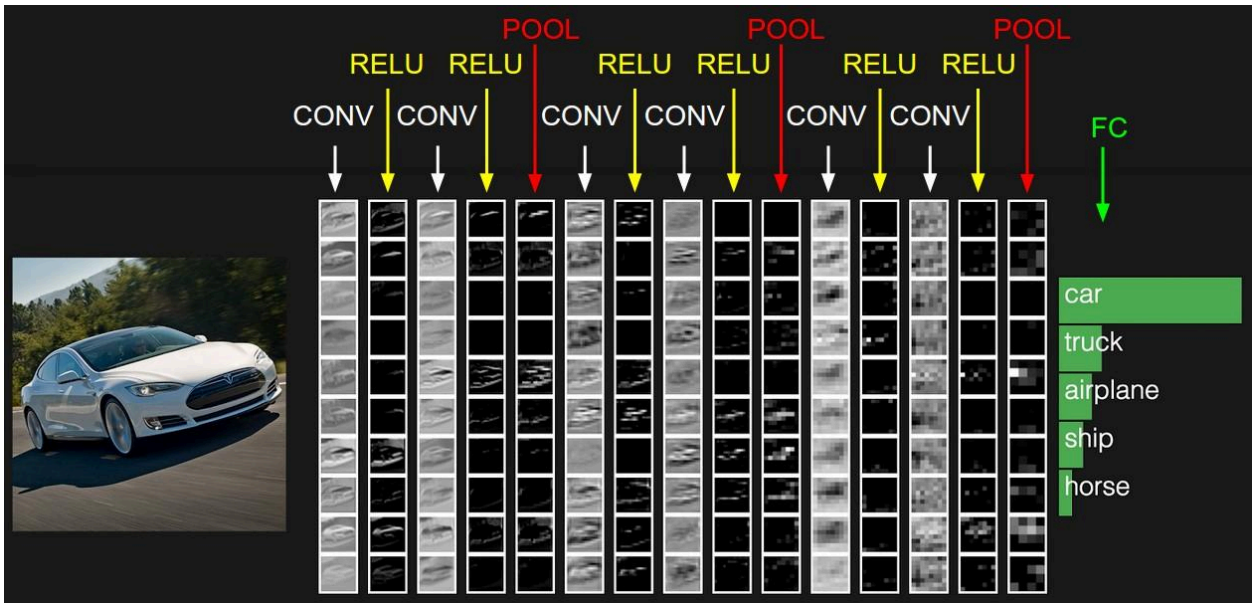


Fig 3.0.1 The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full web-based demo is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every

spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each **CONV layer** (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a **hyperparameter** called the **receptive field of the neuron** (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

Example 1. For example, suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

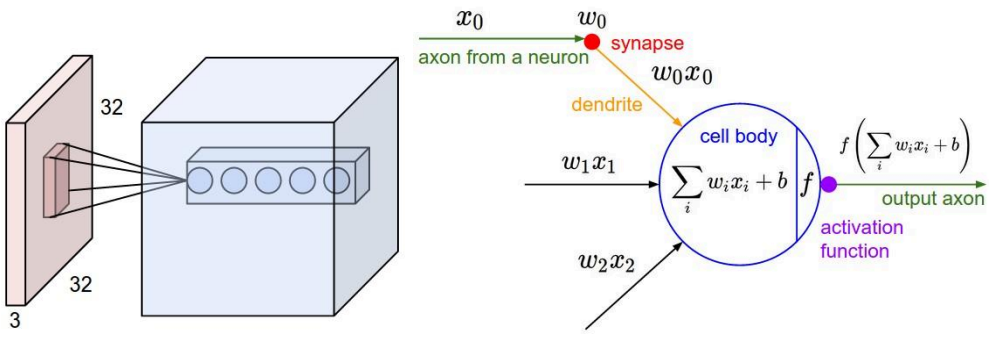


Fig 3.0.2 Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local

region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input. **Right :**The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

---

convince yourself that the correct formula for calculating how many neurons "fit" is given by $(W_1 - F + 2P)/S + 1$. For example for a 7×7 input and a 3×3filter with stride 1 and pad 0 we would get a 5×5output. With stride 2 we would get a 3×3 output.Spatial arrangement. We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the depth, stride and zero-padding.

1. First, the depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre).
2. Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
3. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly as we'll see soon we will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border.
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
- Number of filters K,
- their spatial extent F,
- the stride S,
- the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
- $W_2 = (W_1 + F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
- D2=K

With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling Layer

It is common to periodically insert a **Pooling layer in-between successive Conv layers** in a **ConvNet architecture**. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the **MAX operation**. The most common form is a pooling layer with filters of size 2×2 applied with a stride of **2** downsamples every depth sliced in the input by **2** along both width and height, discarding **75% of the activations.** Every **MAX operation** would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

Accepts a volume of size $W_1 \times H_1 \times D_1$

Requires two hyperparameters: their spatial extent **F**, the stride S,

Produces a volume of size $W_2 \times H_2 \times D_2$where:

$W_2 = (W_1 - F)/S + 1$

$H_2 = (H_1 - F)/S + 1$

$D_2 = D_1$

Introduces zero parameters since it computes a fixed function of the input. For Pooling layers, it is not common to pad the input using zero-padding. It is worth noting that there are only two commonly seen variations of the max pooling layer found in practice: A pooling layer with $F = 3,\ S = 2$ (also called **overlapping pooling**), and more commonly $F = 3,\ S = 2$. Pooling sizes with larger receptive fields are too destructive.

General pooling, in addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. **Average pooling** was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

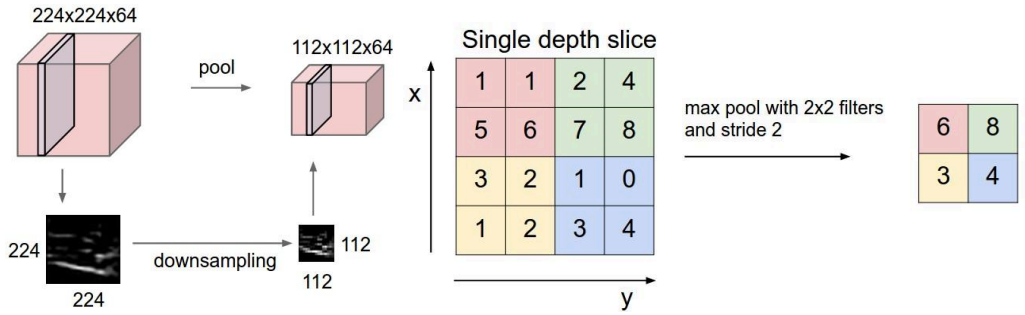


Fig 3.0.4 Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Backpropagation.

Recall from the backpropagation chapter that the backward pass for a $max(x, y)$ operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.

Getting rid of pooling. Many people dislike the pooling operation and think that we can get away without it. For example, Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated **CONV layers**. To reduce the size of the representation they suggest using larger strides in the **CONV layer** once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders **(VAEs)** or generative adversarial networks **(GANs)**. It seems likely that future architectures will feature very few to no pooling layers.

Normalization Layer

Many types of normalization layers have been proposed for use in ConvNet architectures, sometimes with the intentions of implementing inhibition schemes observed in the biological brain. However, these layers have since fallen out of favor because in practice their contribution has been shown to be minimal, if any. For various types of normalizations, see the discussion in Alex Krizhevsky's cuda-convnet library API.

Fully-connected layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. See the Neural Network section of the notes for more information.

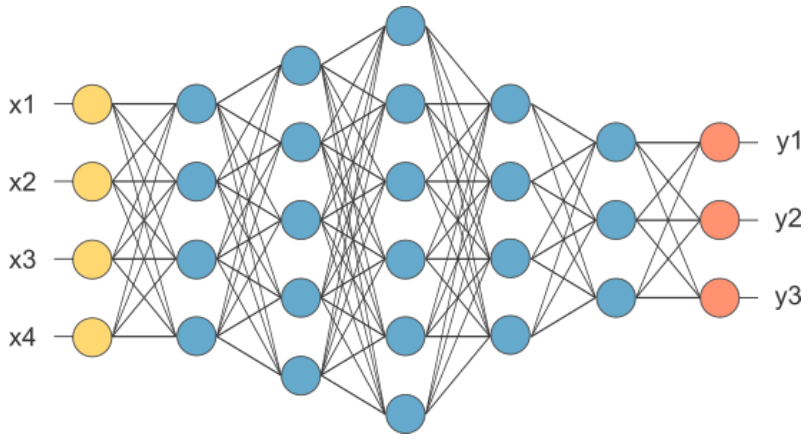


Figure3.0.5 : After pooling layer, flattened as FC layer

In the above diagram, the feature map matrix will be converted as a vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,

Summary

1. Provide input image into convolution layer
2. Choose parameters, apply filters with strides, padding if required. Perform convolution on the image and apply ReLU activation to the matrix.
3. Perform pooling to reduce dimensionality size
4. Add as many convolutional layers until satisfied

5. Flatten the output and feed into a fully connected layer (FC Layer)
6. Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Mode of Operation of Capsule Method

Hinton and Sabour borrowed ideas from neuroscience that suggest the brain is organized into modules called capsules. These capsules are particularly good at handling features of objects like pose (position, size, orientation), deformation, velocity, albedo, hue, texture, etc.

A capsule is a collection or group of neurons that stores different information about the object it is trying to identify in a given image; information mostly about its position, rotation, scale, and so on in a high dimensional vector space (8 dimension or 16 dimension), with each dimension representing something special about the object than can be understood intuitively

In computer graphics there is a concept of rendering, which simply means taking into account various internal representations of an object like its position, rotation, and scale, and converting them to an image on screen. In contrast to this approach our brain works in the opposite way, called inverse graphics. When we look at any object, we internally deconstruct it into different hierarchical sub parts, and we tend to develop a relationship between these internal parts of the whole object. This is how we recognize objects, and because of this our recognition does not depend on a particular view or orientation of the objects. This concept is the building block of capsule networks.

To understand how this works in a capsule network let's look at its architectural design. The architecture of a capsule network is divided into three main parts and each part has sub operations in it. They are:

1. Primary capsules
    a. Convolution
    b. Reshape
    c. Squash
2. Higher layer capsules
    a. Routing by agreement
3. Loss calculation
    a. Margin loss
    b. Reconstruction loss

Primary Capsules

This is the first layer of the capsule network and this is where the process of inverse graphics takes place. Suppose you are feeding the network with the image of a boat or a house, like in the following images. Now, these images are broken down into their sub hierarchical parts in this layer. Let's assume for the sake of simplicity that these images are constructed out of two distinct sub parts; that is, one rectangle and one triangle.

Fig 3.0.6 Capsule representing triangles

In this layer, capsules representing the triangle and rectangle will be constructed. Let us suppose we initialize this layer with 100 capsules, 50 representing the rectangle and 50 representing the triangle. The output of these capsules is represented with the help of arrows in the image below; **the black arrows representing the rectangle's output**, and **the blue arrows representing the triangle's**. These capsules are placed in every location of the

image, and the output of these capsules denotes whether or not that object is located in that position.

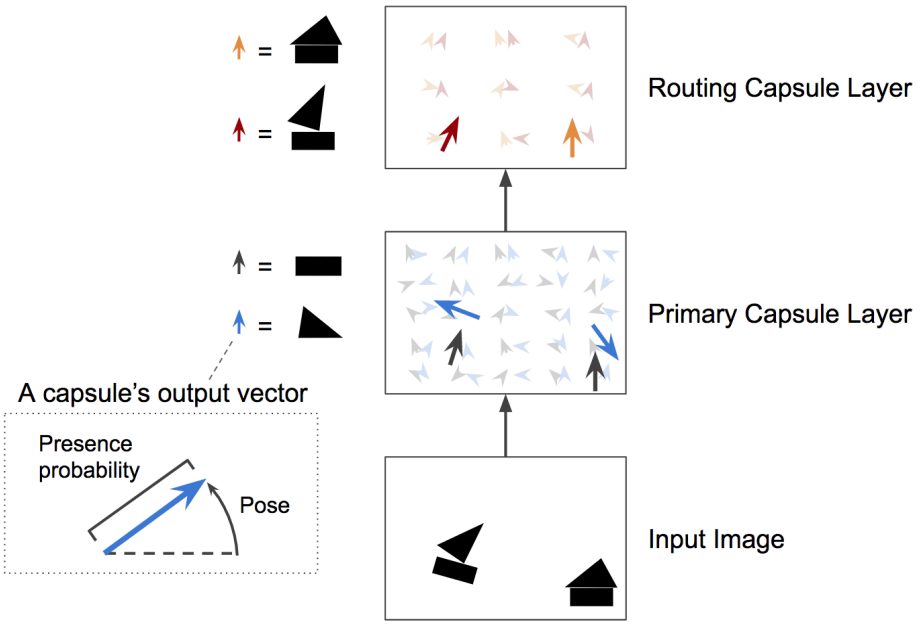


Fig 3.0.7[Angle estimation as performed by capsule networks]

In the picture below you can see that in the location where the object is not placed, the length of the arrow is shorter and where the object is placed, the arrow is longer. The length represents whether the object is present, and the pose of the arrow represents the orientation of that particular object (**position, scale, rotation**, and so on) in the given image.

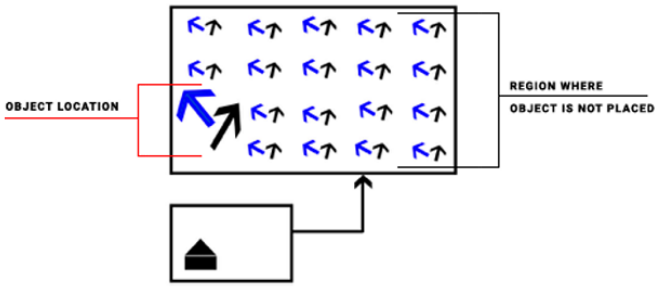


Fig 3.0.8 Capsule representing equivariance

An interesting thing about this representation is that, if we slightly rotate the object in our input image, the arrows representing these objects will also slightly rotate with proportion to its input counterpart. This slight change in input resulting in a slight change

in the corresponding capsule's output is known as **equivariance**. This enables the capsule networks to locate the object in a given image with its precise **location, scale, rotation,** and other positional attributes associated with it.

