CHAPTER – 1
INTRODUCTION

1.INTRODUCTION

1.1 Background:

Seismic data is collected using reflection seismology, or seismic reflection. The method requires a controlled seismic source of energy, such as compressed air or a seismic vibrator, and sensors record the reflection from rock interfaces within the subsurface. The recorded data is then processed to create a 3D view of earth's interior. Reflection seismology is similar to X-ray, sonar and echolocation.

A seismic image is produced from imaging the reflection coming from rock boundaries. The seismic image shows the boundaries between different rock types. In theory, the strength of reflection is directly proportional to the difference in the physical properties on either sides of the interface. While seismic images show rock boundaries, they don't say much about the rock themselves; some rocks are easy to identify while some are difficult.

There are several areas of the world where there are vast quantities of salt in the subsurface. One of the challenges of seismic imaging is to identify the part of subsurface which is salt. Salt has characteristics that makes it both simple and hard to identify. Salt density is usually 2.14 g/cc which is lower than most surrounding rocks. The seismic velocity of salt is 4.5 km/sec, which is usually faster than its surrounding rocks. This difference creates a sharp reflection at the salt-sediment interface. Usually salt is an amorphous rock without much internal structure. This means that there is typically not much reflectivity inside the salt, unless there are sediments trapped inside it. The unusually high seismic velocity of salt can create problems with seismic imaging.

1.2 Seismic imaging

Seismic imaging directs an intense sound source into the ground to evaluate subsurface conditions and to possibly detect high concentrations of contamination. Receivers called geophones, analogous to microphones, pick up "echoes" that come back up through the ground and record the intensity and time of the "echo" on computers. Data processing turns these signals into images of the geologic structure.

There are two types of seismic images produced as the sound waves travel into the ground. *Reflected waves* travel downward, bounce off a layer or object in the soil or rock, and return to the surface. *Refracted waves* are those that travel downward, then turn at a geologic boundary (such as the surface of a rock layer) and travel along it before returning back to the surface. Reflected waves generally show more subsurface detail. However, multiple "echoes" can make reflections very difficult to interpret. Refracted waves are typically used to profile shallow bedrock (*i.e.,* rock less than 100 feet below the ground).

During the survey process, the reflections provide a three-dimensional digital model of the subsurface. This information can be used to identify preferential flow paths, determine the placement and screening of wells, and help select a remediation technology. In addition to providing information about subsurface formations, the indirect detection of dense contaminants including dense non-aqueous phase liquids (DNAPLs) may be possible from the seismic data.

1.2.1 Limitations and Concerns

While this technique may provide images of subsurface geology, demonstrations have shown that it does not directly locate and define contaminant plumes. Unless pooled by some geologic feature, the overall density contrast of DNAPL may be imperceptible. Seismic reflection does not work well in formations that are geologically heterogeneous.

1.2.2 Technology Development Status

High resolution, three-dimensional seismic reflection imaging has been used in exploration for oil, salt and gas, as well as for subsurface fresh water, since the 1950s. It is still being field tested for use at contaminated sites. Recent technological advances have made it possible to generate high-resolution images of formations up to 3,000 feet deep. The down hole applications to detect contaminant plumes are still in development.

CHAPTER – 2
LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 Machine Learning

The term machine learning refers to the automated detection of meaningful patterns in data. Machine Learning is an application of Artificial Intelligence (AI) that provides the systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine Learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observation or data, such as examples, direct experiences, or instruction, in order to look for patterns in data and make better decisions in future based on the examples we provide. The primary aim is to allow the computers learn automatically without intervention or assistance and adjust actions accordingly. ML can play a key role in a wide range of applications, such as Data Mining, Natural Language Processing, Image recognition, and Expert Systems.

2.1.1 Types of Learning

The crucial types on ML include

 Supervised Machine Learning

The program is “trained” on a pre-defined set of “training examples”, which then facilitate its ability to reach an accurate conclusion when new data is supplied. Used when information provided for training is classified or labelled. In majority of applications Learning consists of optimizing hypothesis function, h(x) so that, for given input x it will accurately predict some interesting values of h(x).

 Unsupervised Machine Learning

The program is given bunch of data and must find patterns and relationships. Used when information used to train is neither classified nor labelled.

 Reinforcement Learning

Reinforcement learning, in the context of artificial intelligence, is a type of Dynamic programming that trains algorithms using a system of reward and punishment. A Reinforcement learning algorithm, or agent learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty.

2.2 Computer Vision

As the name suggests, the aim of computer vision (CV) is to imitate the functionality of human eye and brain components responsible for your sense of sight. Doing actions such as recognizing an animal, describing a view, differentiating among visible objects are really a cake-walk for humans. You'd be surprised to know that it took decades of research to discover and impart the ability of detecting an object to a computer with reasonable accuracy.

The field of computer vision has witnessed continual advancements in the past 5 years. One of the most stated advancement is Convolution Neural Networks (CNNs). Today, deep CNNs form the crux of most sophisticated fancy computer vision application, such as self-driving cars, auto-tagging of friends in our facebook pictures, facial security features, gesture recognition, automatic number plate recognition, etc.

Object detection is considered to be the most basic application of computer vision. Rest of the other developments in computer vision are achieved by making small enhancements on top of this.In real life, every time we(humans) open our eyes, we unconsciously detect objects.

The main tasks of computer vision are given as

2.2.1 Object Classification

Object Classification usually focuses on the classification of a small part of the image into two or more classes. For many of these tasks both local information on object appearance and global contextual information on object location are required for accurate classification.

2.2.2 Object Segmentation

The segmentation of required objects in images allows quantitative analysis related to volume and shape. The task of segmentation is typically defined by identifying the set of voxels which make up either the contour or the interior of the objects of interest. Segmentation is the most common subject of papers applying deep learning to image processing and as such has also seen the widest variety in methodology, including the development of unique CNN-based segmentation architectures and the wider application of RNNs.

In computer vision, Image segmentation is the process of partitioning a digital Image into multiple segments. The goal of segmentation is to simplify and or change the representation of an image into something that is more meaningful and easier to analyse. Image segmentation is typically used to locate objects and boundaries in images. More precisely, Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. The result of image segmentation is a set of segments that collectively cover the entire image or a set of contours extracted from the image. Each of the pixels in a region is similar with respect to some characteristic or computed property, such as colour, intensity or texture.

The segmentation in seismic image can be shown as:

Figure 1.1: <u>Original image:</u>

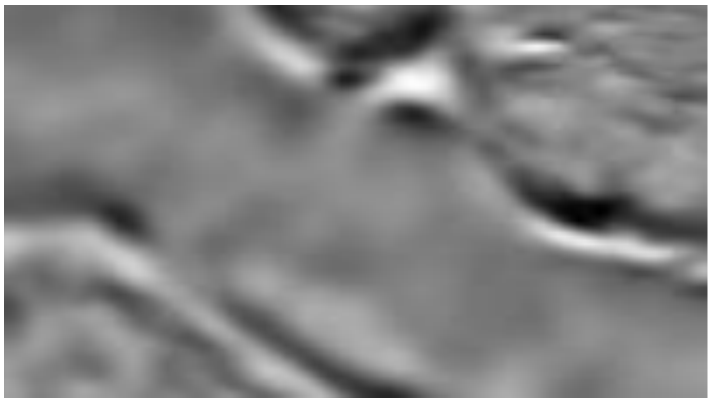
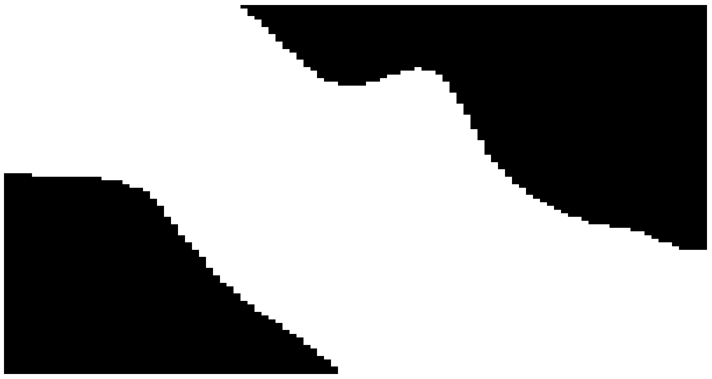


Figure 1.2: <u>segmented image:</u>



2.2.3 Object Detection

Object Detection involves detecting instances of objects from a particular class in an image. The goal of object detection is to detect all instances of objects from a known class such as, people, car or faces in an image. Typically, only a small no of instances of the object are present in the image, but there is a very large possible location and scales at which they can occur and a need to somehow be explored. The detection of objects in the image is the key part. Most of the published deep learning object detection systems use CNNs to perform pixel classification, after which some form of post processing is applied to obtain object candidates.

There are some aspects which are significantly different between object detection and object classification. One key point is that because every pixel is classified, typically the class balance is skewed severely towards the non-object class in attaining setting. Object detection systems construct a model for an object class from a set of training examples.

In the application under consideration, the object that need to be detected is the region where the salt deposits are present in the image.

2.3 Deep Learning

Deep learning is an aspect of artificial intelligence (AI) that is concerned with emulating the learning approach that human beings use to gain certain types of knowledge. At its simplest, deep learning can be thought of as a way to automate predictive analytics.

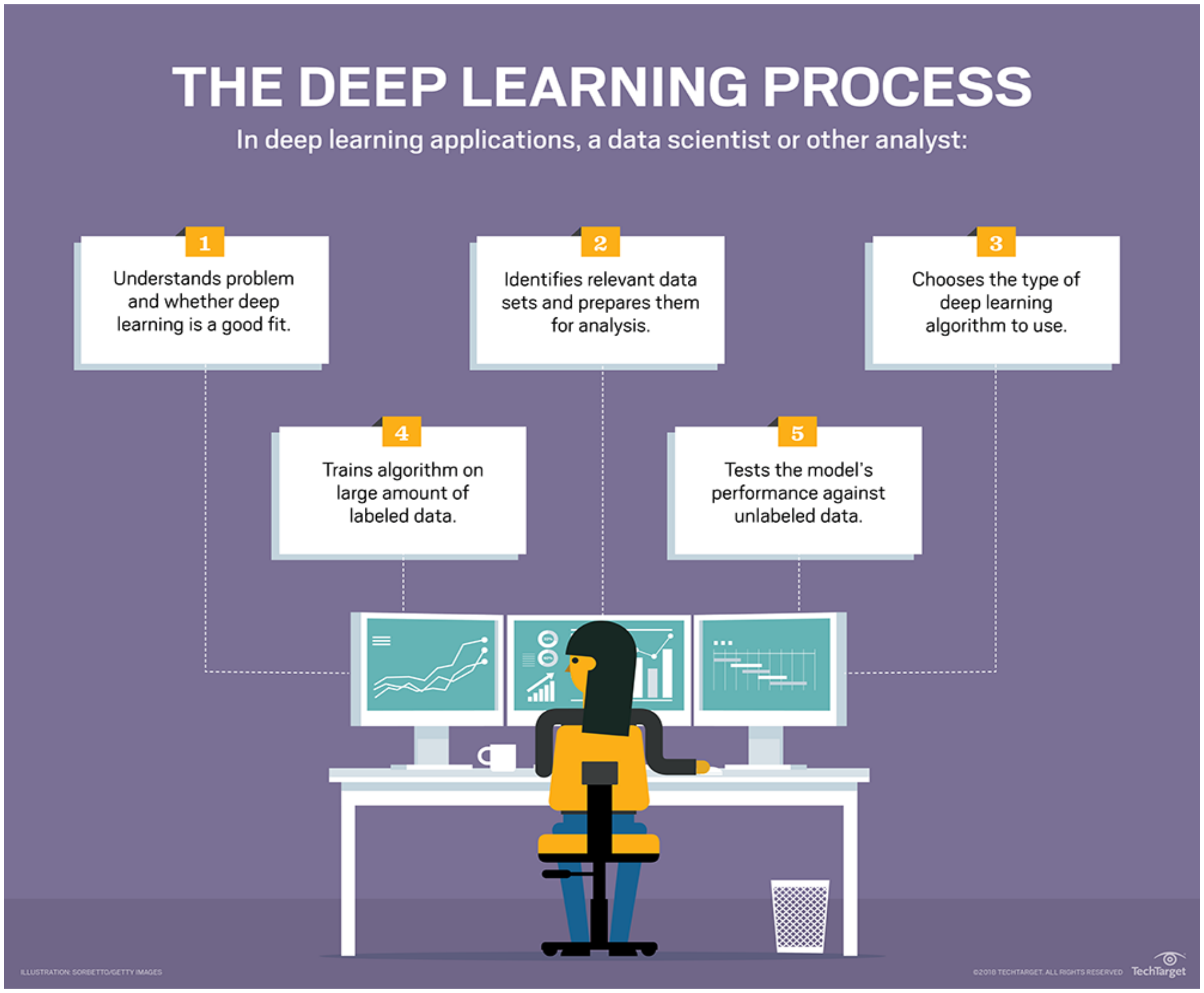
While traditional machine learning algorithms are linear, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction. To understand deep learning, imagine a toddler whose first word is *dog*. The toddler learns what a dog is (and is not) by pointing to objects and saying the word *dog*. The parent says, "Yes, that is a dog," or, "No, that is not a dog." As the toddler continues to point to objects, he becomes more aware of the features that all dogs possess. What the toddler does, without knowing it, is clarify a complex abstraction (the concept of dog) by building a hierarchy in which each level of abstraction is created with knowledge that was gained from the preceding layer of the hierarchy.

Computer programs that use deep learning go through much the same process. Each algorithm in the hierarchy applies a nonlinear transformation on its input and uses what it learns to create a statistical model as output. Iterations continue until the output has reached an acceptable level of accuracy. The number of processing layers through which data must pass is what inspired the label deep.

In traditional machine learning, the learning process is supervised and the programmer has to be very, very specific when telling the computer what types of things it should be looking for when deciding if an image contains a dog or does not contain a dog. This is a laborious process called *feature extraction* and the computer's success rate depends entirely upon the programmer's ability to accurately define a feature set for "dog." The advantage of deep learning is that the program builds the feature set by itself without supervision. Unsupervised learning is not only faster, but it is usually more accurate.

Initially, the computer program might be provided with training data, a set of images for which a human has labeled each image "dog" or "not dog" with meta tags. The program uses the information it receives from the training data to create a feature set for dog and build a predictive model. In this case, the model the computer first creates might predict that anything in an image that has four legs and a tail should be labeled "dog." Of course, the program is not aware of the labels "four legs" or "tail;" it will simply look for patterns of pixels in the digital data. With each iteration, the predictive model the computer creates becomes more complex and more accurate.

Because this process mimics a system of human neurons, deep learning is sometimes referred to as deep neural learning or deep neural networking. Unlike the toddler, who will take weeks or even months to understand the concept of "dog," a computer program that uses deep learning algorithms can be shown a training set and sort through millions of images, accurately identifying which images have dogs in them within a few minutes.


THE DEEP LEARNING PROCESS
In deep learning applications, a data scientist or other analyst:
1 Understands problem and whether deep learning is a good fit.
2 Identifies relevant data sets and prepares them for analysis.
3 Chooses the type of deep learning algorithm to use.
4 Trains algorithm on large amount of labeled data.
5 Tests the model's performance against unlabeled data.
ILLUSTRATION: SORBETTO/GETTY IMAGES
©2018 TECHTARGET. ALL RIGHTS RESERVED
TechTarget

To achieve an acceptable level of accuracy, deep learning programs require access to immense amounts of training data and processing power, neither of which were easily available to programmers until the era of big data and cloud computing. Because deep learning programming is able to create complex statistical models directly from its own iterative output, it is able to create accurate predictive models from large quantities of unlabeled, unstructured data. This is important as the internet of things (IoT) continues to become more pervasive, because most of the data humans and machines create is unstructured and is not labeled.

2.4 Neural Networks

An artificial intelligence function that imitates the work of the human brain in processing data and creating patterns for use in decision making is neural network model. Deep learning is a subset of machine learning in AI that has networks which are capable of learning unsupervised from data that is unstructured or unlabelled, Also known as Deep Neural Learning or Deep Neural Network.

In information technology, a neural network is a system of hardware and/or software patterned after the operation of neurons in the human brain. Neural networks -- also called artificial neural networks -- are a variety of deep learning technologies. Commercial applications of these technologies generally focus on solving complex signal processing or pattern recognition problems. Examples of significant commercial applications since 2000 include handwriting recognition for check processing, speech-to-text transcription, oil-exploration data analysis, weather prediction and facial recognition.

