Ministry of Higher Education
Modern Academy for Computer Science

Brain Minions!

By Supervisors:

Dr. Abbass Rostomme

Eng. Maryam Hossam

2019-2020

Modern Academy
Since 1993
maadi
مودرن اکادیمی بالمعادی

[Brain Minions!]

Angry Nerds

Angry Nerds Team:

1- Mostafa Nabieh

2-Abdalrhman Sherif

3- Ahmed Hamada

Supervisors:

1-Dr. Abbass Rostomme

2-Eng. Maryam Hossam

Table of Contents

List of Figures

[1-Acknowledgements]

First of all, we would like to express our gratitude and appreciation to all those who gave us the opportunity to complete this project. A special thanks to our final year project coordinator, Dr. Abbass Rostomme, Eng. Maryam Hossam, whose help, stimulating suggestions and encouragement, helped us to coordinate our project especially in writing this report.

We would also like to acknowledge with much appreciation the crucial role of the staff of Computer Science and Engineering Lab, who gave the permission to use all required machinery and the necessary material to complete the project.

Last, but not least, many thanks go to the head of the project, Dr. Abbass Rostomme, who gives us his full effort for guiding the team in achieving the goal as well as his encouragement to maintain our progress in track. We would like to appreciate the guidance given by other supervisors as well as the panels especially in our project presentation that has improved our presentation skills by their comments and tips.

[2-Abstract]

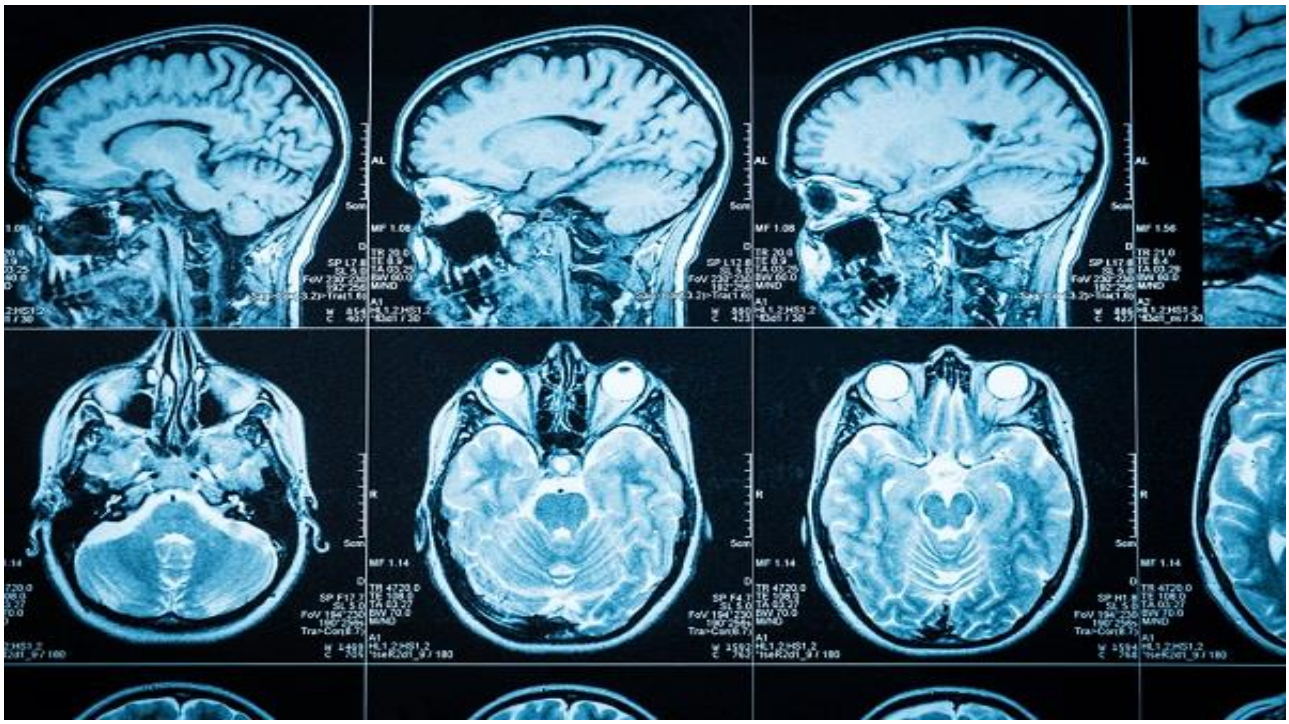

Figure 2.1 Brain tumor

For disease diagnosis, monitoring and treatment preparation, automated segments of brain tumors from 3D magnetic resonance imaging (MRIs) are required. Anatomical knowledge is necessary for manual delineation practices and are costly and time consuming, and can be incorrect due to human error. Here we identify an encoder-decoder-based semitone segmentation network for the 3D MRI tumor subregion segmentation. Due to a limited training data size, the input image itself is reconstructed by a variational self-encoder branch in order to regularize the common decoder and to impose further constraints on its layers. The field of medical imaging is gaining importance with an increase in the demand for automated, reliable, fast and efficient diagnosis which can provide insight to the image better than human eyes.

Brain tumor is the second leading cause for cancer-related deaths in men in age 20 to 39 and fifth leading cause cancer among women in same age group. Brain tumors are painful and may result in various

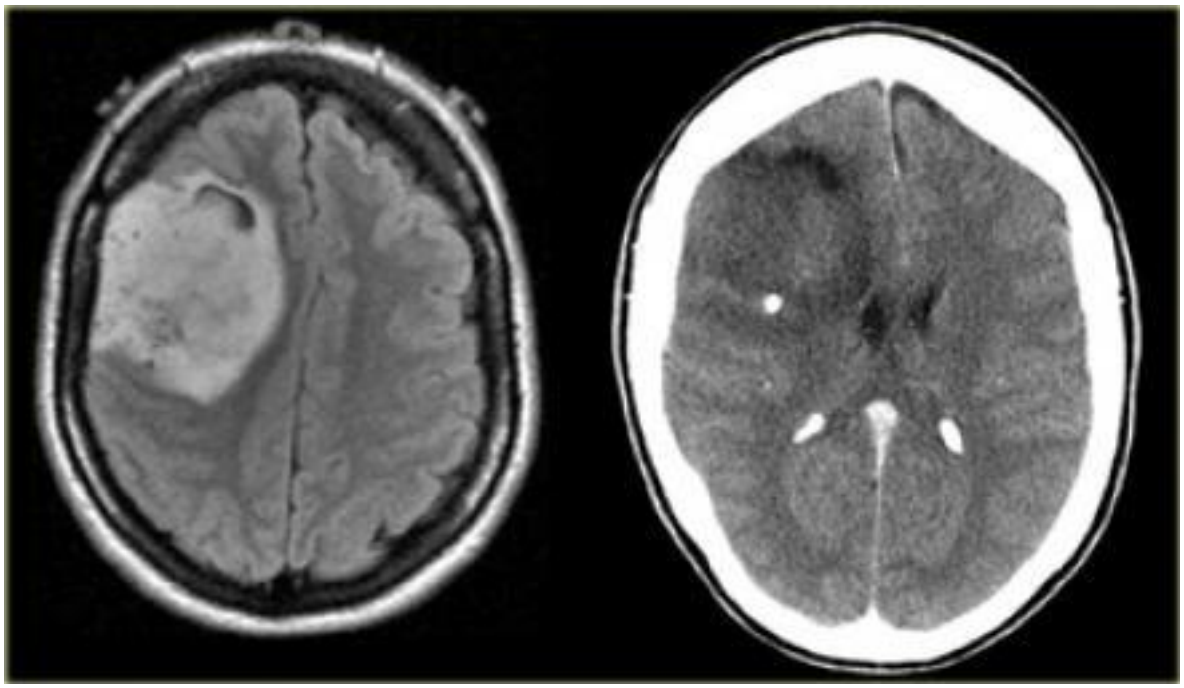

Figure 2.2 Tumor type

diseases if not cured properly. Diagnosis of tumor is a very important part in its treatment. Identification plays an important part in the diagnosis of benign and malignant tumors. A prime reason behind an increase in the number of cancer patients worldwide is the ignorance towards treatment of a tumor in its early stages. This paper discusses such an algorithm that can inform the user about details of tumor using basic image processing techniques.

These methods include:

noise removal and sharpening of the image along with basic morphological functions, erosion and dilation, to obtain the background.

[3- Introduction]

[3-1-Overview]

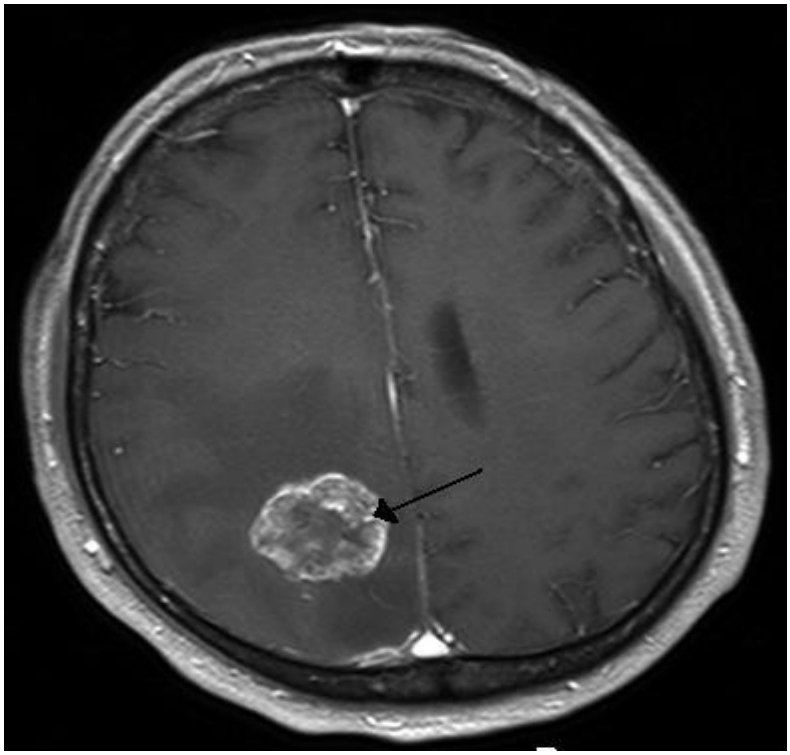

Figure 3.1.1 Brain

In this era of unprecedented changes, with the increase in ailments, the field of heath demands the use of new technology to provide solutions to these problems.

The brain tumor is one such malady that is the most life-threatening disease known to mankind.

Even if this disease is detected in the primal stage, it's not completely possible to find a cure within the time frame and save the patient's life. The Tumor can be classified into two types:

- ***Malignant*** (meaning the cells are cancerous and lethal to patient's life).
- ***Benign*** (meaning the presence of tumor won't affect the health of the patient in any manner).

It is commonly accepted that the most important part of the human body is the human brain, therefore, if anything shall happen to it, it will directly impact the life expectancy of the patient.

The development of a prediction model for the diagnose of brain tumors would greatly benefit the medical community.

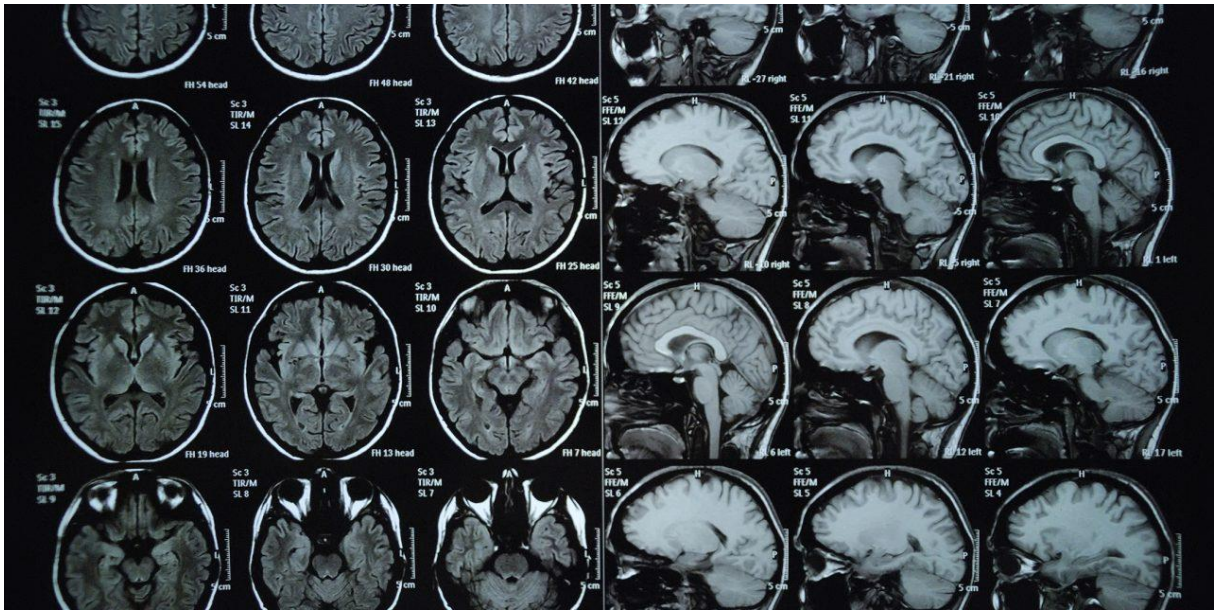

Figure 3.1.2 MRI Brain

In this project, we will build a convolutional neural network model that classifies an image set of MRI scans based on the presence of tumors by comparing it with the previous scans of a brain tumor.

With this project, we seek to help and devise a solution that could be useful to detect this lethal disease.

Furthermore, this model may be trained iteratively on larger and larger datasets, using a process called Ensemble Learning.

[3-2-Problem Definition]

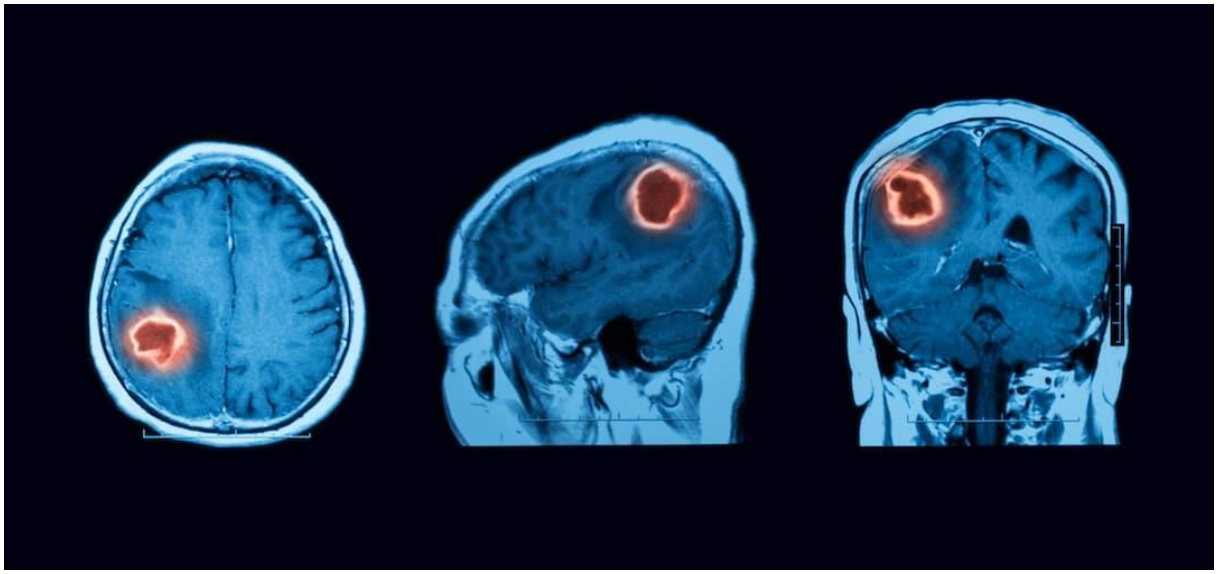

Figure 3.2.1 Primary Brain tumors

A brain tumor is defined as abnormal growth of cells within the brain or central spinal canal. Some tumors can be cancerous thus they need to be detected and cured in time. The exact cause of brain tumors is not clear and neither is exact set of symptoms defined, thus, people may be suffering from it without realizing the danger.
Primary brain tumors can be either *malignant* (contain cancer cells) or *benign* (do not contain cancer cells).
Brain tumors are divided into primary and secondary types of tumors:

- **Primary Brain tumors:** come from brain cells,
- **Secondary**: metastasizing of tumors Other organ brain.

In the brain. Gliomas, which arise from brain glial cells, are the most common type of primary brain tumor. Low grade gliomas can be Subtypes (LGG) and high quality (HGG). Gliomas of high quality are aggressive. There are two types of brain tumor which is primary brain tumor and metastatic brain tumor. Primary brain tumor is the condition when the tumor is formed in the brain and tended to stay there while the metastatic brain tumor is the tumor that is formed elsewhere in the body and spread through the brain.

Form of malignant brain tumor that typically needs surgery and fast-growing Radiation therapy and poor prediction for survival.
Imaging of magnet resonance (MRI) is a key tool for *diagnosis*, *surveillance*, and *surgery of brain tumors*.
It is appeared to be a solid mass when it diagnosed with diagnostic medical imaging techniques.

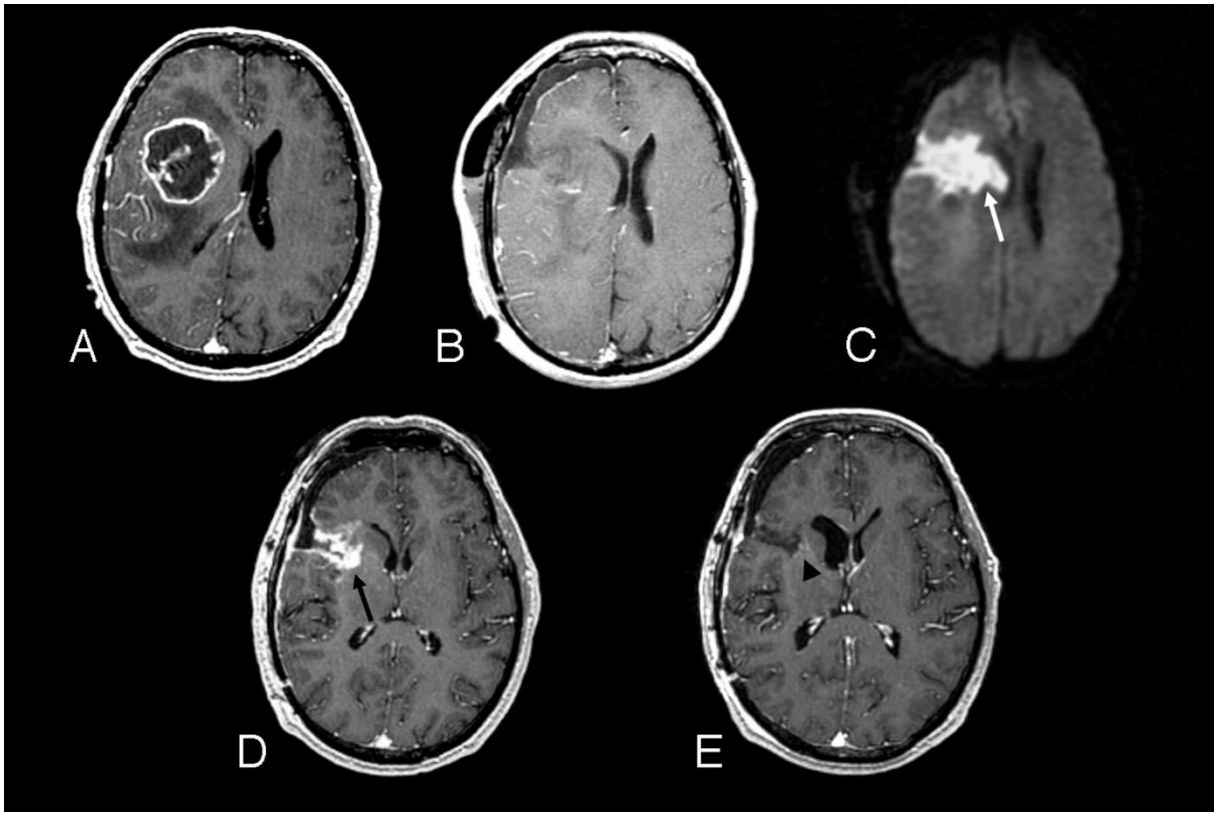

Figure 3.2.2 Type of the tumor

The symptom having of brain tumor depends on the **location**, **size** and **type of the tumor**. It occurs when the tumor compressing the surrounding cells and gives out pressure.