This is achieved using three distinct processes:

1. Convolution
2. Reshape function
3. Squash function

In this layer the input image is fed into a couple of convolution layers. This outputs some array of feature maps; let's say this outputs an array of 18 feature maps. Now, we apply the Reshaping function to these feature maps and let's say we reshape it into two vectors of nine dimensions each (18 = 2 x 9) for every location in the image, which is similar to the image above representing the rectangle and triangle capsules. Now, the last step is to make sure that the length of each vector is not greater than 1; this is because the length of each vector is the probability of whether or not that object is located in that given location in the image, so it should be between 1 and 0. To achieve this we apply something called the **Squash function**. This function simply makes sure that the length of each vector is between 1 and 0 and will not destroy the positional information located in higher dimensions of the vector.

Now we need to figure out what these sub parts or capsules are related to. That is, if we consider our example of boat and house, we need to figure out which triangle and rectangle is part of the house and which is part of the boat. So far, we know where in the image these rectangles and triangles are located, using these convolutions and squashing functions. Now we need to figure out whether a boat or a house is located there and how these triangles and rectangles are related to the boat and the house.

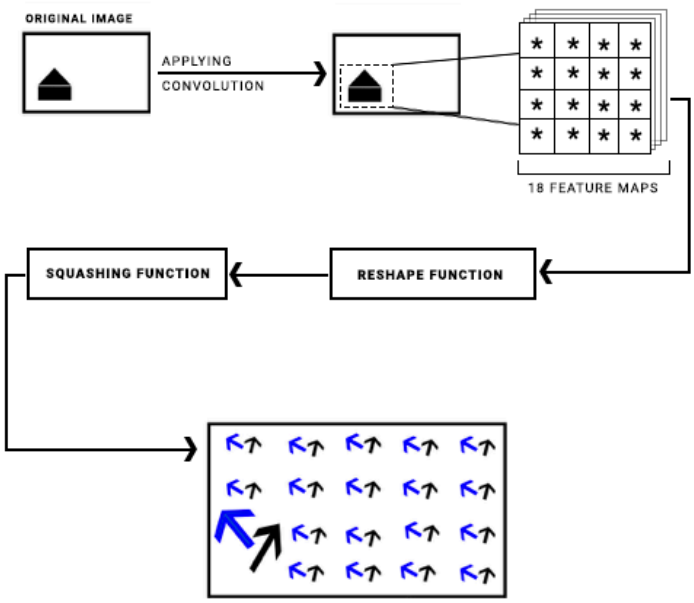


Fig 3.0.9 CapsNet process

Higher Layer Capsules

Before we get into higher layer capsules, there is still one major function left by the primary capsule layer. That is, before the higher layer capsule can operate, right after the Squash function in the primary layer, every capsule in the primary layer will try to predict the output of every capsule in the higher layer in the network; for example, we have 100 capsules, 50 rectangles, and 50 triangles. Now, suppose we have two types of capsules in the higher layer, one for house and another for boat. Depending upon the orientation of both triangle and rectangle capsules, these capsules will make the following predictions on higher layer capsules. This will give rise to the following scenario:

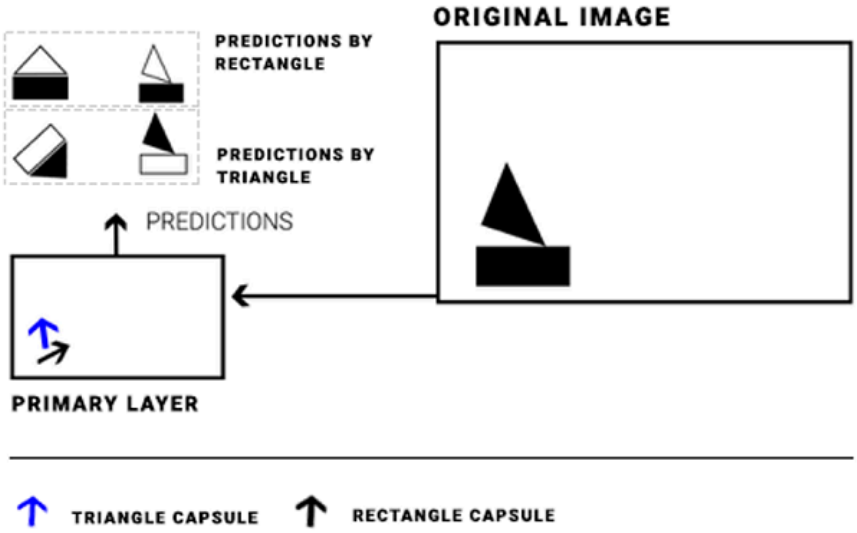


Fig 3.1.0

And as you can see, with respect to its original orientation, the rectangle capsule and the triangle capsule both predicted the boat present in the picture in one of their predictions. They both agree that it's the boat capsule that should be activated in the higher layer capsule. This means that the rectangle and triangle are more a part of a boat than of a house. This also means that the rectangle and triangle capsules think that selecting the boat capsule will explain their own orientation in the primary capsule. In this, both primary layer capsules agree to select the boat capsule in the next layer as a possible object located in the image. This is called routing by agreement.

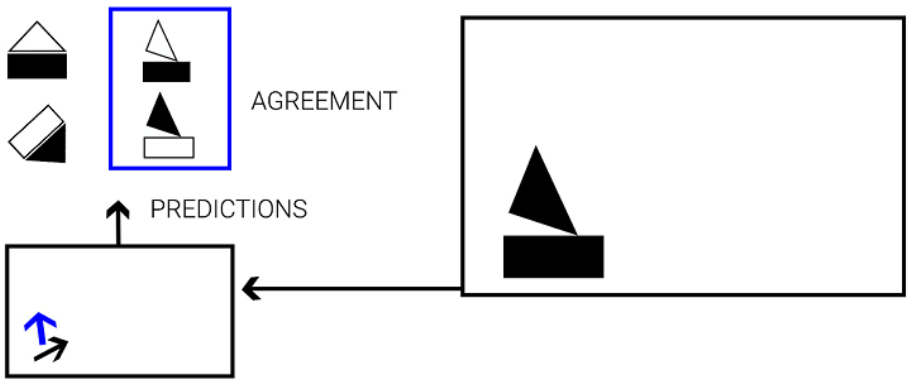


Fig 3.1.0 Routing by agreement

This particular technique has several benefits. Once the primary capsules agree to select a certain higher-level capsule, there is no need to send a signal to another capsule in another higher layer, and the signal in the agreed-on capsule can be made stronger and

cleaner and can help in accurately predicting the pose of the object. Another benefit is that if we trace the path of the activation, from the triangle and rectangle to the boat capsule in a higher layer, we can easily sort out the hierarchy of the parts and understand which part belongs to which object; in this particular case, rectangle and triangle belong to the boat object.

So far we have dealt with the primary layer; now the actual work of the higher capsule layer comes in. Even though the primary layer predicted some output for the higher layer, it still needs to calculate its own output and cross-check which prediction matches with its own calculation.

The first step the higher capsule layer takes to calculate its own output is to set up something called routing weights. We have some predictions given by our primary layer now for each prediction; at first iteration it declares its routing weights to be zero for all. These initial routing weights are fed into a Softmax function and the output is assigned to each prediction.

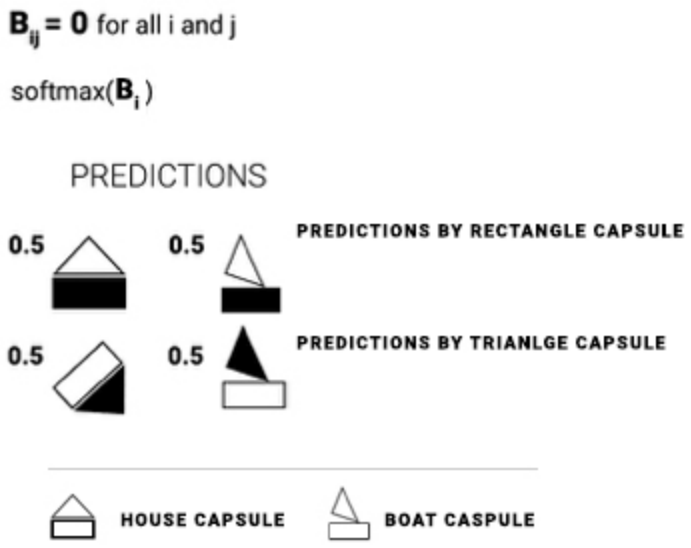
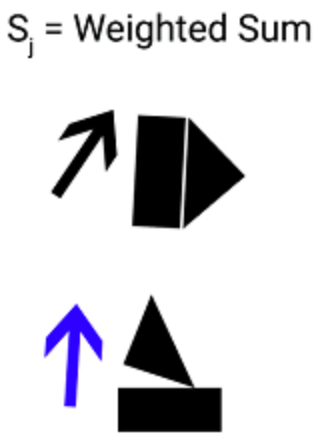


Fig 3.1.1 softmax application

Now, after assigning the Softmax output to the predictions, it calculates the weighted sum to each capsule in this higher layer. This gives us two capsules from a bunch of predictions. This is the actual output of the higher layer for the first round or first iteration.



Now we can find which prediction is the most accurate compared to the actual output of the layer.

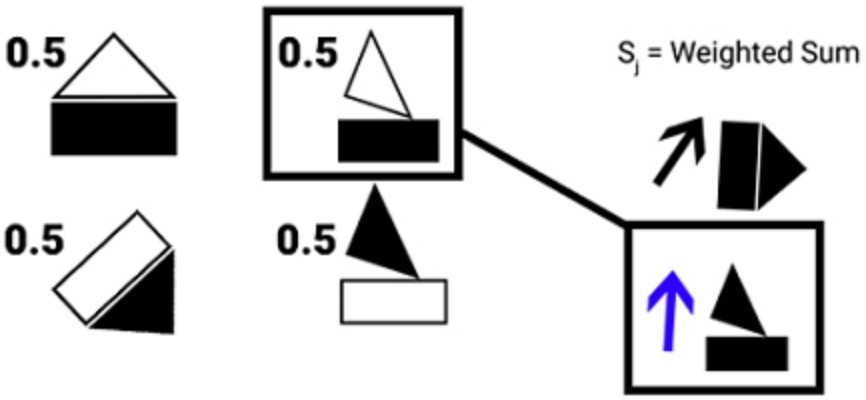


Fig 3.1.1

After selecting the accurate prediction, we again calculate another routing weight for the next round by scalar product of the prediction and the actual output of the layer and adding it to the existing routing weight. Given by equation:

U^ij (Prediction by primary layer)

Vj (Actual output by the higher layer)

Bij += U^ij + Vj

Now, if the prediction and the output match, the new routing weights will be large, and if not, the weight will be low. Again, the routing weights are fed into the Softmax function and the values are assigned to the predictions. You can see that the strong agreed predictions have large weights associated with them, whereas others have low weights.

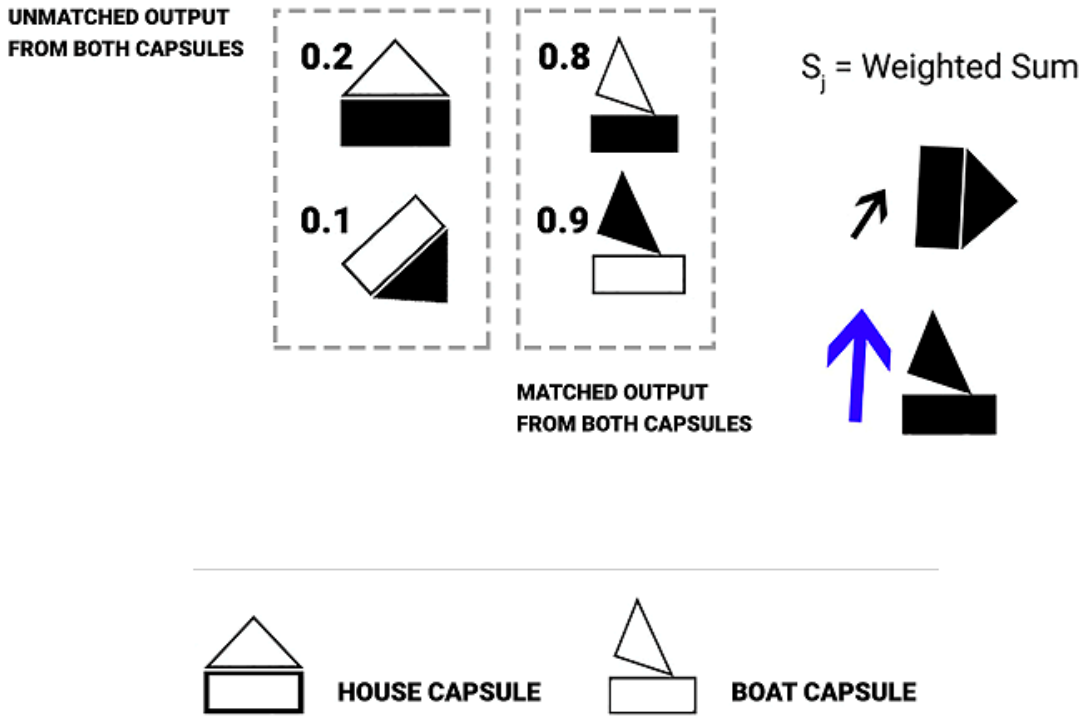


Fig 3.1.1

Again we compute the weighted sum on these predictions with new weights given to it. But now we find that the boat capsule has a longer vector associated with it compared to the house capsule, as the weights were in favor of the boat capsule, so the layer chooses the boat capsule over the house capsule in just two iterations. In this way we can compute the output in this higher layer and choose which capsule to select from this higher layer for the next steps in the capsule network.