Most deep architectures are based on neural networks can be considered as a generalization of a linear or logistic regression. The activation 'a' of each neuron in such a network represents a linear combination of some input x and a set of learned parameters, w and b followed by an element wise linearity σ :

$$\mathbf{a=\sigma\ (wTx + b)}$$

2.4.1 Artificial Neural Networks

Artificial Neural Networks are the biologically inspired simulations performed on the computer to perform certain specific tasks like clustering, classification, pattern recognition etc.

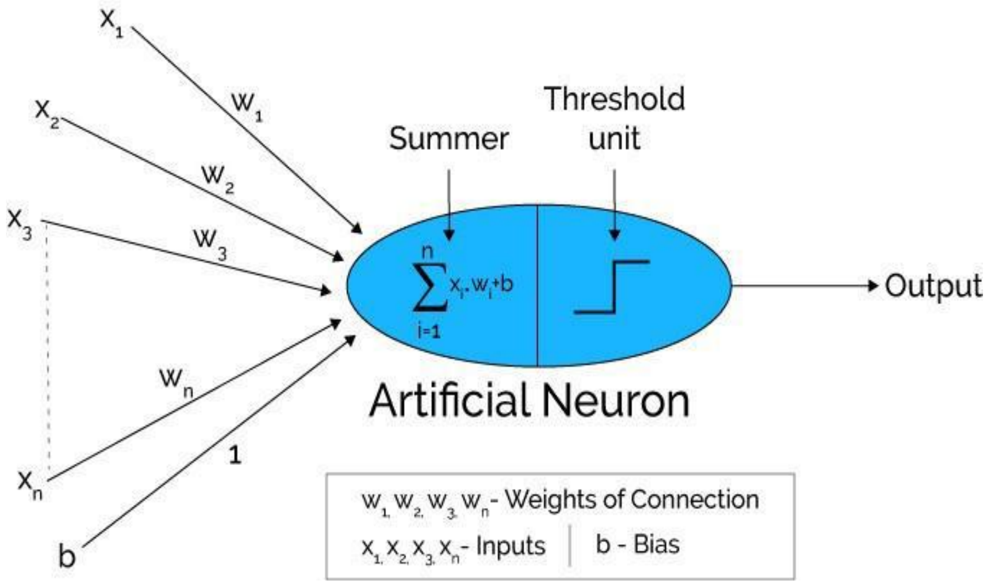

X_1
X_2
X_3
X_n
b
W_1
W_2
W_3
W_n
1
Summer
Threshold unit
$\sum_{i=1}^{n} x_i \cdot w_i + b$
Output
Artificial Neuron
w_1, w_2, w_3, w_n - Weights of Connection
x_1, x_2, x_3, x_n - Inputs
b - Bias

Fig 2.1: Structure of Artificial Neuron

2.4.2 Convolutional Neural Network

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 convolutional neural network (CNN or Convnet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal processing. A CNN consists of input output layer, as well as multiple hidden layers. The hidden layers of CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. Several deep learning software frameworks have appeared to enable efficient development.

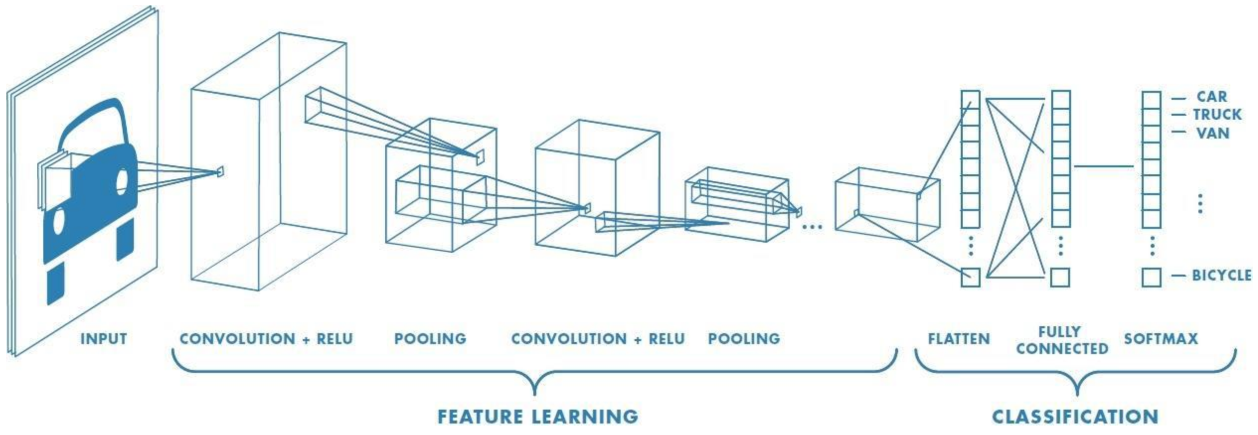


Fig 2.2:CNN architecture

Convolutional Layer

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

Overview and intuition without brain stuff. Let’s first discuss what the CONV layer computes without brain/ neuron analogies. 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. 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 colour on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. We will stack these activation maps along the depth dimension and produce the output volume.

Local Connectivity

When dealing with high-dimensional inputs such as images, as we saw above 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 hyper parameter 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.

To summarize, the Conv Layer

● Accepts a volume of size W1×H1×D1W1×H1×D1

● requires four hyperparameters

○ Number of filters KK,

○ Their spatial extent FF,

○ The stride SS

○ The amount of zero padding PP.

● Produces a volume of size W2×H2×D2W2×H2×D2 where:

○ W2= (W1−F+2P)/S+1W2= (W1−F+2P)/S+1

○ H2= (H1−F+2P)/S+1H2= (H1−F+2P)/S+1 (i.e. width and height are computed equally by symmetry)

○ D2=KD2=K

● With parameter sharing, it introduces F·F·D1F·F·D1 weights per filter, for a total of (F·F·D1)·K(F·F·D1)·K weights and KK biases.

● In the output volume, the dd-th depth slice (of size W2×H2W2×H2) is the result of performing a valid convolution of the dd-th filter over the input volume with a stride of SS, and then offset by dd-th bias.

A common setting of the hyper parameters is F=3,S=1,P=1F=3,S=1,P=1. However, there are common conventions and rules of thumb that motivate these hyperparameters.

Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Convlayers 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 2x2 applied with a stride of 2 downsamples every depth slice 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 2x2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

● Accepts a volume of size W1×H1×D1W1×H1×D1

● Requires two hyperparameters

○ Their spatial extent F

○ The stride S

● Produces a volume of size $W2\times H2\times D2W2\times H2\times D2$ where

  ○ $W2=(W1-F)/S+1W2=(W1-F)/S+1$

  ○ $H2=(H1-F)/S+1H2=(H1-F)/S+1$

  ○ $D2=D1D2=D1$

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.

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. For Image Segmentation fully connected layer is replaced with a convolutional layer.

2.5 Learning Process

Deep convolutional neural networks have led to a series of breakthroughs for image classification. Many other visual recognition tasks have also greatly benefited from very deep models. So, over the years there is a trend to go more deeper, to solve more complex tasks and to also increase /improve the classification/recognition accuracy. But, as we go deeper; the training of neural network becomes difficult and also the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems.

2.5.1 Residual Learning

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

The RNN is represented as

$$Y=x + F(x, WR)$$

As one can distill from this equation, the network only needs to learn the residual F(x, WR). This way, the model is preconditioned towards learning 'simple' parsimonious representations in each layer that are close to the identity function. Function F(x, WR) is known as residual block, for which several variants have been proposed. ResNet architectures have also been combined with inception blocks.

ResNet50 is a 50 layer Residual Network. There are other variants like ResNet101 and ResNet152 also.

CHAPTER – 3
SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 Problem Statement:

Imaging salt has been a huge topic in the seismic industry, basically since they imaged salt the first time. The Society of Exploration geophysicist alone has over 10,000 publications with the keyword salt. Salt bodies are important for the hydrocarbon industry, as they usually form nice oil traps. So there's a clear motivation to delineate salt bodies in the subsurface. Several areas of Earth with large accumulations of oil and gas also have huge deposits of salt below the surface. But unfortunately, knowing where large salt deposits are precisely is very difficult. Professional seismic imaging still requires expert human interpretation of salt bodies. This leads to very subjective, highly variable renderings. More alarmingly, it leads to potentially dangerous situations for oil and gas company drillers. One the main challenges of seismic imaging is to identify the salt deposits present in the seismic images.

The objective is to build an algorithm that can detect the salt regions or salt deposits present in the seismic images. Specifically the algorithm needs to locate the regions which only consists of salt and not the remaining materials like oil and natural gases.

3.2 Existing Approach:

A manual method of predicting or spotting the salt deposits by experts is done till now. But this might be time taking and waste of valuable time of experts. This may not be accurate enough all the time and if they are plenty images to be predicted or analyzed they get stacked leading to lag in work.

3.3 Proposed System:

We are going to use U-net model. To train the U-net model we are giving 4000 images and testing on the 1800 images. U-net is a segmentation architecture which accepts input images in the power of 2 pixels. So we are converting our 101*101 gray scale images into the 128*128 sized images. We are trying to build a model in keras and tensorflow environment by using kaggle kernel platforms. An inbuilt optimizer ‘adam’, an inbuilt binary_cross_entropy loss function and mean_iou metrics are used to build the model.

3.3 Requirements

Processor: CPU or GPU
RAM: 8GB or more

Environment : Kaggle kernel environment

GPUs are highly parallel computing engines, which have an order of magnitude more execution threads than CPUs. Deep learning on GPUs is 10 to 30 times faster than on CPUs. Next to hardware developments, the other driving force for deep learning is wide availability of open source packages. These libraries provide efficient GPU implementations of important operations in neural networks.

3.4 System Flow

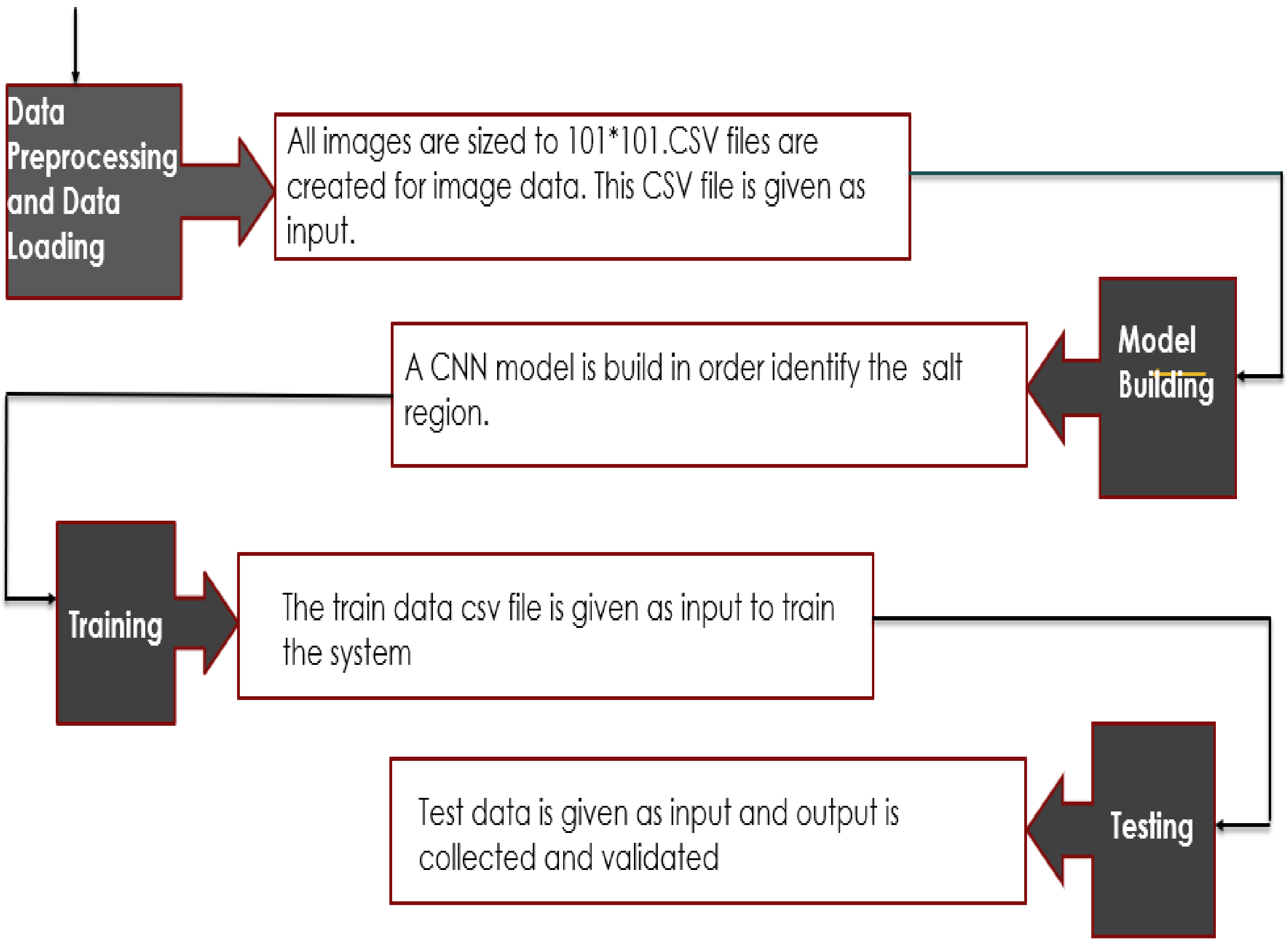

Data Preprocessing and Data Loading
All images are sized to 101*101.CSV files are created for image data. This CSV file is given as input.
Model Building
A CNN model is build in order identify the salt region.
Training
The train data csv file is given as input to train the system
Testing
Test data is given as input and output is collected and validated

Figure 3.1: System Flow

CHAPTER – 4 METHODOLOGY

4. METHODOLOGY

4.1 Segmentation model

In semantic segmentation, to get a finer result, it is very important to use low level details while retaining high level semantic information. However, training such a deep neural network is very hard especially when only limited training samples are available. One way to solve this problem is employing a pre-trained network then fine tuning it on the target dataset. Another way is employing extensive data augmentation, as done in U-Net. In addition to data augmentation, the architecture of U-Net also contributes to relieving the training problem. The intuition behind this is that copying low level features to the corresponding high levels actually creates a path for information propagation allowing signals propagate between low and high levels in a much easier way, which not only facilitating backward propagation during training, but also compensating low level finer details to high level semantic features.

From the study of various papers and analyzing each model, we decided to use U-net.

There are different deep learning libraries that can be used for training purpose. Some of them are Caffe, DeepLearning4.J deepmat, Eblearn, Neon, PyLearn, TensorFlow, Theano, Torch etc. The libraries TensorFlow and Theano are made easier by introducing KERAS. It provides an API where the programmer can simply write functions without any knowledge of its actual executing process. Keras internally process the code using either TensorFlow or Theano. The application under consideration uses Tensor Flow with Keras as it received much attention since its first release.

U-Net is a Fully Convolutional Network (FCN) that does image segmentation. Its goal is then to predict each pixel’s class.

The U-Net architecture is built upon the Fully Convolutional Network and modified in a way that it yields better segmentation in imaging tasks.

Compared to FCN-8, the two main differences are

(1) U-net is symmetric

(2) the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum. These skip connections intend to provide local information to the global information while upsampling.

Because of its symmetry, the network has a large number of feature maps in the upsampling path, which allows to transfer information. By comparison, the basic FCN architecture only had *number of classes* feature maps in its upsampling path.

NETWORK ARCHITECTURE:

The U-Net owes its name to its symmetric shape, which is different from other FCN variants.

U-Net architecture is separated in 3 parts:

- 1 : The contracting/downsampling path
- 2 : Bottleneck
- 3 : The expanding/upsampling path

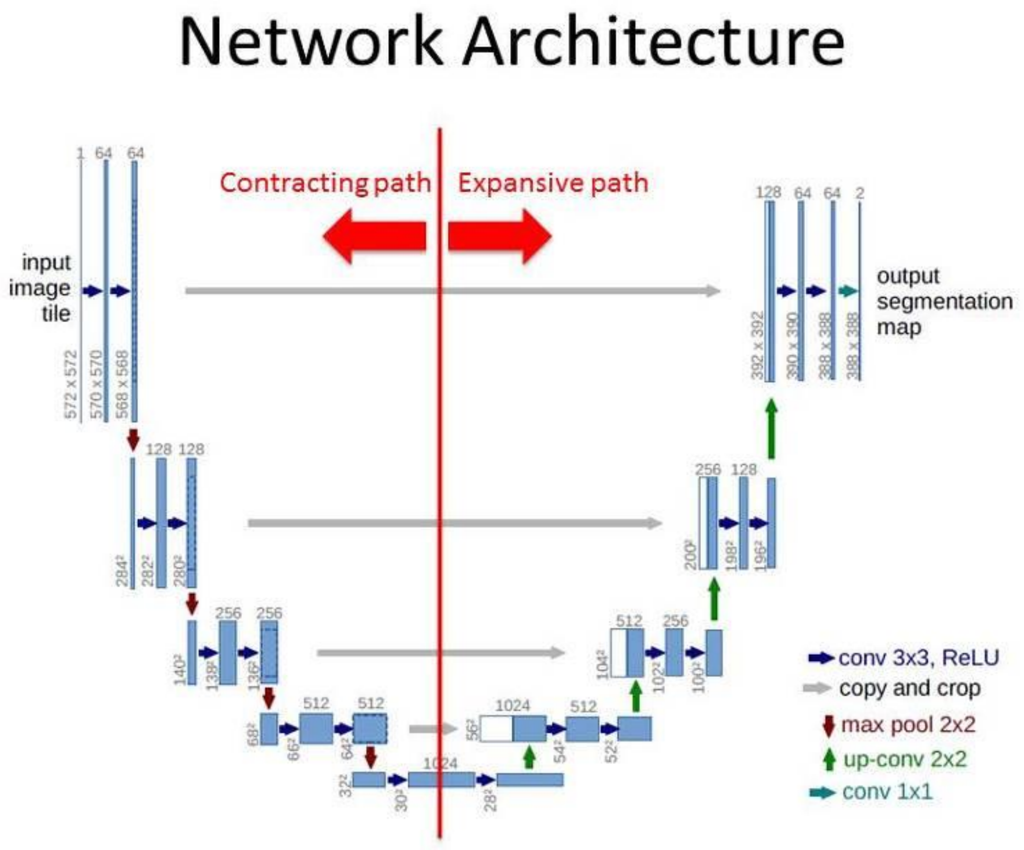


Figure 4.1 : Illustration of U-Net architecture (from U-Net paper)

NOTE: Input image size is taken as 101*101

Contracting/downsampling path:

The contracting path is composed of 4 blocks. Each block is composed of

- 3x3 Convolution Layer + activation function (with batch normalization)
- 3x3 Convolution Layer + activation function (with batch normalization)
- 2x2 Max Pooling

Note that the number of feature maps doubles at each pooling, starting with 64 feature maps for the first block, 128 for the second, and so on. The purpose of this contracting path is to capture the context of the input image in order to be able to do segmentation. This coarse contextual information will then be transfered to the upsampling path by means of skip connections.

Bottleneck:

This part of the network is between the contracting and expanding paths. The bottleneck is built from simply 2 convolutional layers (with batch normalization), with dropout.

Expanding/upsampling path:

The expanding path is also composed of 4 blocks. Each of these blocks is composed of

- Deconvolution layer with stride 2
- Concatenation with the corresponding cropped feature map from the contracting path
- 3x3 Convolution layer + activation function (with batch normalization)
- 3x3 Convolution layer + activation function (with batch normalization)

The purpose of this expanding path is to enable precise localization combined with contextual information from the contracting path.

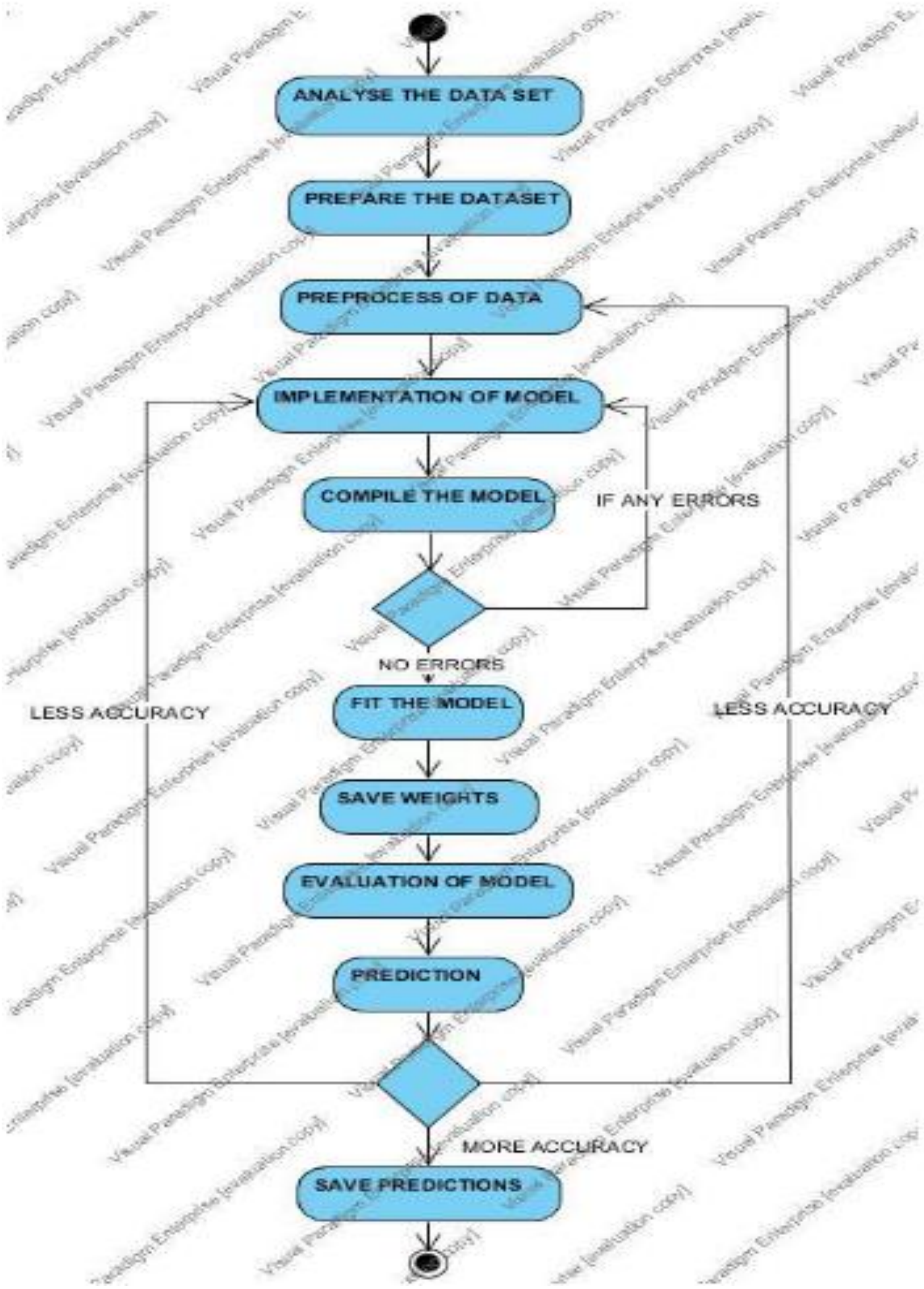


Fig 4.2 : Activity diagram

4.2 Software Analysis

4.2.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, and a syntax that allows programmers to express concepts in fewer lines of code, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

4.2.2 KERAS

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MxNet. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.

It also allows use of distributed training of deep learning models on clusters of Graphics Processing Units (GPU).

Keras helps us with-

- easy and fast prototyping.
- supporting both convolution networks and recurrent networks, as well as combination of both.
- runs on CPU and GPU.

4.2.3 TENSORFLOW

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. These arrays are referred to as "tensors". TensorFlow provides a python API as well as C++, Java, and Rust API's. Third party packages are also available for C#, and R etc.

Among the applications for which TensorFlow is the foundation, are automated image captioning software, such as DeepDream. RankBrain now handles a substantial number of search queries, replacing and supplementing traditional static algorithm-based search results.

4.3 Kaggle Kernels

Kaggle Kernels is a cloud computational environment that enables reproducible and collaborative analysis. Kernels supports scripts in R and Python, Jupyter Notebooks, and RMarkdown reports.

Kernel runs using Docker containers. For every user, it mounts the input to the container with docker images preloaded with the most common data science languages and libraries. It can be considered a script/notebook with environment/data already setup. Advantages are as follows,

1. Docker containerization allows easy setup of kaggle projects.
2. No requirement to download data due to usage of same common volume mounted into the container.
3. Allows, user to share and learn from different approaches.

4.4 Jupyter notebook:

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

4.5 Numpy:

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

4.6 Pandas:

Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.Pandas library uses most of the functionalities of NumPy.

4.7 Mean_iou:

Intersection over Union

The Intersection over Union (IoU) metric, also referred to as the Jaccard index, is essentially a method to quantify the percent overlap between the target mask and our prediction output. This metric is closely related to the Dice coefficient which is often used as a loss function during training.
Quite simply, the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across *both*masks.

$$ IoU = \frac{{target \cap prediction}}{{target \cup prediction}} $$

As a visual example, let's suppose we're tasked with calculating the IoU score of the following prediction, given the ground truth labeled mask.

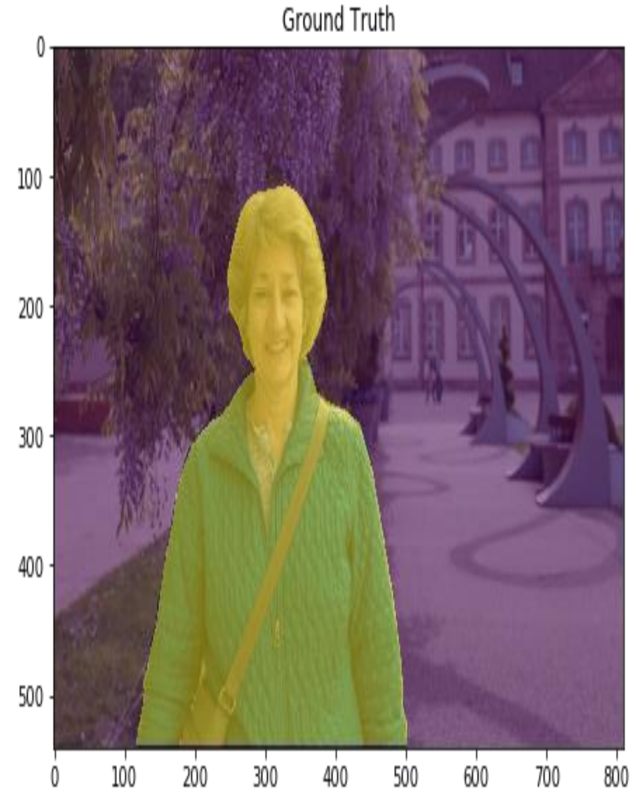


Figure 4.2:Ground Truth Image

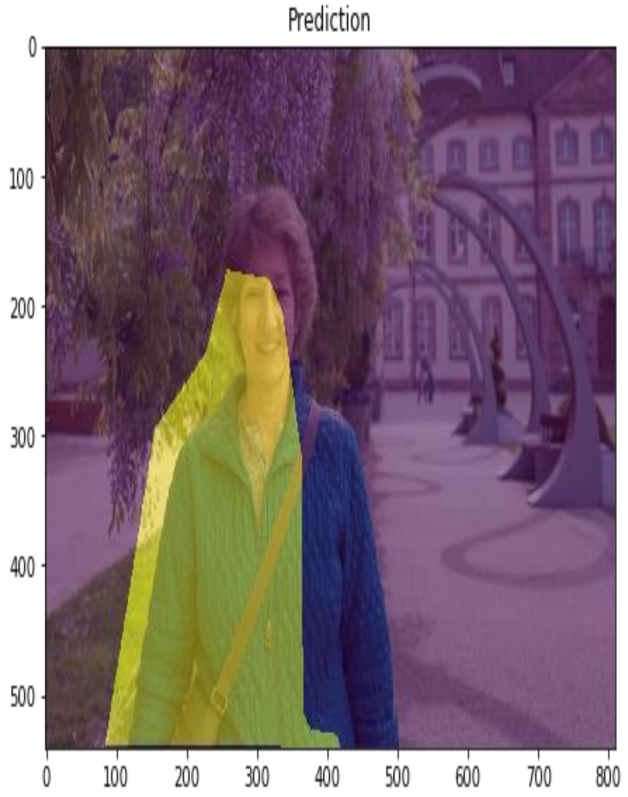


Figure 4.3:Predicted Image

The **intersection** ($A \cap B$) is comprised of the pixels found in both the prediction mask *and* the ground truth mask, whereas the **union** ($A \cup B$) is simply comprised of all pixels found in either the prediction *or* target mask.

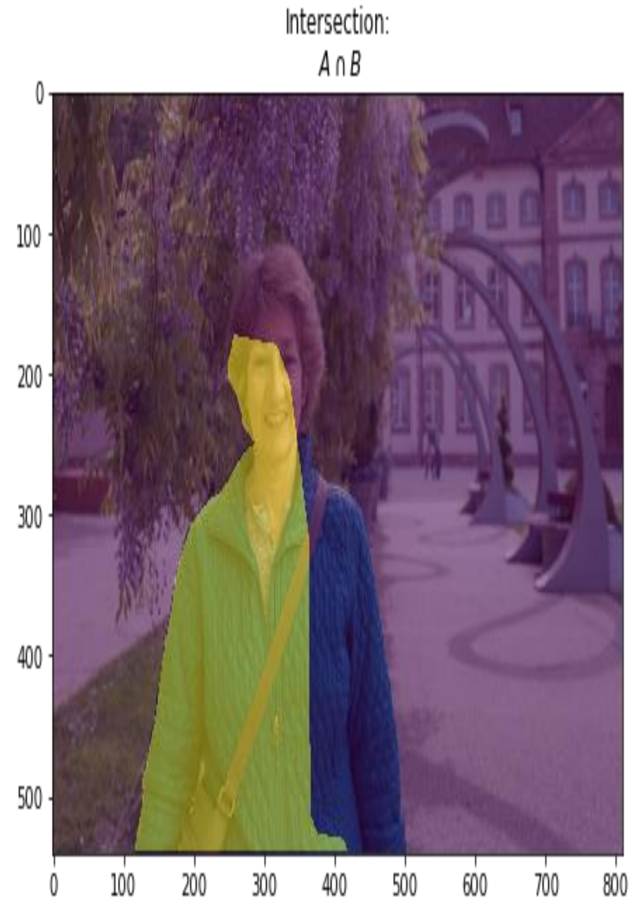


Figure 4.4: Intersection Image

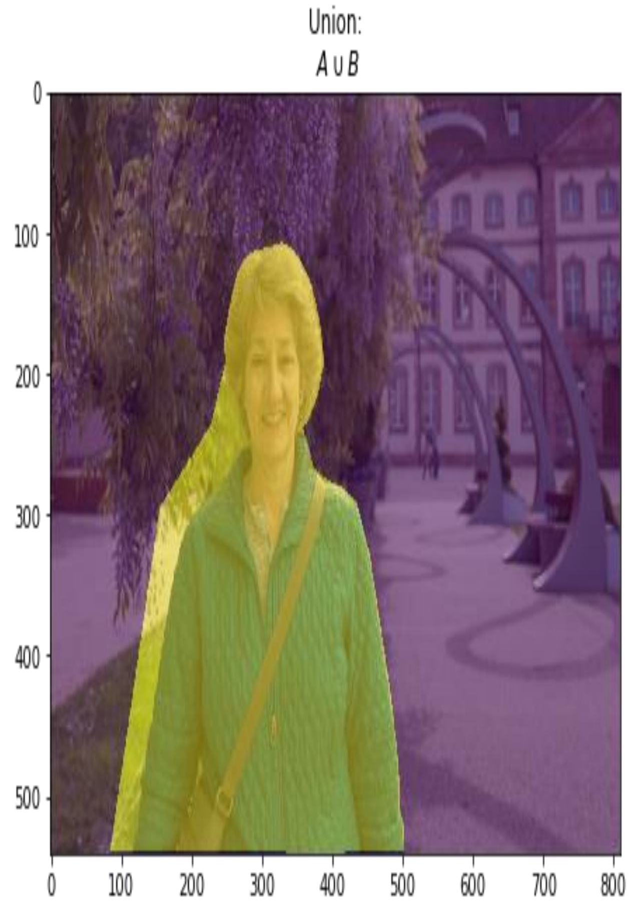
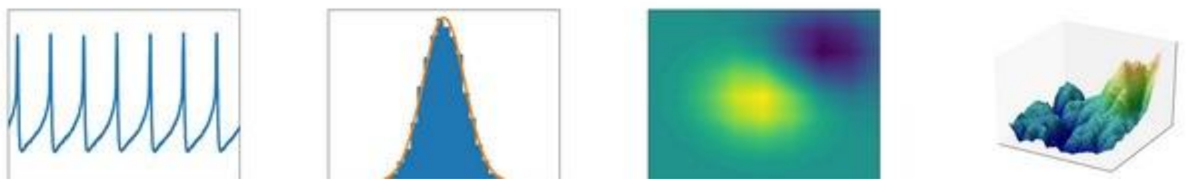


Figure 4.5 Union Image

4.8 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.



Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

CHAPTER – 5 IMPLEMENTATION

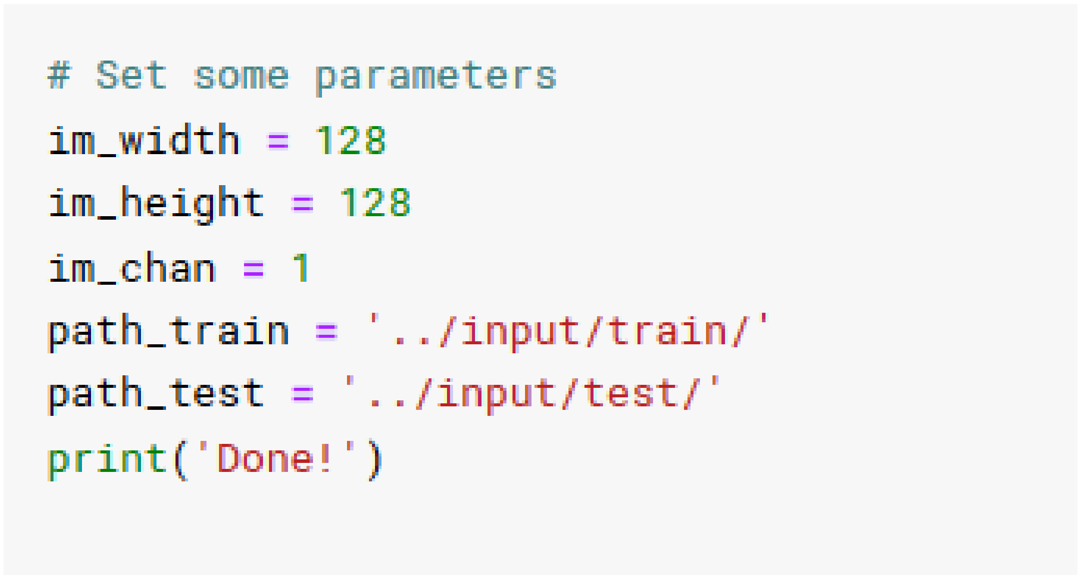
5. IMPLEMENTATION

5.1 Creating new kernal:

1. https://www.kaggle.com/competitions
2. join the TGS salt identification challenge using Gmail
3. click on the kernels and add new kernel
4. Click On New kernel button (on right side of the screen)
5. Click on Notebook (if notebook is required else click script)
6. Enter a title for your notebook
7. Delete the existing code in the first cell and try printing a test message
8. Commit the notebook (its like saving the file to see the output)
9. View the output by executing the cell

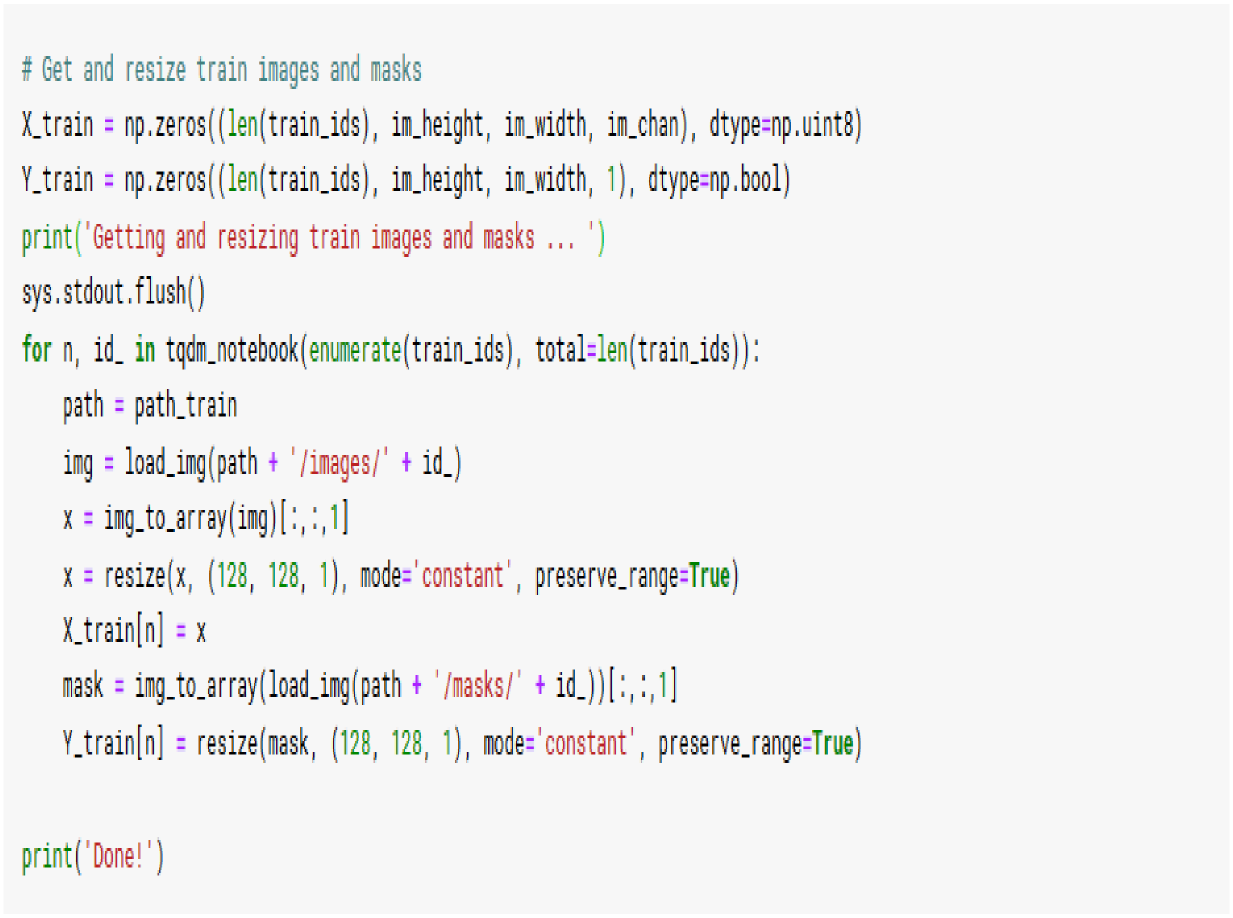
By using kaggle kernals the advantage is that the data of the predefined problem is already available in the input directory. If we want to add extra dataset then also we can add it.

Setting some parameters

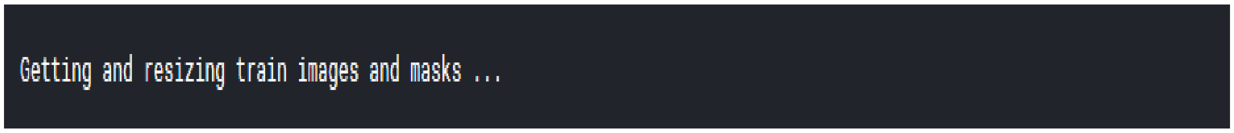

```
# Set some parameters
im_width = 128
im_height = 128
im_chan = 1
path_train = '../input/train/'
path_test = '../input/test/'
print('Done!')
```

5.2 Preprocessing:

We have images of shape (101,101,1) in gray scale. So we first need to extract those images as a pixel array and the size of image for training the model is(128,128,1). So we need to resize the images to 128*128.



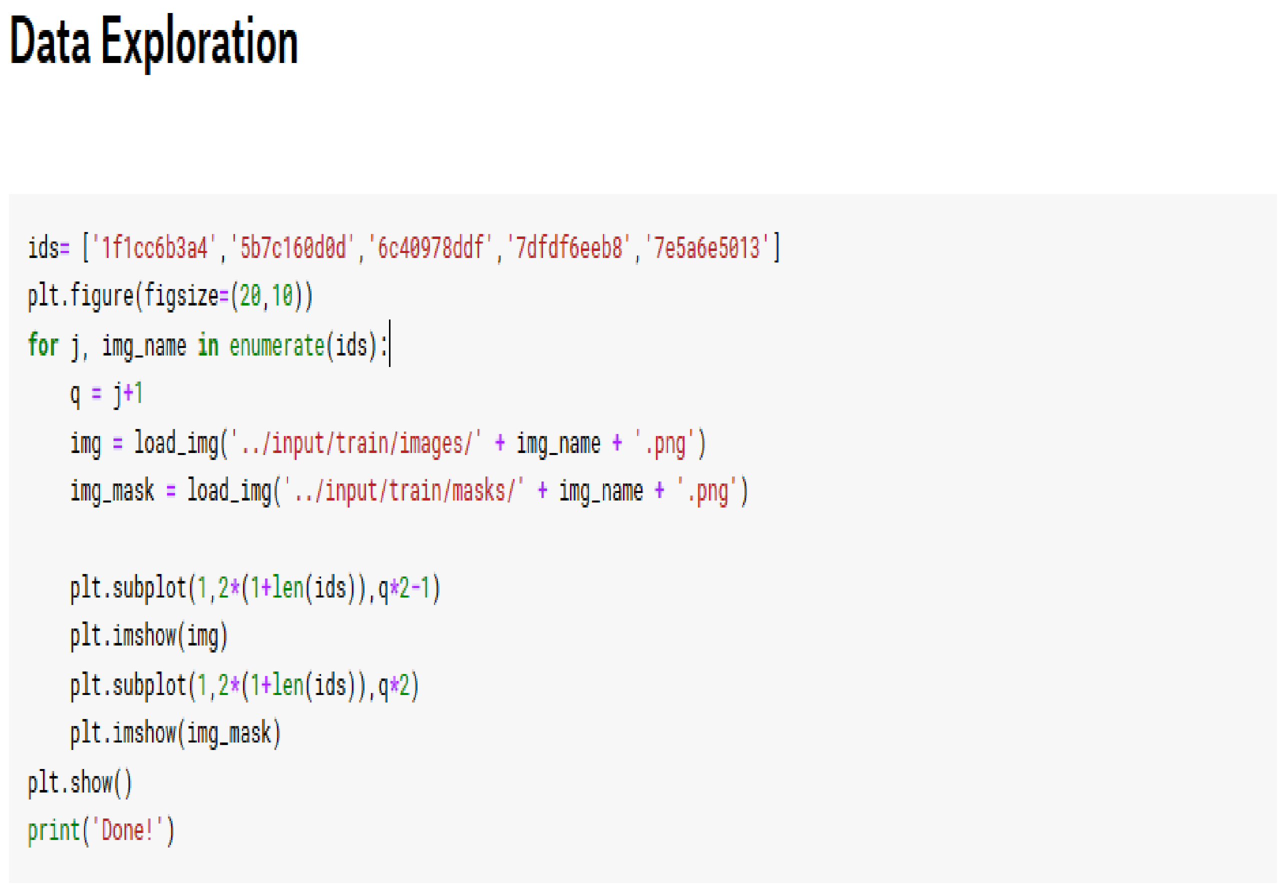
```
# Get and resize train images and masks
X_train = np.zeros((len(train_ids), im_height, im_width, im_chan), dtype=np.uint8)
Y_train = np.zeros((len(train_ids), im_height, im_width, 1), dtype=np.bool)
print('Getting and resizing train images and masks ... ')
sys.stdout.flush()
for n, id_ in tqdm_notebook(enumerate(train_ids), total=len(train_ids)):
    path = path_train
    img = load_img(path + '/images/' + id_)
    x = img_to_array(img)[:,:,1]
    x = resize(x, (128, 128, 1), mode='constant', preserve_range=True)
    X_train[n] = x
    mask = img_to_array(load_img(path + '/masks/' + id_))[:,:,1]
    Y_train[n] = resize(mask, (128, 128, 1), mode='constant', preserve_range=True)

print('Done!')
```



```
Getting and resizing train images and masks ...
```

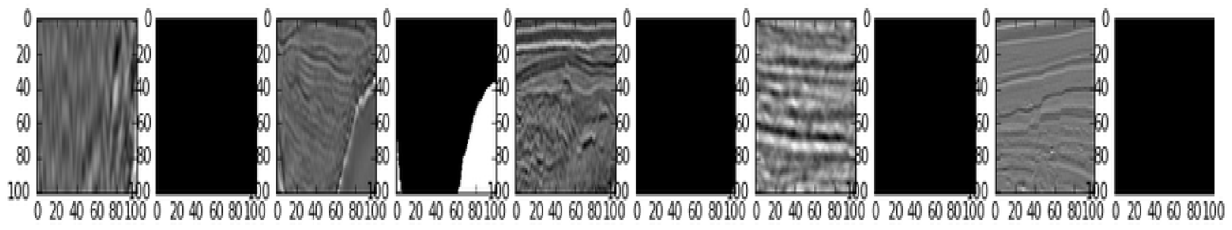
5.3 Data exploration:

After preprocessing the images some of the train image ids are taken and their respective mask images are plotted by using the matplotlib functions like imshow and subplot.



Data Exploration

```
ids= ['1f1cc6b3a4','5b7c160d0d','6c40978ddf','7dfdf6eeb8','7e5a6e5013']
plt.figure(figsize=(20,10))
for j, img_name in enumerate(ids):
    q = j+1
    img = load_img('../input/train/images/' + img_name + '.png')
    img_mask = load_img('../input/train/masks/' + img_name + '.png')

    plt.subplot(1,2*(1+len(ids)),q*2-1)
    plt.imshow(img)
    plt.subplot(1,2*(1+len(ids)),q*2)
    plt.imshow(img_mask)
plt.show()
print('Done!')
```



5.4 Building Segmentation Model

```
# Build U-Net model
inputs = Input((im_height, im_width, im_chan))
s = Lambda(lambda x: x / 255) (inputs)

c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (s)
c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (p1)
c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (p2)
c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (p3)
c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(128, (3, 3), activation='relu', padding='same') (p4)
c5 = Conv2D(128, (3, 3), activation='relu', padding='same') (c5)

u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (u6)
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (c6)
```

```
u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (u7)
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (c7)

u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (u8)
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (c8)

u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (u9)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[mean_iou])
model.summary()
```

Lambda Layer:
keras.layers.Lambda(function, output_shape=**None**, mask=**None**, arguments=**None**)

function: The function to be evaluated. Takes input tensor or list of tensors as first argument.

Input layer:
used to instantiate a Keras tensor simply an entry point.
A Keras tensor is a tensor object from the underlying backend (Theano, TensorFlow or CNTK), which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

```
Input(shape = NULL, batch_shape = NULL, name = NULL,
  dtype = NULL, sparse = FALSE, tensor = NULL)
```

Arguments:

shape	Shape, not including the batch size. For instance, shape=c(32) indicates that the expected input will be batches of 32-dimensional vectors.
batch_shape	Shape, including the batch size. For instance, shape = c(10,32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_shape = list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
name	An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.
dtype	The data type expected by the input, as a string (float32, float64, int32...)
sparse	Boolean, whether the placeholder created is meant to be sparse.
tensor	Existing tensor to wrap into the Input layer. If set, the layer will not create a placeholder tensor.

Returns

A tensor.

Conv2D layer:

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid',
data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)
```

Arguments

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).Here taken level wise 8 16 32 64..

- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Here (3,3) is taken.
- **activation**: Activation function to use .If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).Here rectified linear unit (**ReLU**) is used because of its efficient computational power, no expensive exponential operations and better convergence performance than other functions.

padding: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1. VALID =without padding and SAME= with zero padding.

MaxPooling2D:

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=**None**, padding='valid', data_format=**None**)

Used to perform Max pooling operation for spatial data.
Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

Arguments

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

Conv2DTranspose

keras.layers.Conv2DTranspose(filters, kernel_size, strides=(1, 1), padding='valid', output_padding=**None**, data_format=**None**, dilation_rate=(1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**).

Transposed convolution layer (sometimes called Deconvolution).The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution.

Arguments

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).64 32 16 … are used here.

- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.(2,2) is used here.
- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).Here rectified linear unit (**ReLU**) is used because of its efficient computational power,no expensive exponential operations and better convergence performance than other functions.
- **padding**: one of "valid" or "same" (case-insensitive). VALID =without padding and SAME= with zero padding.

Concatenate

keras.backend.concatenate(tensors, axis=-1)

Concatenates a list of tensors alongside the specified axis.

Arguments

tensors: list of tensors to concatenates.
axis: concatenation axis.

Returns

A tensor.

COMPILE:

Configures the model for training. Given a set of training images and the corresponding ground truth segmentations, our goal is to estimate parameters of the network, such that it produces accurate and robust results. This is achieved through minimizing the loss between the segmentations generated by model and the ground truth values. Using small batches of random data is called stochastic training -- in this case, stochastic gradient descent. The cross-entropy measures how inefficient our predictions are for describing the truth. The loss function used is binary_crossentropy cross entropy as we consider binary classification. The metrics are the measures used to evaluate the model based on particular set of values. Here we use MEAN_IOU as metric. The compile function in keras is given as:

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=[mean_iou])

optimizer:

- An optimizer. This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the Optimizer class.

Adam:

This is used to perform optimization and is one of the best optimizer at present. The author claims that it inherits from RMSProp and AdaGrad (Well it inherits from them).

Features:

- Parameters update are invariant to re-scaling of gradient—It means that if we have some objective function f(x) and we change it to k*f(x) (where k is some constant). There will be no effect on performance. We will come to it later.
- The step-size is **approximately** bounded by the step-size hyper-parameter. (Notice that something is in bold here!).

- It doesn't require stationary objective. That means the f(x) we talked about might change with time and still the algorithm will converge.
- Naturally performs step size annealing. Well remember the classical SGD, we used to decrease step size after some epochs, nothing as such is needed here.

keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=**None**, decay=0.0, amsgrad=**False**)

Arguments:

- **lr**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If None, defaults to K.epsilon().
- **decay**: float >= 0. Learning rate decay over each update.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm.

Loss function:

A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function or it can be an objective function.

Binary_crossentropy:

Binary_crossentropy is the default loss function to use for binary classification problems. It is intended for use with binary classification where the target values are in the set {0, 1}. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0. Cross-entropy can be specified as the loss function in Keras by specifying '*binary_crossentropy*' when compiling the model.

Function:
keras.losses.binary_crossentropy(y_true, y_pred)

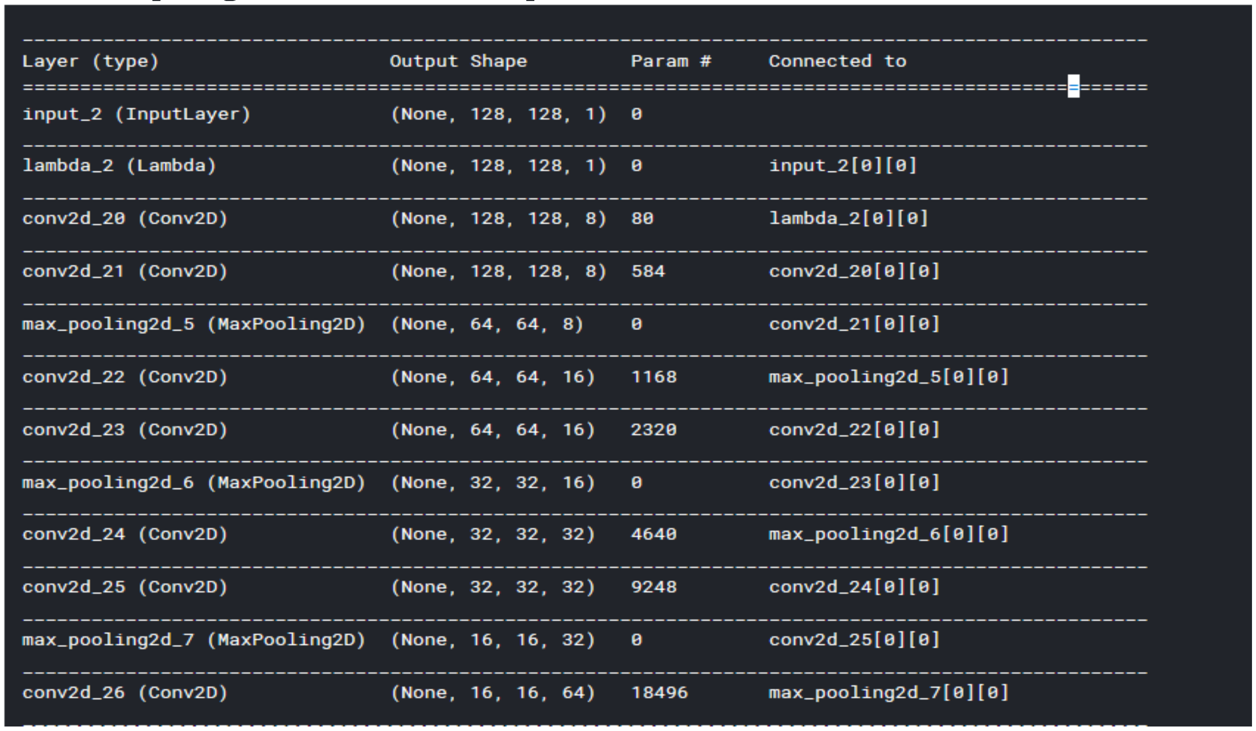
Arguments:

  - **y_true**: True labels. TensorFlow/Theano tensor.
  - **y_pred**: Predictions. TensorFlow/Theano tensor of the same shape as y_true.
- A list of metrics. A metric could be the string identifier of an existing metric or a custom metric function,mean_iou(intersection over union) is used here.

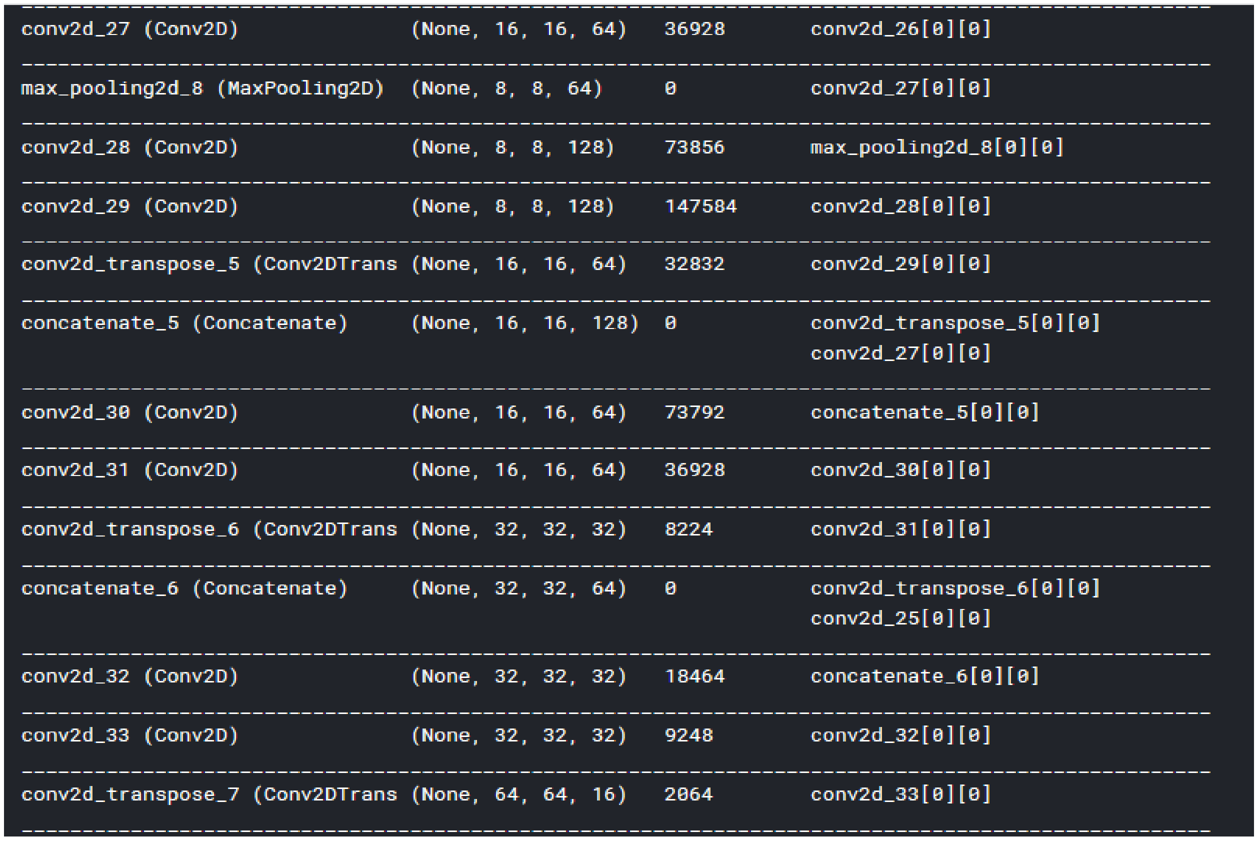
Model.summary():

Using to_json() or to_yaml() is not user friendly for quickly browsing a model, the output of .summary() is at the correct level. Having .summary() return a string is useful for logging.

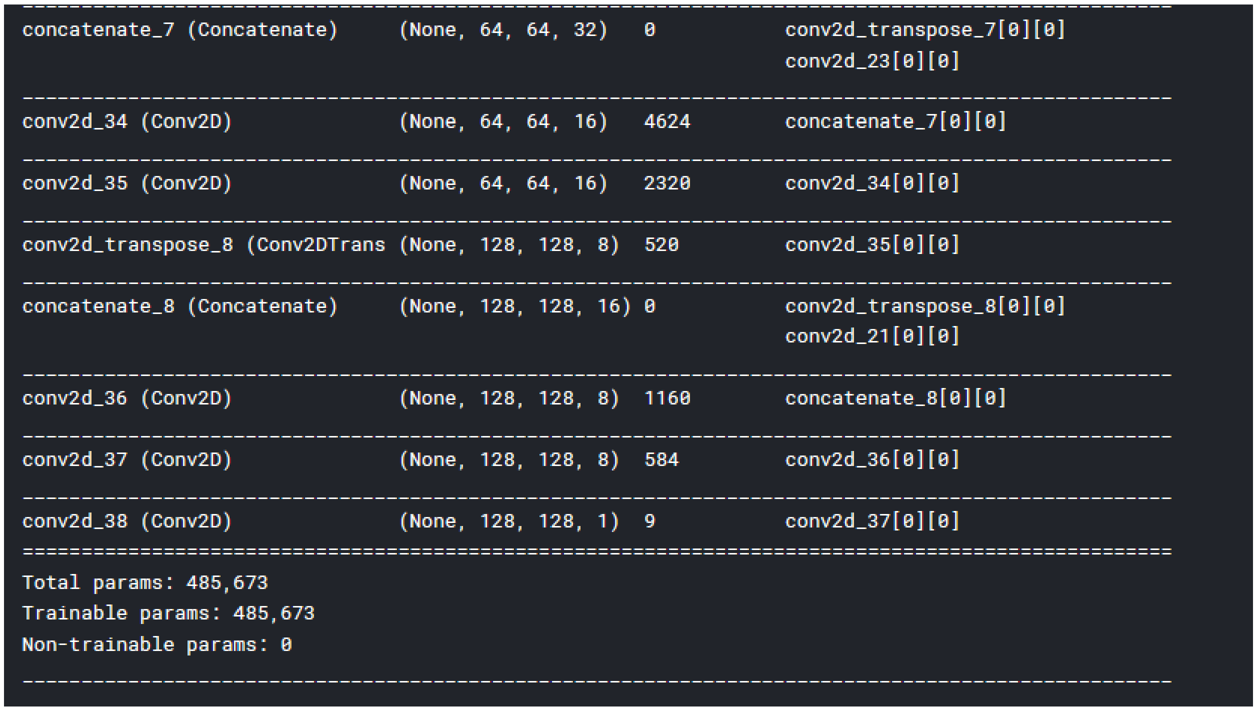
After compiling the model the output is



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 128, 128, 1)  0
__________________________________________________________________________________________________
lambda_2 (Lambda)               (None, 128, 128, 1)  0           input_2[0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 128, 128, 8)  80          lambda_2[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 128, 128, 8)  584         conv2d_20[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 64, 64, 8)    0           conv2d_21[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 64, 64, 16)   1168        max_pooling2d_5[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 64, 64, 16)   2320        conv2d_22[0][0]
__________________________________________________________________________________________________
max_pooling2d_6 (MaxPooling2D)  (None, 32, 32, 16)   0           conv2d_23[0][0]
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 32, 32, 32)   4640        max_pooling2d_6[0][0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 32, 32, 32)   9248        conv2d_24[0][0]
__________________________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)  (None, 16, 16, 32)   0           conv2d_25[0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 16, 16, 64)   18496       max_pooling2d_7[0][0]
__________________________________________________________________________________________________
```



```
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 16, 16, 64)   36928       conv2d_26[0][0]
__________________________________________________________________________________________________
max_pooling2d_8 (MaxPooling2D)  (None, 8, 8, 64)     0           conv2d_27[0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 8, 8, 128)    73856       max_pooling2d_8[0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 8, 8, 128)    147584      conv2d_28[0][0]
__________________________________________________________________________________________________
conv2d_transpose_5 (Conv2DTrans (None, 16, 16, 64)   32832       conv2d_29[0][0]
__________________________________________________________________________________________________
concatenate_5 (Concatenate)     (None, 16, 16, 128)  0           conv2d_transpose_5[0][0]
                                                                 conv2d_27[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 16, 16, 64)   73792       concatenate_5[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 16, 16, 64)   36928       conv2d_30[0][0]
__________________________________________________________________________________________________
conv2d_transpose_6 (Conv2DTrans (None, 32, 32, 32)   8224        conv2d_31[0][0]
__________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 32, 32, 64)   0           conv2d_transpose_6[0][0]
                                                                 conv2d_25[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 32, 32, 32)   18464       concatenate_6[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 32, 32, 32)   9248        conv2d_32[0][0]
__________________________________________________________________________________________________
conv2d_transpose_7 (Conv2DTrans (None, 64, 64, 16)   2064        conv2d_33[0][0]
__________________________________________________________________________________________________
```



```
concatenate_7 (Concatenate)     (None, 64, 64, 32)   0          conv2d_transpose_7[0][0]
                                                                conv2d_23[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 64, 64, 16)   4624       concatenate_7[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 64, 64, 16)   2320       conv2d_34[0][0]
__________________________________________________________________________________________________
conv2d_transpose_8 (Conv2DTrans (None, 128, 128, 8)  520        conv2d_35[0][0]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 128, 128, 16) 0          conv2d_transpose_8[0][0]
                                                                conv2d_21[0][0]
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 128, 128, 8)  1160       concatenate_8[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 128, 128, 8)  584        conv2d_36[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 128, 128, 1)  9          conv2d_37[0][0]
==================================================================================================
Total params: 485,673
Trainable params: 485,673
Non-trainable params: 0
__________________________________________________________________________________________________
```

5.5 Training :

The machine must to be trained with the defined model. The weights/values of feature maps calculated from the given dataset during training phase are saved for future prediction.

The steps of training with a model include

```
earlystopper = EarlyStopping(patience=5, verbose=1)
checkpointer = ModelCheckpoint('model-tgs-salt-1.h5', verbose=1 , save_best_only=True)
results = model.fit(X_train, Y_train, validation_split=0.1, batch_size=4, epochs=30,
                    callbacks=[earlystopper, checkpointer])
```

CALLBACKS:

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes. The relevant methods of the callbacks will then be called at each stage of the training.

EARLYSTOPPING:

Early stopping is a technique for controlling overfitting in machine learning models, especially **neural networks**, by **stopping** training before the weights have converged. Often we **stop** when the performance has **stopped** improving on a held-out validation set.

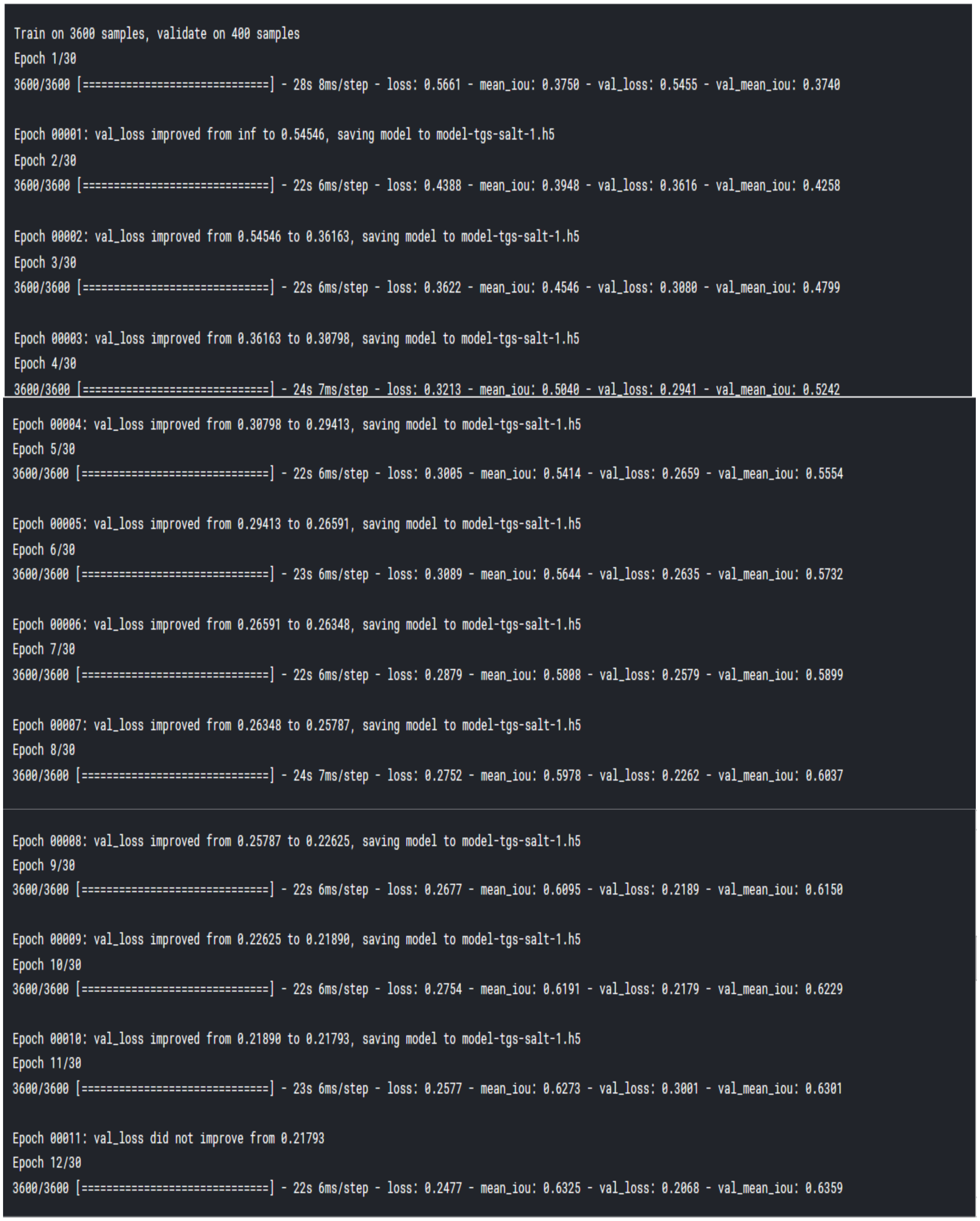
ModelCheckpoint:
Save the model after every epoch.
For example: if filepath is weights **model-tgs-salt-1.h5**, then the model checkpoints will be saved with the epoch number and the validation loss in the filename.

FIT:

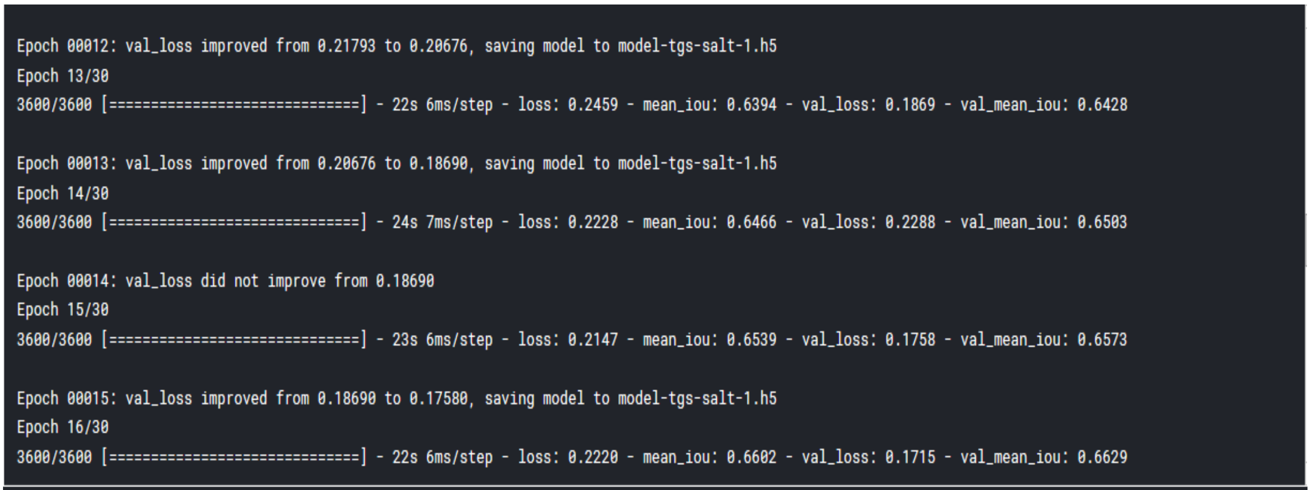
Trains the model for a fixed number of epochs (iterations on a dataset). Takes training and validation images as parameter inputs along with the batch size and no of epochs. The keras implementation of fit function is be shown as

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None, validation_freq=1)
```

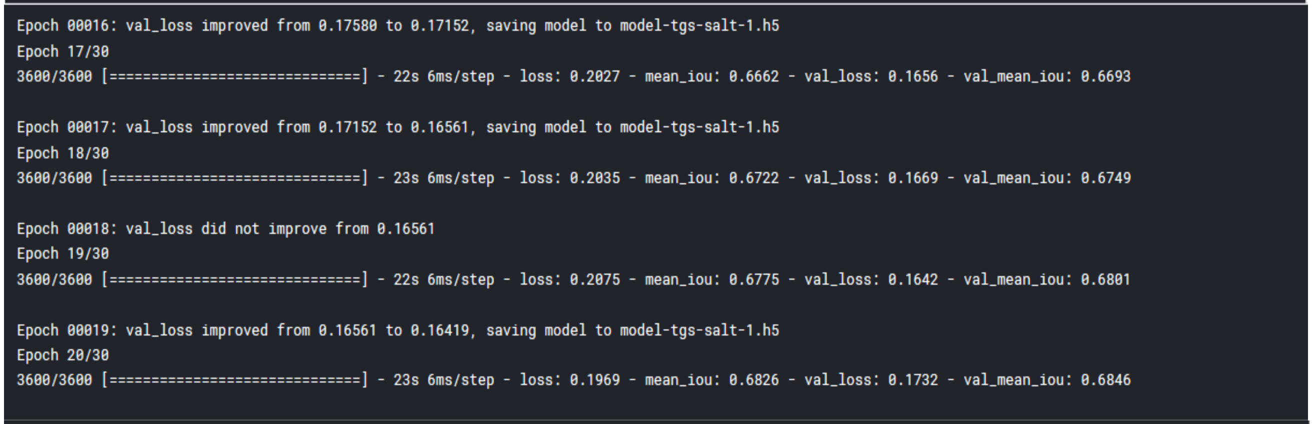
After executing the above one the output is



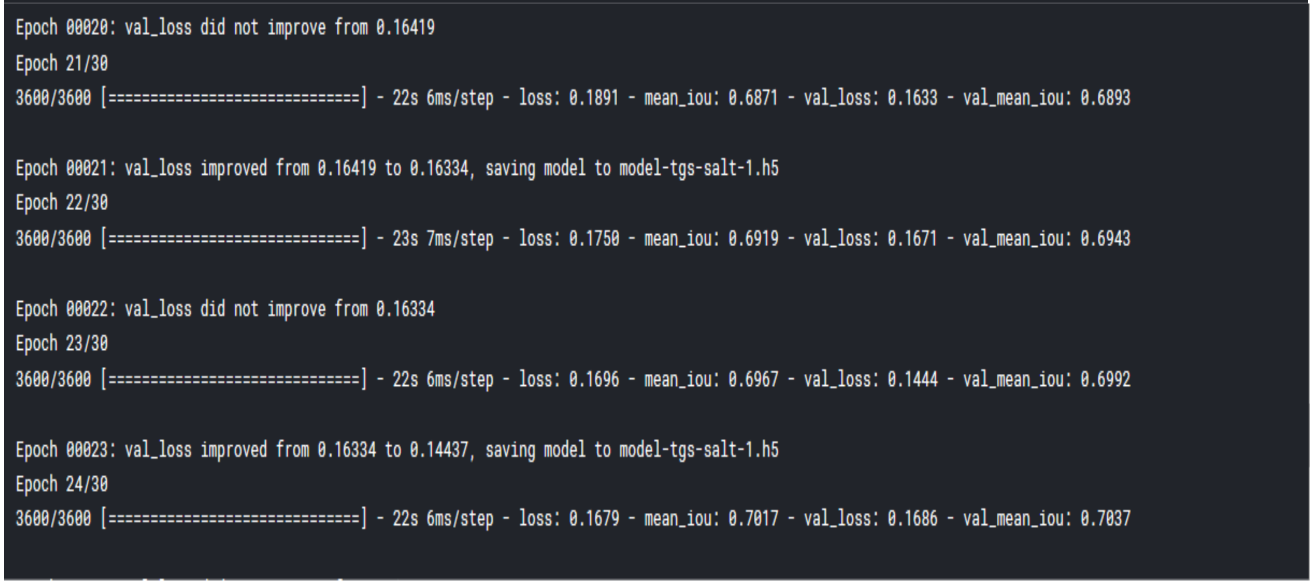
```
Train on 3600 samples, validate on 400 samples
Epoch 1/30
3600/3600 [==============================] - 28s 8ms/step - loss: 0.5661 - mean_iou: 0.3750 - val_loss: 0.5455 - val_mean_iou: 0.3740

Epoch 00001: val_loss improved from inf to 0.54546, saving model to model-tgs-salt-1.h5
Epoch 2/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.4388 - mean_iou: 0.3948 - val_loss: 0.3616 - val_mean_iou: 0.4258

Epoch 00002: val_loss improved from 0.54546 to 0.36163, saving model to model-tgs-salt-1.h5
Epoch 3/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.3622 - mean_iou: 0.4546 - val_loss: 0.3080 - val_mean_iou: 0.4799

Epoch 00003: val_loss improved from 0.36163 to 0.30798, saving model to model-tgs-salt-1.h5
Epoch 4/30
3600/3600 [==============================] - 24s 7ms/step - loss: 0.3213 - mean_iou: 0.5040 - val_loss: 0.2941 - val_mean_iou: 0.5242
Epoch 00004: val_loss improved from 0.30798 to 0.29413, saving model to model-tgs-salt-1.h5
Epoch 5/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.3005 - mean_iou: 0.5414 - val_loss: 0.2659 - val_mean_iou: 0.5554

Epoch 00005: val_loss improved from 0.29413 to 0.26591, saving model to model-tgs-salt-1.h5
Epoch 6/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.3089 - mean_iou: 0.5644 - val_loss: 0.2635 - val_mean_iou: 0.5732

Epoch 00006: val_loss improved from 0.26591 to 0.26348, saving model to model-tgs-salt-1.h5
Epoch 7/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2879 - mean_iou: 0.5808 - val_loss: 0.2579 - val_mean_iou: 0.5899

Epoch 00007: val_loss improved from 0.26348 to 0.25787, saving model to model-tgs-salt-1.h5
Epoch 8/30
3600/3600 [==============================] - 24s 7ms/step - loss: 0.2752 - mean_iou: 0.5978 - val_loss: 0.2262 - val_mean_iou: 0.6037

Epoch 00008: val_loss improved from 0.25787 to 0.22625, saving model to model-tgs-salt-1.h5
Epoch 9/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2677 - mean_iou: 0.6095 - val_loss: 0.2189 - val_mean_iou: 0.6150

Epoch 00009: val_loss improved from 0.22625 to 0.21890, saving model to model-tgs-salt-1.h5
Epoch 10/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2754 - mean_iou: 0.6191 - val_loss: 0.2179 - val_mean_iou: 0.6229

Epoch 00010: val_loss improved from 0.21890 to 0.21793, saving model to model-tgs-salt-1.h5
Epoch 11/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.2577 - mean_iou: 0.6273 - val_loss: 0.3001 - val_mean_iou: 0.6301

Epoch 00011: val_loss did not improve from 0.21793
Epoch 12/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2477 - mean_iou: 0.6325 - val_loss: 0.2068 - val_mean_iou: 0.6359
```



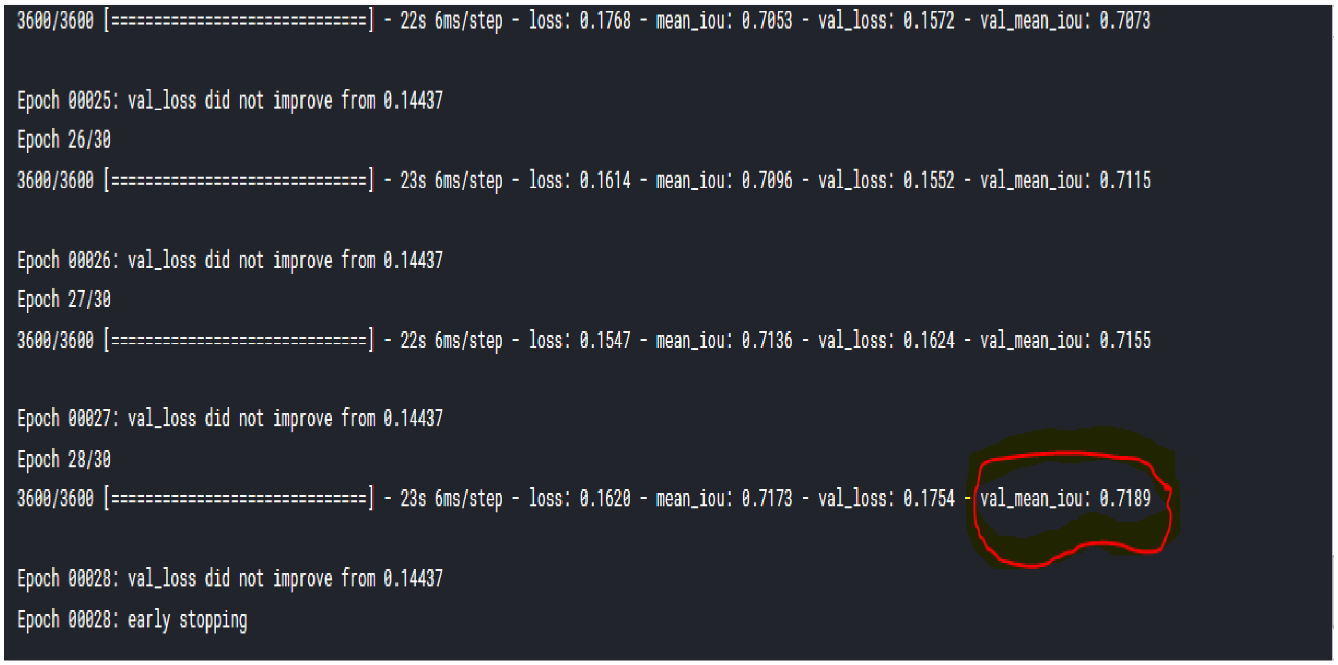
```
Epoch 00012: val_loss improved from 0.21793 to 0.20676, saving model to model-tgs-salt-1.h5
Epoch 13/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2459 - mean_iou: 0.6394 - val_loss: 0.1869 - val_mean_iou: 0.6428

Epoch 00013: val_loss improved from 0.20676 to 0.18690, saving model to model-tgs-salt-1.h5
Epoch 14/30
3600/3600 [==============================] - 24s 7ms/step - loss: 0.2228 - mean_iou: 0.6466 - val_loss: 0.2288 - val_mean_iou: 0.6503

Epoch 00014: val_loss did not improve from 0.18690
Epoch 15/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.2147 - mean_iou: 0.6539 - val_loss: 0.1758 - val_mean_iou: 0.6573

Epoch 00015: val_loss improved from 0.18690 to 0.17580, saving model to model-tgs-salt-1.h5
Epoch 16/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2220 - mean_iou: 0.6602 - val_loss: 0.1715 - val_mean_iou: 0.6629
```



```
Epoch 00016: val_loss improved from 0.17580 to 0.17152, saving model to model-tgs-salt-1.h5
Epoch 17/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2027 - mean_iou: 0.6662 - val_loss: 0.1656 - val_mean_iou: 0.6693

Epoch 00017: val_loss improved from 0.17152 to 0.16561, saving model to model-tgs-salt-1.h5
Epoch 18/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.2035 - mean_iou: 0.6722 - val_loss: 0.1669 - val_mean_iou: 0.6749

Epoch 00018: val_loss did not improve from 0.16561
Epoch 19/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.2075 - mean_iou: 0.6775 - val_loss: 0.1642 - val_mean_iou: 0.6801

Epoch 00019: val_loss improved from 0.16561 to 0.16419, saving model to model-tgs-salt-1.h5
Epoch 20/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.1969 - mean_iou: 0.6826 - val_loss: 0.1732 - val_mean_iou: 0.6846
```



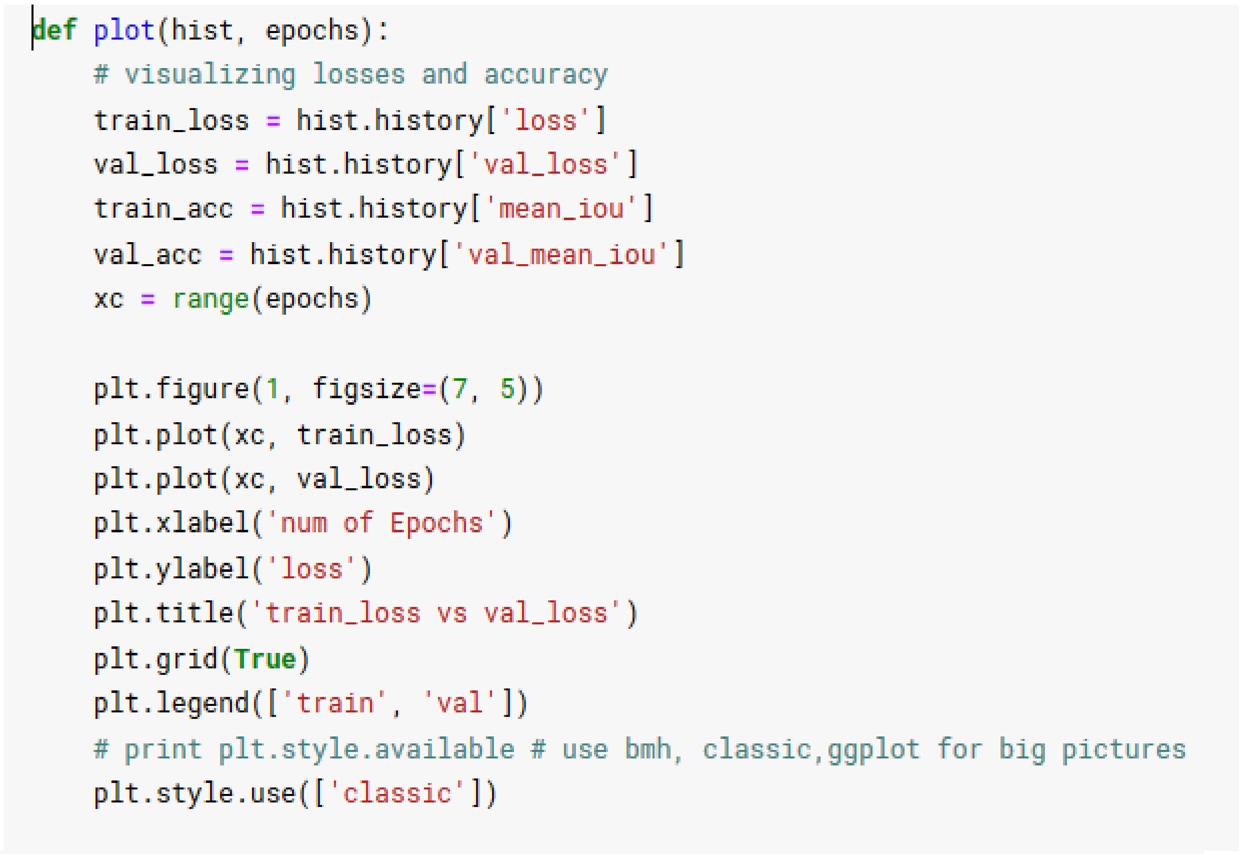
```
Epoch 00020: val_loss did not improve from 0.16419
Epoch 21/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.1891 - mean_iou: 0.6871 - val_loss: 0.1633 - val_mean_iou: 0.6893

Epoch 00021: val_loss improved from 0.16419 to 0.16334, saving model to model-tgs-salt-1.h5
Epoch 22/30
3600/3600 [==============================] - 23s 7ms/step - loss: 0.1750 - mean_iou: 0.6919 - val_loss: 0.1671 - val_mean_iou: 0.6943

Epoch 00022: val_loss did not improve from 0.16334
Epoch 23/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.1696 - mean_iou: 0.6967 - val_loss: 0.1444 - val_mean_iou: 0.6992

Epoch 00023: val_loss improved from 0.16334 to 0.14437, saving model to model-tgs-salt-1.h5
Epoch 24/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.1679 - mean_iou: 0.7017 - val_loss: 0.1686 - val_mean_iou: 0.7037
```



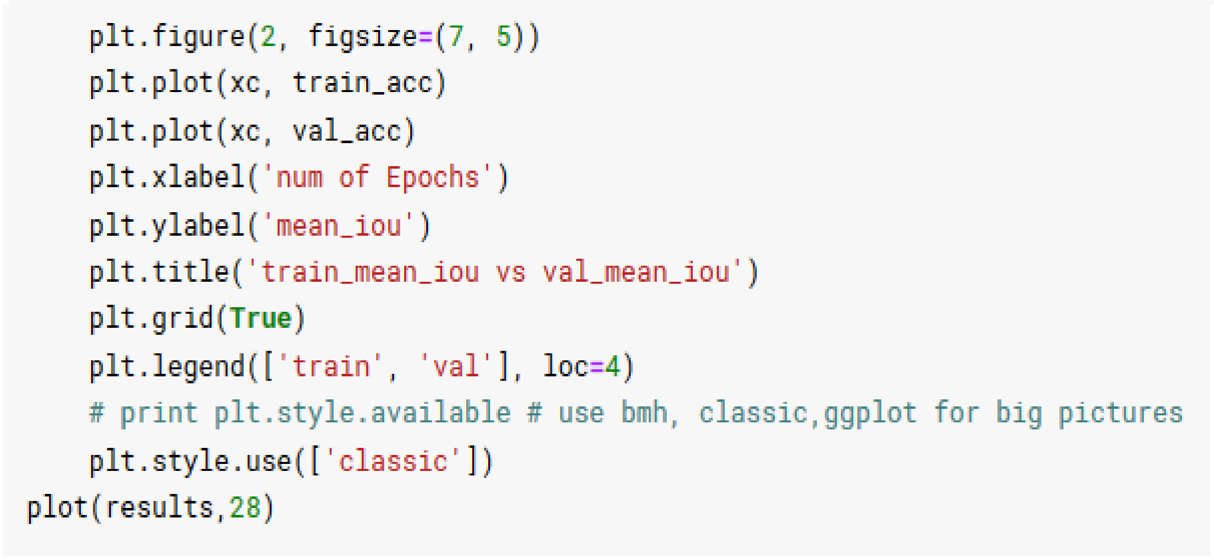
```
3600/3600 [==============================] - 22s 6ms/step - loss: 0.1768 - mean_iou: 0.7053 - val_loss: 0.1572 - val_mean_iou: 0.7073

Epoch 00025: val_loss did not improve from 0.14437
Epoch 26/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.1614 - mean_iou: 0.7096 - val_loss: 0.1552 - val_mean_iou: 0.7115

Epoch 00026: val_loss did not improve from 0.14437
Epoch 27/30
3600/3600 [==============================] - 22s 6ms/step - loss: 0.1547 - mean_iou: 0.7136 - val_loss: 0.1624 - val_mean_iou: 0.7155

Epoch 00027: val_loss did not improve from 0.14437
Epoch 28/30
3600/3600 [==============================] - 23s 6ms/step - loss: 0.1620 - mean_iou: 0.7173 - val_loss: 0.1754 - val_mean_iou: 0.7189

Epoch 00028: val_loss did not improve from 0.14437
Epoch 00028: early stopping
```

So the training mean_iou is 0.7189 and training loss is 0.14437

Plotting graphs between the train loss and validation loss & train mean_iou and validation mean_iou

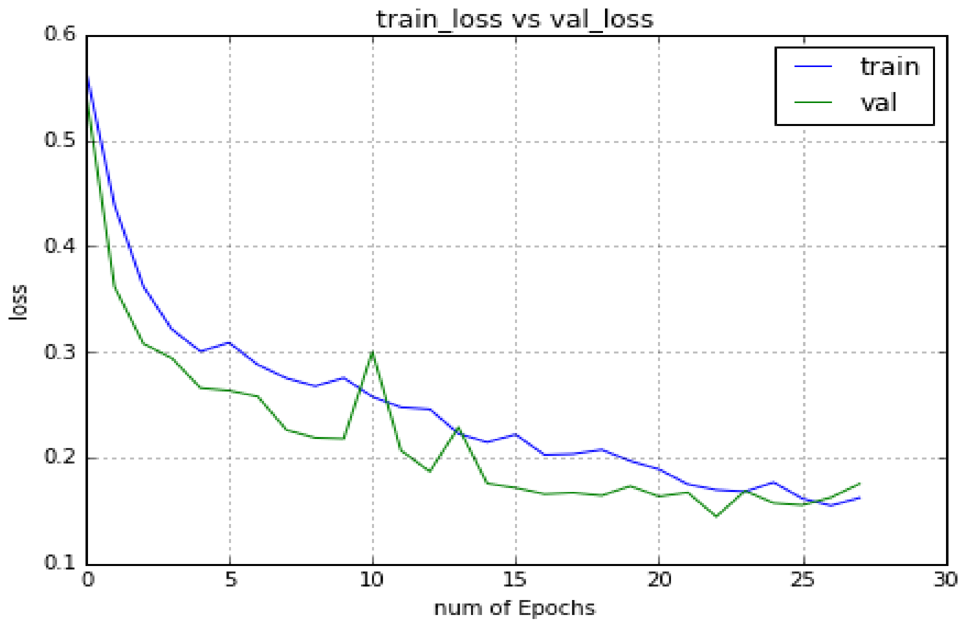


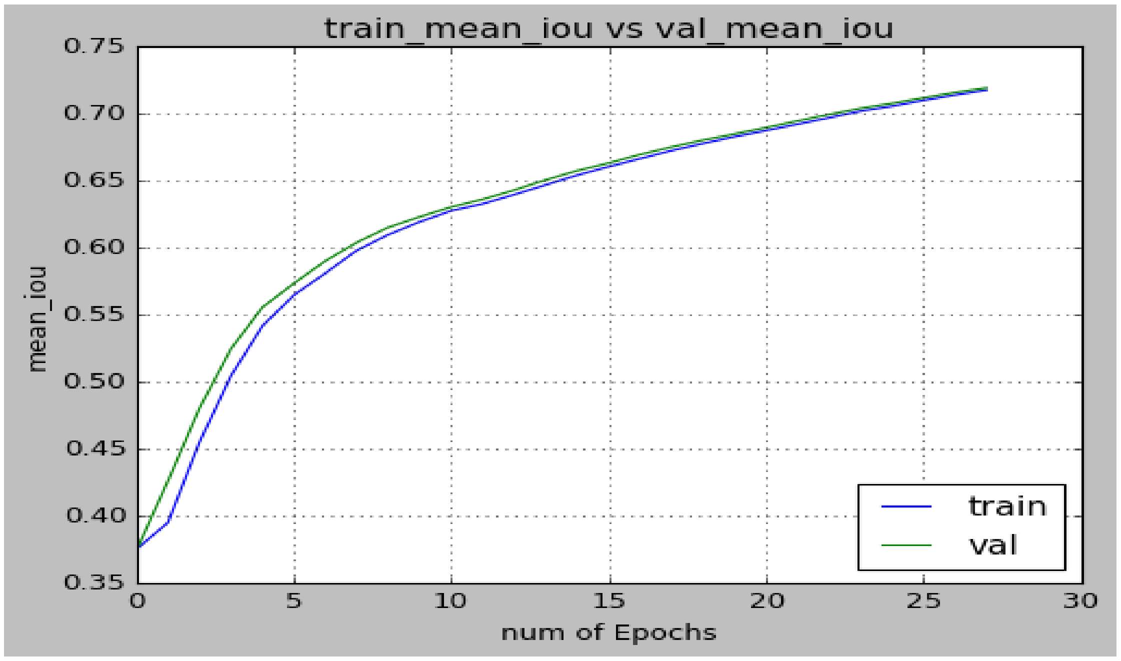
```python
def plot(hist, epochs):
    # visualizing losses and accuracy
    train_loss = hist.history['loss']
    val_loss = hist.history['val_loss']
    train_acc = hist.history['mean_iou']
    val_acc = hist.history['val_mean_iou']
    xc = range(epochs)

    plt.figure(1, figsize=(7, 5))
    plt.plot(xc, train_loss)
    plt.plot(xc, val_loss)
    plt.xlabel('num of Epochs')
    plt.ylabel('loss')
    plt.title('train_loss vs val_loss')
    plt.grid(True)
    plt.legend(['train', 'val'])
    # print plt.style.available # use bmh, classic,ggplot for big pictures
    plt.style.use(['classic'])
```



```python
    plt.figure(2, figsize=(7, 5))
    plt.plot(xc, train_acc)
    plt.plot(xc, val_acc)
    plt.xlabel('num of Epochs')
    plt.ylabel('mean_iou')
    plt.title('train_mean_iou vs val_mean_iou')
    plt.grid(True)
    plt.legend(['train', 'val'], loc=4)
    # print plt.style.available # use bmh, classic,ggplot for big pictures
    plt.style.use(['classic'])
plot(results,28)
```

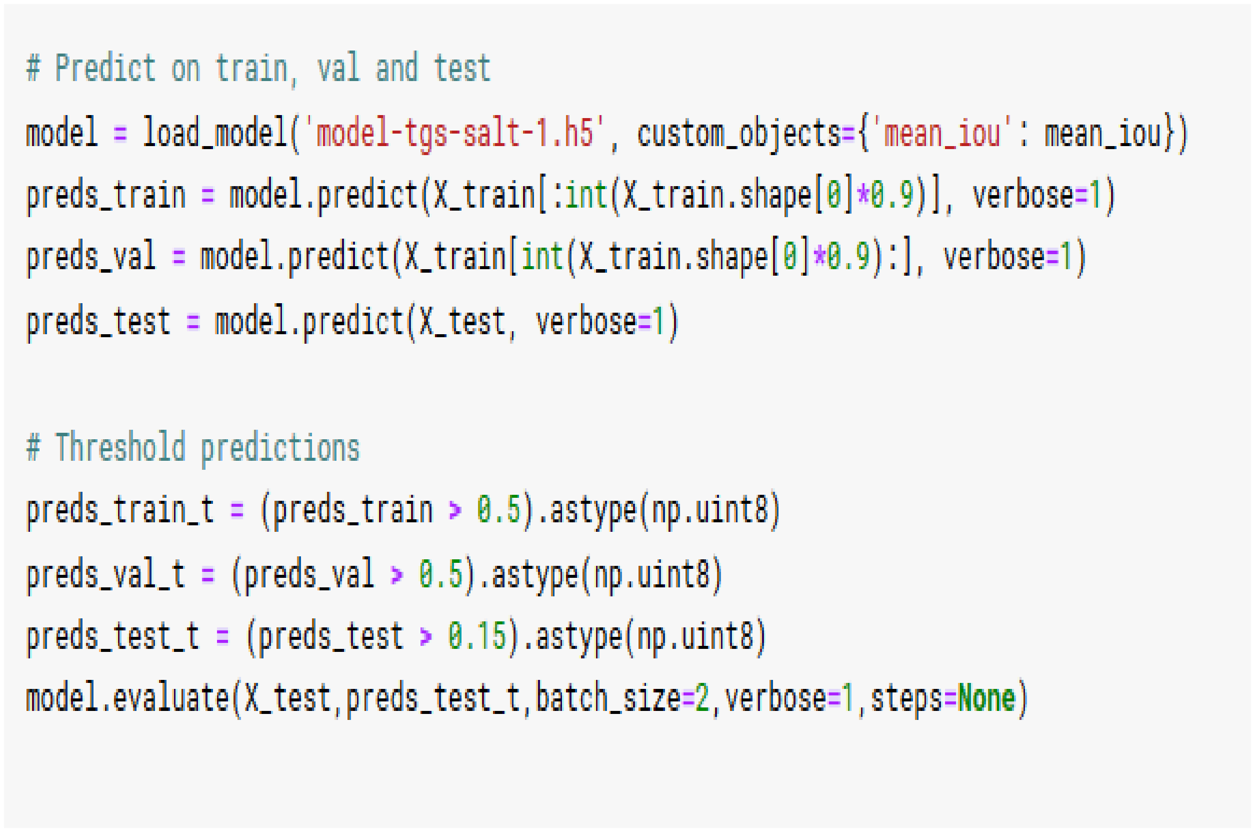
after executing we get two graphs





5.6 TESTING AND PREDICTION:

In this phase the final weights of trained model are used to predict the segmentations of test images. The steps in prediction include



```
# Predict on train, val and test
model = load_model('model-tgs-salt-1.h5', custom_objects={'mean_iou': mean_iou})
preds_train = model.predict(X_train[:int(X_train.shape[0]*0.9)], verbose=1)
preds_val = model.predict(X_train[int(X_train.shape[0]*0.9):], verbose=1)
preds_test = model.predict(X_test, verbose=1)

# Threshold predictions
preds_train_t = (preds_train > 0.5).astype(np.uint8)
preds_val_t = (preds_val > 0.5).astype(np.uint8)
preds_test_t = (preds_test > 0.15).astype(np.uint8)
model.evaluate(X_test,preds_test_t,batch_size=2,verbose=1,steps=None)
```

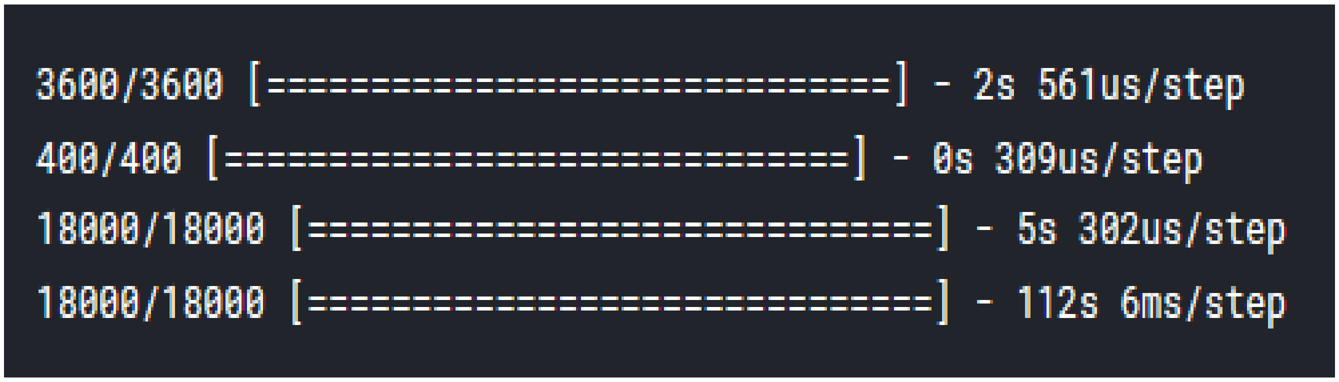
Load weights: The weights saved in the training phase are loaded. The keras implementation of load weights method is given by:

model = load_model('model-tgs-salt-1.h5', custom_objects={'mean_iou': mean_iou})

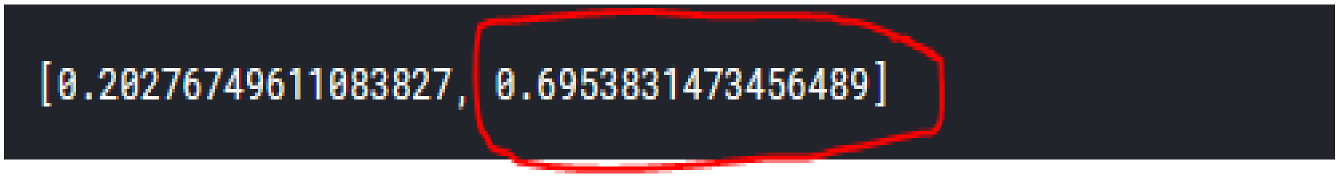
Predict the sample: Generates output predictions for the input samples as numpy arrays. The keras implementation of predict function is given as output = model.predict(imgs_test, batch_size = 1, verbose=1)

Evaluate the model: Returns the loss value & metrics values for the model in test mode. Takes test data samples as input. The keras implementation of evaluate function is given as

model.evaluate(X_train, Y_train,batch_size=2,verbose=1,steps=None)



```
3600/3600 [==============================] - 2s 561us/step
400/400 [==============================] - 0s 309us/step
18000/18000 [==============================] - 5s 302us/step
18000/18000 [==============================] - 112s 6ms/step
```

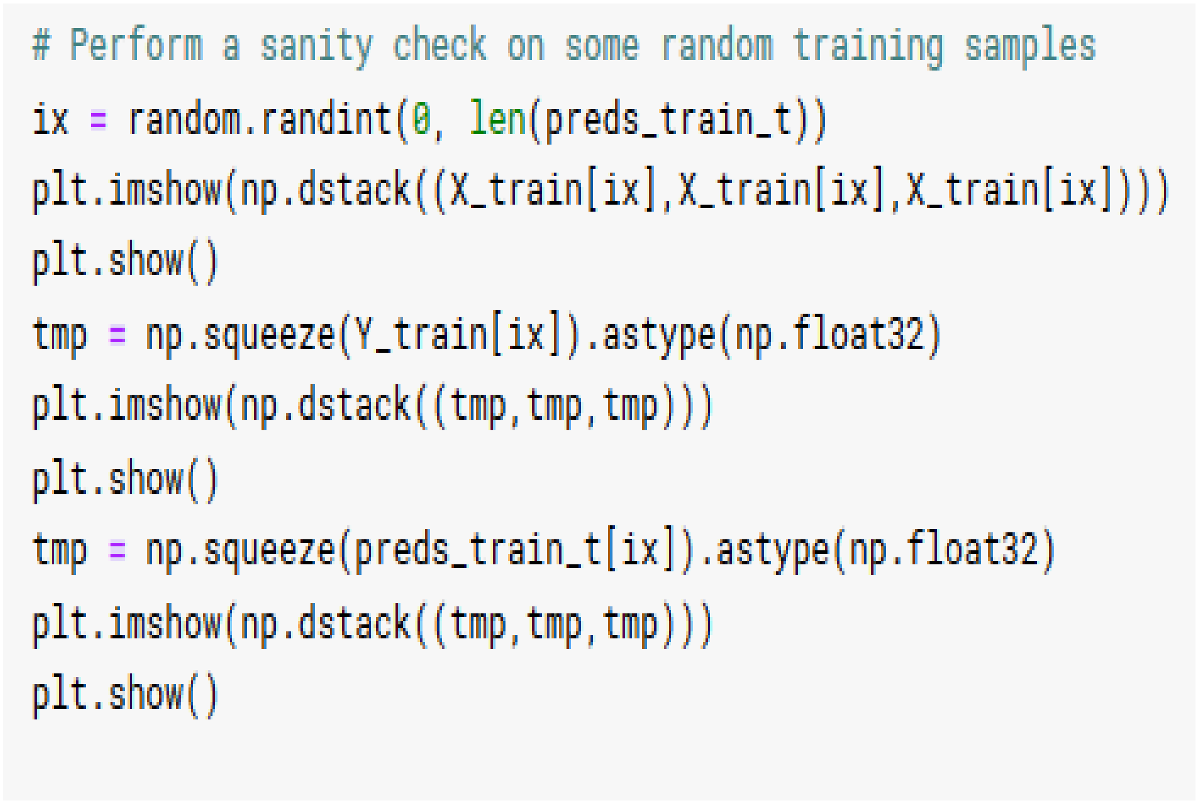


```
[0.20276749611083827, 0.6953831473456489]
```

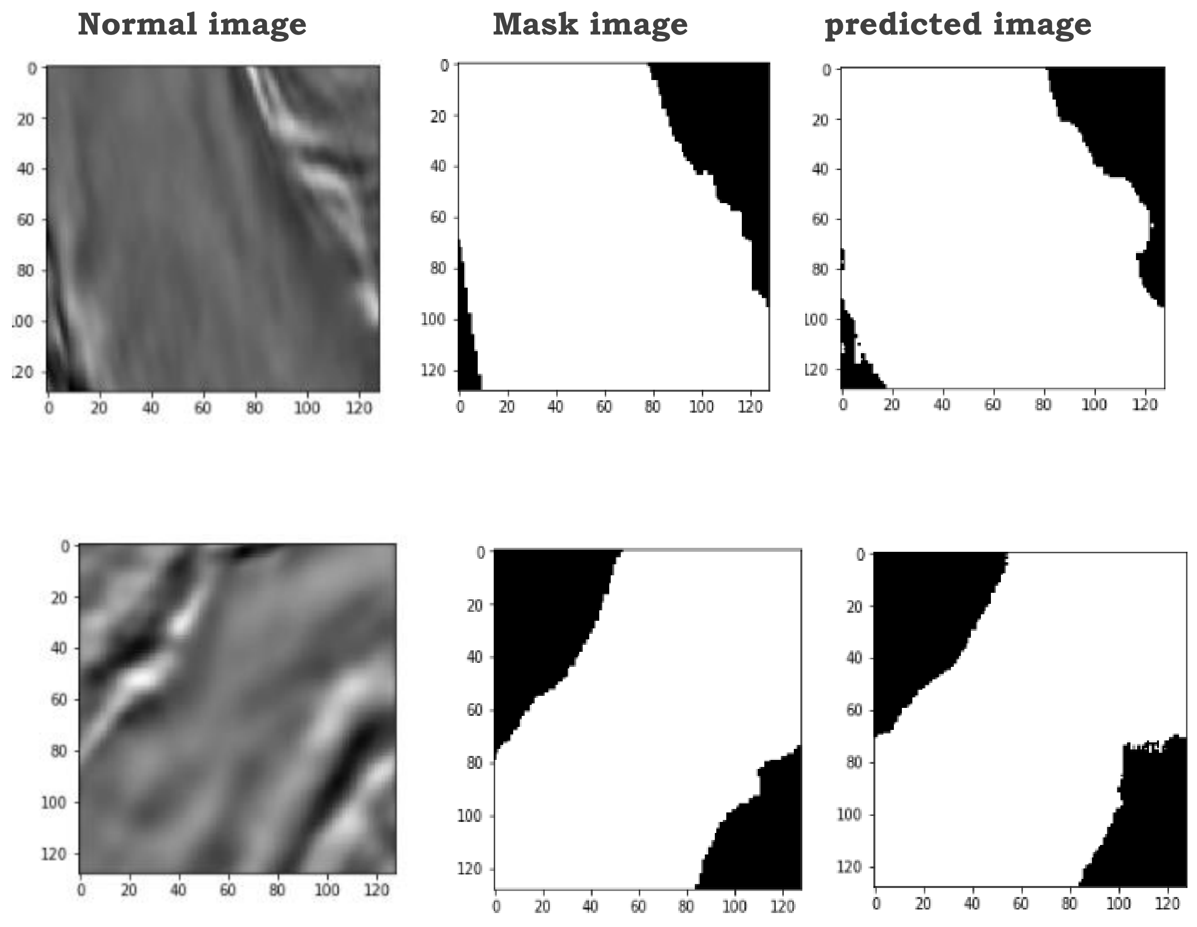
The accuracy that obtained after testing the test images is 69.5

5.7 Sanity Check:

After the prediction some random images are selected from the training data set their respective mask images and predicted images are plotted using matplotlib library functions.



```
# Perform a sanity check on some random training samples
ix = random.randint(0, len(preds_train_t))
plt.imshow(np.dstack((X_train[ix],X_train[ix],X_train[ix])))
plt.show()
tmp = np.squeeze(Y_train[ix]).astype(np.float32)
plt.imshow(np.dstack((tmp,tmp,tmp)))
plt.show()
tmp = np.squeeze(preds_train_t[ix]).astype(np.float32)
plt.imshow(np.dstack((tmp,tmp,tmp)))
plt.show()
```


Normal image
Mask image
predicted image

CHAPTER – 6
SOFTWARE TESTING

6.SOFTWARE TESTING

6.1 Testing Methodology:

Software Testing is a critical element of ultimate review of specification design and coding. Testing of software leads to the uncovering of errors in the software, functional and performance requirements are met. Testing also provides a good indication of software reliability and software quality as a whole. The result of different phases of testing are evaluated and then compared with the expected results. If the errors are uncovered they are debugged and corrected. A strategy approach to software testing has the generic characteristics:

- Testing begins at the module level and works outward towards the integration of entire computer based systems.
- Testing and debugging are different activities but debugging must be accommodated in the testing strategy.

Unit Testing:

The module interface is tested to ensure that information properly flows into and out of the program unit under test. The unit testing isnormally considered as an adjunct (something added to other) step to coding step. Because modules are not standalone programs , drivers and/or stubs software must be developed for each unit. A driver is nothing more than a main program that accepts test cases data and passes it to the module. A stub serves to replace the modules that are subordinate to the modules to be tested. A stub may do minimal data manipulation, prints verification of entry and returns.

Approaches used for Unit Testing are:

Functional Test: Each part of the code was tested individually and the panels were tested individually on all platforms to see if they are working properly. Ensures that the requirements are properly satisfied by the application. Functions (or features) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application.

Performance Test: These determined the amount of execution time spent on various parts of units and the resulting throughput, response time given by the module. Ensure software applications will perform well under their expected workload. Features and Functionality supported by a software system is not the only concern.

CHAPTER – 7 & 8
CONCLUSION & FUTURE ENHANCEMENT

7.CONCLUSION

In our project, we are training the machine to detect Salt deposits in the earth crust by Semantic Segmentation using Convolutional Neural Networks using seismic imaging. The model we are using to train the machine is UNet. When the machine gets trained, based on the probabilities of the region it determines to which class it belongs to(salt or no-salt).This provides an accurate method of finding out the salt deposits in less time.

8.FUTURE ENHANCEMENT

Large datasets can be provided to get more accurate predictions using compatible system architectures. High-end GPU usage would improve processing speed. This can also be extended to predict other natural resources like oil, gases etc. A Web application can be developed and given to TGS for easy prediction purpose.

REFERENCES:

[1] Road Extraction by Deep Residual U-Net by Zhengxin Zhang†, Qingjie Liu†∗, Member, IEEE and Yunhong Wang, Senior Member, IEEE

[2] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in CVPR, 2015, pp. 3431–3440. [24] O.

[3] Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in MICCAI, 2015, pp. 234–241.

[4] F. Chollet et al., “Keras,” https://github.com/fchollet/keras, 2015. [5] Keras documentation [6] cs231n.github.io/convolutional-networks/

[7] DeepResidualLearningforImageRecognition by Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Microsoft Research {kahe, v-xiangz, v-shren, jiansun}@microsoft.com

[8] A survey on Deep Learning in Medical Image Analysis, Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyososetio, Fancesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, Clara I. Sanchez. Diagnostic Image Analysis Group, Radboud University Medical Center Nijemgen, The Netherlands.

[9] Fully Convolutional Networks for Semantic Segmentation, Jonathan Long, Evan Shelhamer, Trevor Darrell, UC Berkley.

[10] Daniil’s blog, Machine learning and Computer vision artisan.
[11] Deep Learning applied for Food Classification, Stratospark.

[12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[13] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556, 2014.