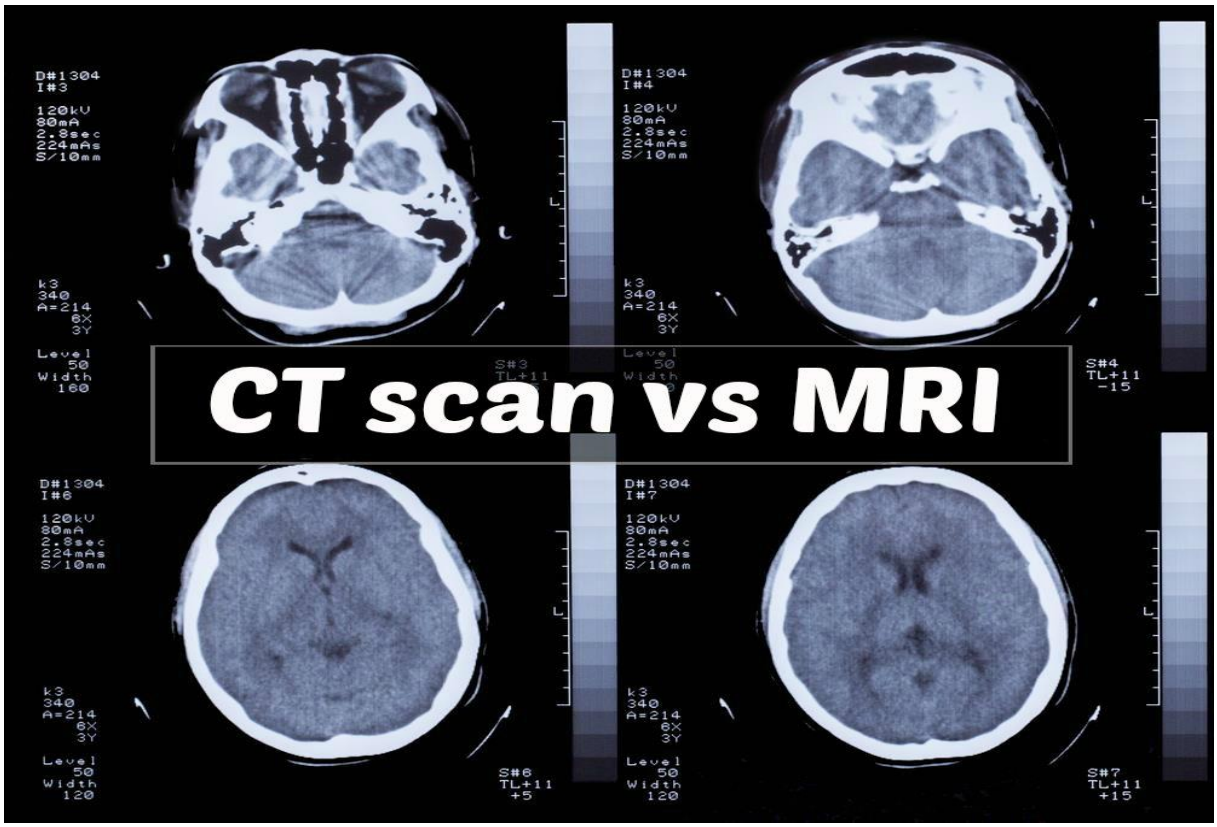

Figure 3.2.3 CT Scan vs MRI

The reasons for selecting CT images upon MRI images are as follows:

1. CT is much faster than MRI, making it the study of choice in cases of trauma and other acute neurological emergencies. CT can be obtained at considerably less cost than MRI.

2. CT is less sensitive to patient motion during the examination.

3. The imaging can be performed much more rapidly, so CT may be easier to perform in claustrophobic or very heavy patients.

4. CT can be performed at no risk to the patient with implantable medical devices, such as cardiac pacemakers, ferromagnetic vascular clips and nerve stimulators.

Figure 3.2.4 CT brain images

The focus of this project is :

CT brain images' tumor extraction and its representation in simpler form such that it is understandable by everyone.
Humans tend to understand colored images better than black and white images, thus, we are using colors to make the representation simpler enough to be understood by the patient along with the medical staff.
Contour plot and c-label of tumor and its boundary is programmed to give 3D visualization from 2D image using different colors for different levels of intensity.

A user-friendly GUI is also created which helps medical staff to attain the above objective without getting into the code.

[3-3- Aim & Objective]

The main reason for detection of brain tumors is to provide aid to clinical diagnosis. The aim is to provide an algorithm that guarantees the presence of a tumor by combining several procedures to provide a foolproof method of tumor detection in CT brain images.

The methods utilized are filtering, contrast adjustment, negation of an image, image subtraction, erosion, dilation, threshold, and outlining of the tumor. The objective of this work is to bring some useful information in simpler form in front of the users, especially for the medical staff treating the patient.

The resultant image will be able to provide information like size, dimension and position of the tumor, aims to evaluate state-of-the-art methods for the segmentation of brain tumors by providing a 3D MRI dataset with ground truth tumor segmentation labels annotated by physicians. Images better with the help of different colors for different levels of intensity, giving 3D visualization from a 2D image.

[4- Overall Description]

[4-1- Project Features]

The subtraction of background and its negative from different sets of images results in extracted tumor image.
Plotting contour and c-label of the tumor and its boundary provides us with information related to the tumor that can help in a better visualization in diagnosing cases.
This process helps in identifying the ***size***, ***shape***, and ***position of the tumor***.
It helps the medical staff as well as the patient to understand the seriousness of the tumor with the help of different color-labeling for different levels of elevation.
A GUI for the contour of the tumor and its boundary can provide information to the medical staff on the click of user choice buttons.

[4-2- Constraints]

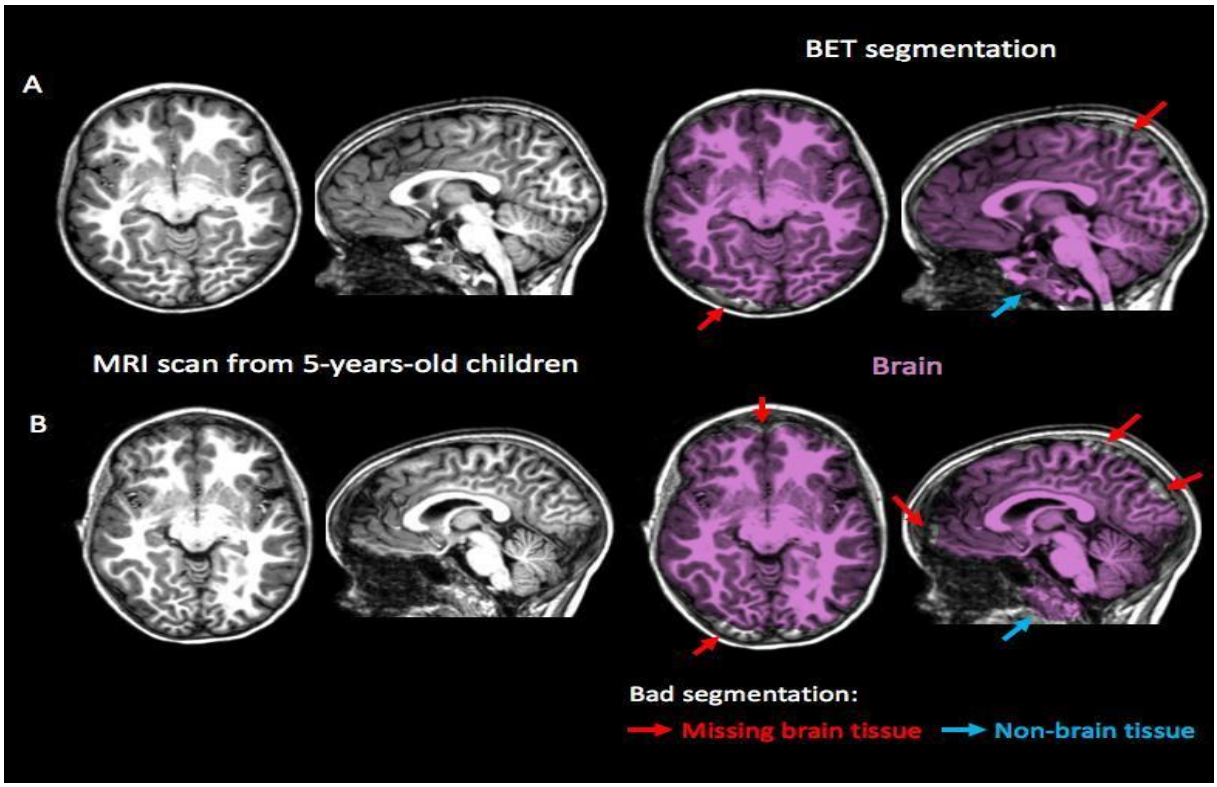

Figure 4.2.1 Segmentation

Robust MRI brain segmentation is a hard task, especially in presence of tumor that affects the normal brain tissue texture and intensity features. Brain tumors can take various shapes, positions and intensity levels. Brain segmentation is an essential preprocessing task to develop a good tumor detection and segmentation algorithm.

The structure of human brain is complicated, usually overlapping, uncertain, vague, and indiscernible in nature. Moreover, brain magnetic resonance images (MRI) often suffer from outliers, noise and other artifacts.

[5-Requirements Project]

[5-1-Tools & Prerequisites]

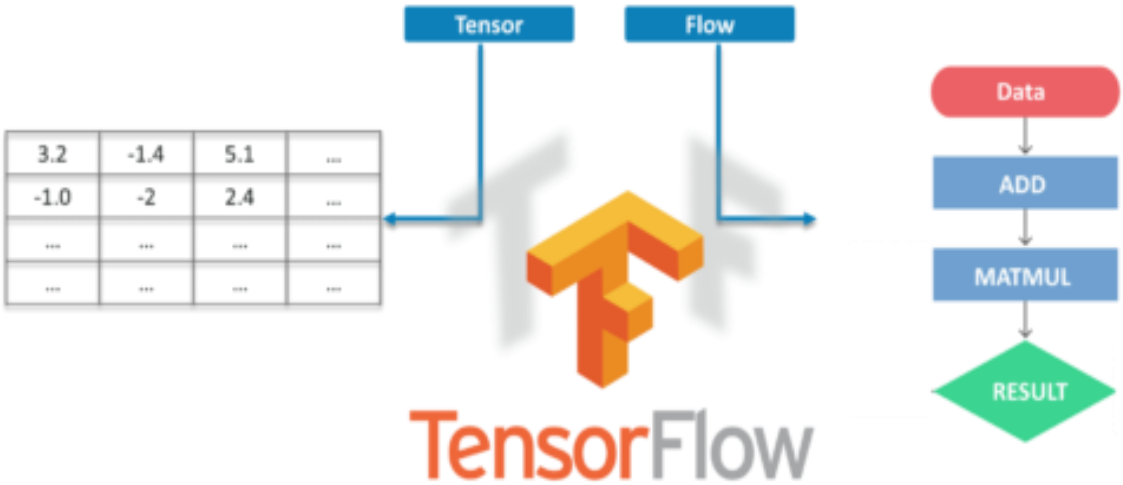

Figure 5.1.1 TensorFlow

Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Figure 5.1.2 TensorFlow Features

Powerful experimentation for research

A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.
TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.
TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.
TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units).
TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

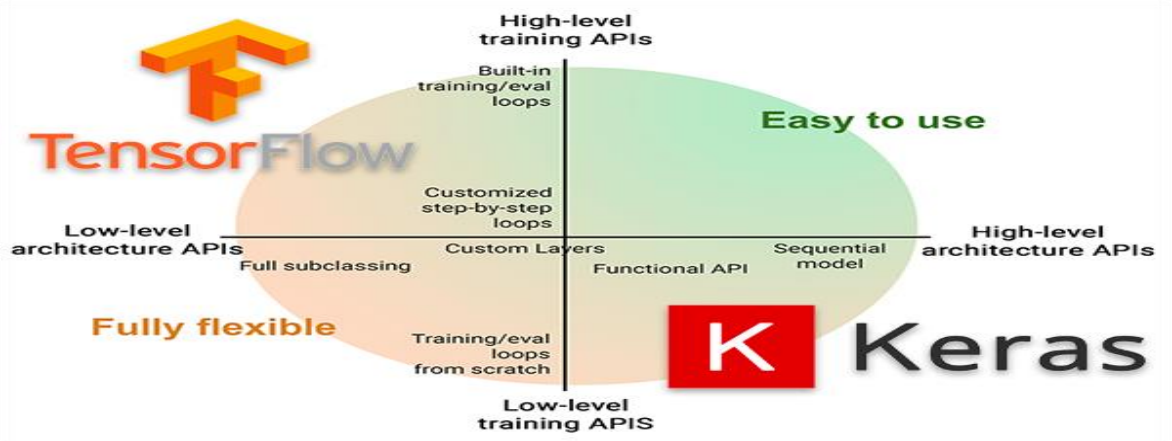

Figure 5.1.3 TensorFlow vs Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Keras is TensorFlow's high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production, with three key advantages:

- **User-friendly**

  Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- **Modular and Composable**

  Keras models are made by connecting configurable building blocks together, with few restrictions.
- **Easy to extend**

  Write custom building blocks to express new ideas for research. Create new layers, metrics, loss functions, and develop state-of-the-art models.

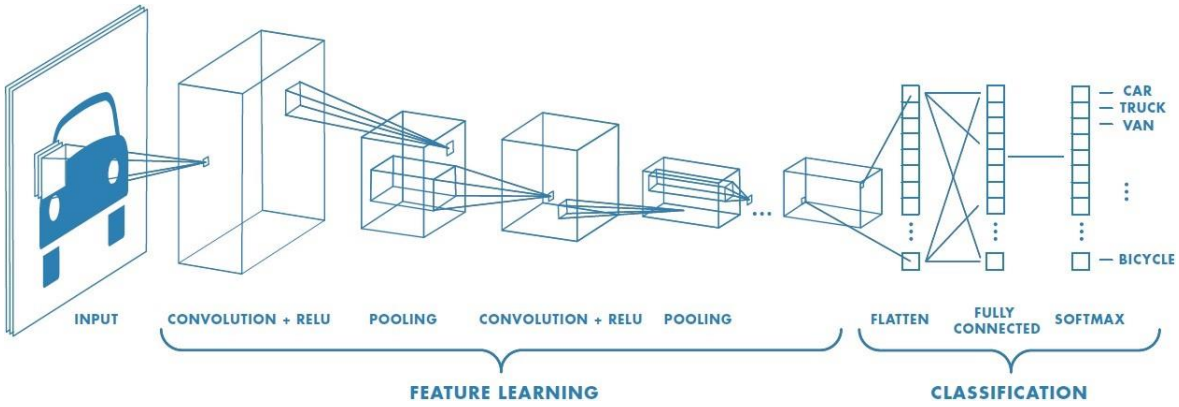

Figure 5.1.4

Image Embeddings + CNN = Image Classification

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used. CNN image classifications take an input image, process it and classify it under certain categories. Computers sees an input image as 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). E.g., 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.

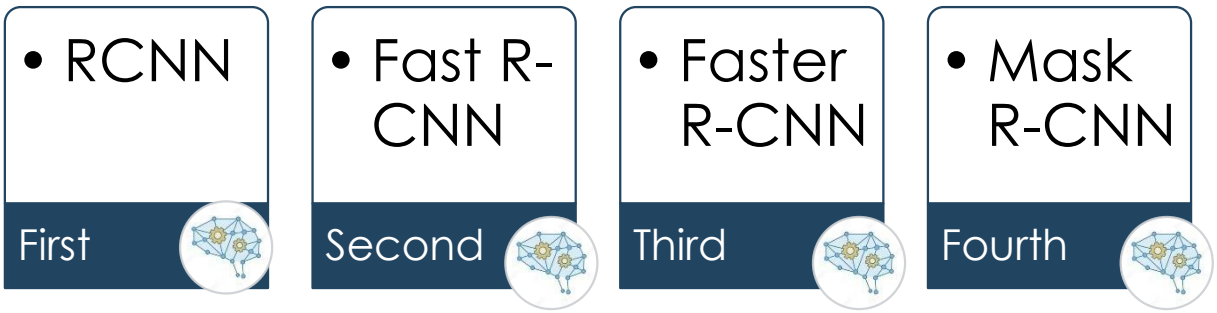

Figure 5.1.5 RCNN vs Fast RCNN vs Faster vs Mask

Computer vision is an interdisciplinary field that has been gaining huge amounts of traction in the recent years (since CNN) and self-driving cars have taken center stage. Another integral part of computer vision is object detection. Object detection aids in pose estimation, vehicle detection, surveillance etc. The difference between object detection algorithms and classification algorithms:
is that in detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image. Also, you might not necessarily draw just one bounding box in an object detection case, there could be many bounding boxes representing different objects of interest within the image and you would not know how many beforehand. The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable — not constant, this is because the number of occurrences of the objects of interest is not fixed. A naive approach to solve this problem would be to take different regions of interest from the image, and use a CNN to classify the presence of the object within that region.

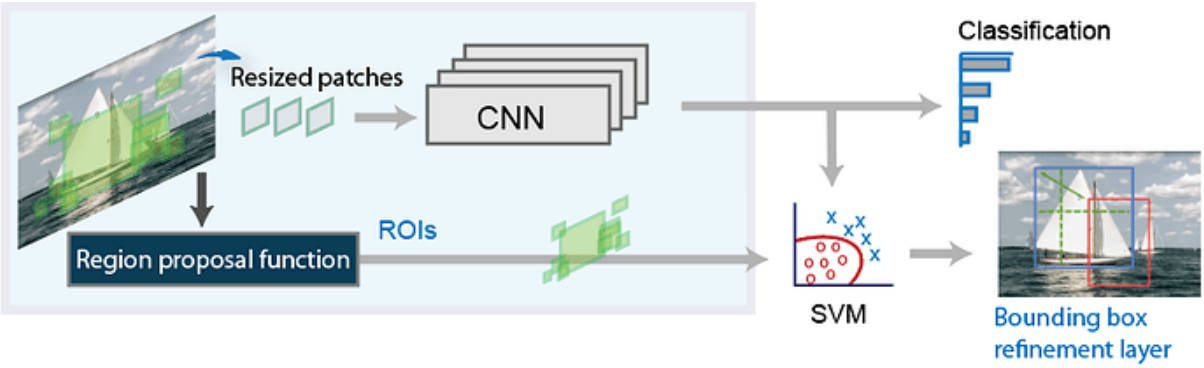

Figure 5.1.6 RCNN

RCNN

Proposed a method where we use selective search to extract just 2000 regions from the image and he called them region proposals.
Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions.
These 2000 region proposals:

- generated using the selective search algorithm.
- warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output.

The CNN acts as a **feature extractor** and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal.
In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box.
For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could’ve been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.