I have only described two iterations or rounds for the sake of simplicity, but actually it can take longer, depending upon the task you are performing.

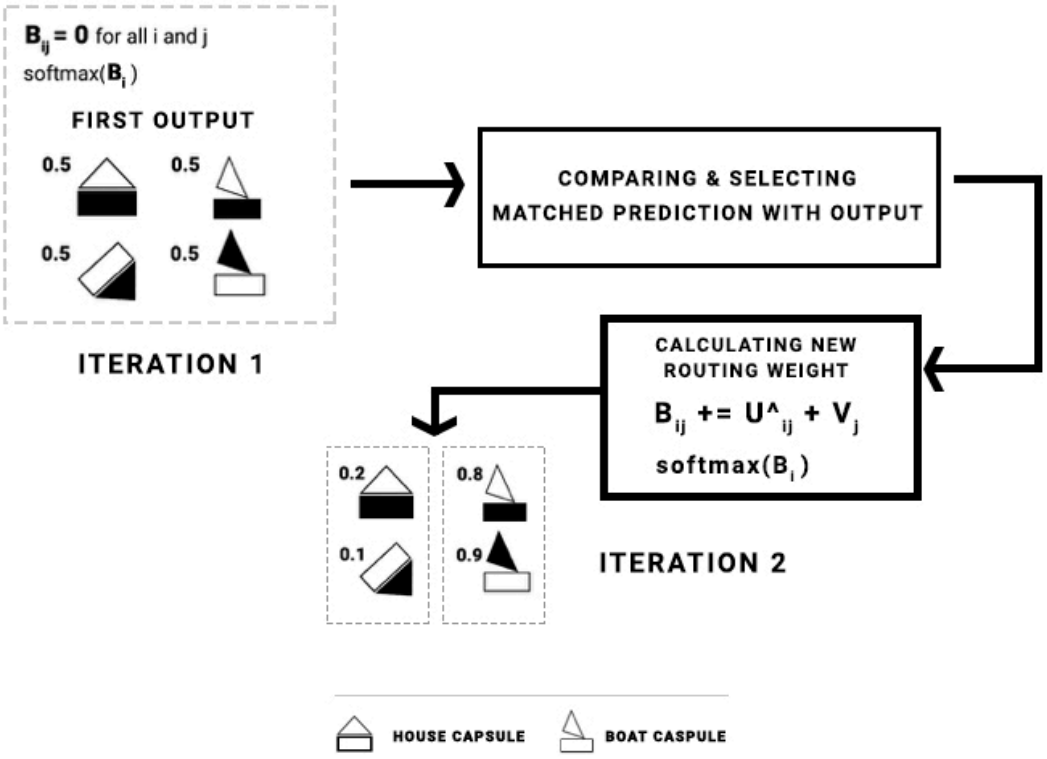


Fig 3.1.2 Capsule by Iteration

Loss Calculation

Now that we have made a decision on what object is in the image using the routing by agreement method, you can perform classification. As with our previous higher layer, one capsule per class, that is, one capsule for boat and one for house, we can easily add a layer on top of this higher layer and compute the length of the activation vector and, depending upon the length, we can assign a class probability to make an image classifier.

In the original paper, margin loss was used to calculate the class probability of multiple classes to create such an image classifier. Margin loss simply means that if a certain object of a class is present in the image then the squared length of the corresponding vector of that object's capsule must not be less than 0.9. Similarly, if that object of that class is not present in the image, then the squared length of the corresponding vector of that object should not be more than 0.1.

Suppose Vk is the length of the output vector of that class K object. Now, if that object of class K is present then its squared value should not be less than 0.9; that is, |Vk|2 >=0.9. Similarly, if the class K object is not present then |Vk|2 <=0.1.

In addition to margin loss, there is an additional unit called the decoder network connected to the higher capsule layer. This decoder network is three fully connected layers, two of them being rectified linear unit (ReLU) activated units, and the last one the sigmoid activated layer, which is used to reconstruct the input image.

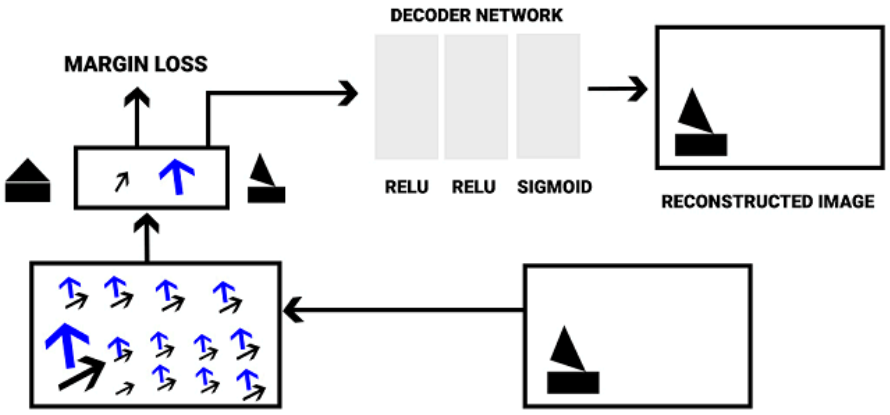


Fig 3.1.3 decoder network

This decoder network learns to reconstruct the input image by minimizing the squared difference between the reconstructed image and the input image:

Reconstruction Loss = (Reconstructed Image – Input Image)2

Now, we have total loss as:

Total Loss = Margin Loss + alpha * Reconstruction Loss

Here, the value of alpha (a constant to minimize reconstruction loss) in the paper1 it is 0.0005 (no extra information is given on why this particular value was chosen). Here the reconstruction loss is scaled down considerably so as to give more importance to the margin loss and so it can dominate the training process. The importance of the reconstruction unit and the reconstruction loss is that it forces the network to preserve

the information required to reconstruct the image up to the highest capsule layer. This also acts as a regularizer to avoid over-fitting during the training process.

In the paper1, the capsule network is used to classify between MNIST* digits. As you can see below (in Figure 1), the paper showed different units of CapsNet for MNIST classification. Here, the input after feeding through two convolutional layers is reshaped and squashed to form 32 primary capsules with 6 x 6 x 8 capsules each. These primary capsules are fed into higher layer capsules, a total of 10 capsules with 16 dimensions each, and at last margin loss is calculated on these higher layer capsules to give class probability.

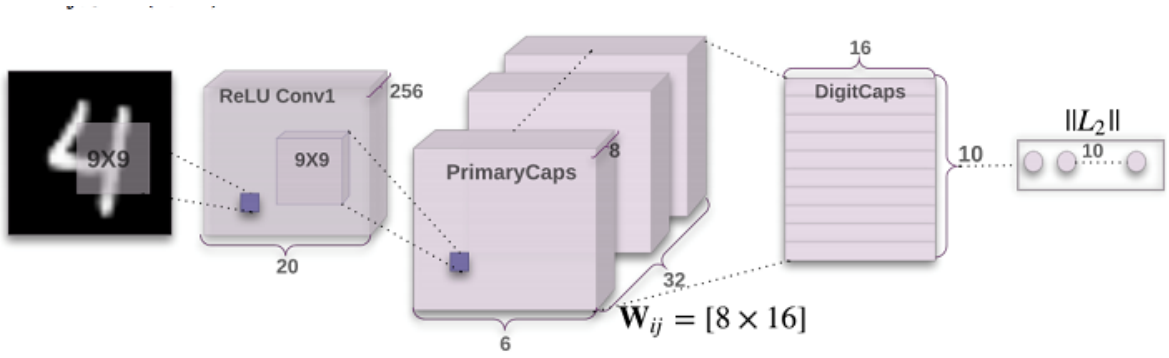


Fig 3.1.5. Dynamic routing between capsules

shows the decoder network used to calculate reconstruction loss. A higher layer capsule is connected to three fully connected layers with the last layer being a sigmoid activated layer, which will output 784-pixel intensity values (28 x 28 reconstructed image).

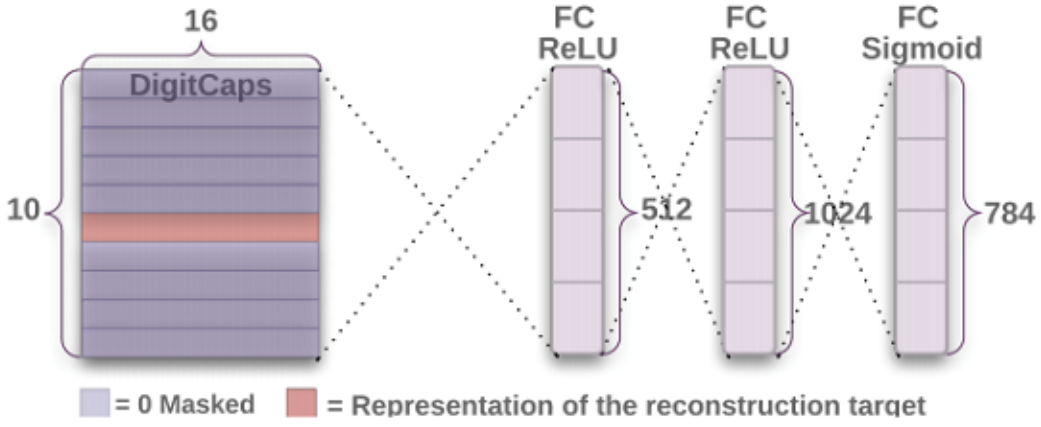


Figure 3.1.6 representing digitcaps

An interesting thing about this higher capsule layer is that each dimension on this layer is interpretable. That is, if we take the example from the paper on the MNIST dataset, each dimension from the 16-dimension activation vector can be interpreted and signifies certain characteristics of the object. If we modify one of the 16 dimensions we can play with the scale and thickness of the input; similarly, another can represent stroke thickness, another width and translation, and so on.

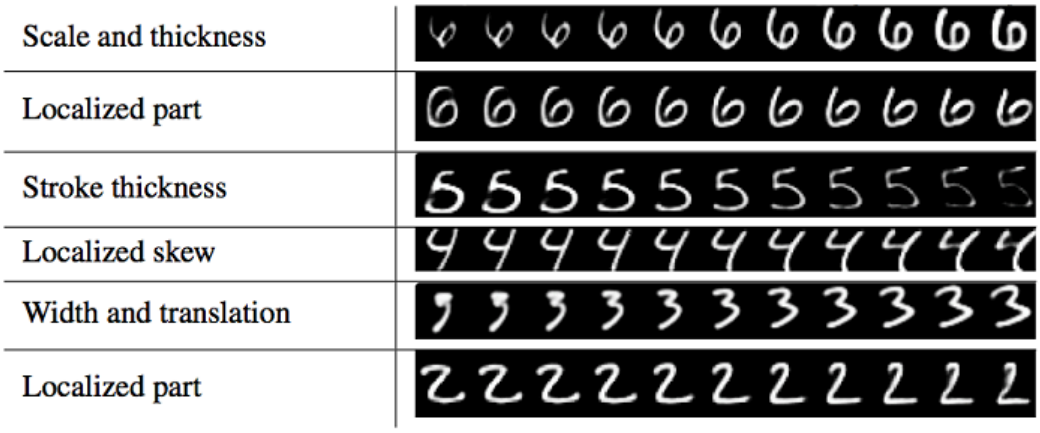


Fig 3.1.8 output of capsnet

Strength and Weakness of the Methods

Capsule networks (CapsNet) are the new architecture in neural networks, an advanced approach to previous neural network designs, particularly for computer vision tasks.

Strengths of Capsule Networks

- Capsule networks have the highest success in the MNIST dataset when compared to other state-of-the-art techniques.
- It's successful with smaller datasets. (By forcing the model to learn the feature variant in a capsule, it can extrapolate possible variants more effectively with less training data.)
- The routing-by-agreement algorithm allows us to distinguish objects in overlapping images.
- It's easier to interpret the image with activation vectors.
- Capsule networks maintain information such as equivariance, hue, pose, albedo, texture, deformation, speed, and location of the object.
- Automatically calculates hierarchy of parts in a given object

Weakness of Capsule Networks

- CIFAR10 does not succeed, in other words results are not state of the art in a difficult dataset as compared to state of the art models.
- Have not been tested on very large datasets like ImageNet.
- Due to the routing-by-agreement algorithm (due to inner loop), more time is required for training the model.
- The application of the capsule network model with different routing algorithms shows that it's a subject that needs more experimentation and is still developing.
- Problem of crowding—not being able to distinguish between two identical objects of the same type placed close to one another.

Chapter 4 Implementation & Testing

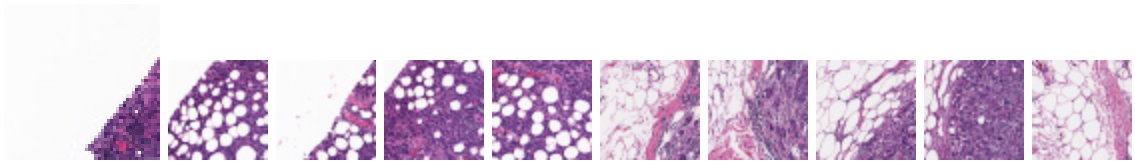
Dataset Analysis

Breast Cancer - IDC

Invasive Ductal Carcinoma (IDC) is the most common subtype of all breast cancers. To assign an aggressiveness grade to a whole mount sample, pathologists typically focus on the regions which contain the IDC. As a result, one of the common pre-processing steps for automatic aggressiveness grading is to delineate the exact regions of IDC inside of a whole mount slide.

The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive). Each patch's file name is of the format: uxXyYclassC.png — > example 10253idx5x1351y1101class0.png . Where u is the patient ID (10253idx5), X is the x-coordinate of where this patch was cropped from, Y is the y-coordinate of where this patch was cropped from, and C indicates the class where 0 is non-IDC and 1 is IDC.

IDC Patches with



Non-IDC Patches with

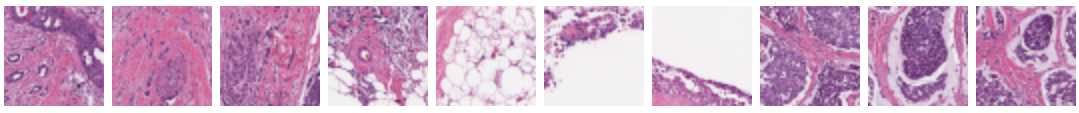


Fig 4.0.1 IDC and Non IDC Patches

Brain MRI Images for Brain Tumor Detection

The image data that was used for this problem is Brain MRI Images for Brain Tumor Detection. It consists of MRI scans of two classes:

NO - no tumor, encoded as 0
YES - tumor, encoded as 1

Unfortunately, the data set description doesn't hold any information about where this MRI scans come from and so on.

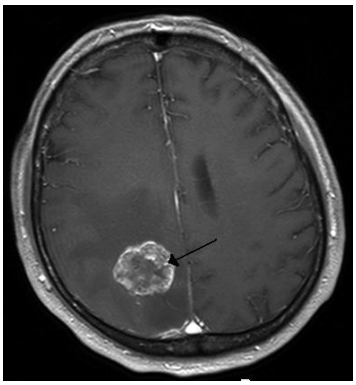


Fig 4.0.2 Brain metastasis in the right cerebral hemisphere from lung cancer, shown on magnetic resonance imaging.

Data Preparation

To build our convolutional networks, we chose Keras backend which works with python for the implementation. Keras is extremely easy to use extremely easy to use extremely easy to use and follows a pythonic style of coding. It is modular, meaning, we can easily add and remove modularity building blocks of Neural Network code, while easily changing cost functions, optimizers, initialization schemes, activation functions, regularization schemes. It also allows us to build powerful Neural Networks quickly and efficiently quickly and efficiently quickly and efficiently. TensorFlow is not as user friendly and modular as Keras

In the implementation process, we will go through the following steps in getting out the classifier.

- Loading Our Data & Getting our data in Shape
- Visualizing the data and distribution
- Building & Compiling Our Model (Specific with capsule and cnn)
- Training Our Classifier
- Plotting Loss and Accuracy Charts
- Saving and Loading Your Model

Practical Implementation of CNN and Capsule Network on breast cancer

Loading the data is a simple but obviously integral first step in creating a deep learning model. Fortunately, Keras has some built-in data loaders that are simple to execute. Data is stored in an array.
When using Keras is getting their dataset in the correct shape (dimensionality) required for Keras. correct shape (dimensionality) required for Keras.

However, the input required by Keras requires us to define it in the following format:
Number of Samples, **Rows**, **Cols**, **Depth Number of Samples**, **Rows, Cols**, **Depth**
Therefore, since our MNIST dataset is grayscale grayscale grayscale, we need it in the form of:
60000, 28, 28, 1 (if it were in color, it would be 60000, 28, 28, 3)
Using Numpy reshape reshape reshape function, we can easily add that 4th dimension to our data

But we can easily get this done in keras and get data augmentation out of the box in addition. In deep learning, the more training data/examples we have the better our model will be on unseen data (test data).Keras provides an easy way with the imagaDataGenerator to load images from the generator without going through the steps above. Remember we said CNN requires more understanding of the image since it is

position and position invariance. Adding variations such as rotations, shifts, zooming etc make our classifier much more invariant to changes in our images. Thus making it far more robust.

To use the generator in loafing our images

1. Create variables to the image directory

```
train_data_dir = '/content/drive/My Drive/DeepLearning/cancer/train'
validation_data_dir = '/content/drive/My Drive/DeepLearning/cancer/validation'
```

2. Create a generator for the test

```
# Creating our data generator for our test data
validation_datagen = ImageDataGenerator(
    # used to rescale the pixel values from [0, 255] to [0, 1] interval
    rescale = 1./255)

# Creating our data generator for our training data
train_datagen = ImageDataGenerator(
      rescale = 1./255,              # normalize pixel values to [0,1]
      rotation_range = 30,           # randomly applies rotations
      width_shift_range = 0.3,       # randomly applies width shifting
      height_shift_range = 0.3,      # randomly applies height shifting
      horizontal_flip = True,        # randonly flips the image
      fill_mode = 'nearest')         # uses the fill mode nearest to fill gaps created by the above
```

3. Use the flow from directory method to

```
# Specify criteria about our training data, such as the directory, image size, batch size and type
# automagically retrieve images and their classes for train and validation sets
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size = (img_width, img_height),
        batch_size = batch_size,
        class_mode = 'binary',
        shuffle = True)

validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size = (img_width, img_height),
        batch_size = batch_size,
        class_mode = 'binary',
        shuffle = False)
```

This produces an output like

```
Found 1857 images belonging to 2 classes.
Found 1502 images belonging to 2 classes.
```

Breast Cancer DataSet Analysis

Let's take a look at the distribution of classes among sets:

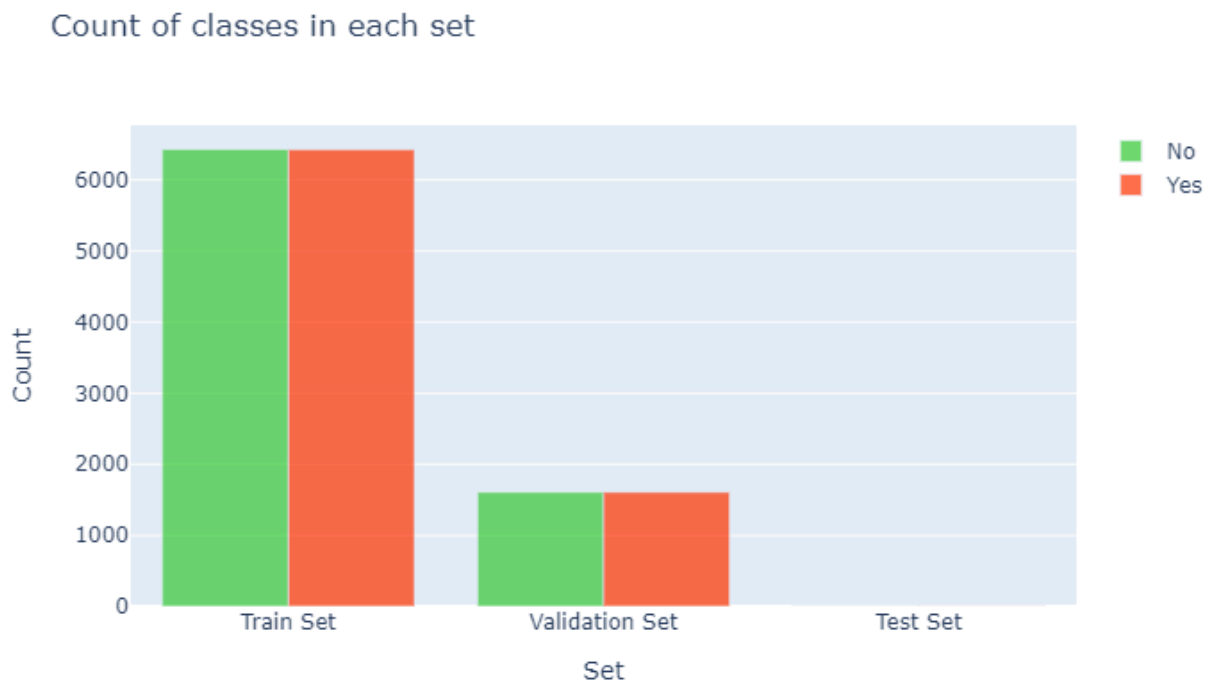
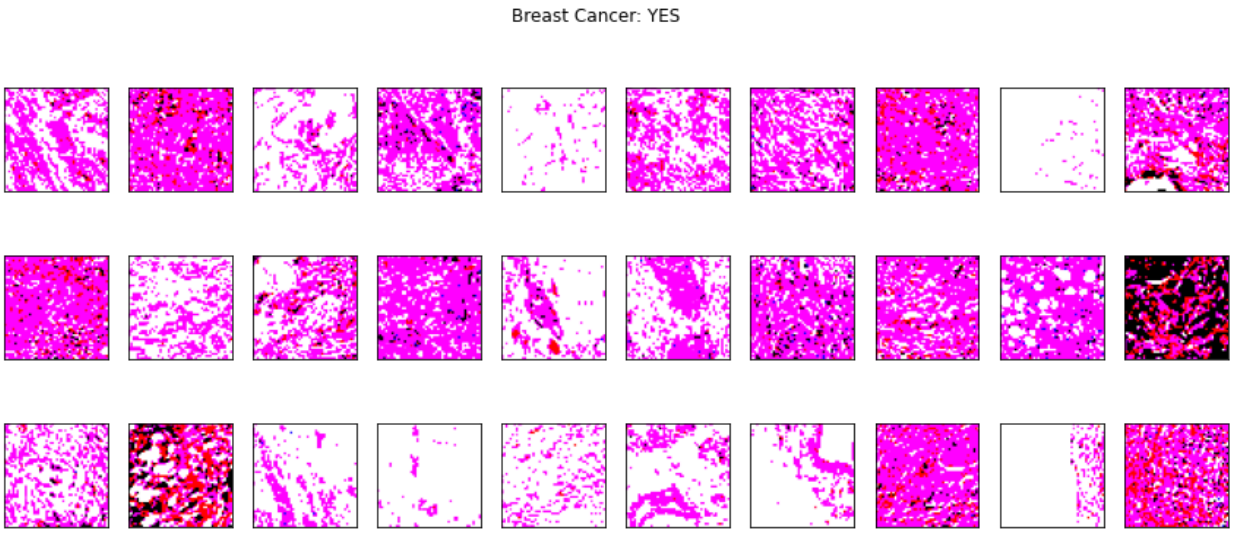
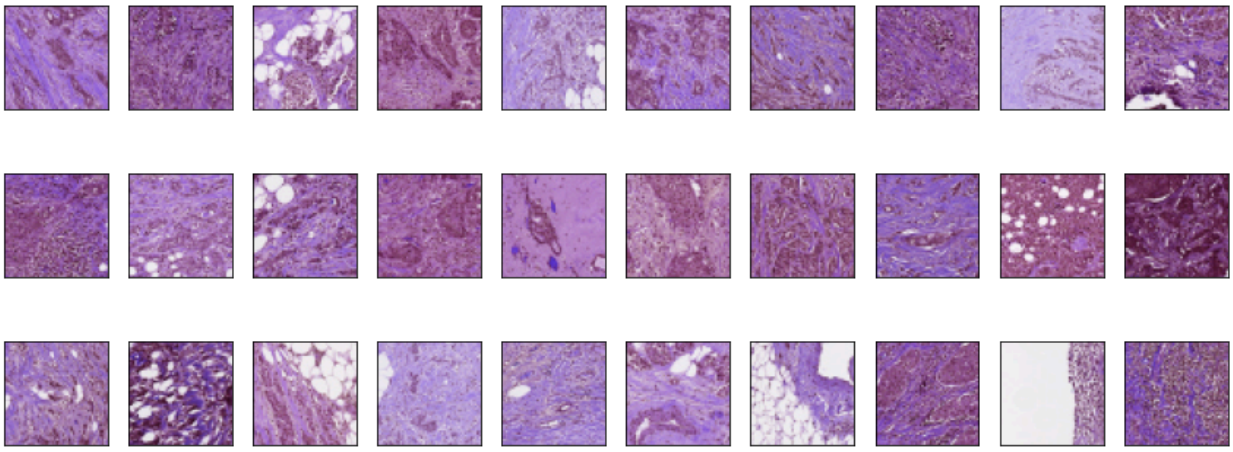
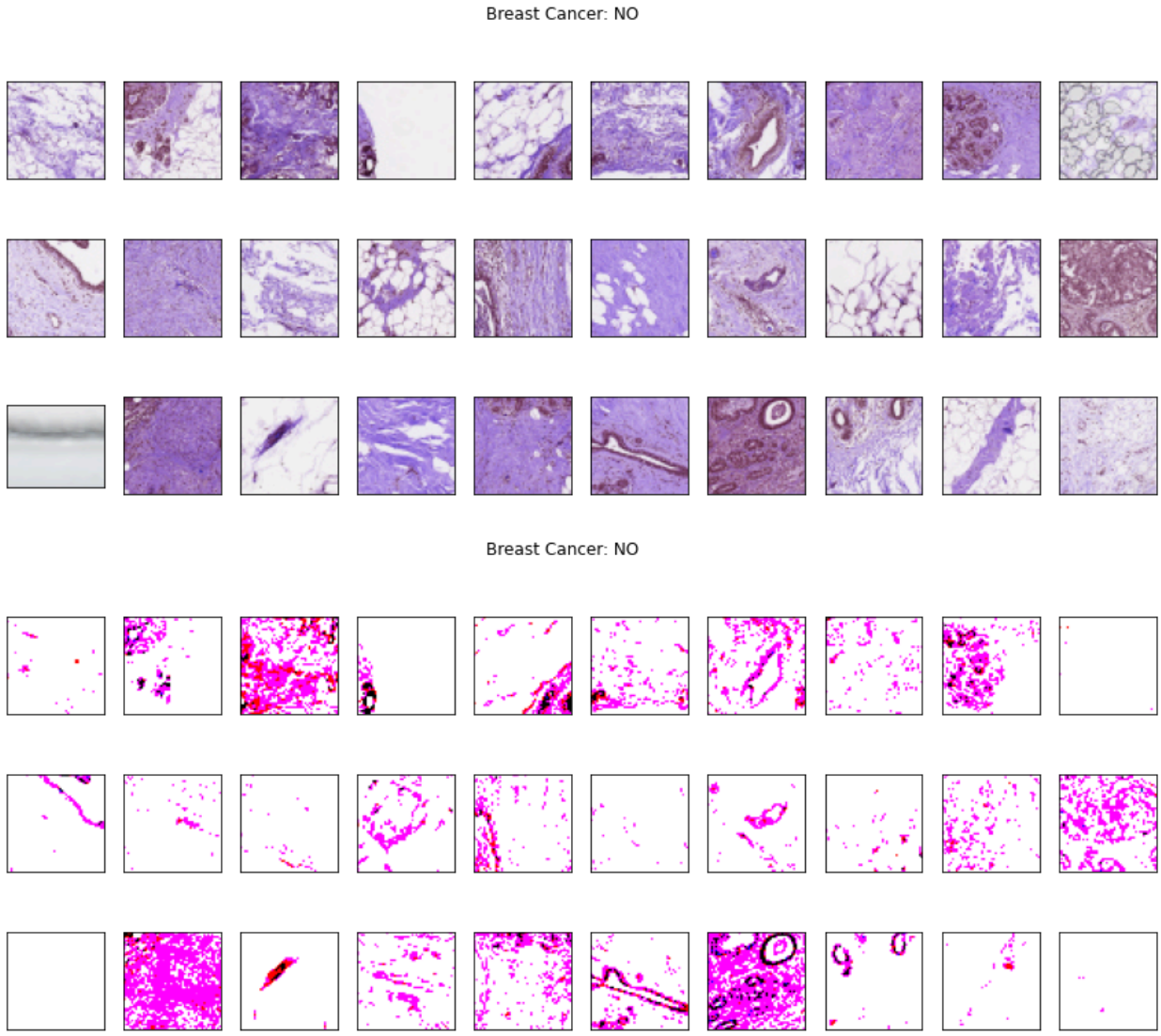