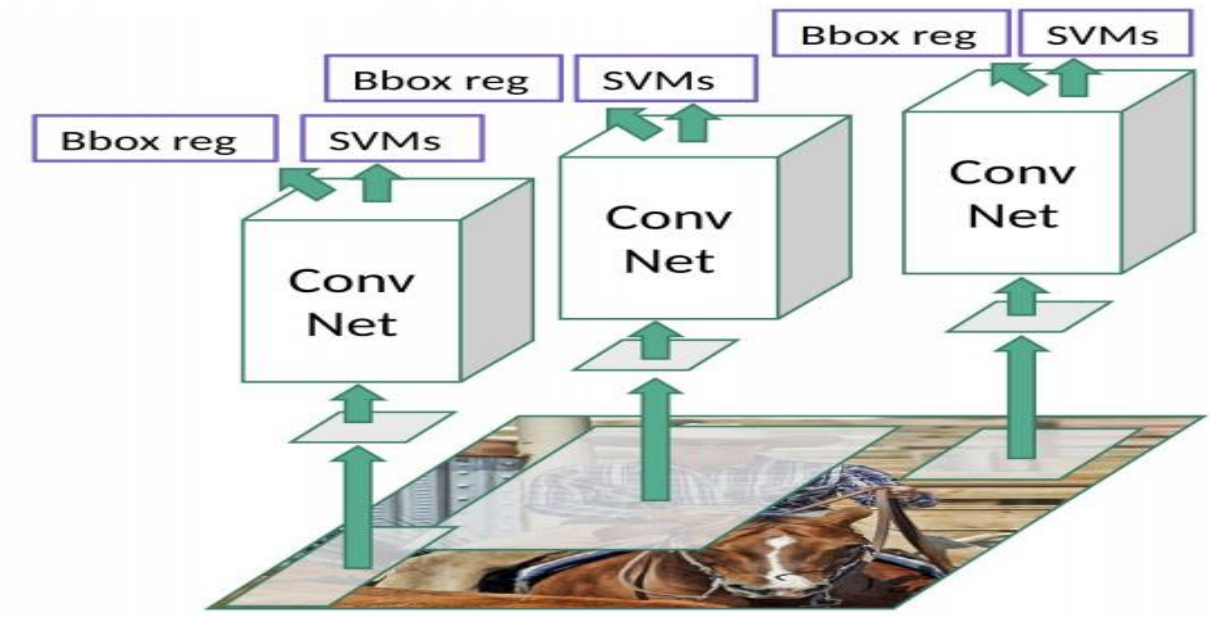

Figure 5.1.7 Problems with R-CNN

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

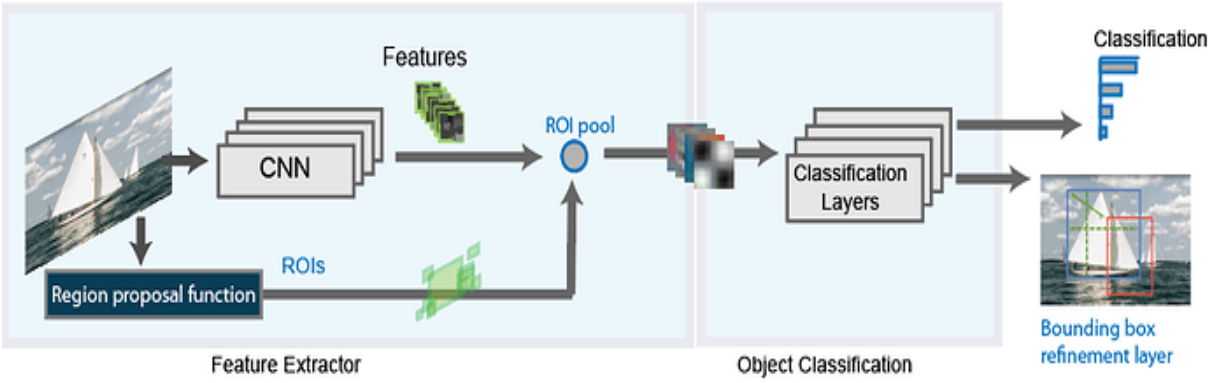

Figure 5.1.8 Fast RCNN

Fast R-CNN

solved some of the drawbacks of R-CNN to build a faster object detection algorithm.
The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
From the convolutional feature map:

- we identify the region of proposals,
- warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.
The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.

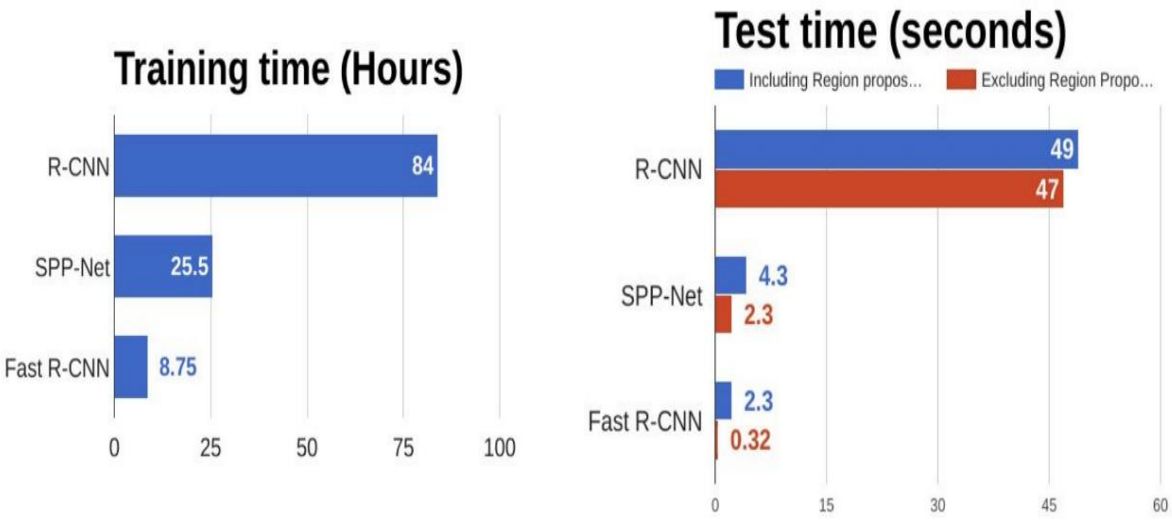

Figure 5.1.9 Training vs Test RCNN

From the above graphs, you can infer that Fast R-CNN is significantly faster in training and testing sessions over R-CNN. When you look at the performance of Fast R-CNN during testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals. Therefore, region proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

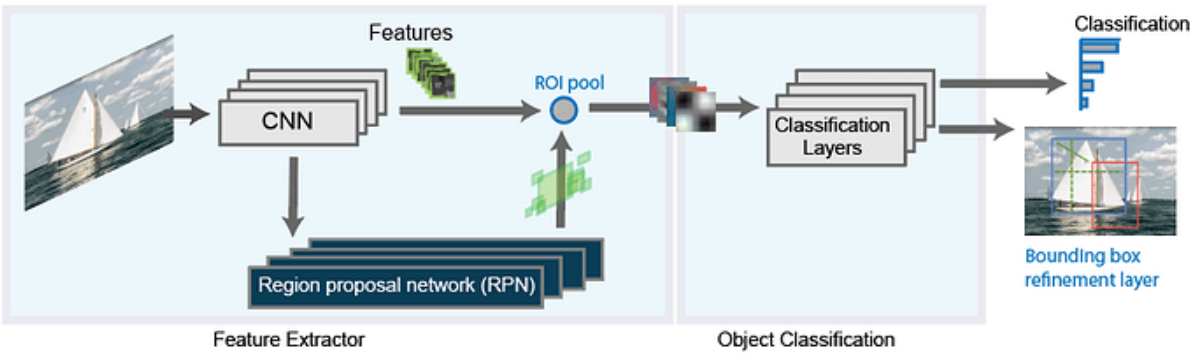

Figure 5.1.10 Faster R-CNN

Faster R-CNN

Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals.
Selective search is a slow and time-consuming process affecting the performance of the network. Therefore, Shaoqing Ren et al. Came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

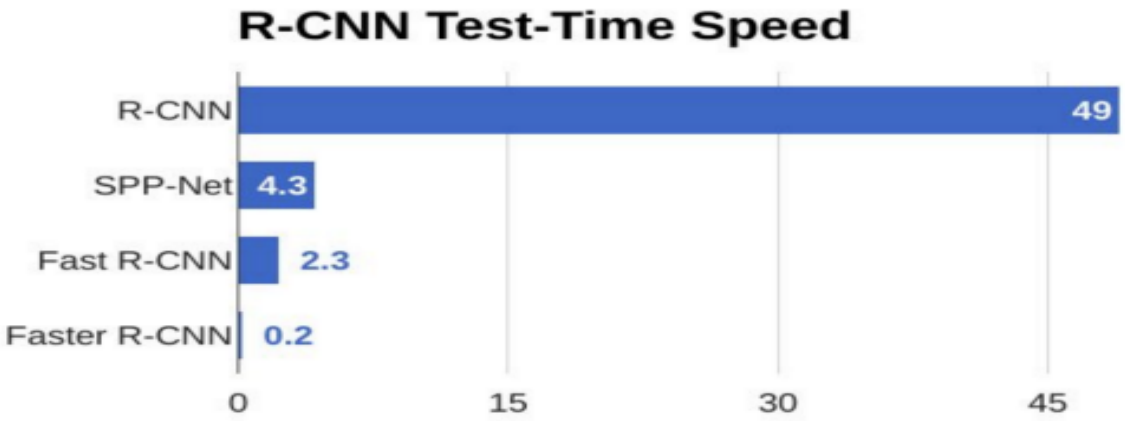

Figure 5.1.11 RCNN Test-Time Speed

From the above graph, you can see that Faster R-CNN is much faster than it's predecessors. Therefore, it can even be used for real-time object detection. Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a roi pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

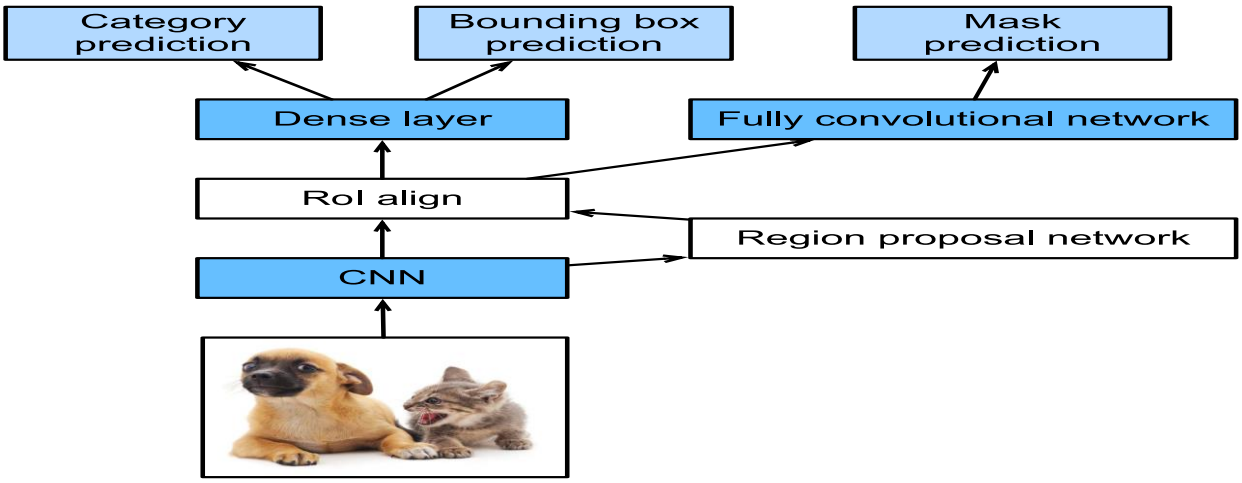

Figure 5.1.12 Mask RCNN

Mask R-CNN

is a two-stage framework:

- the first stage: scans the image and generates proposals (areas likely to contain an object).
- the second stage: classifies the proposals and generates bounding boxes and masks. Both stages are connected to the backbone structure.

Summary

1- An R-CNN model selects several proposed regions and uses a CNN to perform forward computation and extract the features from each proposed region. It then uses these features to predict the categories and bounding boxes of proposed regions.

2-Fast R-CNN improves on the R-CNN by only performing CNN forward computation on the image as a whole. It introduces an RoI pooling layer to extract features of the same shape from RoIs of different shapes.

3- Faster R-CNN replaces the selective search used in Fast R-CNN with a region proposal network. This reduces the number of proposed regions generated, while ensuring precise object detection.

4-Mask R-CNN uses the same basic structure as Faster R-CNN, but adds a fully convolution layer to help locate objects at the pixel level and further improve the precision of object detection.

Figure 5.1.13 Python Flask

Flask API is a drop-in replacement for Flask that provides an implementation of browsable APIs similar to what Django REST framework provides. It gives you properly content negotiated-responses and smart request parsing. Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

Figure 5.1.14 Python

Python is a high-level, general-purpose programming language with a reference implementation that compiles source code into bytecode before being executed on a process virtual machine.
Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Figure 5.1.15 Android

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance, with the main contributor and commercial marketer being Google. Android's source code has been used as the basis of different ecosystems, most notably that of Google which is associated with a suite of proprietary software called Google Mobile Services (GMS),that frequently comes pre-installed on said devices. This includes core apps such as Gmail, the digital distribution platform Google Play and associated Google Play Services development platform, and usually apps such as the Google Chrome web browser. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google.
Other competing Android ecosystems include Amazon.com's Fire OS, or LineageOS.

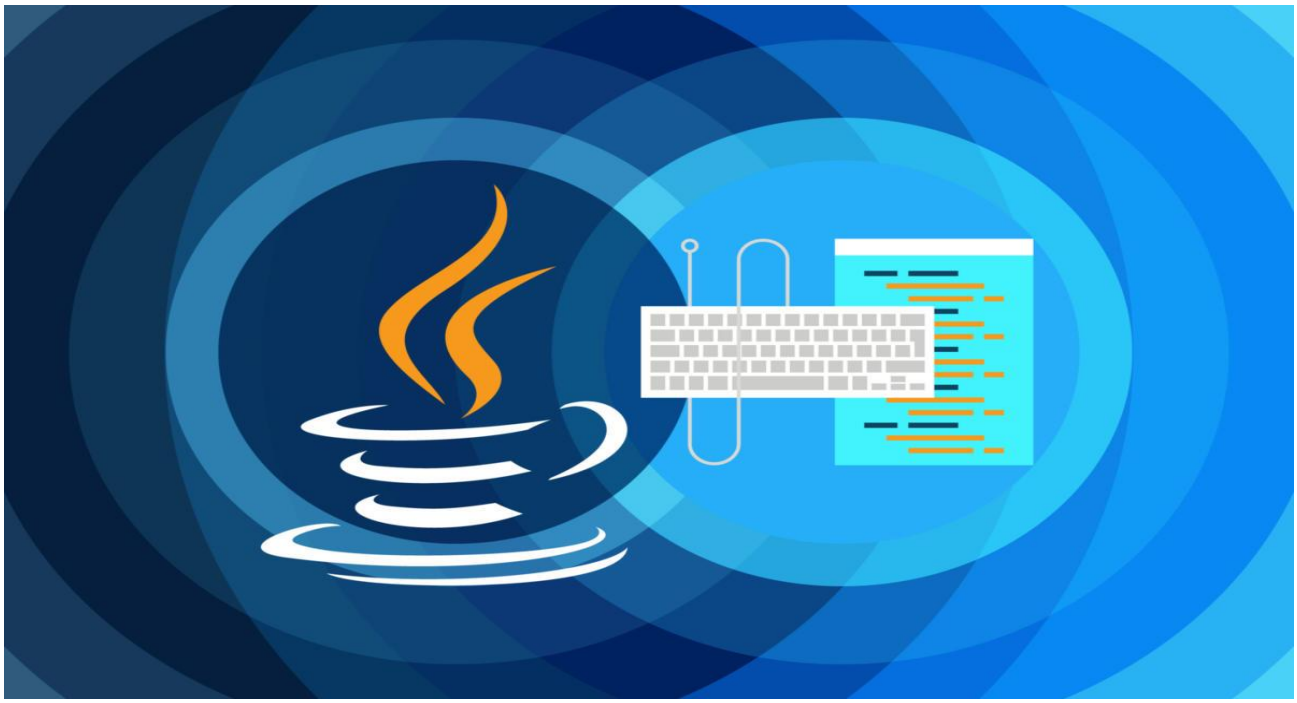

Figure 5.1.16 Java

Java Indonesia, bordered by the Indian Ocean on the south and the Java Sea on the north. With a population of over 141 million (Java only) or 145 million (including the inhabitants of its surrounding islands), Java has 56.7 percent of the Indonesian population and is the world's most populous island.The Indonesian capital city, Jakarta, is located on its northwestern coast. Much of the well-known part of Indonesian history took place on Java. It was the centre of powerful Hindu-Buddhist empires, the Islamic sultanates, and the core of the colonial Dutch East Indies. Java was also the center of the Indonesian struggle for independence during the 1930s and 1940s. Java dominates Indonesia politically, economically and culturally. Four of Indonesia's eight UNESCO world heritage sites are located in Java: Ujung Kulon National Park, Borobudur Temple, Prambanan Temple, and Sangiran Early Man Site.

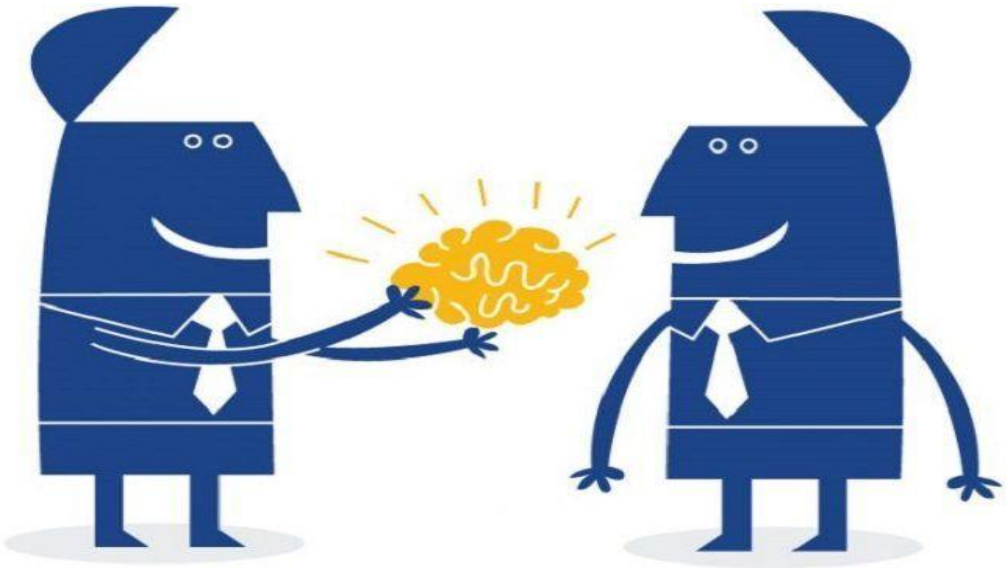

Figure 5.1.17 transfer learning

very few people train an entire Convolutional Network from scratch(with random initialization), because it is relatively rare tohave a dataset of sufficient size. Instead, it is common to pretrain a ConvNe t on a very large dataset (e.g. ImageNet, which contains 1.2 million im ages with 1000 categories), and then use the ConvNet either as an initi alization or a fixed feature extractor for the task of interest.
We always hear that we **do not have to reinvent the wheel**. Well, this is always true. Why do not we work and climb over the giant shou lders? Why do not we build something even if we change it a bit? Well, that's not a theft. In fact, everything on the Internet without a lic ense is open source. You can deal with a simple modification that you can get on your next research paper, but the purpose is to understand what has been completed and not just use it.
These two major Transfer learning scenarios look as follows:
Finetuning the convnet: Instead of random initializaion, we initialize t he network with a pretrained network, like the one that is trained on i magenet 1000 dataset. Rest of the training looks as usual.
ConvNet as fixed feature extractor: Here, we will freeze the weig hts for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with rando m weights and only this layer is trained.

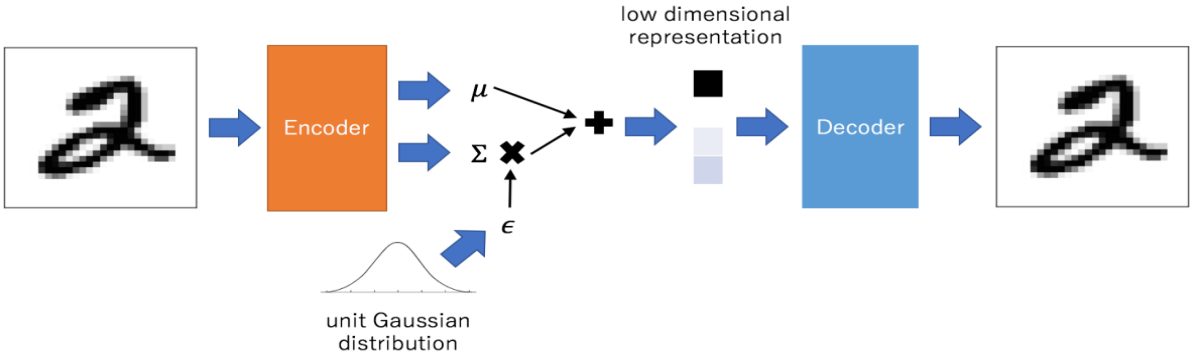

Figure 5.1.18 autoencoder

Variational Autoencoder Models make strong assumptions concerning the distribution of latent variables.
They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes estimator.
It assumes that the data is generated by a directed graphical model and that the encoder is learning an approximation to the posterior distribution where Φ and θ denote the parameters of the encoder (recognition model) and decoder (generative model) respectively.
The probability distribution of the latent vector of a variational autoencoder typically matches that of the training data much closer than a standard autoencoder.

Advantages-

- It gives significant control over how we want to model our latent distribution unlike the other models.
- After training you can just sample from the distribution followed by decoding and generating new data.

Drawbacks

- When training the model, there is a need to calculate the relationship of each parameter in the network with respect to the final output loss using a technique known as backpropagation. Hence, the sampling process requires some extra attention.