Fig 4.0.3

Plotting 30 samples of the Dataset Side by side (Breast Cancer)






Breast Cancer: NO
Breast Cancer: NO

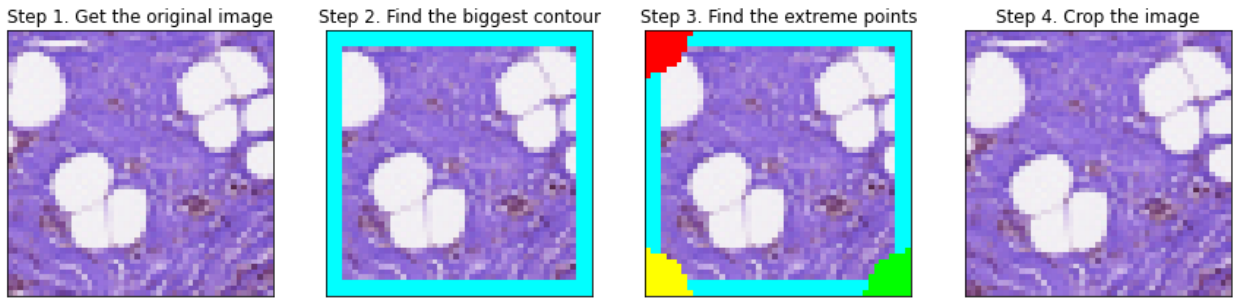

Step 1. Get the original image
Step 2. Find the biggest contour
Step 3. Find the extreme points
Step 4. Crop the image

Fig 4.0.1

Data Augmentation Techniques (Breast Cancer)

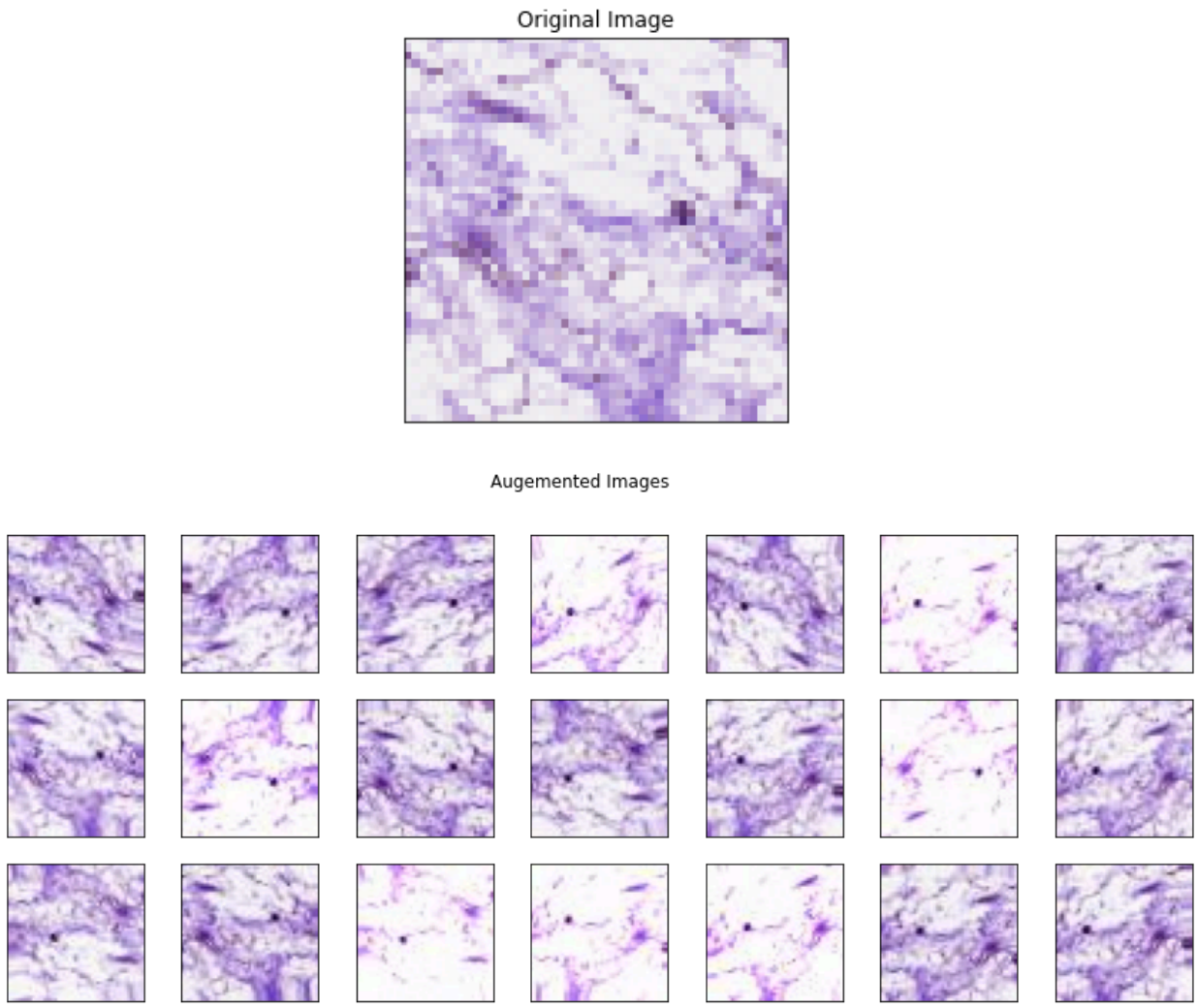


Fig 4.0.1

Building The Model (CNN) - Breast Cancer

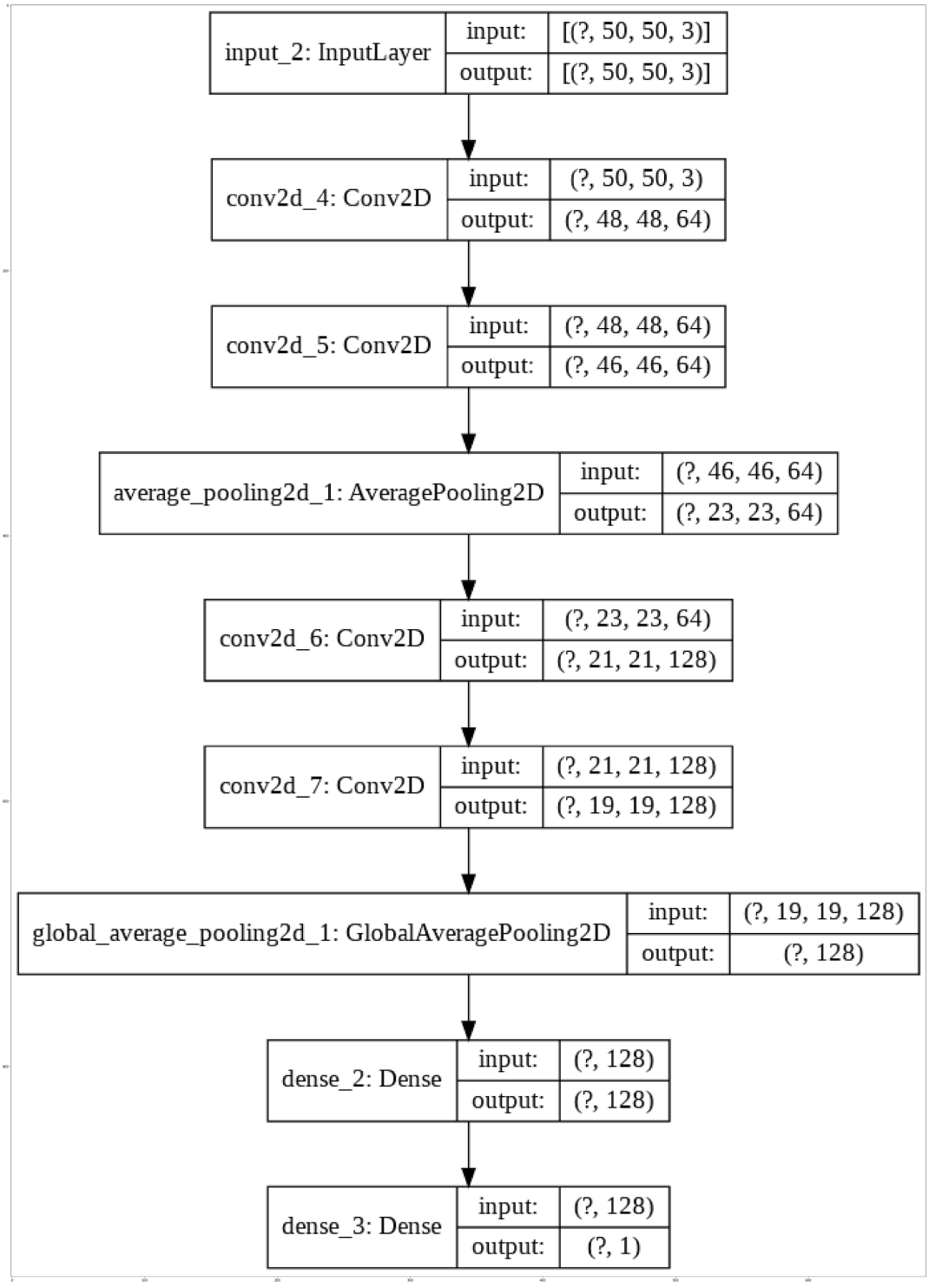
```
batch_size = 128
num_classes = 1
img_rows, img_cols = 224, 224


input_image = Input(shape=(224,224,3))
cnn = Conv2D(64, (3, 3), activation='relu')(input_image)
cnn = Conv2D(64, (3, 3), activation='relu')(cnn)
cnn = AveragePooling2D((2,2))(cnn)
cnn = Conv2D(128, (3, 3), activation='relu')(cnn)
cnn = Conv2D(128, (3, 3), activation='relu')(cnn)
cnn = GlobalAveragePooling2D()(cnn)
dense = Dense(128, activation='relu')(cnn)
output = Dense(NUM_CLASSES, activation='sigmoid')(dense)

cnn_model = Model(inputs=input_image, outputs=output)
# model.compile(loss=lambda y_true,y_pred: y_true*K.relu(0.9-y_pred)**2 + 0.25*(1-y_true)*K.relu(y_pred-0.1)**2,
#               optimizer='adam',
#               metrics=['accuracy'])

cnn_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

cnn_model.summary()


cnnhistory = cnn_model.fit_generator(
    train_generator,
    steps_per_epoch = 128 // batch_size,
    epochs = 5,
    validation_data = validation_generator,

    # validation_steps = nb_validation_samples // batch_size
    )
```

Visualizing the model - Breast Cancer CNN

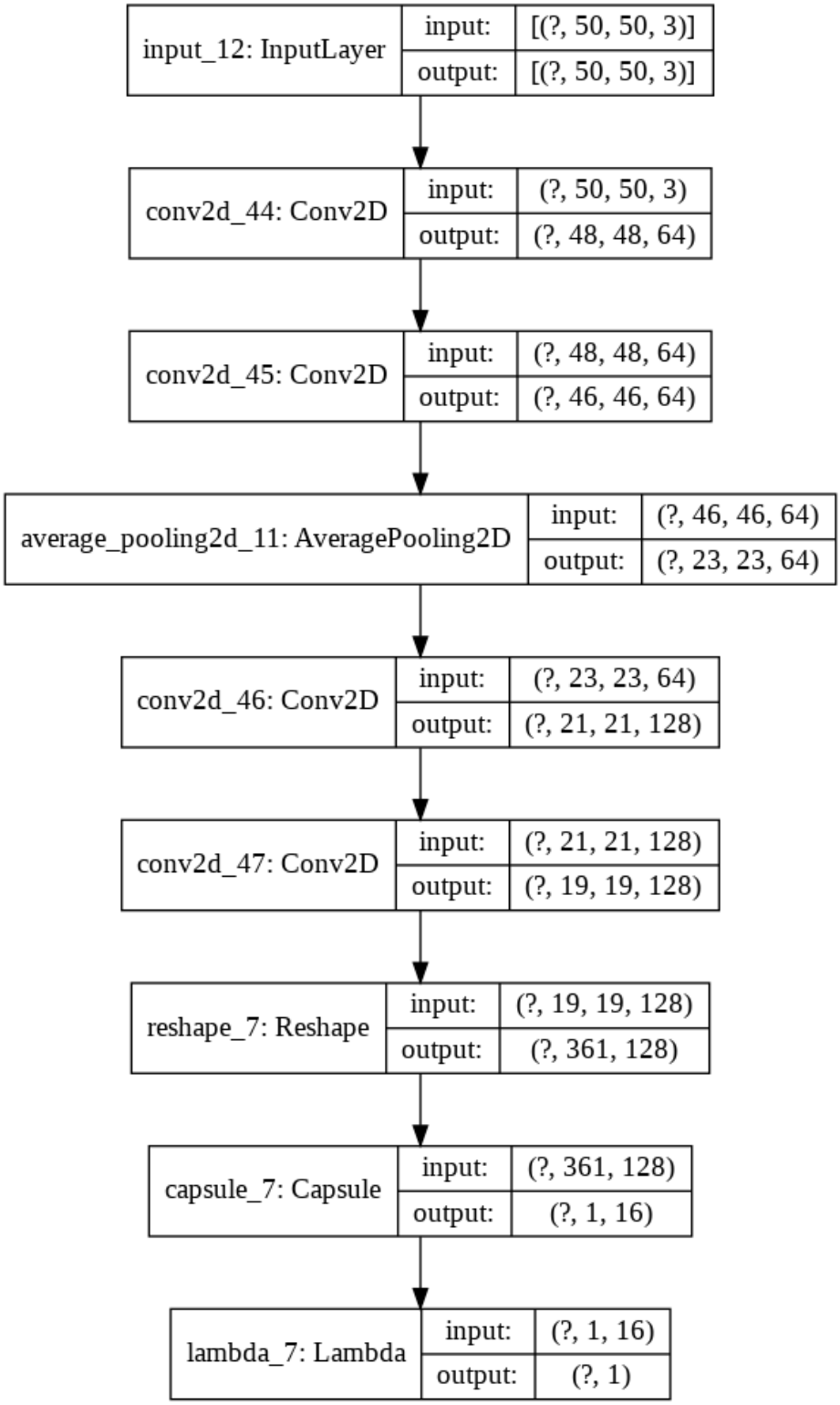
```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 50, 50, 3)]       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 48, 48, 64)        1792
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 46, 46, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 21, 21, 128)       73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 19, 19, 128)       147584
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 276,801
Trainable params: 276,801
Non-trainable params: 0
```

Fig 4.0.4 Visualizing the model

VISUALIZING THE MODEL (CNN) -



VISUALIZING THE MODEL (CAPSULE)


input_12: InputLayer
input: [(?, 50, 50, 3)]
output: [(?, 50, 50, 3)]
conv2d_44: Conv2D
input: (?, 50, 50, 3)
output: (?, 48, 48, 64)
conv2d_45: Conv2D
input: (?, 48, 48, 64)
output: (?, 46, 46, 64)
average_pooling2d_11: AveragePooling2D
input: (?, 46, 46, 64)
output: (?, 23, 23, 64)
conv2d_46: Conv2D
input: (?, 23, 23, 64)
output: (?, 21, 21, 128)
conv2d_47: Conv2D
input: (?, 21, 21, 128)
output: (?, 19, 19, 128)
reshape_7: Reshape
input: (?, 19, 19, 128)
output: (?, 361, 128)
capsule_7: Capsule
input: (?, 361, 128)
output: (?, 1, 16)
lambda_7: Lambda
input: (?, 1, 16)
output: (?, 1)

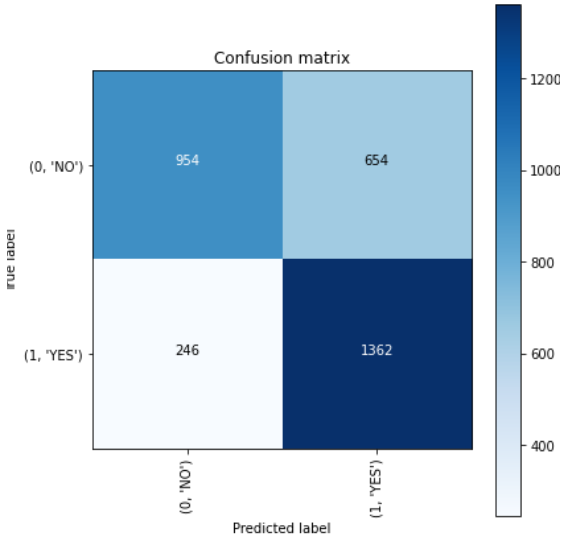
Testing of Breast Cancer Training Methods

After running the model on both dataset, we generate the confusion matrix. Below shows how the validation set of images of 2316 breast cancer images runs with their respective validation accuracy. Overall the performance shows the CNN has better accuracy even though the confusion matrix shows a couple of misclassifications.

Validate on Validation Set

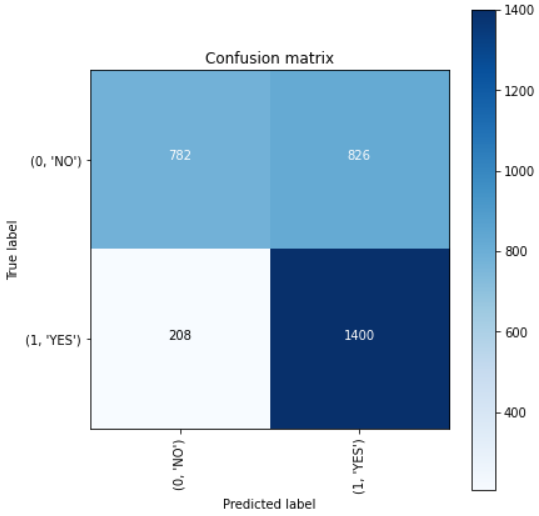
CONFUSION MATRIX CNN

VALIDATION ACC **0.72**



CONFUSION MATRIX CAPSULE

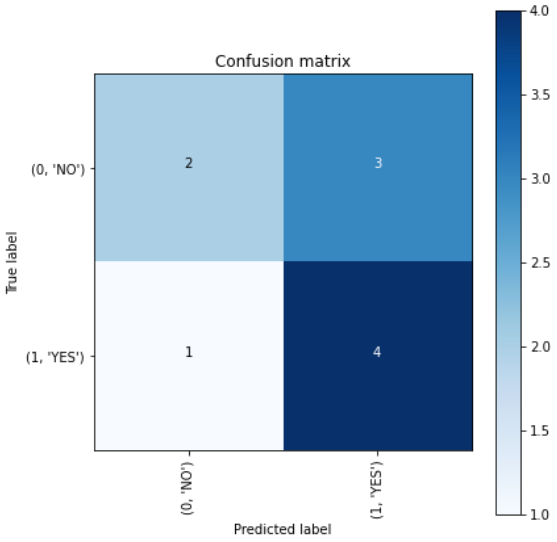
VALIDATION ACC **0.68**



Validate on Test Set

CONFUSION MATRIX CNN

VALIDATION ACC **0.60**



CONFUSION MATRIX CAPSULE

VALIDATION ACC **0.50**

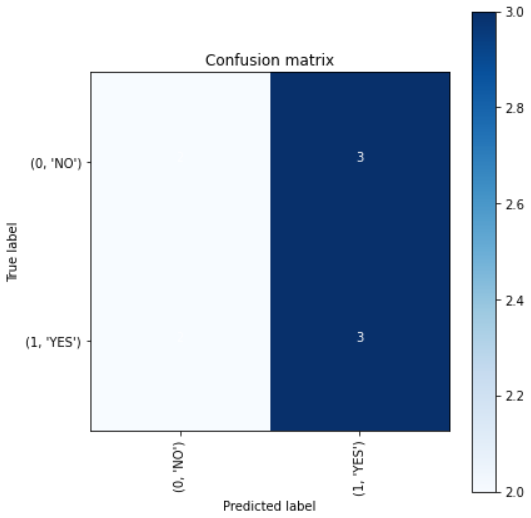
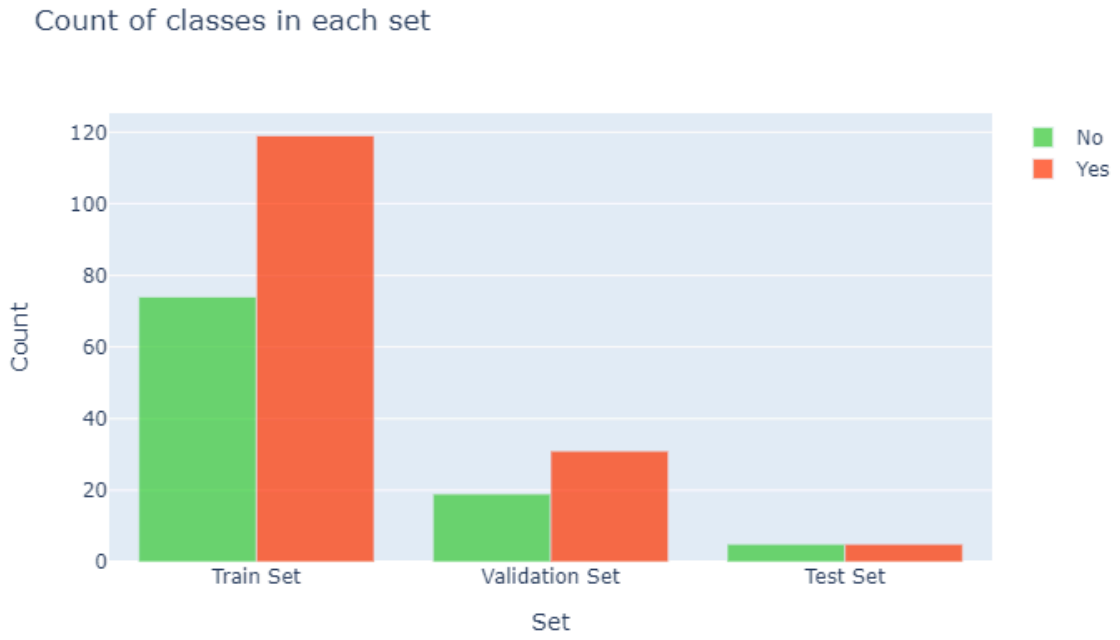


Fig 4.0.5 Confusion Matrix

With the validation on the test set of images of 10 sample brain tumor images, their respective validation accuracy are above them. Overall the performance shows the CNN has better accuracy even though the confusion matrix shows a couple of misclassifications.

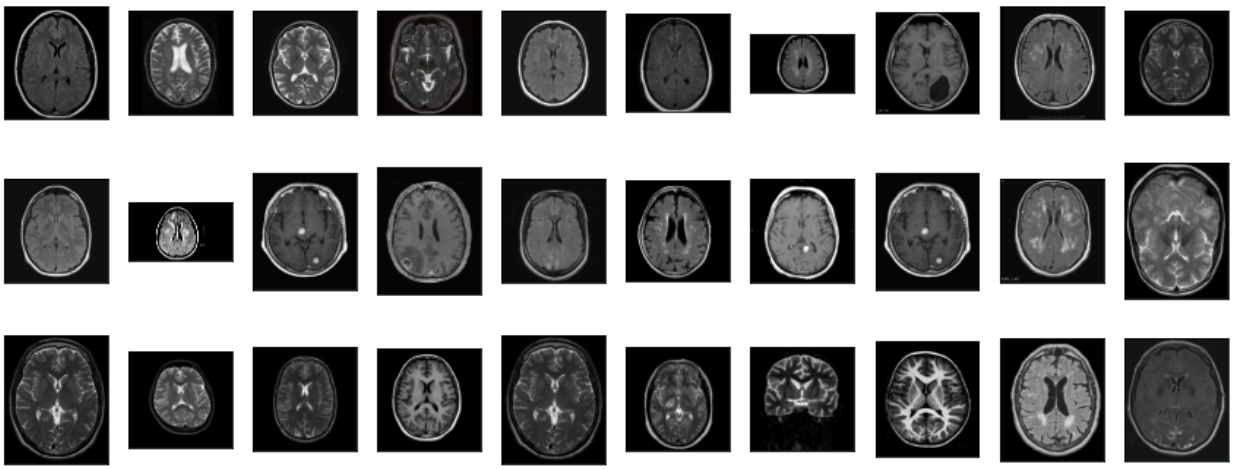
Practical Implementation of CNN and Capsule Network on breast cancer

Most of the concepts discussed from above have the same concepts when it comes to data loading and preparation. But there are a few implementation details we added to the