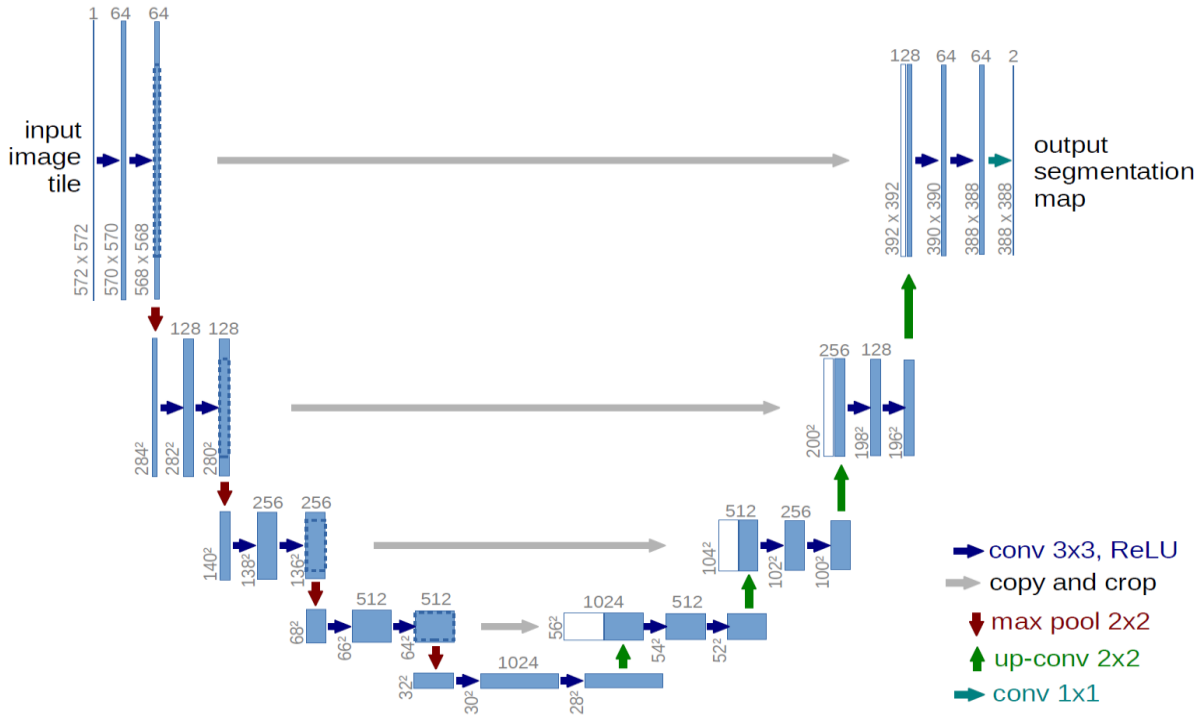

Figure 5.1.19 U-Net

U-Net is a convolutional neural network that was developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg, Germany.
The network is based on the fully convolutional network and its architecture was modified and extended to work with fewer training images and to yield more precise segmentations.
Segmentation of a 512 × 512 image takes less than a second on a modern GPU. The network consists of a contracting path and an expansive path, which gives it the u-shaped architecture.
The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (Relu) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path.

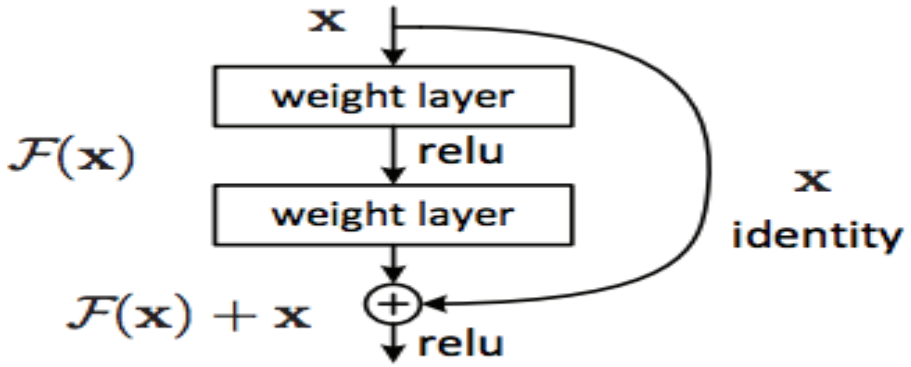

Figure 2. Residual learning: a building block.

Figure 5.1.20 ResNet

A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or shortcuts to jump over some layers.

Typical ResNet models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between.An additional weight matrix may be used to learn the skip weights; these models are known as HighwayNets.Models with several parallel skips are referred to as DenseNets.In the context of residual neural networks, a non-residual network may be described as a plain network.

One motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights.

During training, the weights adapt to mute the upstream layer, and amplify the previously-skipped layer. In the simplest case, only the weights for the adjacent layer's connection are adapted, with no explicit weights for the upstream layer.

This works best when a single nonlinear layer is stepped over, or when the intermediate layers are all linear. If not, then an explicit weight matrix should be learned for the skipped connection (a HighwayNet should be used).

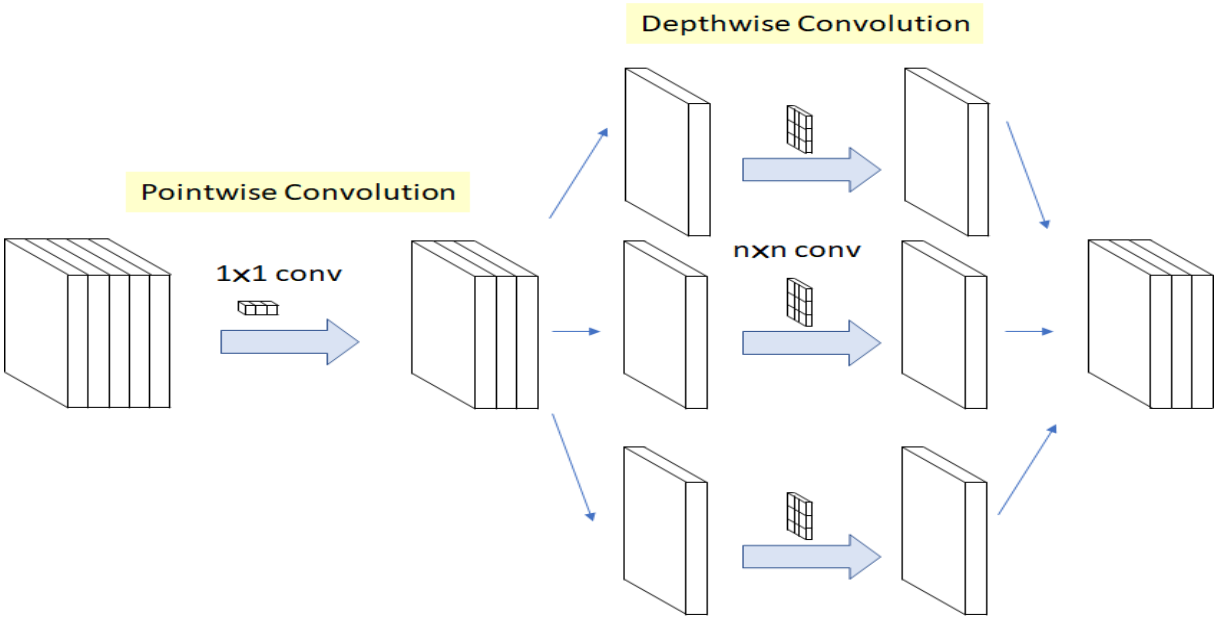

Figure 5.1.21 Xception

In Paper(Xception: Deep Learning with Depthwise Separable Convolutions),we name our proposed architecture Xception, which stands for “Extreme Inception”.

The Xception architecture has 36 convolutional layers forming the feature extraction base of the network.

In our experimental evaluation we will exclusively investigate image classification and therefore our convolutional base will be followed by a logistic regression layer. Optionally one may insert fully-connected layers before the logistic regression layer, which is explored in the experimental evaluation section.

The 36 convolutional layers are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules.

In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections.

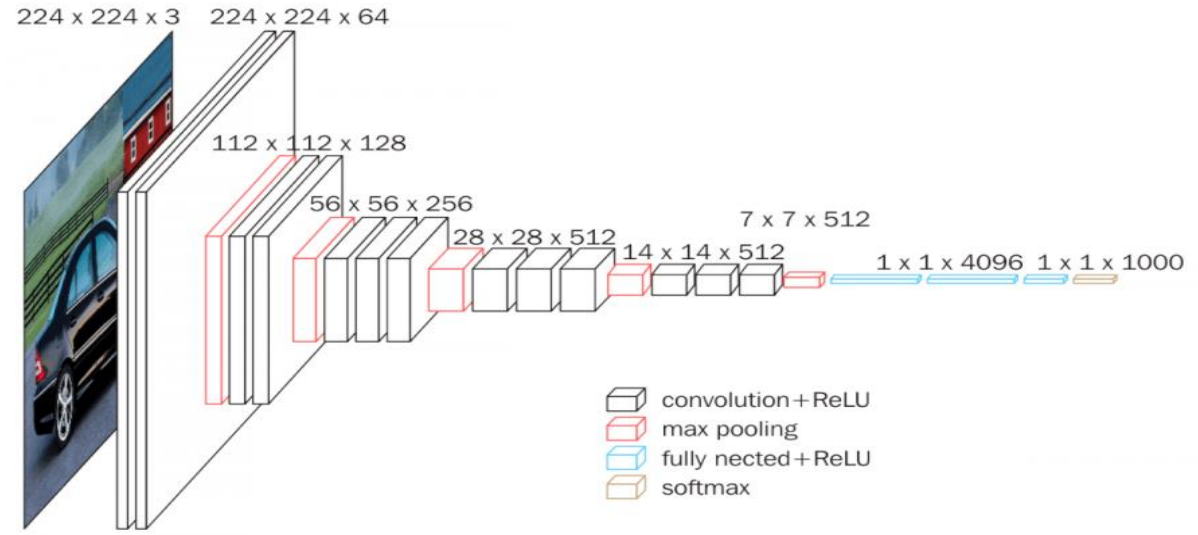

Figure 5.1.22 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

It was one of the famous model submitted to ILSVRC-2014.

It makes the improvement over AlexNet by replacing large kernel-sized filters with multiple 3×3 kernel-sized filters one after another.

VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

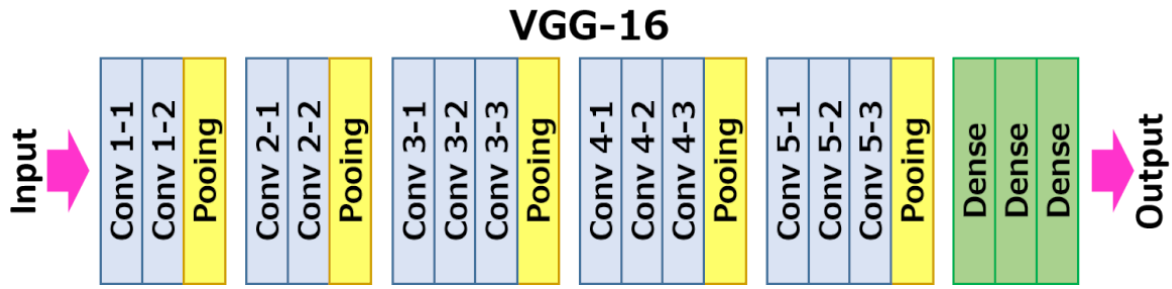

Figure 5.1.23 VGG16 layer

The input to cov1 layer is of fixed size 224 x 224 RGB image.

The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3.

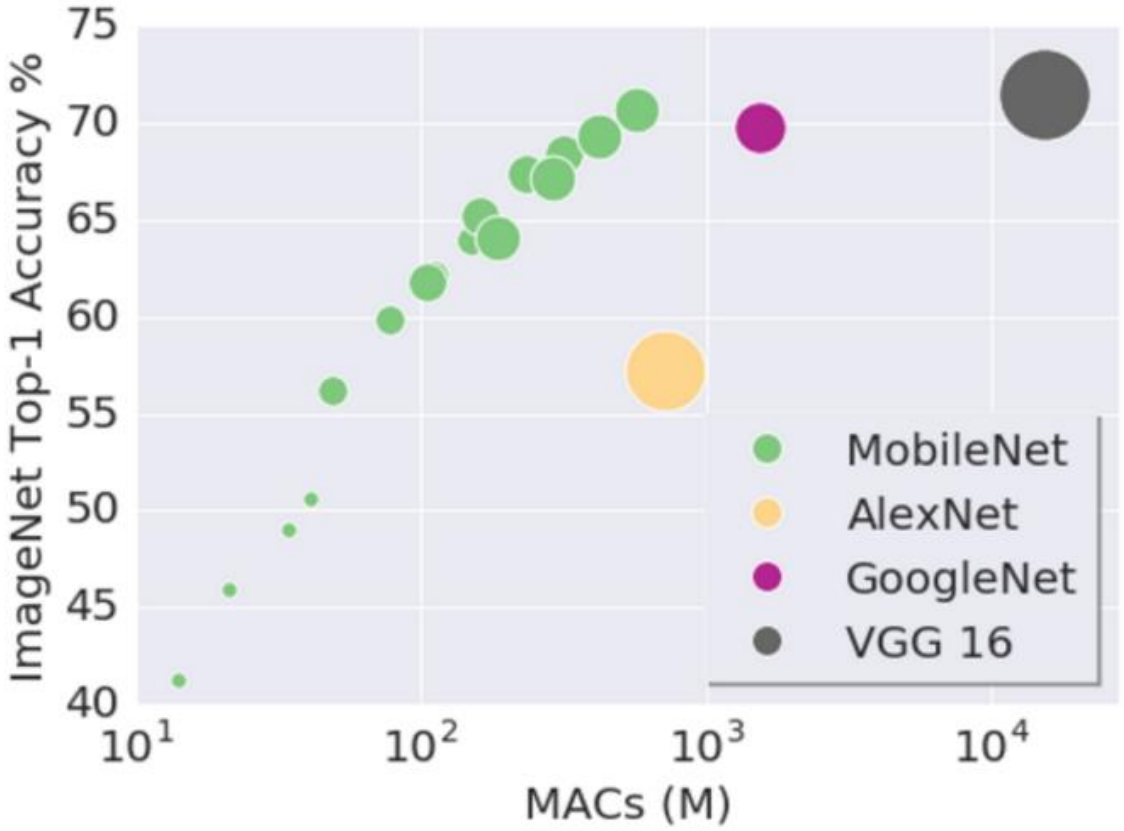

Figure 5.1.24 Mobile Nets

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how other popular large scale models, such as Inception, are used. MobileNets can be run efficiently on mobile devices with TensorFlow Mobile.MobileNets trade off between latency, size and accuracy while comparing favorably with popular models from the literature.

MobileNetsV1 VS MobileNetsV2

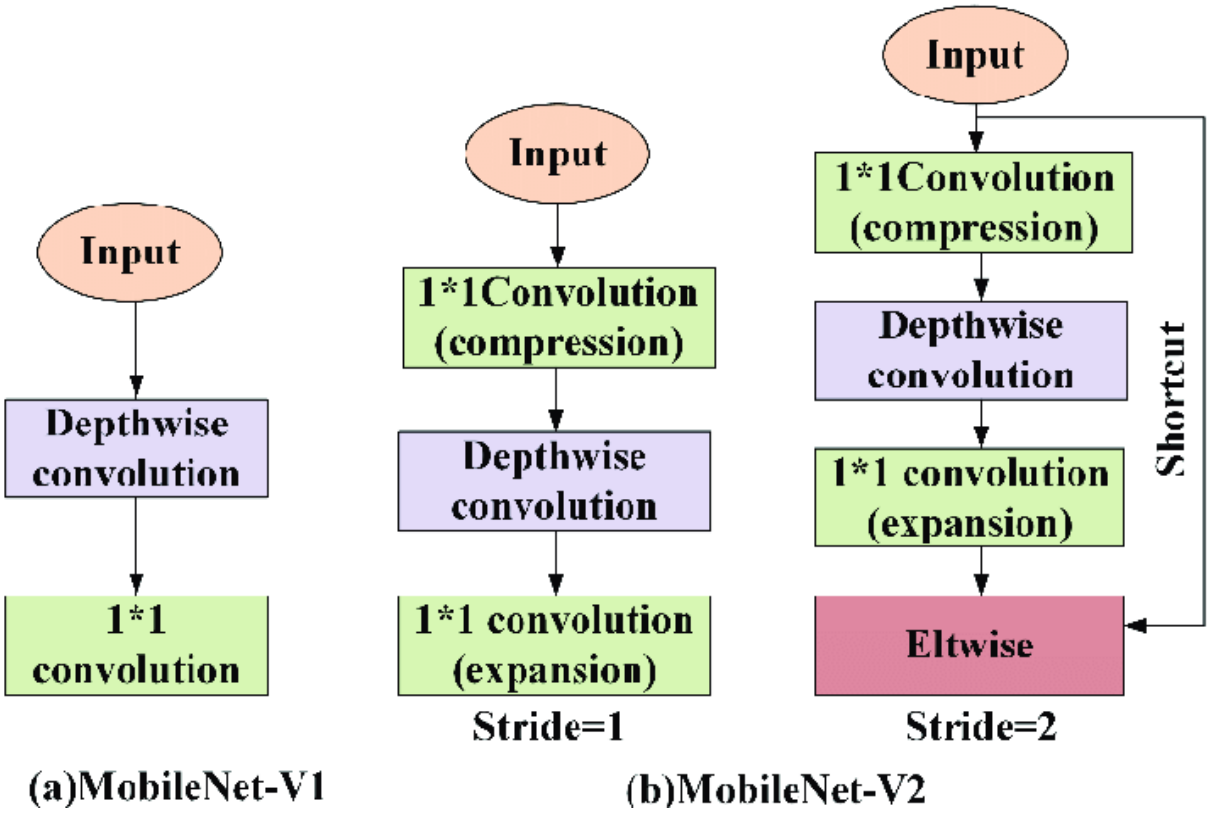

Figure 5.1.25 MoblileNetV1 vs MoblieNetV2

The idea behind V1 was to replace expensive convolutions with cheaper ones, even if it meant using more layers. That was a great success.

The main changes in the V2 architecture are the residual connections and the expand/projection layers.

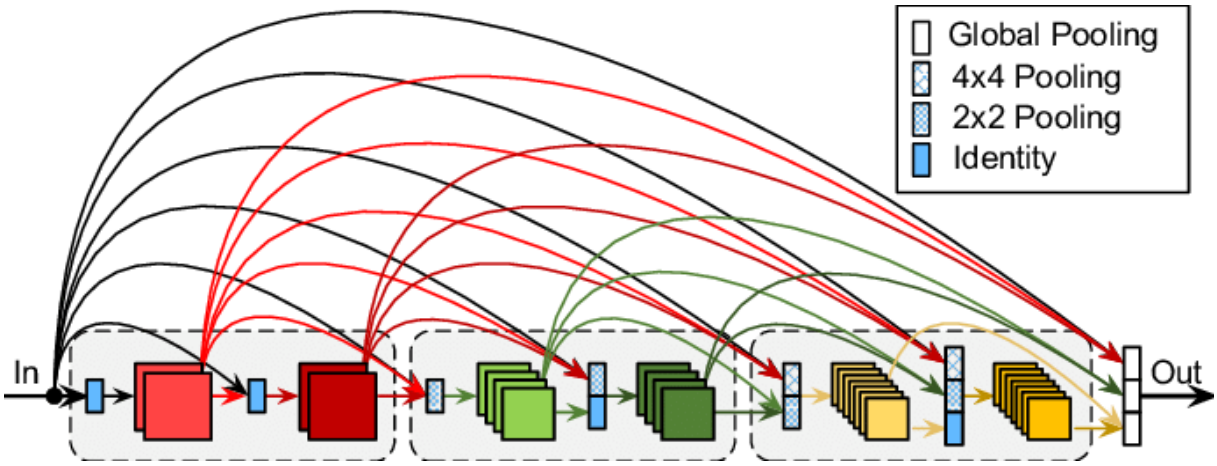

Figure 5.1.26 DenseNet

DenseNet architecture is new, it is a logical extension of ResNet. ResNet architecture has a fundamental building block (Identity) where you merge (additive) a previous layer into a future layer.
Reasoning here is by adding additive merges we are forcing the network to learn residuals (errors i.e. diff between some previous layer and current one). In contrast, DenseNet paper proposes concatenating outputs from the previous layers instead of using the summation.

DenseNet:

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion.
Whereas traditional convolutional networks with L layers have L connections — one between each layer and its subsequent layer — our network has L(L+1)/ 2 direct connections.
For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

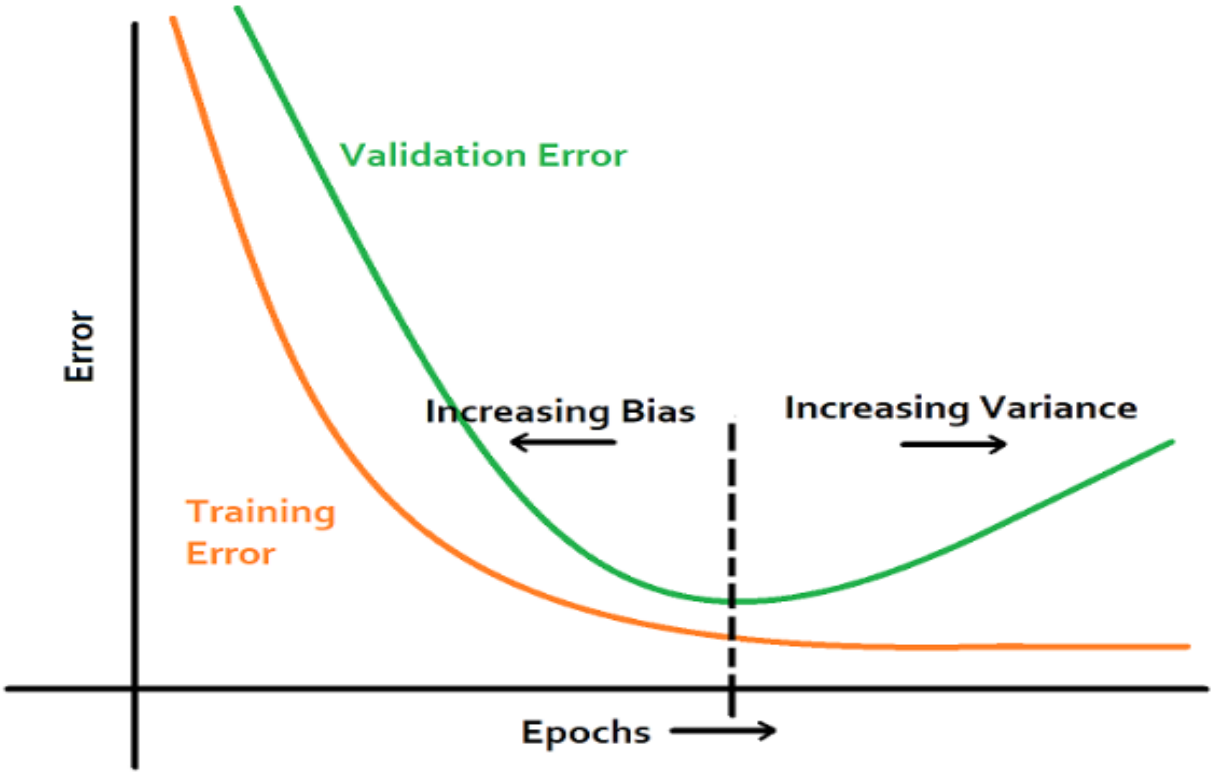

Figure 5.1.27 Validation Error

In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent.

Such methods update the learner so as to make it better fit the training data with each iteration.

Up to a point, this improves the learner's performance on data outside of the training set. Past that point, however, improving the learner's fit to the training data comes at the expense of increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit.

Early stopping rules have been employed in many different machine learning methods, with varying amounts of theoretical foundation.

An alternative approach is to train the model once for a large number of training epochs.During training, the model is evaluated on a holdout validation dataset after each epoch. If the performance of the model on the validation dataset starts to degrade (e.g. loss begins to increase or accuracy begins to decrease), then the training process is stopped.

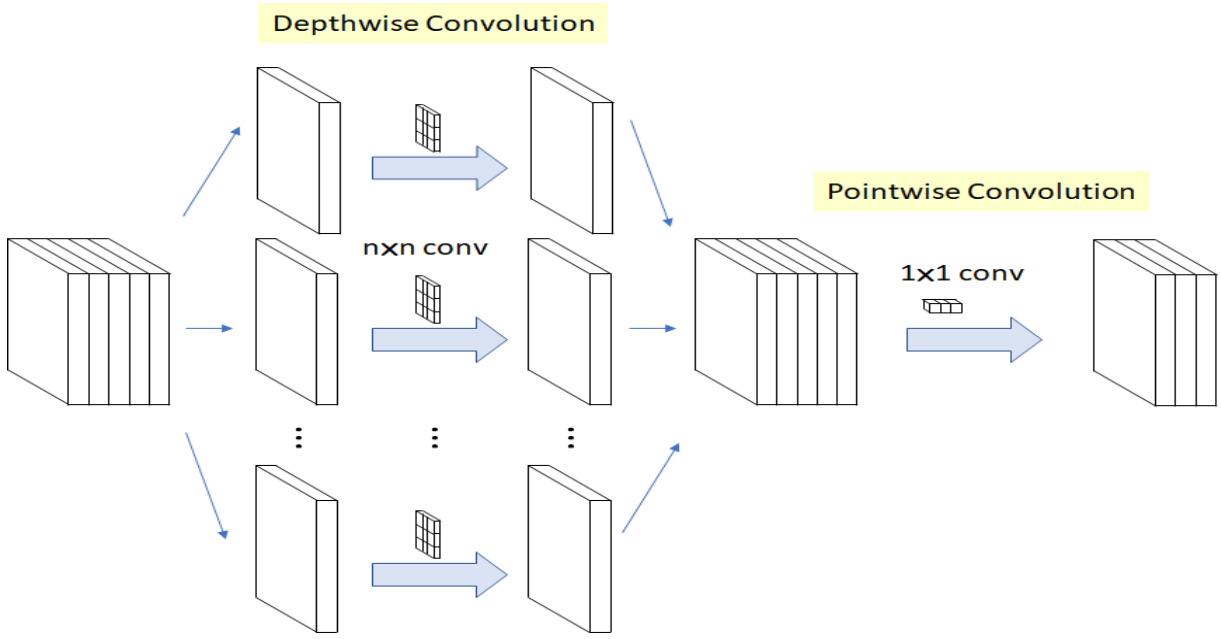

Figure 5.1.28 XCeption

How does XCeption work?

XCeption is an efficient architecture that relies on two main points :

- Depthwise Separable Convolution
- Shortcuts between Convolution blocks as in ResNet

Depthwise Separable Convolution

Depthwise Separable Convolutions are alternatives to classical convolutions that are supposed to be much more efficient in terms of computation time.

The limits of convolutions First of all, let's take a look at convolutions. Convolution is a really expensive operation. Let's illustrate this :

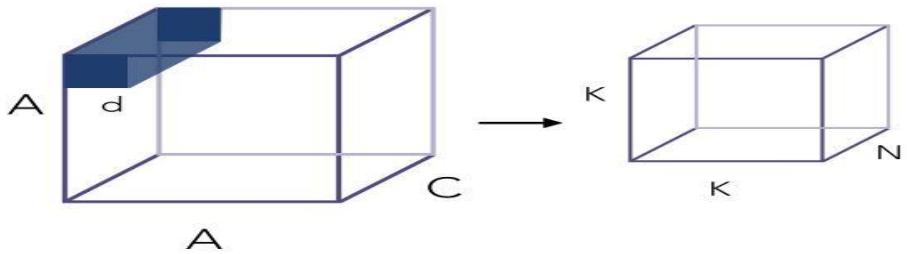

Figure 5.1.29 limits of convolutions

The input image has a certain number of channels C, say 3 for a color image. It also has a certain dimension A, say 100 * 100. We apply on it a convolution filter of size d*d, say 3*3.
To overcome the cost of such operations, depthwise separable convolutions have been introduced. They are themselves divided into 2 main steps:

- Depthwise Convolution
- Pointwise Convolution

Depthwise Convolution is a first step in which instead of applying convolution of size d×d×Cd×d×C, we apply a convolution of size d×d×1d×d×1. In other words, we don't make the convolution computation over all the channels, but only 1 by 1.

Pointwise convolution operates a classical convolution, with size 1×1×N1×1×N over the K×K×CK×K×C volume. This allows creating a volume of shape K×K×NK×K×N, as previously.
XCeption offers an architecture that is made of Depthwise Separable Convolution blocks + Maxpooling, all linked with shortcuts as in ResNet implementations.
Xception models remain expensive to train, but are pretty good improvements compared to Inception. Transfer learning brings part of the solution when it comes to adapting such algorithms to your specific task.

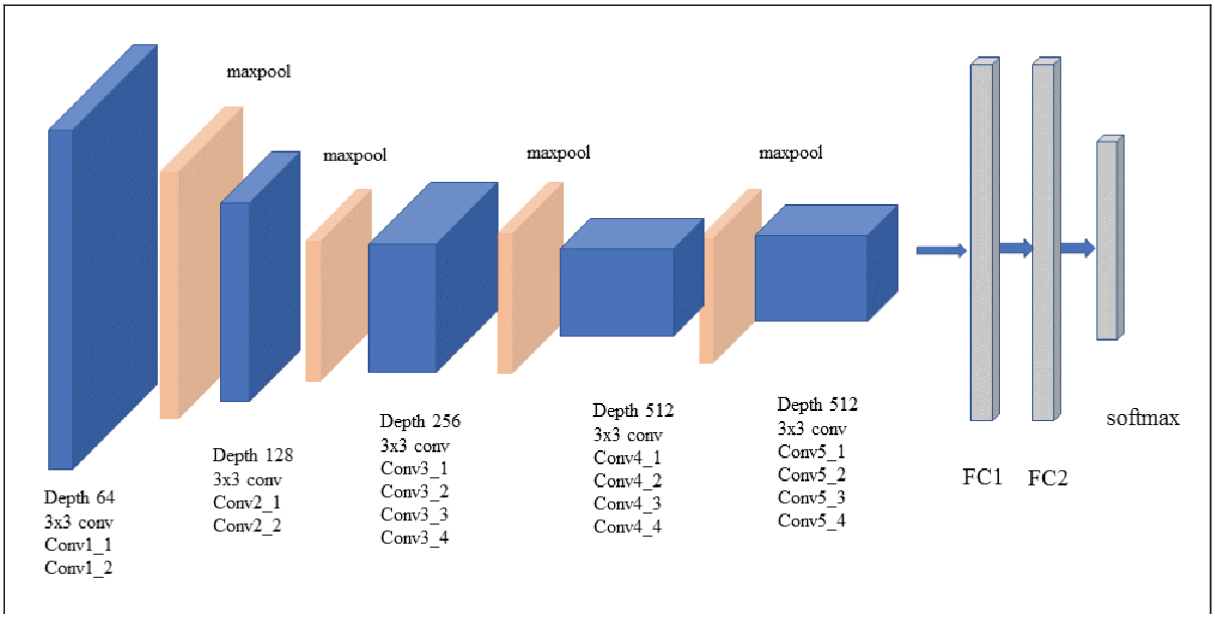

Figure 5.1.30 VGG19

VGG19 is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer).

There are other variants of VGG like VGG11, VGG16 and others. Alex Net came out in 2012 and it improved on the traditional Convolutional neural networks, So we can understand VGG as a successor of the Alex Net but it was created by a different group named as Visual Geometry Group at Oxford's and hence the name VGG, It carries and uses some ideas from it's predecessors and improves on them and uses deep Convolutional neural layers to improve accuracy. First of all, let’s explore what ImageNet is. It is an Image database consisting of 14,197,122 images organized according to the WordNet hierarchy. this is an initiative to help researchers, students and others in the field of image and vision research. ImageNet also hosts contests from which one was ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) which challenged researchers around the world to come up with solutions that yields the lowest top-1 and top-5 error rates (top-5 error rate would be the percent of images where the correct label is not one of the model’s five most likely labels).

The competition gives out a 1,000-class training set of 1.2 million images, a validation set of 50 thousand images and a test set of 150 thousand images and here comes the VGG Architecture, in 2014 it out-shined other state of the art models and is still preferred for a lot of challenging problems.

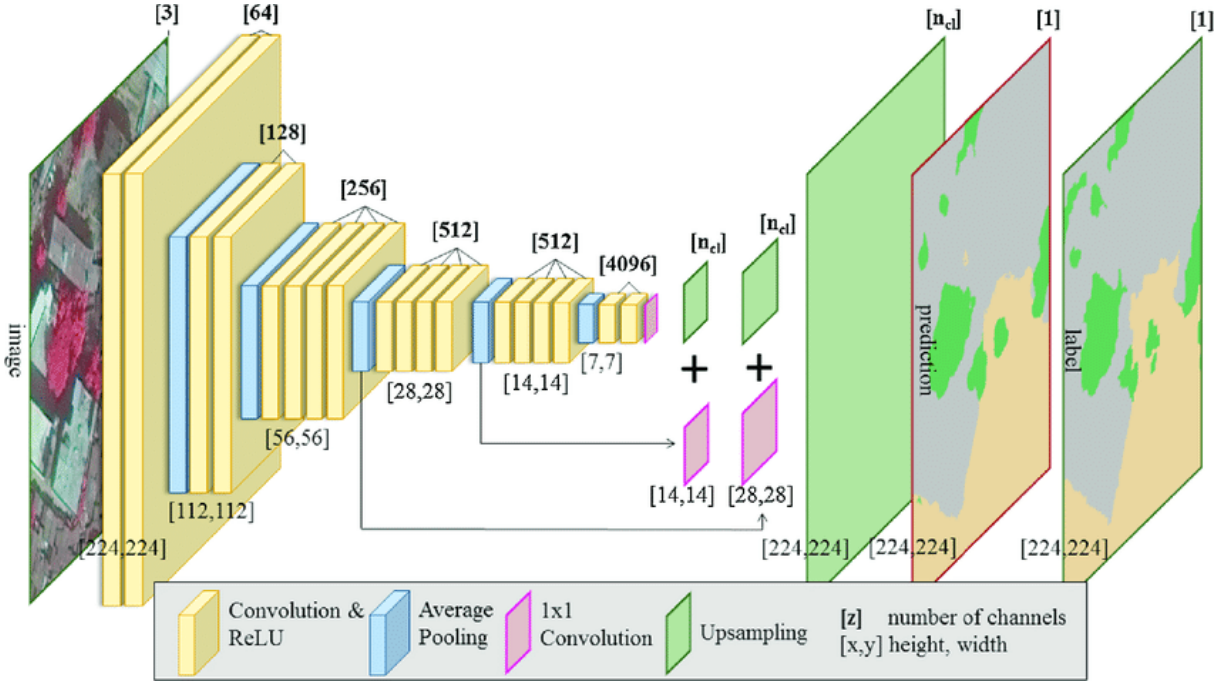

Figure 5.1.31 Architecture VGG19

Architecture

- A fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- spatial padding was used to preserve the spatial resolution of the image.
- max pooling was performed over a 2 * 2 pixel windows with sride 2.
- this was followed by Rectified linear unit(ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.

ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients.

AlexNet, the winner of ImageNet 2012 and the model that apparently kick started the focus on deep learning had only 8 convolutional layers, the VGG network had 19 and Inception or GoogleNet had 22 layers and ResNet 152 had 152 layers. In this blog we will code a ResNet-50 that is a smaller version of ResNet 152 and frequently used as a starting point for transfer learning.

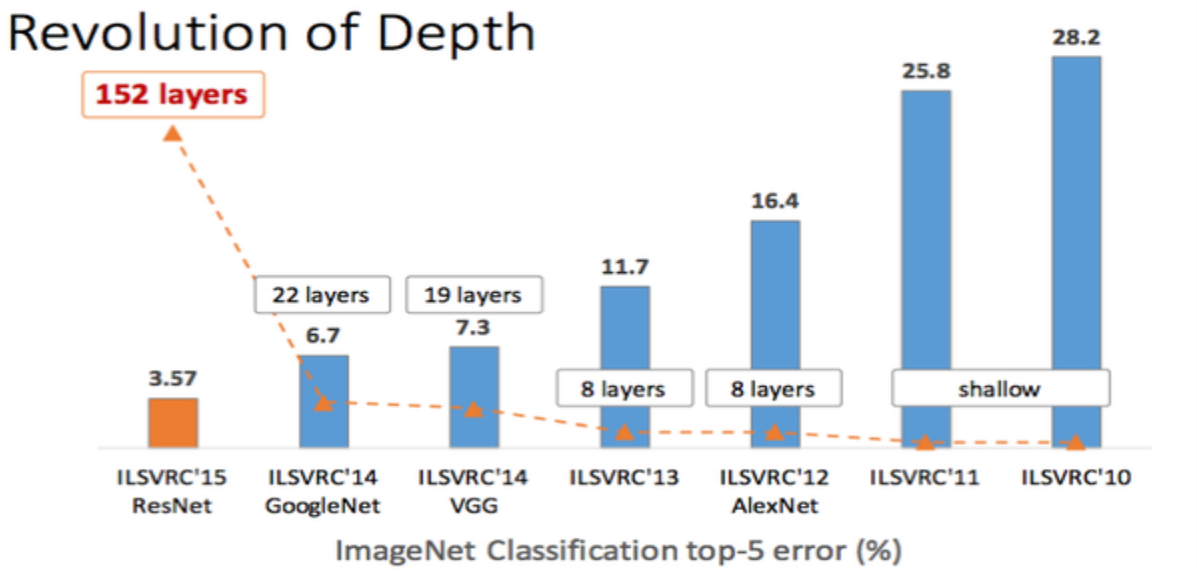

Figure 5.1.32 ImageNet

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

Skip Connection — The Strength of ResNet

ResNet first introduced the concept of skip connection. The diagram below illustrates skip connection. The figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. This is called skip connection.

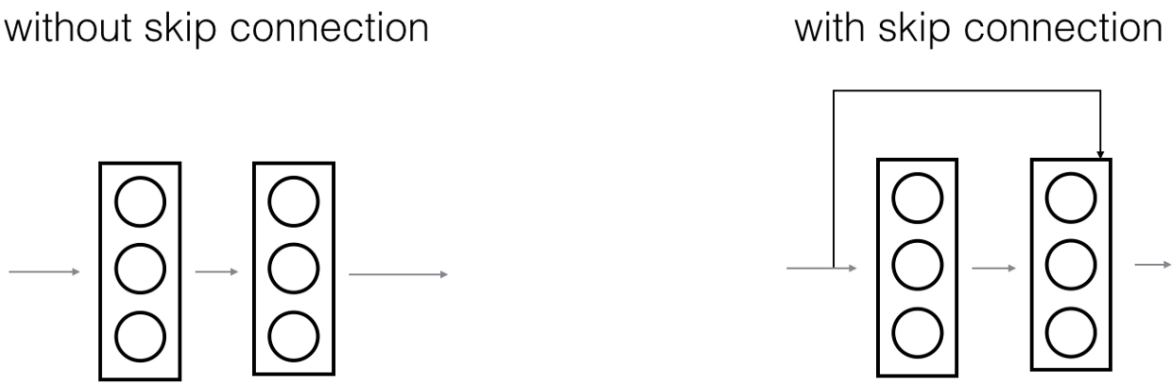

Figure 5.1.33 Skip Connect 1

if the convolution + batch norm operations are done in a way that the output shape is the same,then we can simply add them as shown below.

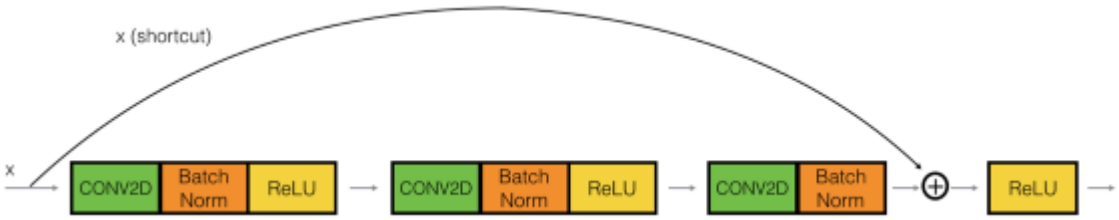

Figure 5.1.34 Skip Connect 2

Otherwise, the x_shortcut goes through a convolution layer chosen such that the output from it is the same dimension as the output from the convolution block as shown below:

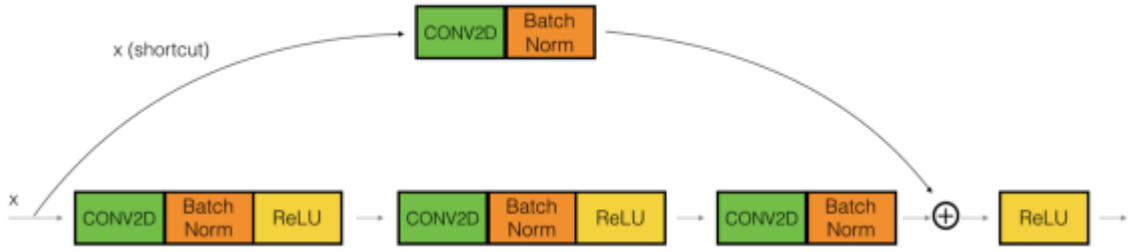

Figure 5.1.35 Skip Connect 3

Why do Skip Connections work?

This is an interesting question. I think there are two reasons why Skip connections work here:

1. They mitigate the problem of vanishing gradient by allowing this alternate shortcut path for gradient to flow through
2. They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse

Infect since ResNet skip connections are used in a lot more model architectures like the Fully Convolutional Network (FCN) and U-Net. They are used to flow information from earlier layers in the model to later layers. In these architectures they are used to pass information from the down sampling layers to the up-sampling layer

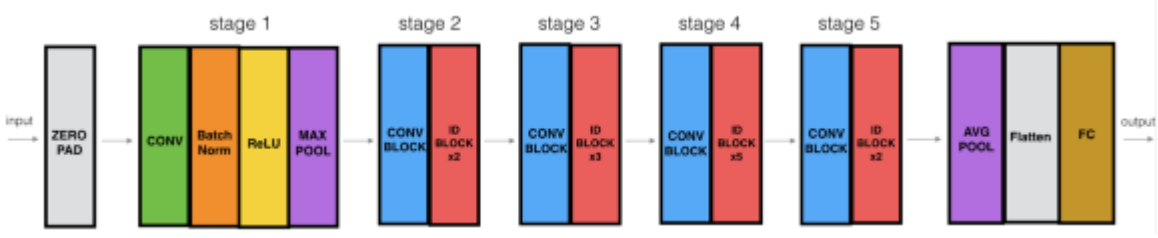

Figure 5.1.36 ResNet50

The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

Was ResNet Successful?

- Won 1st place in the ILSVRC 2015 classification competition with top-5 error rate of 3.57% (An ensemble model)
- Won the 1st place in ILSVRC and COCO 2015 competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation.
- Replacing VGG-16 layers in Faster R-CNN with ResNet-101. They observed a relative improvement of 28%
- Efficiently trained networks with 100 layers and 1000 layers also.

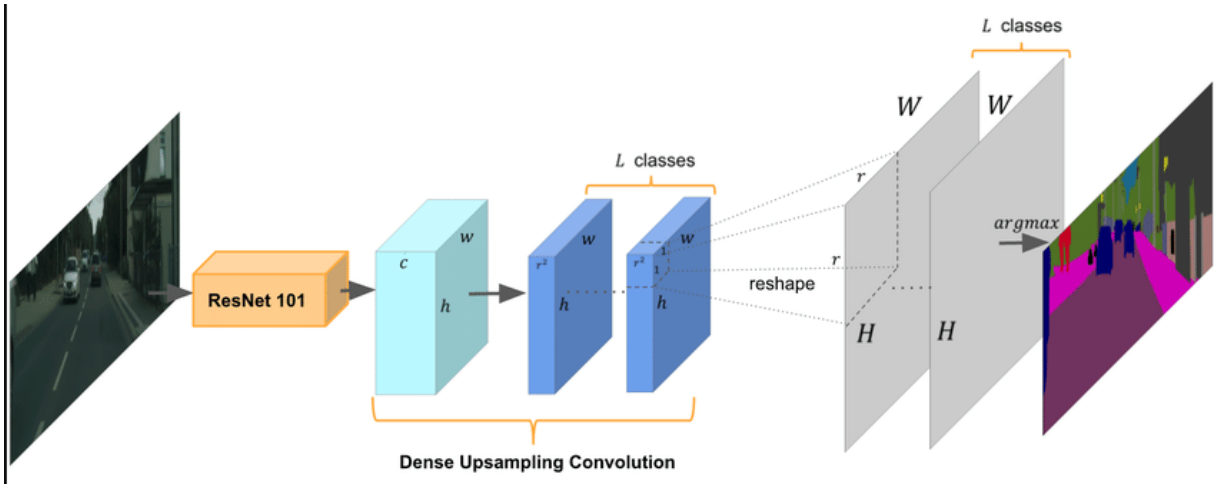

Figure 5.1.37 ResNet-101

ResNet-101 is a convolutional neural network that is **101** layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database

In a nutshell, a **VAE** is an **autoencoder** whose encodings distribution is regularized during the training in order to ensure that its latent space has good properties allowing us to generate some new data.
Moreover, the term “variational” comes from the close relation there is between the regularization and the variational inference method in statistics. If the last two sentences summaries pretty well the notion of VAEs, they can also raise a lot of questions. What is an autoencoder? What is the latent space and why regularizing it? How to generate new data from VAEs? What is the link between VAEs and variational inference? In order to describe VAEs as well as possible, we will try to answer all this questions (and many others!) and to provide the reader with as much insights as we can (ranging from basic intuitions to more advanced mathematical details). Thus, the purpose of this post is not only to discuss the fundamental notions Variational Autoencoders rely on but also to build step by step and starting from the very beginning the reasoning that leads to these notions.

What is dimensionality reduction?

In machine learning, dimensionality reduction is the process of reducing the number of features that describe some data.
This reduction is done either by:

- selection (only some existing features are conserved) or
- extraction (a reduced number of new features are created based on the old features) and can be useful in many situations that require low dimensional data (data visualization, data storage, heavy computation).

Although there exist many different methods of dimensionality reduction, we can set a global framework that is matched by most (if not any!) of these methods.

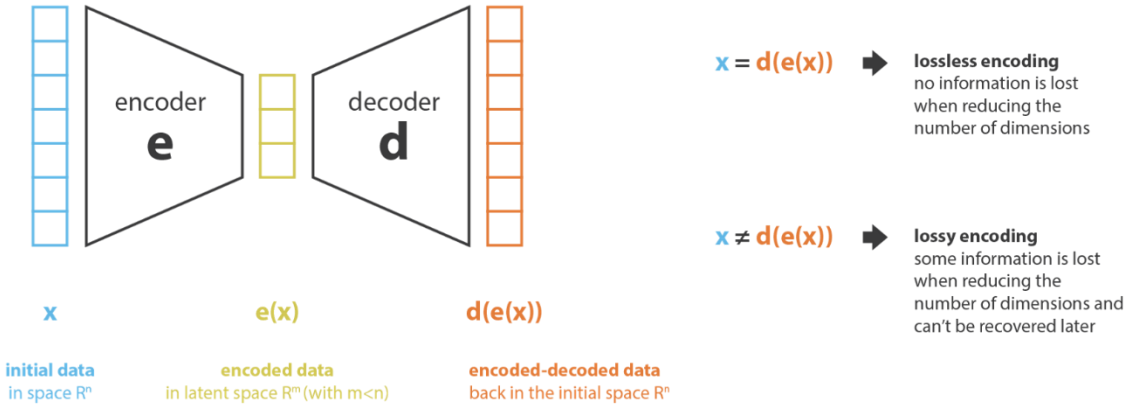

Figure 5.1.38 Encoder and Decoder

First, let's call encoder the process that produce the "new features" representation from the "old features" representation (by selection or by extraction) and decoder the reverse process.
Dimensionality reduction can then be interpreted as data compression where the encoder compresses the data (from the initial space to the encoded space, also called **latent space**) whereas the decoder decompresses them.
Of course, depending on the initial data distribution, the latent space dimension and the encoder definition, this compression can be lossy, meaning that a part of the information is lost during the encoding process and cannot be recovered when decoding.
The main purpose of a dimensionality reduction method is to find the best encoder/decoder pair among a given family.

Autoencoders

The general idea of autoencoders is pretty simple and consists in setting an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimization process**.** So, at each iteration we feed the autoencoder architecture (the encoder followed by the decoder) with some data, we compare the encoded-decoded output with the initial data and backpropagate the error through the architecture to update the weights of the networks. Thus, intuitively, the overall autoencoder architecture (encoder+decoder) creates a bottleneck for data that ensures only the main structured part of the information can go through and be reconstructed.
Looking at our general framework, the family E of considered encoders is defined by the encoder network architecture, the family D of considered decoders is defined by the decoder network architecture and the search of encoder and decoder that minimize the reconstruction error is done by gradient descent over the parameters of these networks.

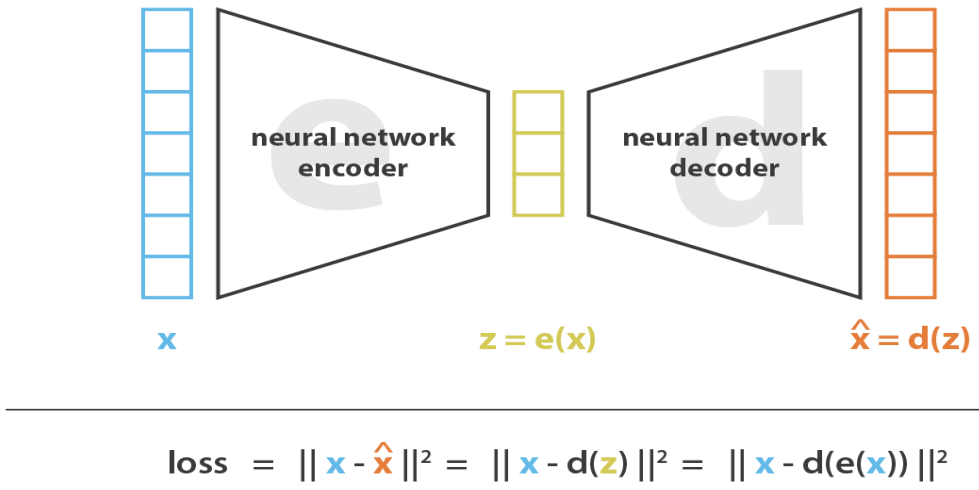

Figure 5.1.39 encoder and decoder architectures

Let’s first suppose that both our encoder and decoder architectures have only one layer without non-linearity (linear autoencoder). Such encoder and decoder are then simple linear transformations that can be expressed as matrices. In such situation, we can see a clear link with PCA in the sense that, just like PCA does, we are looking for the best linear subspace to project data on with as few information loss as possible when doing so. Encoding and decoding matrices obtained with PCA define naturally one of the solutions we would be satisfied to reach by gradient descent, but we should outline that this is not the only one. Indeed, several bases can be chosen to describe the same optimal subspace and, so, several encoder/decoder pairs can give the optimal reconstruction error. Moreover, for linear autoencoders and contrarily to PCA, the new features we end up do not have to be independent (no orthogonality constraints in the neural networks).

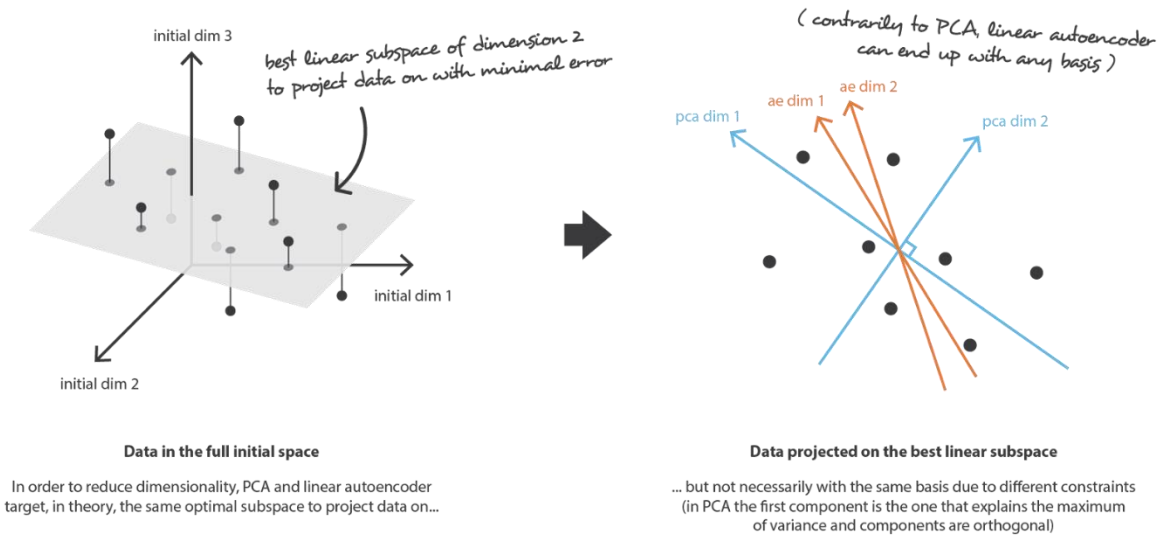

Figure 5.1.40 PCA

Now, let's assume that both the encoder and the decoder are deep and non-linear. In such case, the more complex the architecture is, the more the autoencoder can proceed to a high dimensionality reduction while keeping reconstruction loss low. Intuitively, if our encoder and our decoder have enough degrees of freedom, we can reduce any initial dimensionality to 1. Indeed, an encoder with "infinite power" could theoretically takes our N initial data points and encodes them as 1, 2, 3, … up to N (or more generally, as N integer on the real axis) and the associated decoder could make the reverse transformation, with no loss during the process.

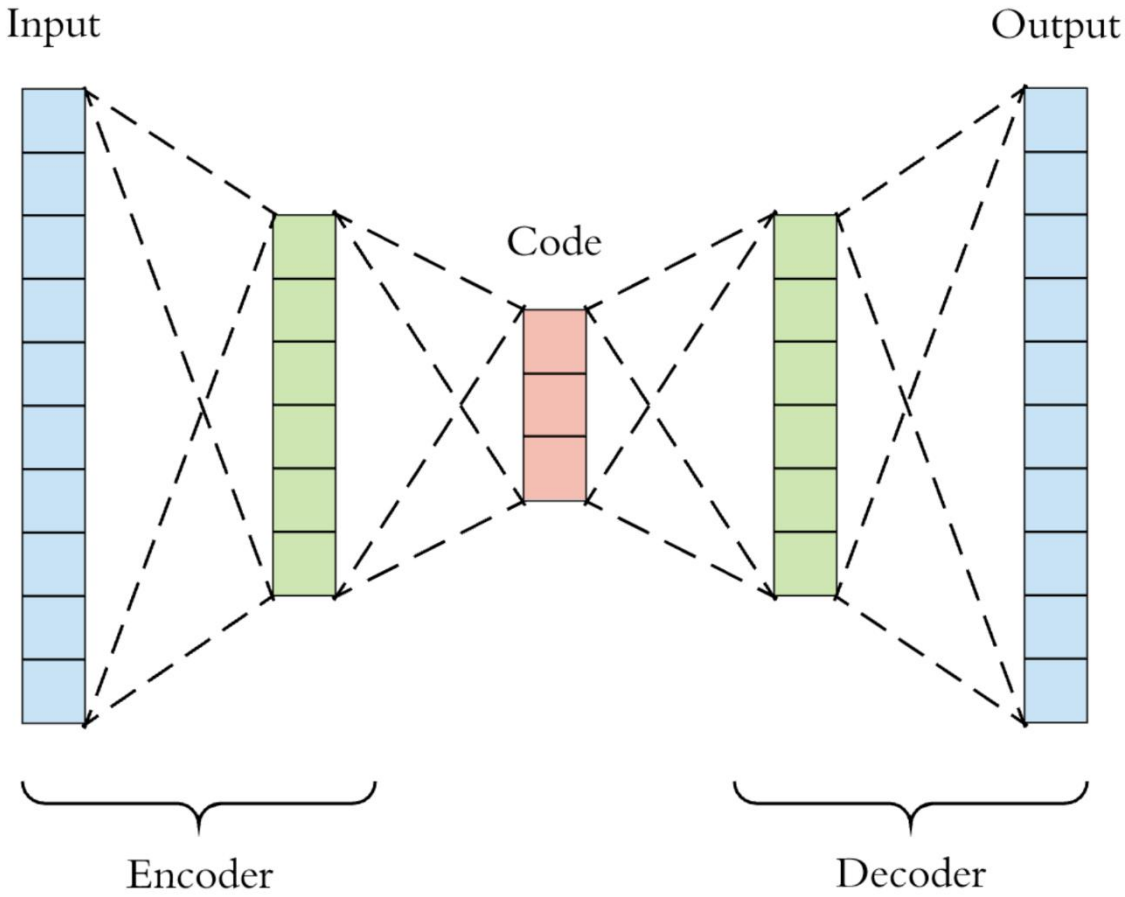

Figure 5.1.41 dimensionality reduction

Here, we should however keep two things in mind. First, an important dimensionality reduction with no reconstruction loss often comes with a price: the lack of interpretable and exploitable structures in the latent space (lack of regularity). Second, most of the time the final purpose of dimensionality reduction is not to only reduce the number of dimensions of the data but to reduce this number of dimensions while keeping the major part of the data structure information in the reduced representations. For these two reasons, the dimension of the latent space and the "depth" of autoencoders (that define degree and quality of compression) have to be carefully controlled and adjusted depending on the final purpose of the dimensionality reduction.

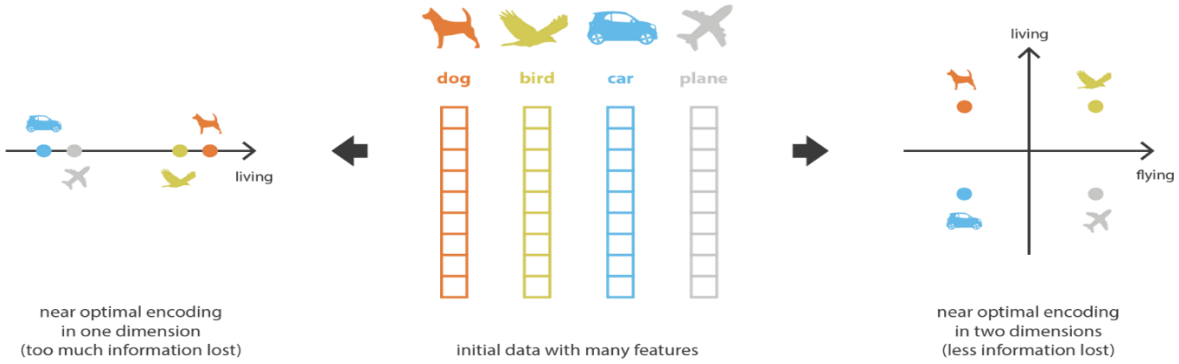

Figure 5.1.42 variational autoencoders

Definition of variational autoencoders

a variational autoencoder can be defined as being an autoencoder whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable generative process.

Just as a standard autoencoder, a variational autoencoder is an architecture composed of both an encoder and a decoder and that is trained to minimize the reconstruction error between the encoded-decoded data and the initial data. However, in order to introduce some regularization of the latent space, we proceed to a slight modification of the encoding-decoding process: instead of encoding an input as a single point, we encode it as a distribution over the latent space. The model is then trained as follows:

- first, the input is encoded as distribution over the latent space
- second, a point from the latent space is sampled from that distribution
- third, the sampled point is decoded and the reconstruction error can be computed
- finally, the reconstruction error is backpropagated through the network

[5-2- Setting up The Environment]

Figure 5.2.1 Colab

Colab Notebooks

Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.

We provide notebooks for several of our models that allow you to interact with them on a hosted Google Cloud instance for free.

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier.

Figure 5.2.2 Kaggle

Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Kaggle got its start in 2010 by offering machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science, and Artificial Intelligence education.

Kaggle Kernels: a cloud-based workbench for data science and machine learning. Allows data scientists to share code and analysis in Python and R.

[6-Implementation]

[6-1-Data Preprocessing]

Data preprocessing is an important step in the data mining process.
The phrase "garbage in, garbage out" is particularly applicable to data mining and deep learning projects.
Data-gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: −100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), missing values, etc.
Analyzing data that has not been carefully screened for such problems can produce misleading results.
Thus, the representation and quality of data is first and foremost before running an analysis.
Often, data preprocessing is the most important phase of a deep learning project, especially in computational biology.

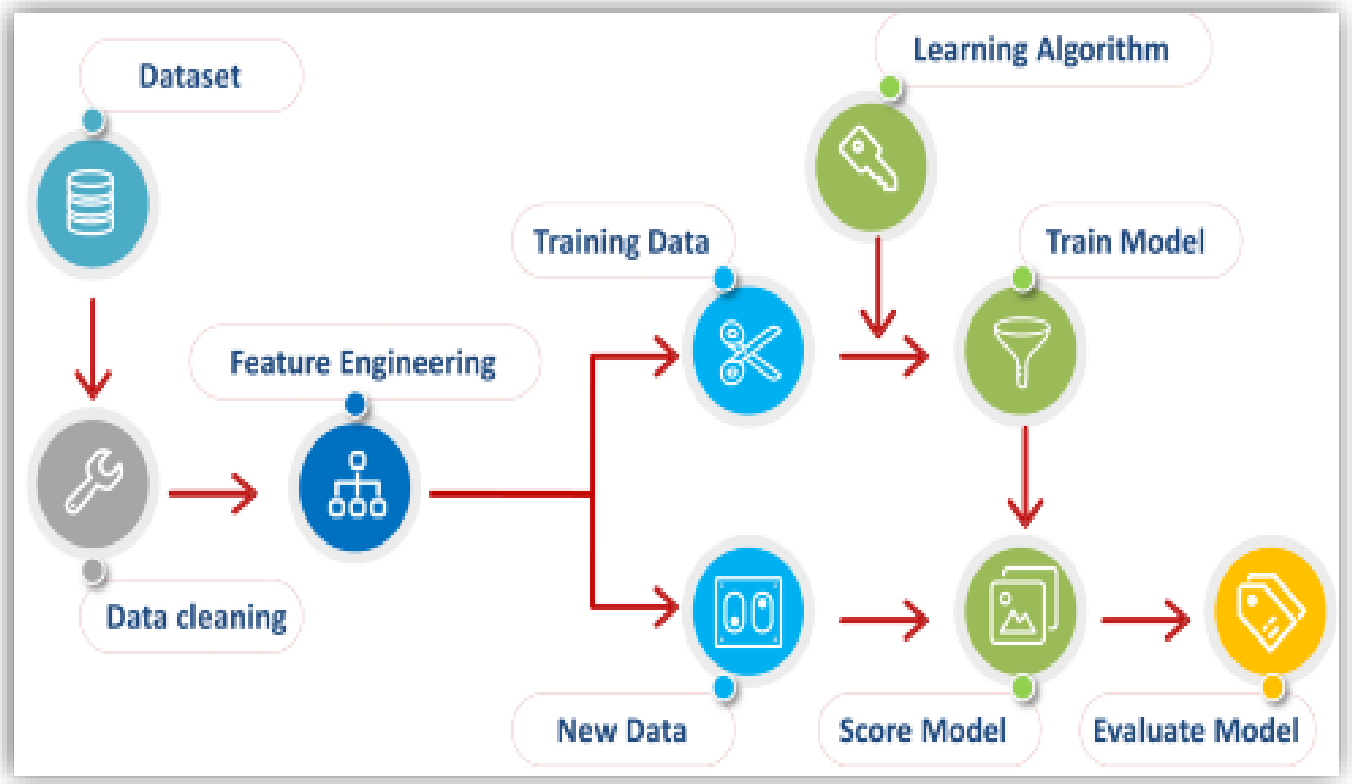

Figure 6.1.1 Data Preprocessing

Image Ratio

The image size of the input layer of the VGG_16 model is (224, 224). But we can see from the figures 9 & 10, that the image sizes differ in terms of height and width including the difference in the size of black corners. Unfortunately, some images that are wide may look weird after resizing. Let's look at the ratio distributions (ratio = width/height).

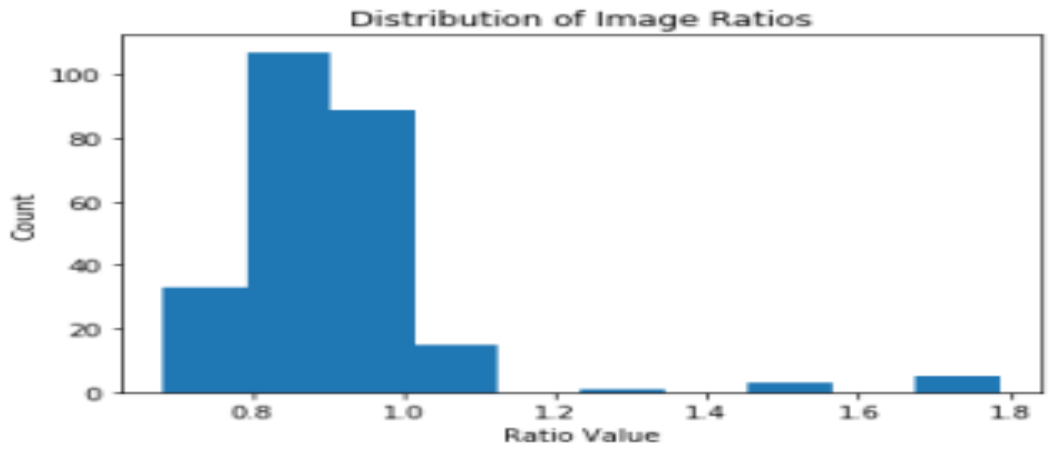

Figure 6.1.2 observe that the size of the images varies widely

From the Figure, we can observe that the size of the images varies widely. Most of the images fall in the ratio value between the interval (0.7 - 1.1), which denotes that the images have more height. However, a few falls into the ratio values of (1.2 – 1.8), meaning the images are wider.

The VGG_16 model only takes 224 by 224-pixel inputs. Thus, the data needs to be further preprocessed and that includes image cropping before providing it as an input to the model.

Cropping the MRI Scans

In order to normalize the data, first, the images need to be cropped. In general, it detects the relevant region from the image. Next, it finds the extreme North, East, West, and South (x, y) coordinates along the contour as described below.

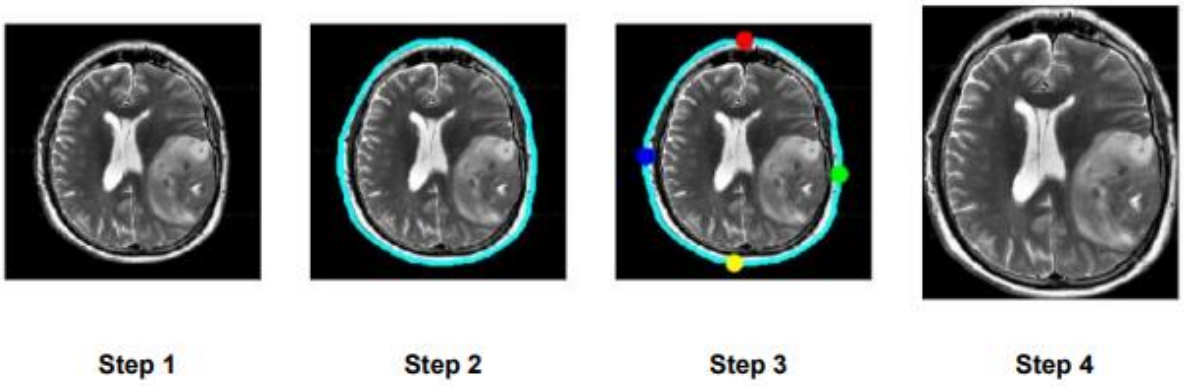

Figure 6.1.3 Cropping the MRI Scans

Step 1. First, we get the original image from the dataset.
Step 2. Next, we find the biggest contour of the brain.
Step 3. Third, we plot the extreme points on all four edges of the contour. Remember the contour is simply a Numpy array of (x, y) – coordinates.
Step 4. Lastly, we crop the image based on those extreme points.
Next, the cropping must be applied to all the images present in each and every dataset. This makes the dataset ready that can be used to train and build the model.

[6-2 Data Augmentation]

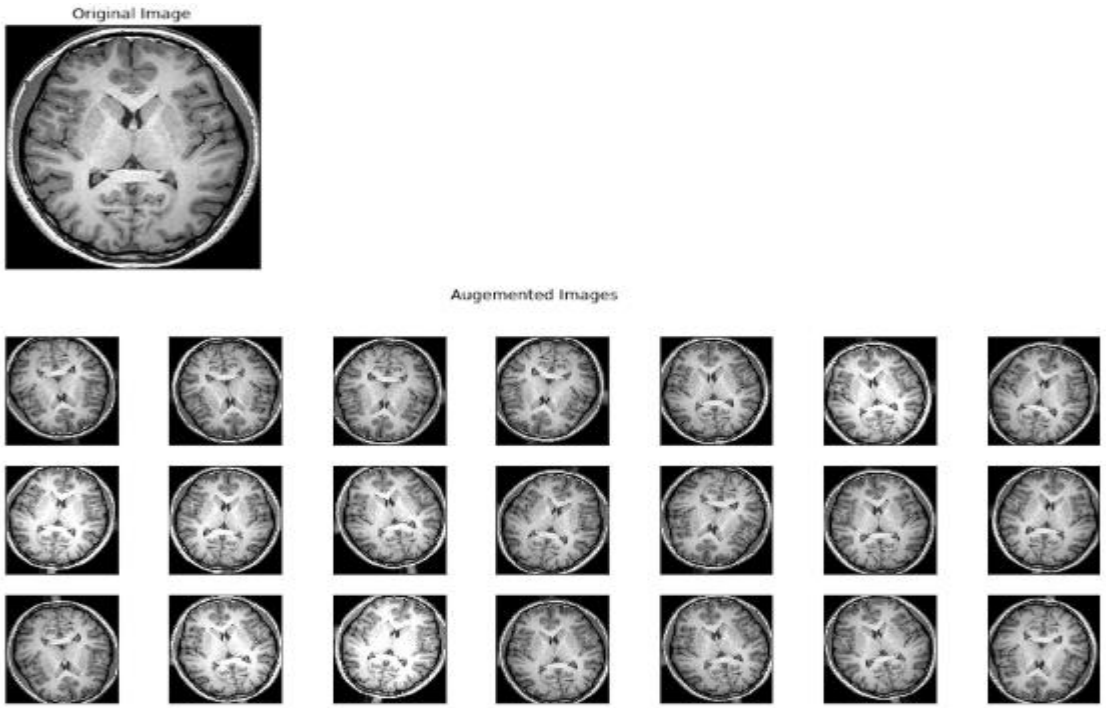

Figure 6.2.1 Data Augmentation

the keras.preprocessing.image.ImageDataGenerator class. This class allows you to:

- **rotation_range** is a value in degrees 15, a range within which to randomly rotate pictures
- **width_shift** and **height_shift** are within which to randomly translate pictures vertically or horizontally value 0.05
- **rescale** is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, so we target values between 0 and 1 instead by scaling with a 1/255. factor.
- **shear_range** is for randomly applying shearing transformations value 0.05
- **horizontal_flip** is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal asymmetry.
- **vertical_flip** is for randomly flipping half of the images vertical -relevant when there are no assumptions of vertical asymmetry.

[6-3-Data Description]

- **Data Description Overview**

Ample multi-institutional routine clinically-acquired pre-operative multimodal MRI scans of glioblastoma (GBM/HGG) and lower grade glioma (LGG), with pathologically confirmed diagnosis and available OS, are provided as the training, validation and testing data for this year's BraTS challenge.

Specifically, the datasets used in this year's challenge have been updated, since BraTS'18, with more routine clinically-acquired 3T multimodal MRI scans, with accompanying ground truth labels by expert board-certified neuroradiologists.

- **Imaging Data Description**

All BraTS multimodal scans are available as NIfTI files (.nii.gz) and describe

a) Native (T1).
b) Post-contrast T1-weighted (T1Gd).
c) T2-weighted (T2).
d) T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes, and were acquired with different clinical protocols and various scanners from multiple (n=19) institutions.

All the imaging datasets have been segmented manually, by one to four raters, following the same annotation protocol, and their annotations were approved by experienced Neuro-radiologists.

Annotations comprise the GD-enhancing tumor (ET — label 4), the peritumoral edema (ED — label 2), and the necrotic and non-enhancing tumor core (NCR/NET — label 1), as described both in the BraTS 2012-2013 TMI paper and in the latest BraTS summarizing paper.

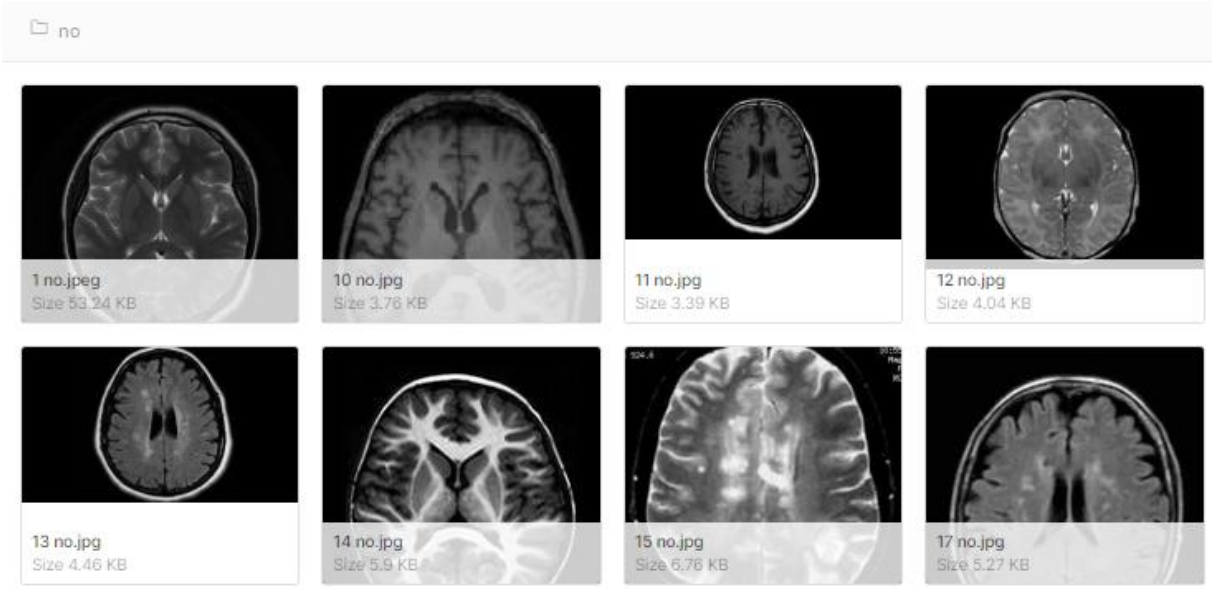

Figure 6.3.1 Dataset 1

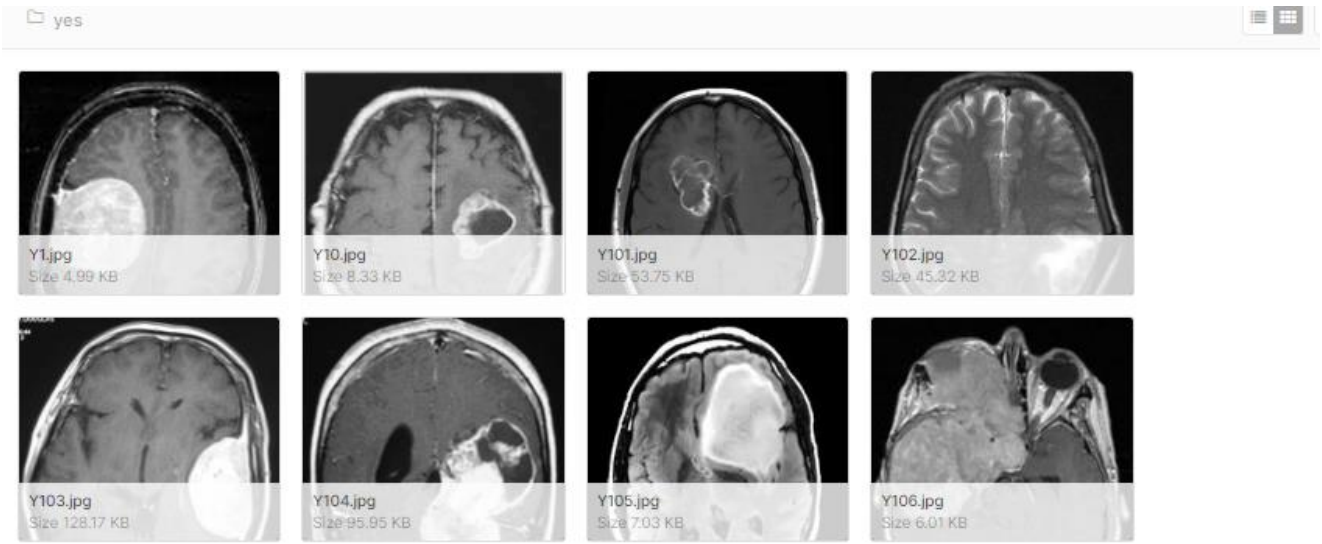

Figure 6.3.2 Dataset 2

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 where this MRI scans come from and so on.

[6-4- Separation training and test dataset]

When you're working on a model and want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you'll a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the obviously solution is to split the dataset you have into two sets, one for training and the other for testing; and you do this before you start training your model. But the question is, how do you split the data? You can't possibly manually split the dataset into two.
And you also have to make sure you split the data in a random manner to help us with this task, the SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

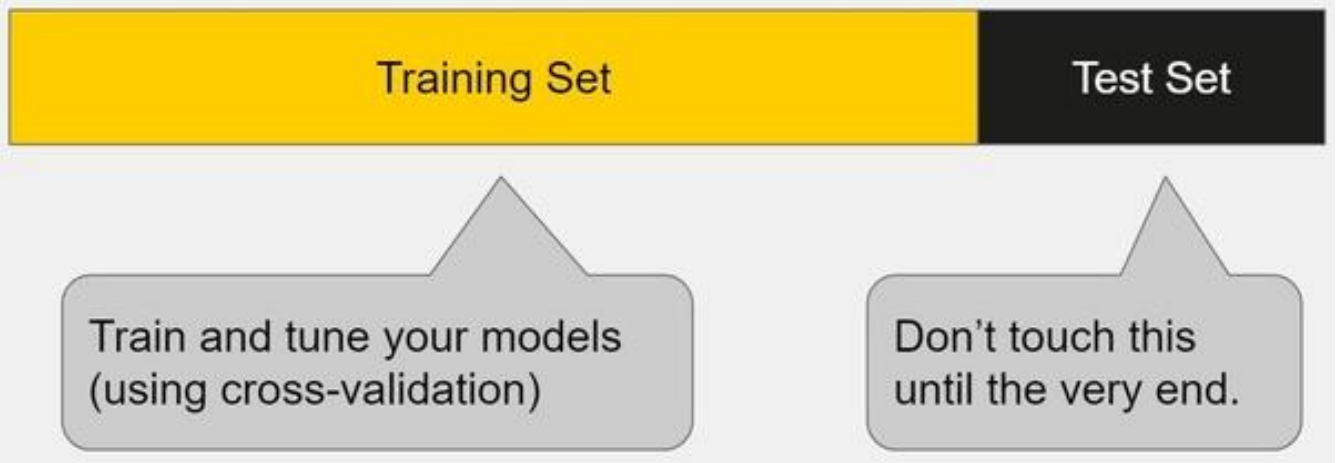

Figure 6.4.1 train_test_split

There are a few parameters that we need to understand before we use the class:

- **Test-Size:**

This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 20% as the test dataset. If you're specifying this parameter, you can ignore the next parameter.

- **Train-Size**:

You have to specify this parameter only if you're not specifying the **Test-Size**.

This is the same as **Test-Size**, but instead you tell the class what percent of the dataset you want to split 60% as the training set.

- **Random-State**:

Here you pass an integer, which will act as the seed for the random number generator during the split 20% as the validation set . Or, you can also pass an instance of the **Random-State** class, which will become the number generator. If you don't pass anything, the **Random-State** instance used by np.random will be used instead.

Figure 6.4.2 Dataset Split

[6-5- Building the models]

A great way to use deep learning to classify images is to build a Convolutional Neural Network (CNN).
The Keras library in Python makes it pretty simple to build a CNN.

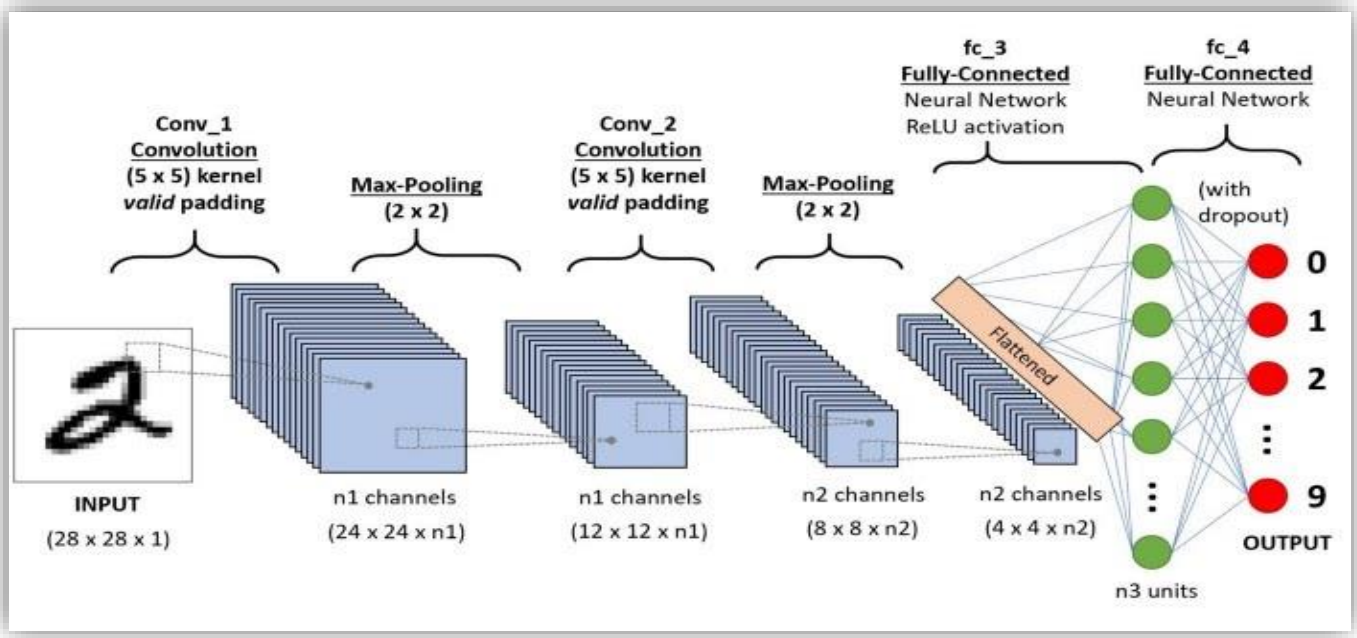

Figure 6.5.1 Building the models

Computers see images using pixels. Pixels in images are usually related. For example, a certain group of pixels may signify an edge in an image or some other pattern. Convolutions use this to help identify images. A convolution multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered. This process is visualized below. A convolution neural network is similar to a multi-layer perceptron network. The major differences are what the network learns, how they are structured and what purpose they are mostly used for.

Convolutional neural networks were also inspired from biological processes, their structure has a semblance of the visual cortex present in an animal. CNNs are largely applied in the domain of computer vision and has been highly successful in achieving state of the art performance on various test cases.

- **What do the hidden layers learn?**

The hidden layers in a CNN are generally convolution and pooling (down sampling) layers.

In each convolution layer, we take a filter of a small size and move that filter across the image and perform convolution operations.

Convolution operations are nothing but element-wise matrix multiplication between the filter values and the pixels in the image and the resultant values are summed.

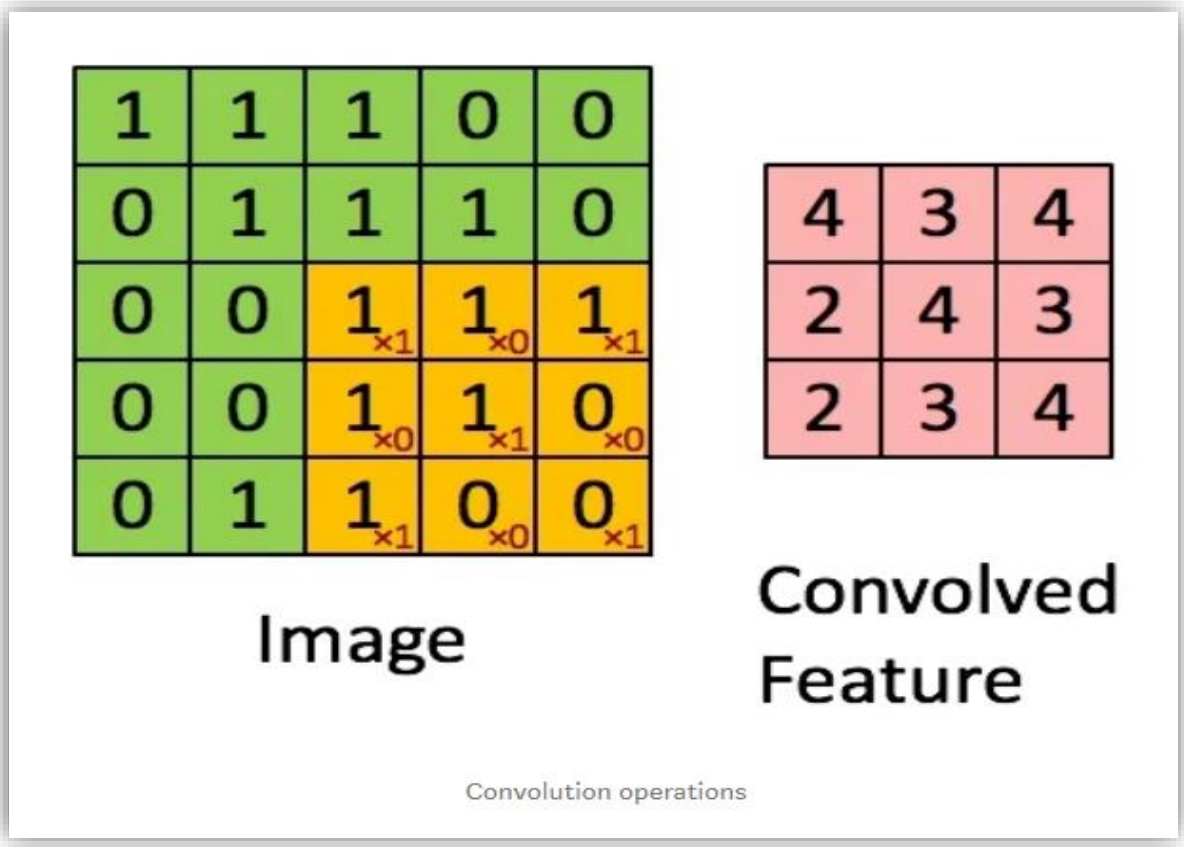

Figure 6.5.2 hidden layers

The filter's values are tuned through the iterative process of training and after a neural net has trained for certain number of epochs, these filters start to look out for various features in the image.

Take the example of face detection using a convolutional neural network. The earlier layers of the network look for simple features such as edges at different orientations etc.

As we progress through the network, the layers start detecting more complex features and when you look at the features detected by the final layers, they almost look like a face.

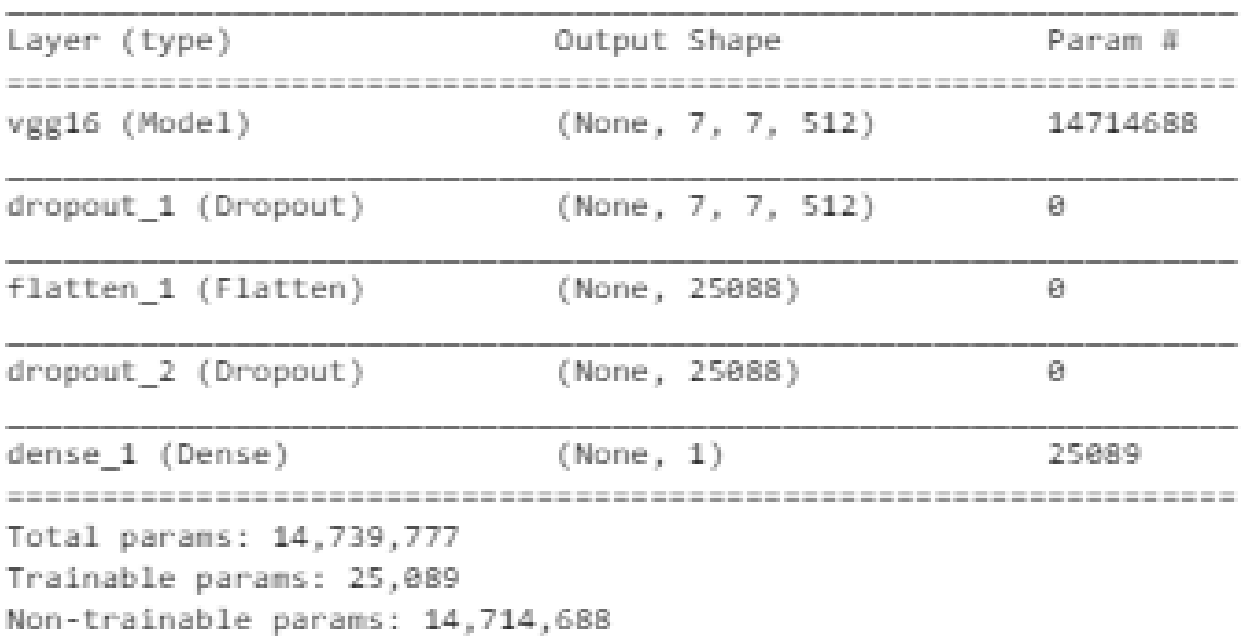

Figure 6.5.3 Vgg16 layer

Now, let's move on to pooling layers. Pooling layers are used to down-sample the image. The image would contain a lot of pixel values and it is typically easy for the network to learn the features if the image size is progressively reduced. Pooling layers help in reducing the number of parameters required and hence, this reduces the computation required. Pooling also helps in avoiding overfitting. There are two types of pooling operation that could be done:

Max Pooling:

Selecting the maximum value.

- Average Pooling:

  Sum all of the values and dividing it by the total number of values.

[6-6- Running predictions on the test set]

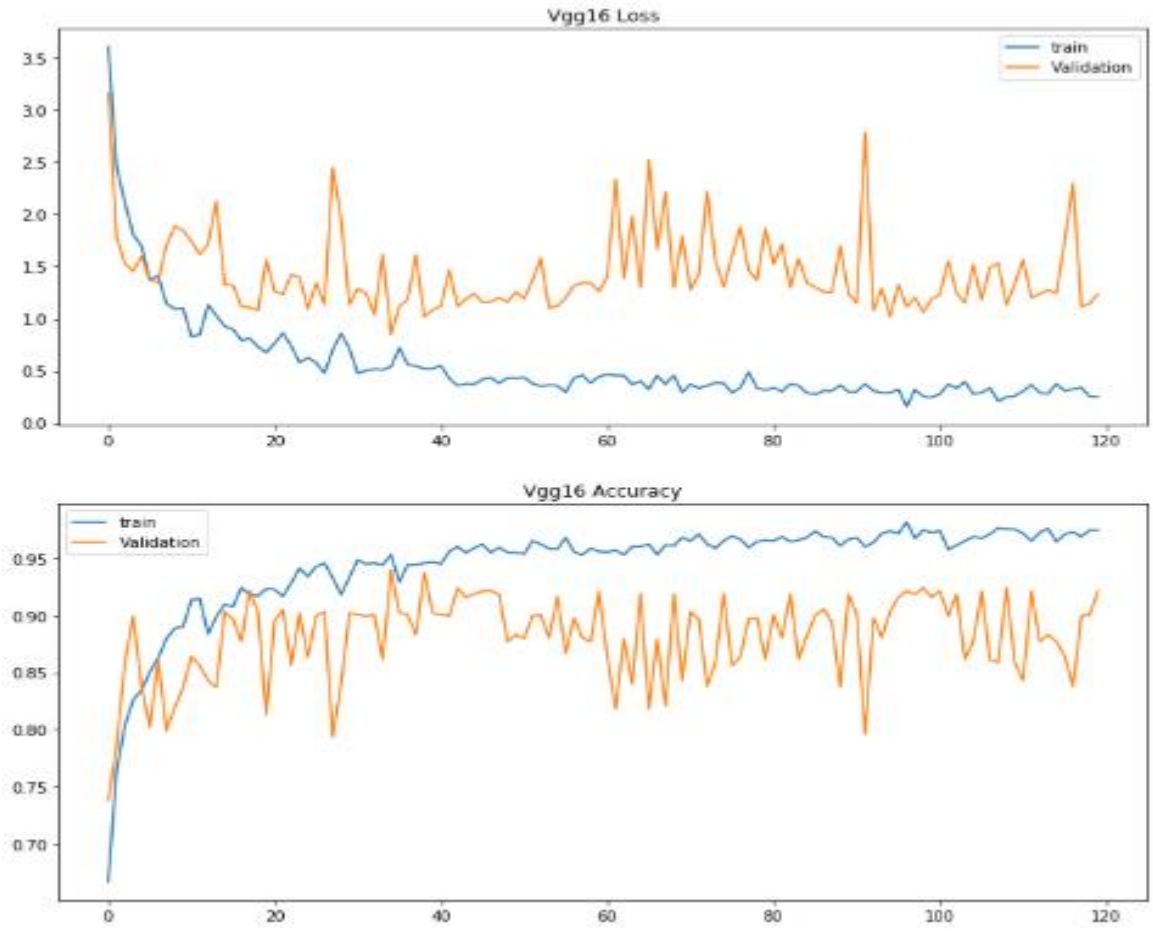

Figure 6.6.1 VGG16 Accuracy

Accuracy: 0.900000
Precision: 1.000000

Recall: 0.800000

F1 score: 0.888889

Train: 0.840, Test: 0.900

This is the highest accuracy reached in the our project .

[6-7- Checking the Confusion Matrix]

Well, it is a performance measurement for deep learning classification problem where output can be two or more classes.
It is a table with 4 different combinations of predicted and actual values.
It is extremely useful for measuring:

- **Recall Score (Sensitivity)**
- **Precision Score (Specificity)**

Accuracy and most importantly AUC-ROC Curve.
Let's understand TP, FP, FN, TN in terms of pregnancy analogy.

- **True Positive:**

Interpretation: You predicted positive and it's true.
You predicted that a woman is pregnant and she actually is.

- **True Negative:**

Interpretation: You predicted negative and it's true.
You predicted that a man is not pregnant and he actually is not.

- **False Positive: (Type 1 Error)**

Interpretation: You predicted positive and it's false.
You predicted that a man is pregnant but he actually is not.

- **False Negative: (Type 2 Error)**

Interpretation: You predicted negative and it's false.
You predicted that a woman is not pregnant but she actually is.

Just Remember, we describe predicted values as Positive and Negative and actual values as True and False.Let's understand confusion matrix through math.

Recall

Out of all the positive classes, how much we predicted correctly. It should be high as possible.

Precision

Out of all the positive classes we have predicted correctly, how many are actually positive.

And **Accuracy** will be…

- **F1-Score = 2 * (Precision * Recall) / (Precision + Recall)**

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F\text{ - }measure = \frac{2*Recall*Precision}{Recall + Precision}$$

y	y pred	output for threshold 0.6	Recall	Precision	Accuracy
0	0.5	0			
1	0.9	1			
0	0.7	1			
1	0.7	1	1/2	2/3	4/7
1	0.3	0			
0	0.4	0			
1	0.5	0			

Figure 6.7.1 Recall

[6-8- Adding Dropout Regularization to fight Over-itting]

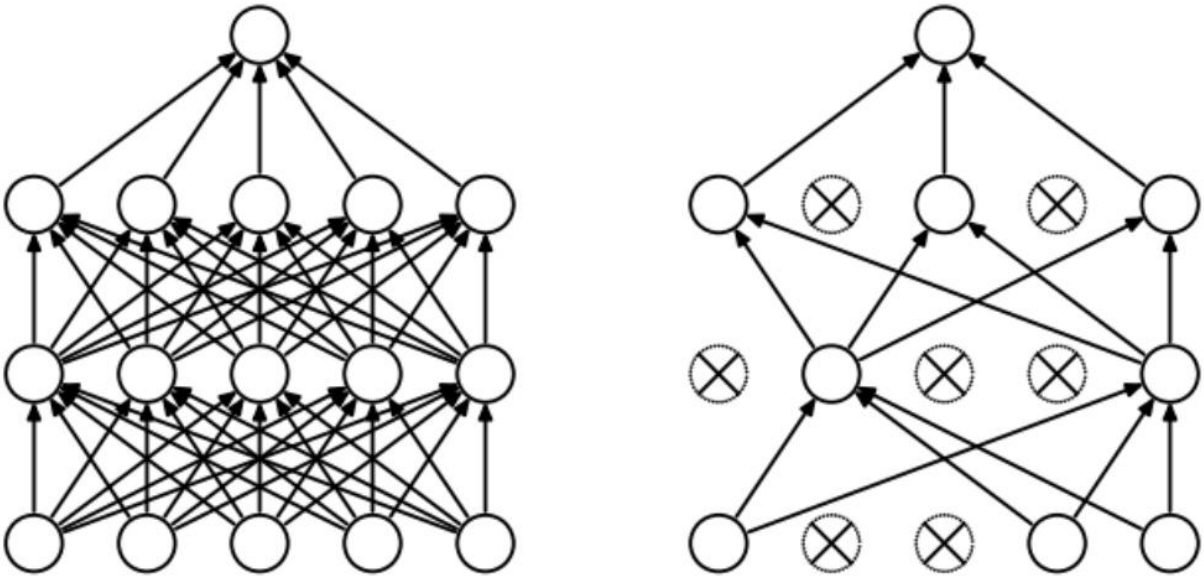

Figure 6.8.1 Dropout

Deep neural nets with a large number of parameters are very powerful deep learning systems. However, over-fitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with over-fitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single “un thinned” network that has smaller weights. This significantly reduces over-fitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

[7- Conclusion and future work]

[7-1- Conclusion]

In conclusion, this project proposes a work on brain cancer detection using MRI scans based on a machine learning algorithm with the combination of computer vision problems, that helps to automate the process of cropping the brain from MRI scans. This dataset of MRI scans consists of:
(i) different types of tumors having different sizes as well as positions and (ii) unaffected brain scans for training purposes. The CNN model used for this project is VGG_16. The existing works use the other CNN models like InceptionV3, ResNet50, FastAI, etc. Some even used the VGG_16 model but the results are not satisfactory because the dataset available is too small. The model I built is more like a prototype because it was trained using 243 samples of brain MRI scans. As the dataset is too small, I implemented a technique called Data Augmentation. This technique did generate approximately 7,776 (243 x 32) augmented images from the 243 images. Now the VGG_16 model has to be trained with these augmented images. After training the model, it is ready to be tested with the test dataset However, the predictions can be improved by implementing Ensemble Learning. This is going to be my future work.

[7-2- Future Work]

This algorithm was built using one of the effective CNN models, VGG_16. This model is 16 layers deep and best suited for image classification problems. Different models (like Inception_V3, Resnet50, fastAI, etc.) produce different accuracy scores. We can also combine multiple models together to build a meta-model for better performance.

A combination of different classifier models to produce a mega algorithm and training this built meta-model is a whole other level of work.
This process is known as Ensemble Learning. “Stacking” is the most common Ensemble learning technique that combines multiple classification models via a metaclassifier.
The base-level models use a complete training set to get trained and the outputs produced by these base models can be used to train the meta-model.
The prediction levels are much more accurate with the use of Ensemble Learning technique.
However, the implementation of this meta-model requires a high-performance Graphics Processing Unit (GPU) in addition to CPU that demands high computational power, cost, and time. My future work lies in the implementation of Brain cancer detection using Ensemble learning.
The combined approach that we presented in this work has some limitations. Although we have used it on state-of-the-art deep learning algorithms, however, we have evaluated its results on only classification tasks. It can be extended to be used for other important problems e.g.

- Work with different CNNs
- Improve and increase dataset
- API

[8- References]

1- The Keras Blog (2016, June 5). “Building powerful image classification models using very little data” Retrieved October 2019.

2- Wadhai, V.M., Chancalani, A., Kachwalla, M., Shinde, P., Katare, R., Agarwal, A. (April 2018). “Classification of Brain MRI images for Cancer Detection using Deep Learning”, in International Journal of Advanced Research in Computer and Communication Engineering, ISO 3297:2007 Certified, Vol. 7, Issue 4.

3- Francois Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions”, Google, Inc,2017.

4- Gao Huang; Zhuang Liu; Kilian Q. Weinberger; Laurens van der Maaten;” Densely Connected Convolutional Networks”, IEEE,2016.

5- Yi Zhu and Shawn Newsam,” Densenet For Dense Flow”, IEEE,2017.

6- Simon Jegou;Michal Drozdzal;David Vazquez;Adriana Romero;Yoshua Bengio;Montreal Institute,” The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation”, IEEE,2016.

7- Jason Brownlee,” A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks”, Deep Learning Performance,2018.

8- Mayo Clinic. “Brain tumor.” Retrieved September 2019.

9- Healthline. “Different types of brain tumors” Retrieved September 2019.

10- Chakrabarty, N. “Dataset for Brain tumor MRI images” Retrieved October 2019.

11-Sharma, K., Kaur, A., Gujral, S., (October 2014). “Brain Tumor Detection based on Machine Learning Algorithms”, in International Journal of Computer Applications (0975 – 8887) Vol. 103, Issue 1.

12- Rosebrock, A (2016, April 11). “Finding extreme points in contours with OpenCV” Retrieved October 2019.

13- Smolyakov, V. (2017, August 22). “Ensemble Learning to Improve Machine Learning Results” Retrieved November 2019.

14- Keras Documentation. “Keras: The Python Deep Learning library” Retrieved October 2019.

15- Heller, M. (2019, January 28). “What is Keras? The deep neural network API explained” Infoworld. Retrieved November 2019.

17- Hassan, M.U. (2018, November 20). “VGG16 – Convolutional Network for Classification and Detection” Retrieved October 2019, from https://neurohive.io/en/popular-networks/vgg16/