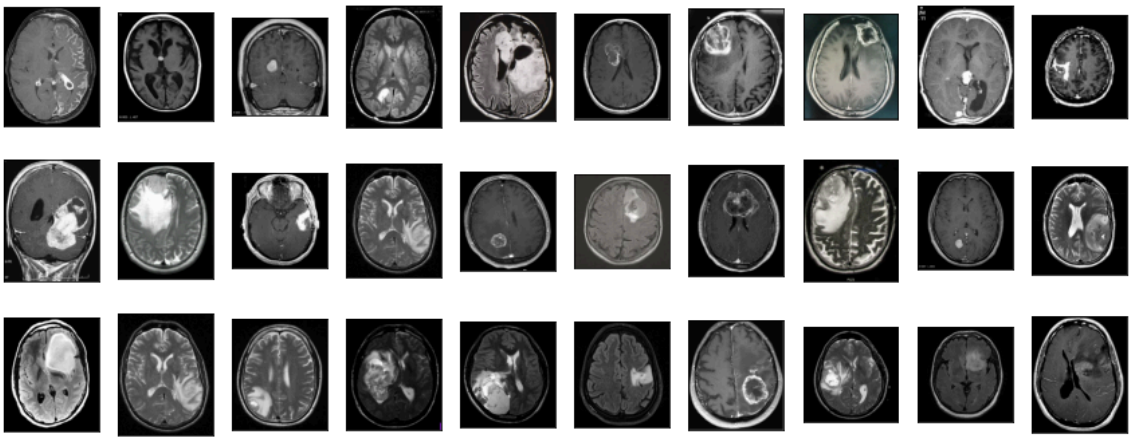


Plotting 30 samples of the Dataset Side by side

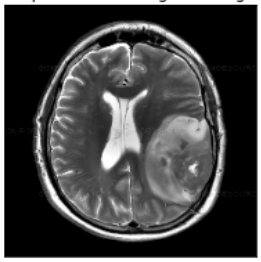
Tumor: NO



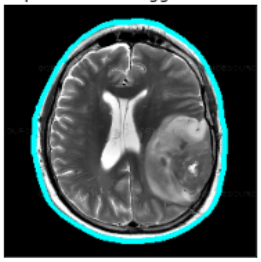
Tumor: YES



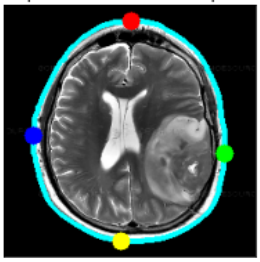
Step 1. Get the original image



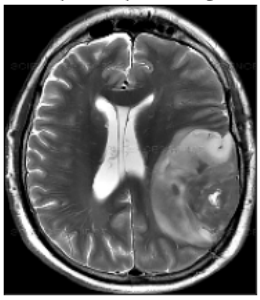
Step 2. Find the biggest contour



Step 3. Find the extreme points



Step 4. Crop the image



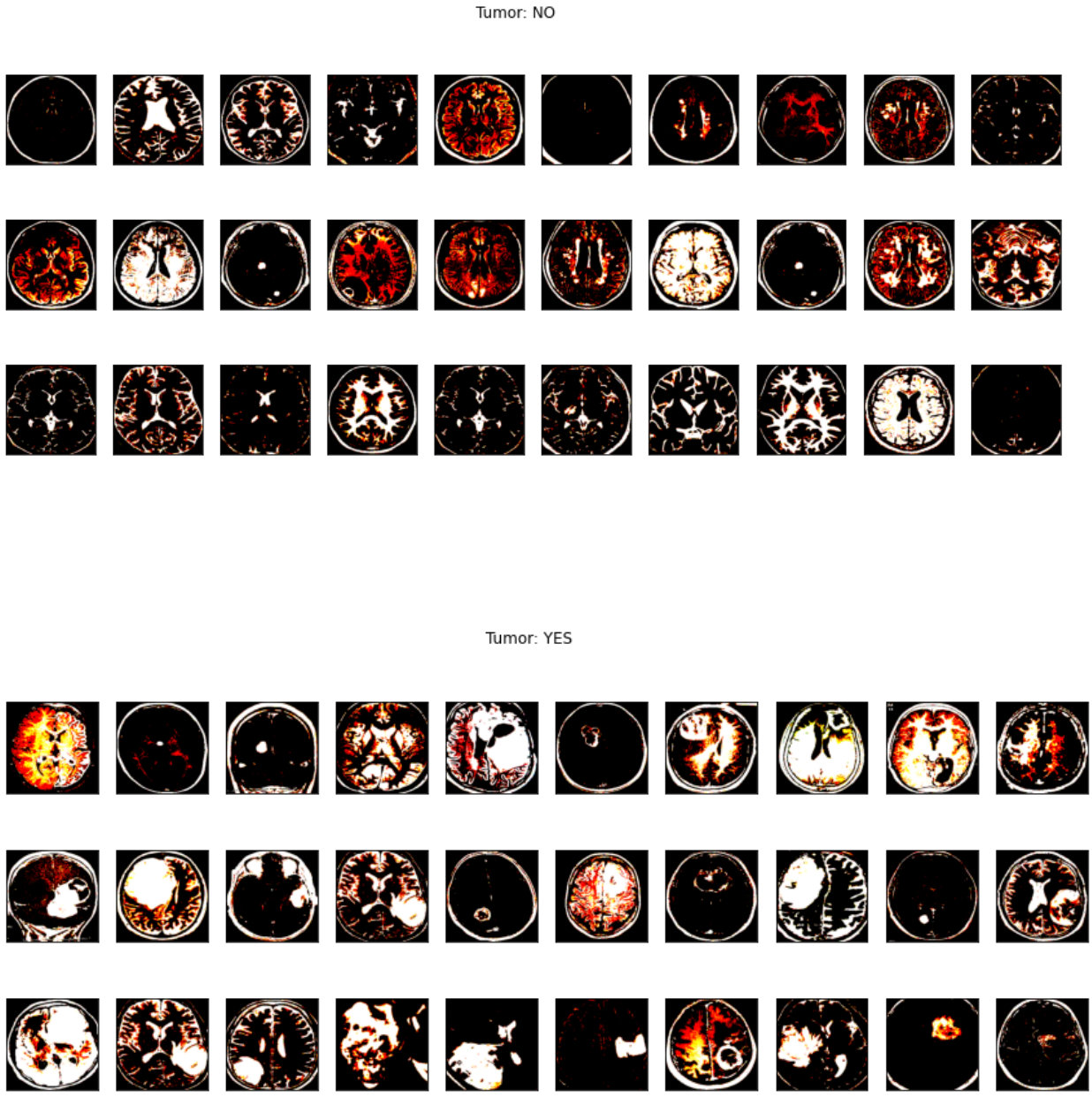
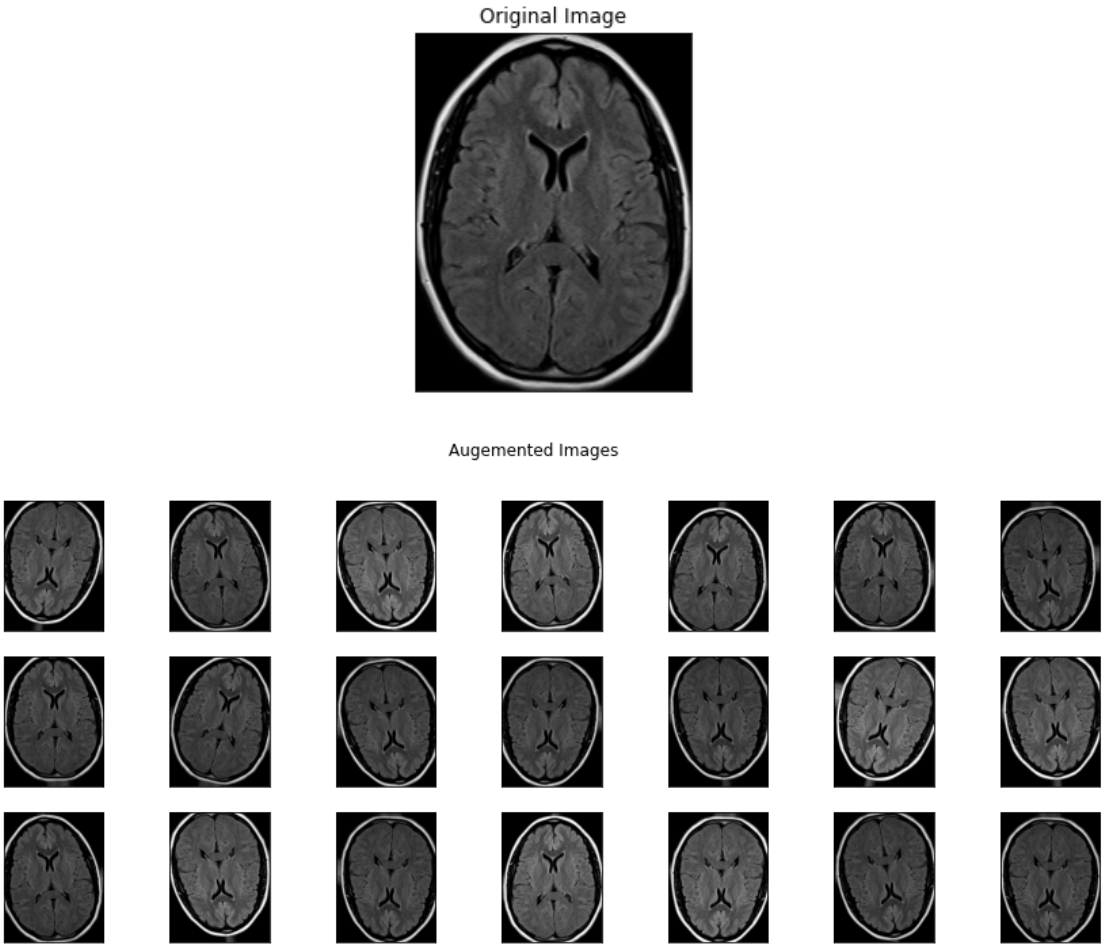
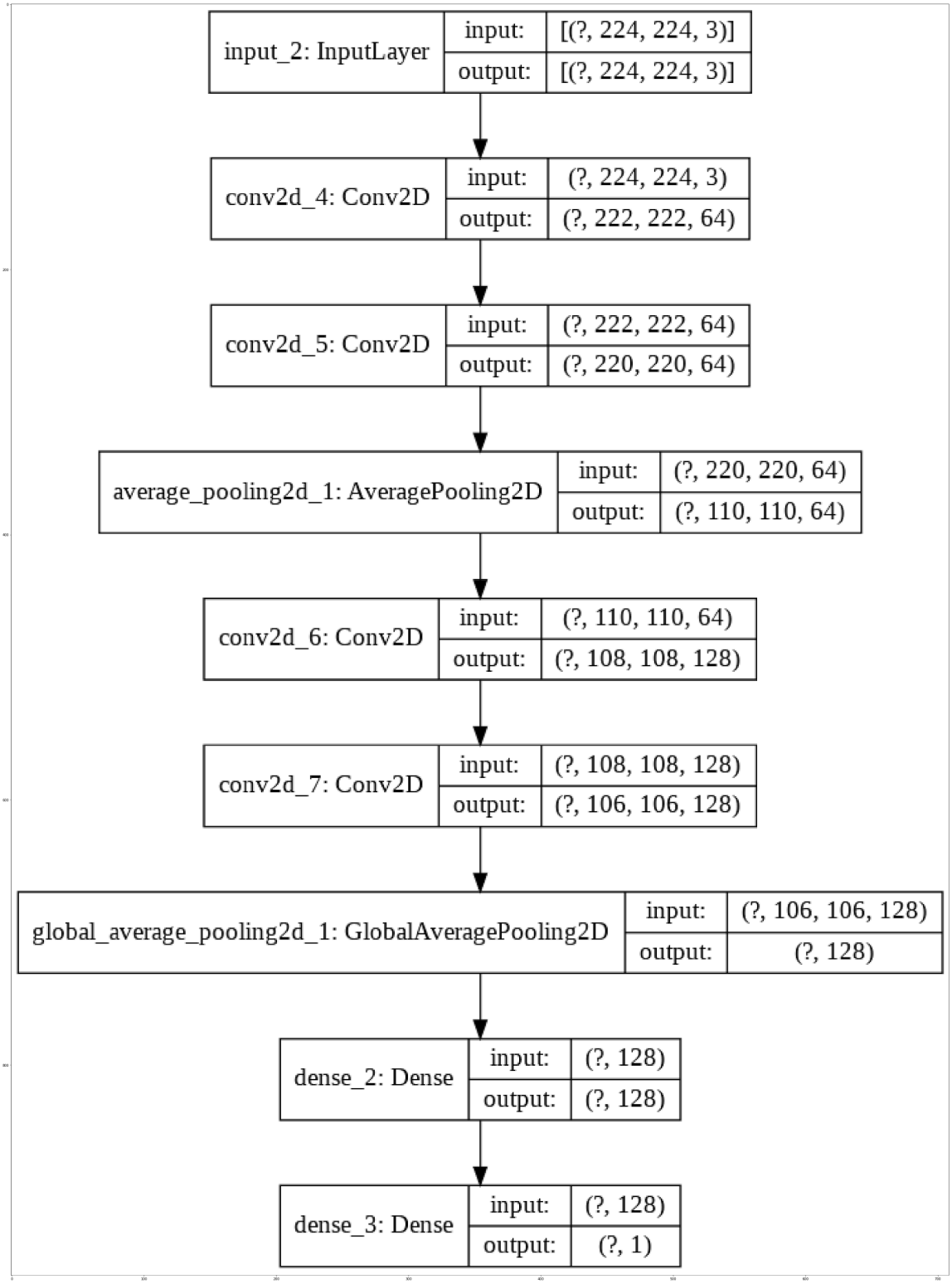


Fig 4.0.6 Brain Tumor Patches

Data Augmentation - Brain Tumor



Building the model (CNN) - Brain Tumor


input_2: InputLayer
input: [(?, 224, 224, 3)]
output: [(?, 224, 224, 3)]
conv2d_4: Conv2D
input: (?, 224, 224, 3)
output: (?, 222, 222, 64)
conv2d_5: Conv2D
input: (?, 222, 222, 64)
output: (?, 220, 220, 64)
average_pooling2d_1: AveragePooling2D
input: (?, 220, 220, 64)
output: (?, 110, 110, 64)
conv2d_6: Conv2D
input: (?, 110, 110, 64)
output: (?, 108, 108, 128)
conv2d_7: Conv2D
input: (?, 108, 108, 128)
output: (?, 106, 106, 128)
global_average_pooling2d_1: GlobalAveragePooling2D
input: (?, 106, 106, 128)
output: (?, 128)
dense_2: Dense
input: (?, 128)
output: (?, 128)
dense_3: Dense
input: (?, 128)
output: (?, 1)

Building the model (CAPSULE) - Brain Tumor

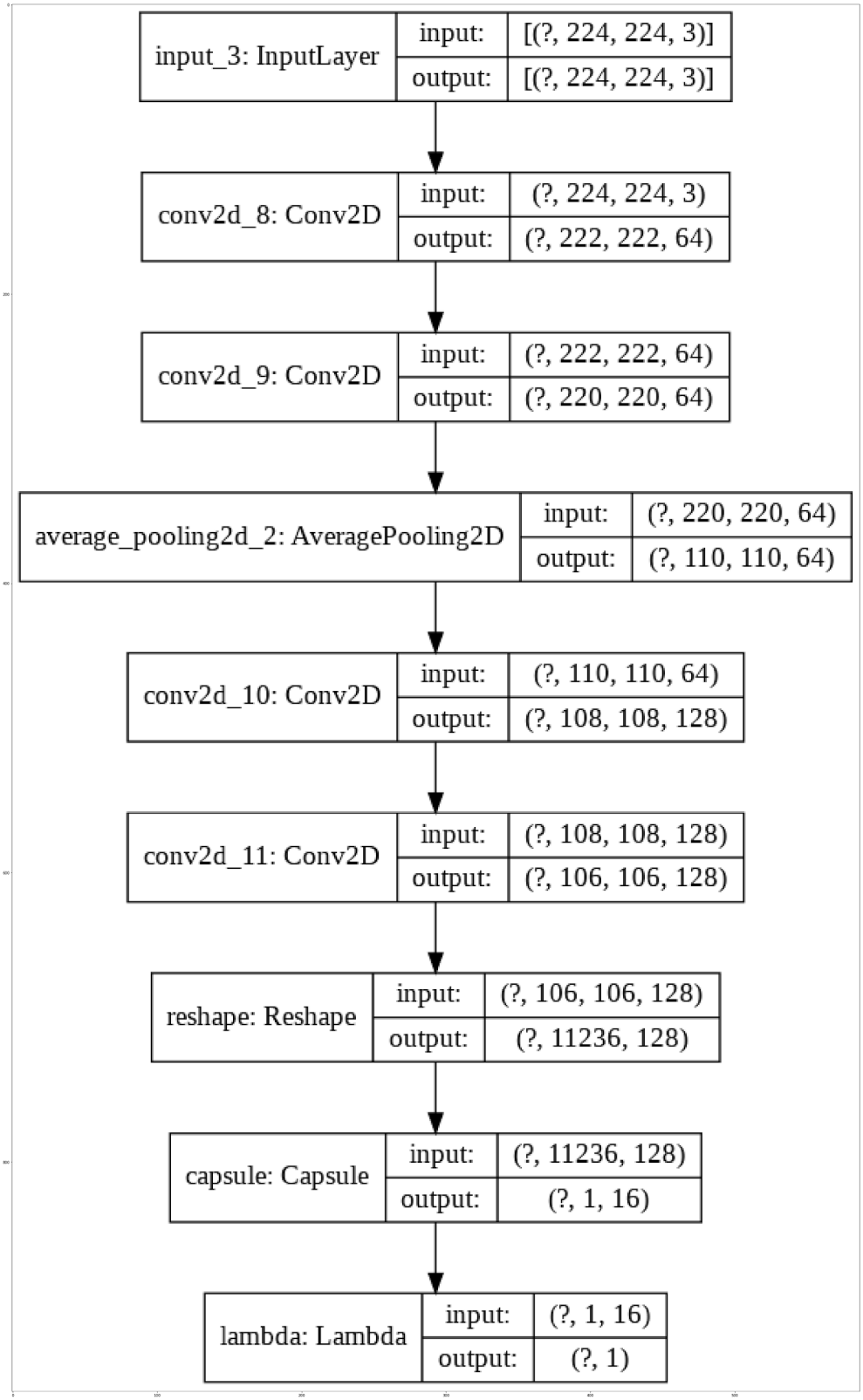


Fig 4.0.7 Model Representation

Testing of Methods

After running the model on both dataset, we generate the confusion matrix. Below shows how the validation set of images of 50 brain tumor images runs with their respective validation accuracy. Overall the performance shows the capsule has better accuracy even though the confusion matrix shows a couple of misclassifications.

Validate on Validation Set

CONFUSION MATRIX CNN
VALIDATION ACC **0.50**

CONFUSION MATRIX CAPSULE
VALIDATION ACC **0.60**

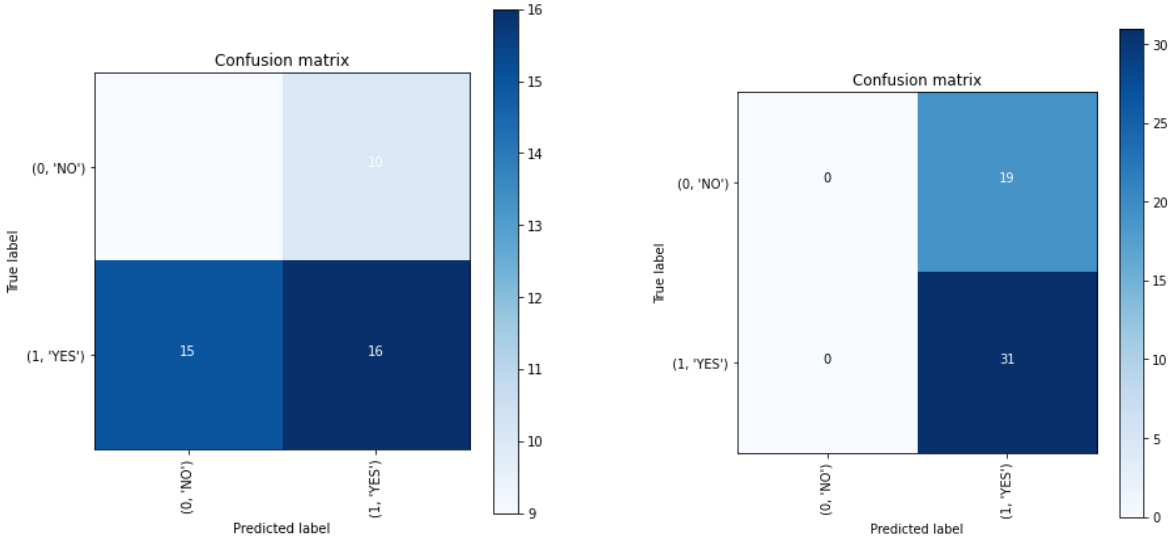


Fig 4.0.8 Confusion matrix for brain tumor

With the validation on the test set of images of 10 sample brain tumor images, their respective validation accuracy are above them. Overall the performance shows the CNN has better accuracy even though the confusion matrix shows a couple of misclassifications.

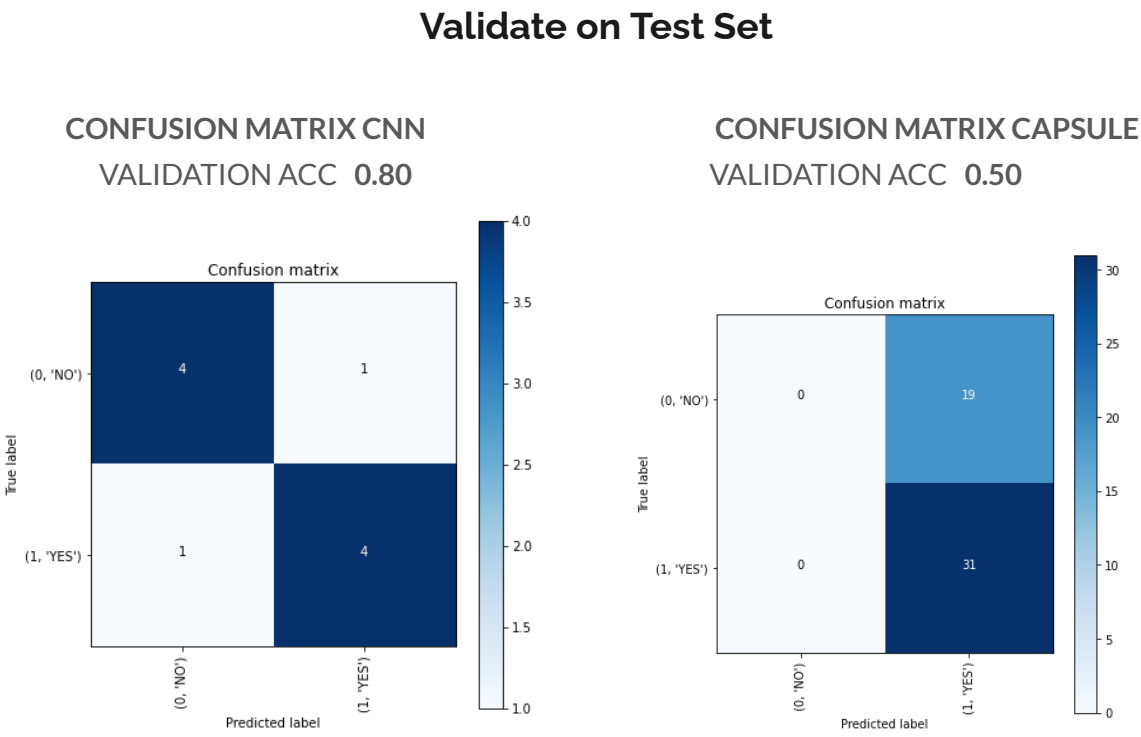


Fig 4.0.8

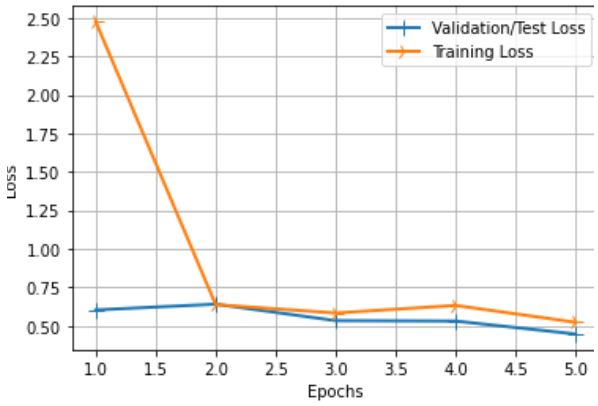
* Note: there might be some misunderstanding in terms of set names so I want to describe what do I mean by test and validation set:
* *validation set - is the set used during the model training to adjust the hyperparameters.
*
* test set - is the small set that I don't touch for the whole training process at all. It's been used for final model performance evaluation.

Evaluation of Methods

Validation Loss

Below summarizes the validation loss of the trained model.

CNN



CAPSULE

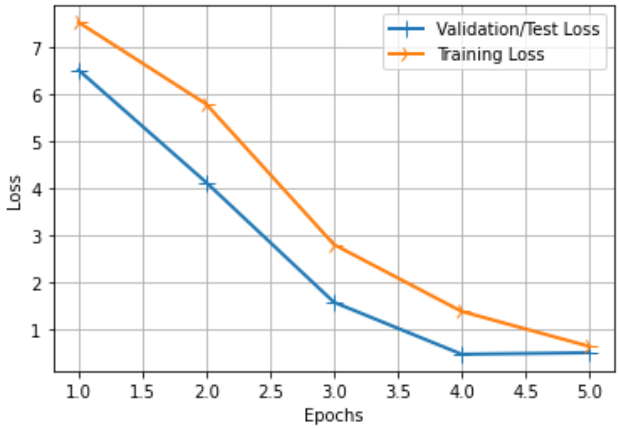


Fig 4.0.9

Results & Discussion of Both Methods

This project was a combination of CNN and capsule model classification problem (to predict whether the subject has a brain tumor or not) & also to predict whether there is a cancerous cell or not. The final accuracy of the capsule is much higher than 50% baseline (random guess). However, it could be increased by a larger number of train images or through model hyperparameters tuning. CNN on the other hand still shows more promises even under this constraints dataset.

Chapter 5 Conclusion and Recommendation

Conclusion

Building and labelling large datasets for solving medical image classification problems is today a bottleneck for many applications. After comparing the behavior of capsule networks against ConvNets under typical datasets constraints of medical image analysis, namely, small amounts of annotated data and class-imbalance and evaluating our experiments on a medical (histological breast cancer images) publicly available datasets and Brain Tumor images also publicly available. Our results suggest that capsule networks can be trained with less amount of data for the same or better performance and are more robust to an imbalanced class distribution, which makes our approach very promising for the medical imaging community.

Recommendation

We recommend using these two models hand in hand. Looking at the results of these two models on the medical images clearly indicates that some methods clearly fit some operations and if used together can be very useful. Clearly from the implementation, our training model used a CNN + Capsule implementation. Also with a little bit more training and more complex data augmentation, better performance can be obtained from the model.

References

1. Ashwin Bhandare, Maithili Bhide,Pranav Gokhale and Rohan Chandavarkar. "Applications of Convolutional Neural Networks." 2016
2. Hubel and Wiesel , Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. 1962 Jan
3. Kunihiko Fukushima , Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position . 1980
4. Simeon Kostadinov, Understanding the Backpropagation algorithm, August 2019
5. Steinkraus, D. & Buck, I. & Simard, Patrice. (2005). Using GPUs for machine learning algorithms. 1115 - 1120 Vol. 2. 10.1109/ICDAR.2005.251.
6. Strigl, Daniel & Kofler, Klaus & Podlipnig, Stefan. (2010). Performance and Scalability of GPU-Based Convolutional Neural Networks. 317-324. 10.1109/PDP.2010.43.
7. Xiaqing Li, Guangyan Zhang, H. Howie Huang, Zhufan Wang, Weimin Zheng (2016). Performance Analysis of GPU-based Convolutional Neural Networks.
8. Faizan Shaikh, Why are GPUs necessary for training Deep Learning models?, May 2017
9. Alex Krizhevsky, Ilya Sutskever & Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks
10. Christian Szegedy,, Wei Liu , Yangqing Jia , Pierre Sermanet , Scott Reed, Dragomir Anguelov , Dumitru Erhan , Vincent Vanhoucke, Andrew Rabinovich (2015). Going Deeper with Convolutions
11. Sara Sabour , Nicholas Frosst ,Geoffrey E. Hinton (2017). Dynamic Routing Between Capsules

Appendices

- https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection
- https://colab.research.google.com/
- https://keras.io/api/models/model_saving_apis/
- https://software.intel.com/content/www/us/en/develop/articles/understanding-capsule-network-architecture.html
- https://www.youtube.com/watch?v=pPN8d0E3900
- https://github.com/tensorflow/datasets
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
- https://www.curiousily.com/posts/hackers-guide-to-fixing-underfitting-and-overfitting-models/
- https://github.com/bojone/Capsule
- https://github.com/keras-team/keras
- https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd