Semantic Segmentation using Convolutional Neural Network

Eashwar Sathyamurthy
Robotics Engineer
A. James Clark School of Engineering
University of Maryland
College Park, USA
eashwar@umd.edu

Dr. Mohammed Charifa
Supervising Faculty
Maryland Applied Graduate Engineering
University of Maryland
College Park, USA
scharifa@umd.edu

***Abstract*—Human beings are easily able to identify and differentiate objects based on their cognitive skills. Cognitive skills like category formation, pattern recognition, working memory etc help humans to segment and process visual information obtained through eyes. But, a machine process visual data differently in the form of matrix containing whole numbers known as pixels. Each pixel indicates the brightness levels at that particular location. As the machine does not posses cognitive capability to segment images, a convolution neural net is designed and built through which the machine learns to segment images. The segmentation performed is called semantic segmentation as it groups every pixel of the image into classes based on labels. The report describes and implements convolutional neural networks like U-net[17], Segnet[15], IC-net[18] and Resnet34[19], compare the results obtained and recommends the best architecture for semantic segmentation.**

I. Introduction

Over the years convolution neural net (CNN) has increased its horizon and made significant impact on pattern recognition and machine learning fields. CNN is one type of neural network which has one input layer, one output layer and two or more hidden layers. The hidden layers is a combination of convolution layers, pooling layers, activation functions, deconvolution layers, unpooling layers and a fully connected layer. The main distinction between a CNN and a regular neural net is that in CNN neurons handles input and output in three dimensions. An example of CNN is shown in the figure below[1]:

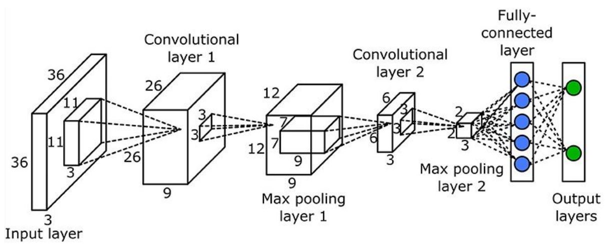


Figure 1: CNN Example

II. Definitions

In this section, different layers of a CNN are explained in detail.

A. *Input Layer*

The input to the input layer of the CNN is generally a color image represented in three dimensional arrays containing pixels in range 0-255. The three dimensions are length, width and number of channels (R,G,B) of the input image. Generally, the input image is resized to the desired dimensions accepted by the CNN.

B. *Convolutional Layer*

In the convolutional layer, convolution operation happens between the kernel and the input image. Kernel is a 2D array whose elements are normally distributed. An example of 3x3 Gaussian Kernel is given below[2]:

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

Figure 2: 3x3 Gaussian kernel

Mathematically, 2D convolution is represented as follows:

$$y[i,j] = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} h[m,n].x[i-m,j-n]$$

In convolution layer, the convolution operation is performed on all the pixels by sliding the kernel over the input image. The following image shows the convolution operation between the image and the kernel[3]:

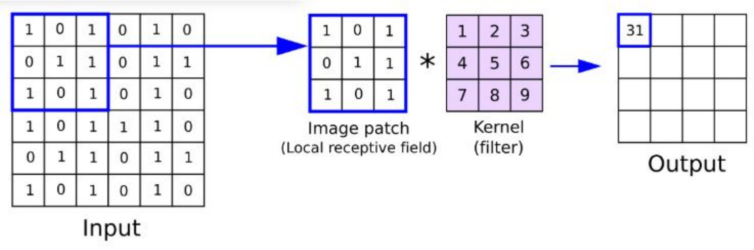


Figure 3: Convolving kernel over the image

Typically, in most neural networks the dimensions of the image after performing convolution are maintained same with the help of padding. But, it is important to understand how convolution operation affects the dimensions of the image. The relation is given as follows:

$$\boldsymbol{OutputWidth = \frac{I_W - K_W + 2 * P}{S} + 1}$$

where I_W is the image width
K_W is the kernel width
P is number of padding bits
S = stride = number of pixels shift over the input image

$$\boldsymbol{OutputHeight = \frac{I_H - K_H + 2 * P}{S} + 1}$$

where I_H is the image height
K_H is the kernel height
P is number of padding bits

C. *Pooling Layers:*

Pooling layers in a CNN are typically connected after convolutional layers. It reduces the size of the image by retaining only the important features in the image and neglecting the rest. Pooling layers have the following advantages:

1) It reduces the number of parameters to train in a CNN.
2) It prevents the neural net from the over-fitting problem. Over fitting is a situation where the neural net obtains high accuracy over training data and less accuracy over the testing data. This usually occurs because of large number of trainable parameters which lets the neural net detect all the patterns in an training image present but fails to learn from the patterns to generalize it on the testing image. This can be observed when the neural net obtains high accuracy on the training data but low accuracy on the testing data.

   Commonly used Pooling Layers: Average pooling and max pooling are the two most commonly used pooling layers in neural network.

   a) **Average Pooling:** In average pooling, the average value is selected from each patch of the feature map to obtained a reduced map size. The following image gives an example of average pooling[4]:

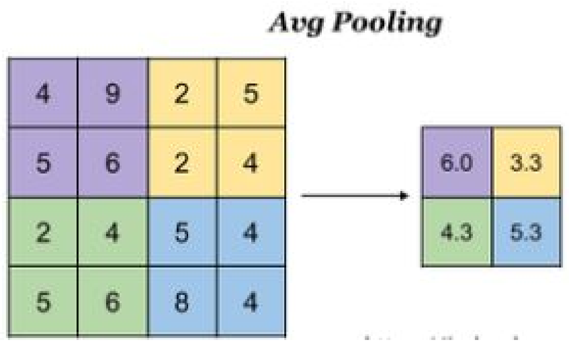


Figure 4: Average Pooling Example

In the above example, the feature map is of size 4x4 and the patch is of size 2x2 and stride = 2 and there are no padding bits. So, if we apply the formula for resulting dimension of the feature map, we get the following:

$$\begin{aligned} OutputWidth &= \frac{I_W - K_W + 2 * P}{S} + 1 \\ &= \frac{4 - 2 + 2 * 0}{2} + 1 \\ OutputWidth &= 2 \end{aligned}$$

$$\begin{aligned} OutputHeight &= \frac{I_H - K_H + 2 * P}{S} + 1 \\ &= \frac{4 - 2 + 2 * 0}{2} + 1 \\ OutputHeight &= 2 \end{aligned}$$

So, the output feature map will be of size 2x2. Let consider the first (violet) patch and calculate the average pooling output:

$$Outputvalue = \frac{4 + 9 + 5 + 6}{4} = 6.0$$

This process is repeated over the entire feature map to obtain the reduced 2x2 feature map output.

   b) **Max Pooling:** In max pooling, the maximum value is selected from each patch of the feature map to obtained a reduced map size. Max pooling is the most commonly used pooling layers. The following image gives an example of max pooling[4]:

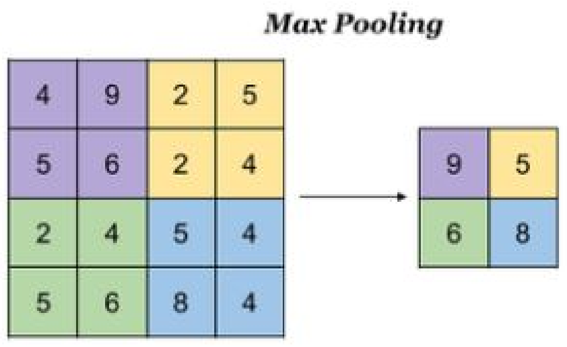


Figure 5: Max Pooling Example

As this is the same example used in average pooling, the output feature map size is 2x2. The following is the max pooling output for the first (violet) patch of the feature map:

$$Outputvalue = max(4, 9, 5, 6) = 9.$$

D. *Unpooling Layers:*

In CNN, inverse operation of pooling is not possible. But unpooling layer obtains the approximate inverse of pooling operation by storing the index of the maximum value from each patch of the feature map in a set of switch variables while pooling operation is being performed. These switch variables are again used in unpooling layers to reconstruct the layer before the pooling operation was performed. The below picture shows unpooling:

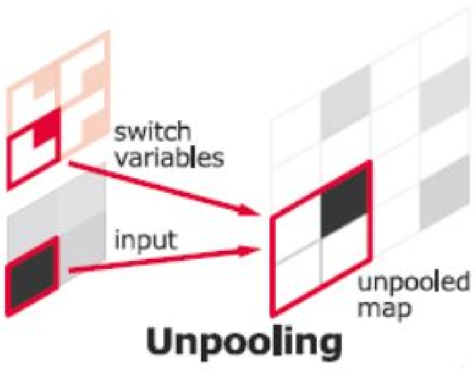


Figure 6

E. *Activation Functions:*

As the name suggests, an activation function decides whether to activate a neuron or not based on calculations of weighted sum and bias. In the case of a CNN, weighted sum is the output image obtained by convolving the kernel over the input image. Bias is a fixed value added pixel wise to the obtained output image. Activation function is applied to this output image which decides whether to select a particular pixel value and pass on to the next layer or not.

Need for Activation Functions: In neural networks, the weights (kernel in the case of CNN) and bias values constantly gets updated based on the error at the output. The values are updated using back-propagation algorithm. Activation functions makes the back propagation possible as the gradients are supplied along with errors to update the weight and bias values[5].

Different Activation Functions: There are different activation functions used in neural networks based on its applications. They are:

1) **Step Function:** Step function is a threshold based activation function which outputs 1 for the values greater than a specified threshold or outputs 0 otherwise. Hence, 1 means the neuron gets activated and 0 means neuron gets deactivated. It is shown in the below figure:[7] The following conditional equations

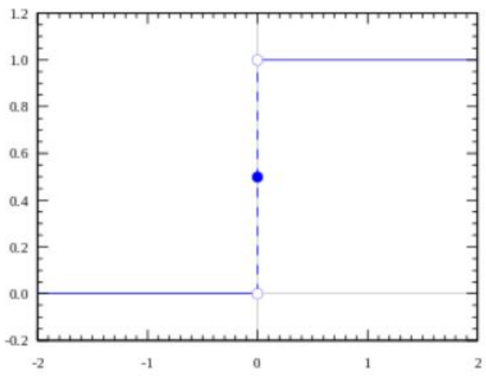


Figure 7: Step Function

defines step function:

$$\begin{aligned}\text{Activation Function } A &= 1 \text{ if } y \geq threshold\\ &= 0 \text{ otherwise}\end{aligned}$$

Step function can be used as an activation function if the neural net is being used for binary classification where the output is just ”YES” or ”NO”. Typical example of such classification is determining the picture is of a dog or a cat.
But, step functions cannot be used as activation functions for multiple(more than 2) classifications as the neural net might give multiple 1’s and 0’s as output.

2) **Linear Function:** Linear function means a straight line functions which is proportional to the input. The below figure shows the example:[7] The following

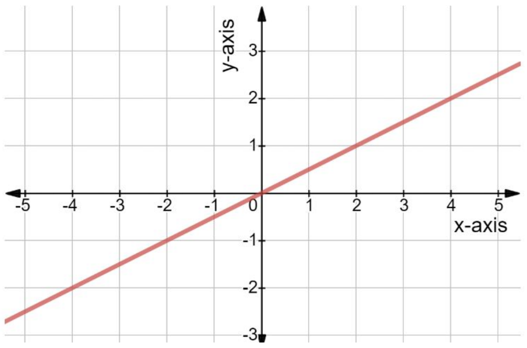


Figure 8: Linear Function

conditional equations defines linear function:

$$\text{Activation Function } A = c * X$$

Linear function eliminates the multiple classification problem faced by employing step function as the activation function. As the activation function varies with input values, it can take many values not only 0 and 1. Hence, it can classify multiple classes.
The main disadvantage of linear function is the derivative becomes constant. This means that the gradient no longer varies with input. Hence, if there was an error in detecting the output, since the gradient is constant there will be constant change in the weights and bias values irrespective of inputs. This means that the neural net will fail to learn.
Another disadvantage occurs due to linearity property. By, employing linear functions as activation functions to all hidden layers, the output will be a linear combination of hidden layers. Hence, multiple hidden layers can be combined into 1 hidden layer. Hence, by employing linear function, 1-hidden layer neural nets are possible.

3) **Sigmoid Function:** The below picture shows the sigmoid function.[7]

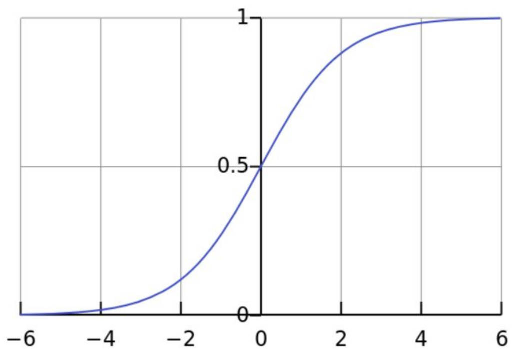


Figure 9: Linear Function

The following conditional equations defines sigmoid function:

$$\text{Activation Function } A = \frac{1}{1+e^{-x}}$$

The sigmoid function is essentially a smoothened version of step function. This makes it far more useful than step function. Firstly, sigmoid function is non-linear which eliminates the linearity problem. Secondly, it can take many values between 0 and 1 which makes it usable in multi class classification problems.
The only disadvantage of sigmoid function is that gradient of the points farther from origin approaches to zero which gives rise to vanishing gradients problem. This makes the neural net to learn very slowly and in worst case scenarios not learn at all.

4) **Tanh Function:** The below picture shows the tanh function.[7]

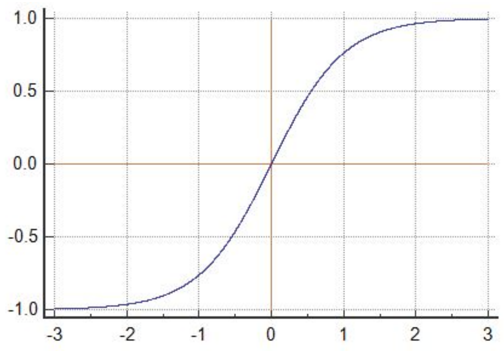


Figure 10: Tanh Function

The following conditional equations defines tanh function:

$$\text{Activation Function } A = \frac{2}{1+e^{-2*x}} - 1$$

Tanh function is a scaled version of sigmoid function. The gradient of the points farther from origin approaches slowly to zero than sigmoid function. But, tanh function also faces the problem of vanishing gradient.

5) **Relu:** Relu stands for Rectified linear unit. It is one of the most commonly used activation function. The figure below shows the function:[7]

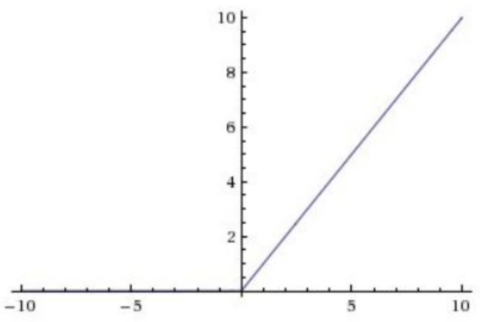


Figure 11: Relu Function

Relu is non-linear in nature as it remains zero for all the negative values which allows neural net to have multiple layers. In addition to this, most of the neurons in the neural net never fire up due to the characteristics of Relu function. This makes the neural net lighter. This makes Relu function less computationally expensive than Sigmoid and Tanh functions.
The disadvantage of Relu function is that for negative values the gradient remains zero. This means that some neurons in the neural net will not respond to the change in the output. This is called dying Relu problem.

F. *Fully Connected Layer:*

In neural network, a fully connected later is that layer which connects the inputs of one layer to every activation unit of the next layer. Typically, a fully connected layer is present before the output layer. In a neural net each unit in a convolutional layer contains a feature of the input image. Hence, each convolutional layer is a collection of features of the input image. A fully connected layer holds the composite and aggregate features of all the convolution layers. Hence, a fully connected layer is a feature vector of the input image. The following figure shows an example of a fully connected layer:[8]

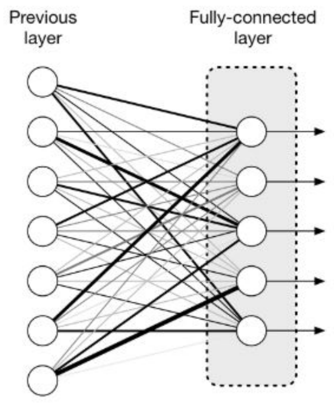


Figure 12: Fully Connected Layer

G. *Deconvolution Layer:*

Deconvolution layers in CNN is used to represent the output data in the same dimensions as the input image but increasing the size of the output data. Deconvolution is a transpose operation of convolution.

In deconvolution, first the data is padded and separately according to the desired output size. The below figure shows the representation of data in padded and separated form to convert data of size 2x2 to 4x4:[11]

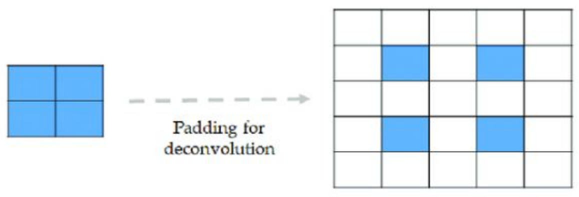


Figure 13

Then the kernel is made to slide over the padded representation to get 4x4 output data size.[12]

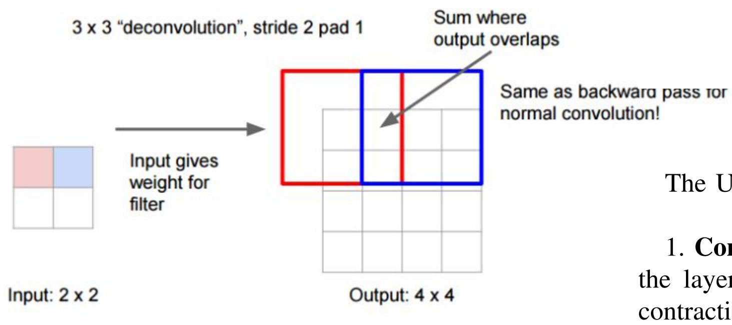

3 x 3 "deconvolution", stride 2 pad 1
Sum where output overlaps
Same as backward pass for normal convolution!
Input gives weight for filter
Input: 2 x 2
Output: 4 x 4

Figure 14

H. *Output Layer:*

The output layer is the final layer of CNN. The output layer takes input from the fully connected layer and assigns probability to it. The assignment of probability is done through softmax activation function. Hence, the output layer outputs the probability of each class in the input image. The below image shows the softmax activation function present in output layer.[10]

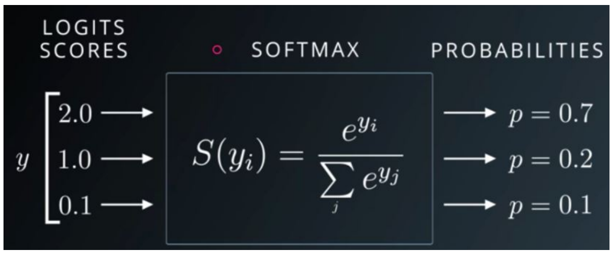


Figure 15: Softmax Activation Function Example

III. Architectures

Generally, there are many architectures found in the literature which are used for different purposes like natural language processing, segmentation and tracking. The upcoming sections will discuss about the main architectures which are used for semantic segmentation.

IV. U-net Architecture

U-net is popularly used CNN for segmentation. The name U-net comes from the fact that the layers in the CNN are arranged in U shape. The below figure shows the U-net architecture[17]:

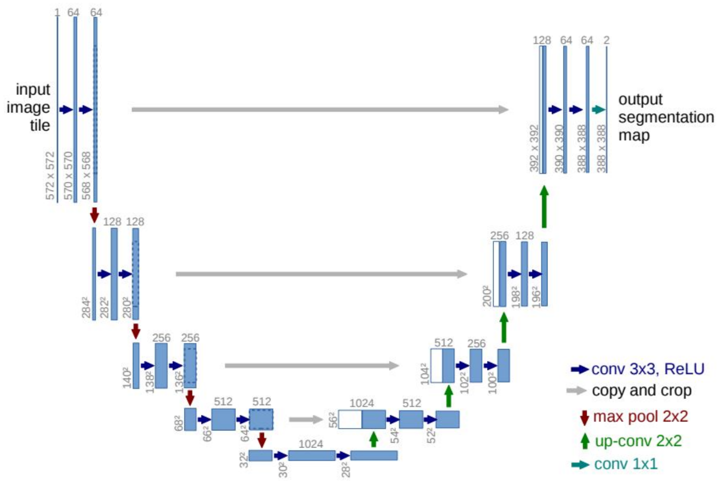


Figure 16: U-net Architecture

The U-net architecture consists of two path. They are:

1. **Contraction Path**: The left half of the U-net where the layers are represented from top to bottom is called contraction path. The contraction path consists of an input layer, sequence of convolution layers, Relu activation layers and max pooling layers. These series of layers reduce the size of the input image simultaneously increasing the number of channels.
2. **Expansion Path**: The right half of the U-net where the layers are represented from bottom to top is called expansion path. The expansion path consists of sequence of deconvolution or up-convolution layers and concatenation with high resolution features from the contraction path and unpooling layers. The expansion path gives output data in the same size as the input image data. The expansion path increase the size of the contraction path output data simultaneously decreasing the number of channels.

V. Segnet Architecture

Segnet is designed to perform pixel wise segmentation more efficiently. It consists of an encoder network followed by a decoder network and finally a pixel classification layer. The encoder layer consists of 13 convolution layers similar to VGG16[20] for object classification. Since each encoder layer has a corresponding decoder layer, there are 13 convolution layers present in decoder layer.

Each encoder layer performs convolution operation to detect the features of the input image. Then, these are batch normalized to reduce the amount by which hidden unit values shift. Finally, a rectified linear unit (ReLU) activation is applied pixel wise followed by max pooling operation. Each decoder layer has a upsampling layer to perform unpooling operation and also to increase the image size by a factor 2 followed by convolution layers.

The last decoder layer has a softmax activation function which gives output images whose length is the number of segmented labels present. The figure below shows the segnet architecture[15]:

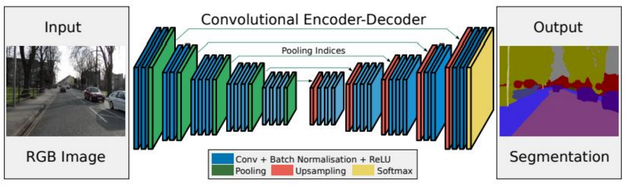


Figure 17: Segnet Architecture

VI. IC-net Architecture

The full form of IC-net is image cascade network. IC-net architecture is designed to make sematic segmentation faster while not compromising too much on the accuracy. IC-net is a highly efficient semantic segmentation system with descent quality. IC-net architecture is shown below[18]:

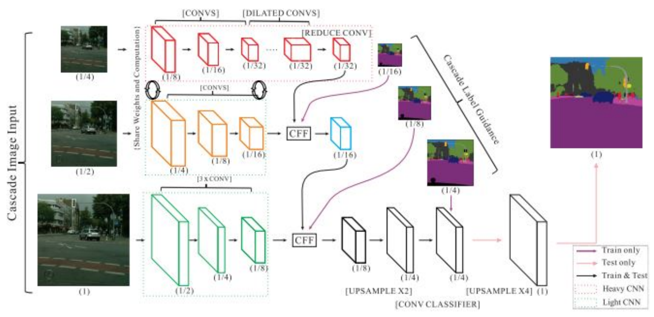


Figure 18: IC-net Architecture

IC-net architecture takes cascade image inputs i.e the resolution of the input image is varied into three categories of low, medium and high resolution images and are fed as inputs to the architecture in a cascaded form. As the semantic segmentation using of resolution images is computationally expensive and slow, the low resolution image (1/4 of original image) is fed into PSPnet[21] architecture for semantic segmentation. The PSPnet[21] architecture outputs segmented image with resolution 1/32 of the original image. As we decreased the resolution of the image, the quality of semantic segmentation is reduced. To preserve the quality, the medium (1/2 of the original image) and high (original image) resolution image are used to refine the prediction process. The medium and high resolution images are passed through light weighted CNNs which are less computationally expensive and takes less to extract features. Thus, the outputs obtained from different resolution branches are fused using cascaded feature fusion unit (CFF) and trained using cascade label guidance.

Cascaded Feature Fusion Unit: Cascaded Feature Fusion Unit combines the features extracted from the outputs obtained from different CNN architectures of different resolution and outputs combined feature vector with the desired resolution. The unit takes 2 input feature maps (F1 and F2) and 1 ground truth label (LABEL) as input. First, the unit converts the feature map F1 into same resolution by performing upsampling operation.Then refinement of features is done through dilation convolution operation. The feature map F2 undergoes projection convolution with 1x1 kernel to output same number of channels as feature map F1. Then, the two feature maps undergoes element wise sum in SUM layer followed by 'RELU' layer to obtained a fused feature map[18].

Cascaded Label Guidance Unit: Cascaded Label Guidance strategy is used to enhance the learning of the IC-net architecture. It uses different resolutions (low, medium and high) of ground truth label for different resolution branches(low, medium and high). The weighted softmax cross entropy loss is calculated for each resolution branch with relative loss weight. This ultimately minimized the loss function of the IC-net architecture[18].

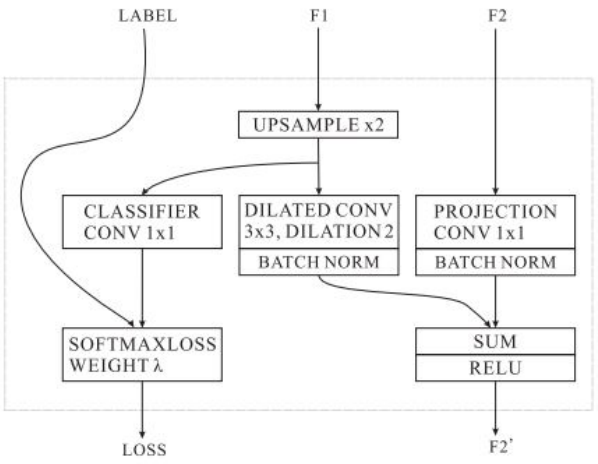


Figure 19: Cascaded Feature Fusion

VII. Resnet34 Architecture

The full form of Resnet is Residual network and the number 34 signifies the number of layers in the Resnet architecture. The Resnet architecture mainly address the degradation problem which arises with architecture's increasing depth. When the number of layers in the CNN increases the accuracy becomes saturated and begin to decrease drastically. Resnet architecture presents a solution to this problem by introducing residual or shortcut connections. The residual or shortcut connections means to add previous layer output with some train error to output obtained after passing it through the convolutional layer. This ensures that the resulting trained error will not increase with increasing depth of the architecture. If the input and output layer have same dimensions, then the shortcut connections are applied directly or the output is padded with zeroes to match input dimension or undergoes 1x1 convolution to match the input layer dimensions. The Resnet34 architecture is shown in the figure below[19]:

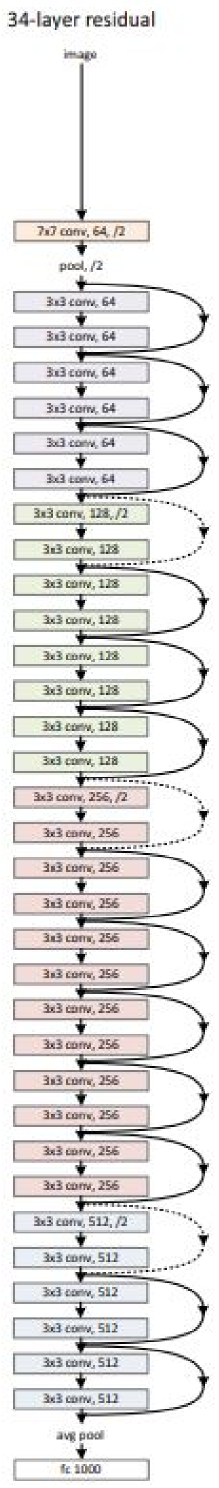


Figure 20: Resnet Architecture

VIII. Implementation:

The details of the implementation of the above mentioned architecture and required dependencies are given below:

1) Language Used: Python version - 3.8
2) Packages Used: Tensorflow version - 2.2.0, Numpy, cv2
3) Software Used: Jupyter Notebook
4) Github Link: https://github.com/Eashwar-S/Semantic_Segmentation
5) Cloud Service Used: AWS Sagemaker
6) Cloud Storage used: AWS S3 bucket storage

IX. Cityscrapes Dataset:

Cityscrapes dataset consisted of 5000 images and labels. 2975 images were used for training, 500 for validation and remaining 1525 images were used for testing. There are a total of 29 classes present in the dataset.

X. Results

Semantic segmentation was performed using U-net, Segnet, IC-net and Resnet34 architectures and the below table represents the results obtained with architectures arranged in ascending order based on their performance and accuracy.

Architecture Results

Sno	**Architecture**	**Img size**	**batch size**	**L.R**	**IOU Accu.**	**Loss**
1.	U-net	256 x256	16	0.001	53.29%	0.3350
2.	IC-net	256 x256	16	0.001	48.39%	0.4134
3.	Segnet	256 x256	16	0.001	48.39%	0.5001
4.	Resnet34	256 x256	16	0.001	47.37%	0.4943

- **Segmented Output** The following are the results of the segmented output obtained by U-net, IC-net, segnet and resnet architectures for the same input image are given below respectively:

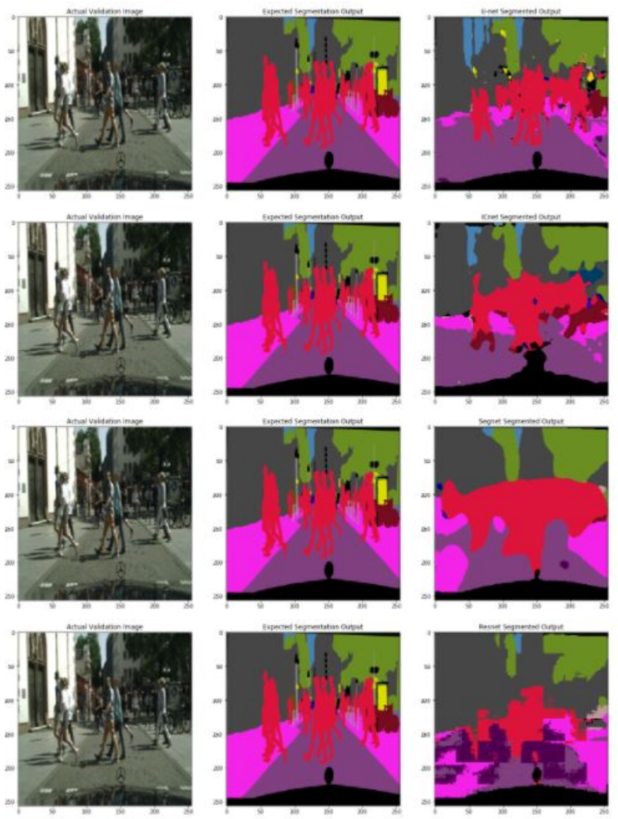


Figure 21: Segmented Results

- **MIOU accuracy and Loss:** The following are the graphs of MIOU accuracy and categorical cross entropy loss with respect to number of EPOCHS. The left half of the graph represents the result of training and the right half is for validation.

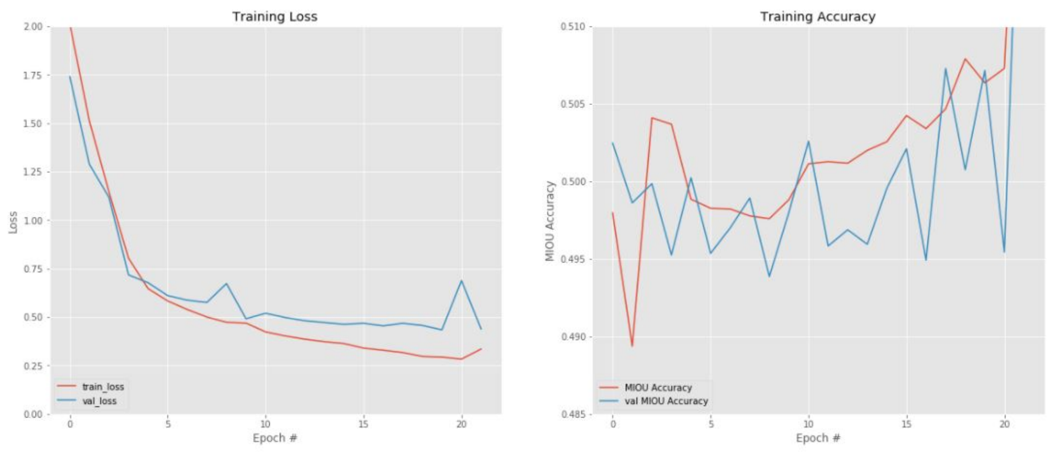


Figure 1: U-net MIOU accuracy and Loss

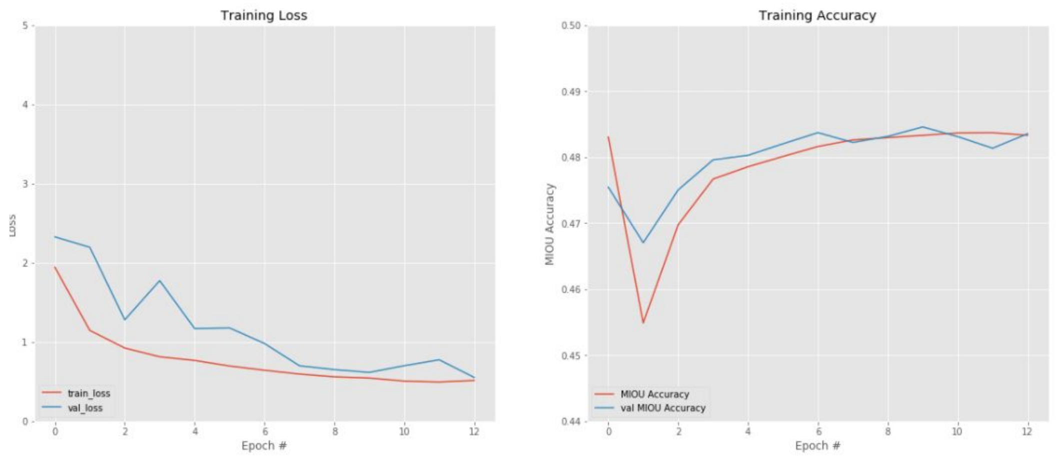


Figure 2: Segnet MIOU accuracy and Loss

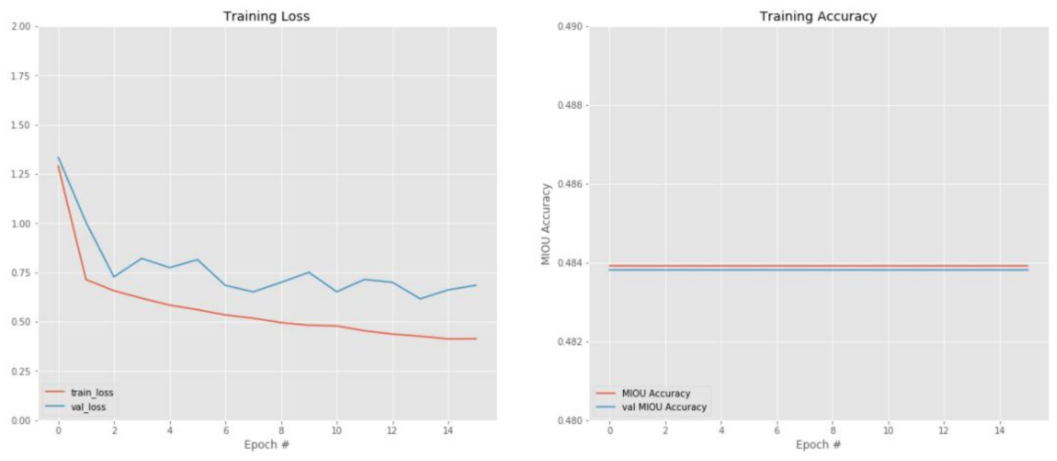


Figure 3: ICnet MIOU accuracy and Loss

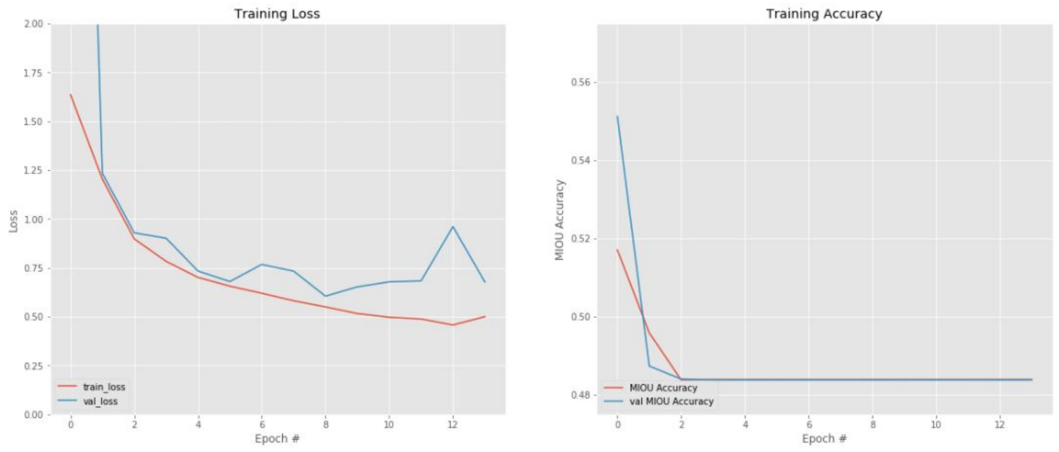


Figure 4: Resnet MIOU accuracy and Loss

- **Precision vs Recall Graph:** The following are the graphs of Precision vs Recall for 4 architectures. The left half of the graph represents the result of training and the right half is for validation.

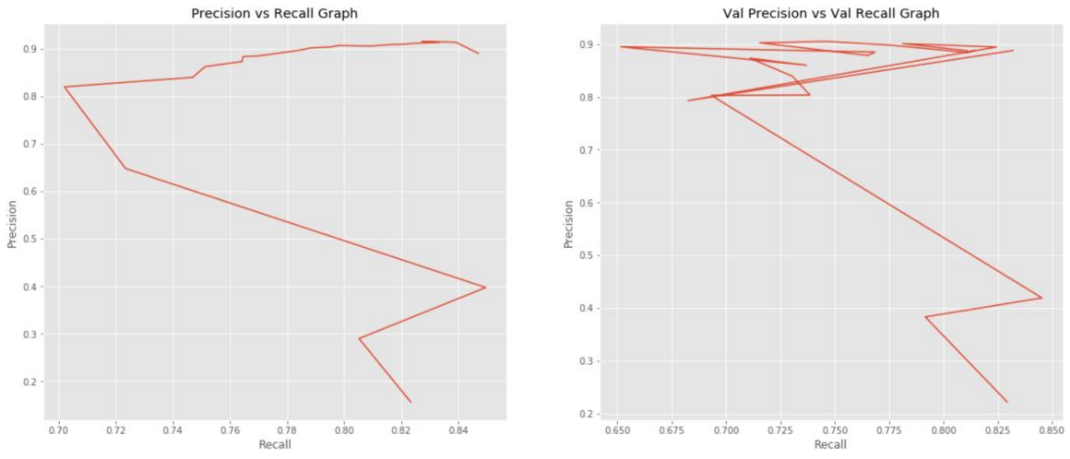


Figure 5: U-net Precision vs Recall Graph

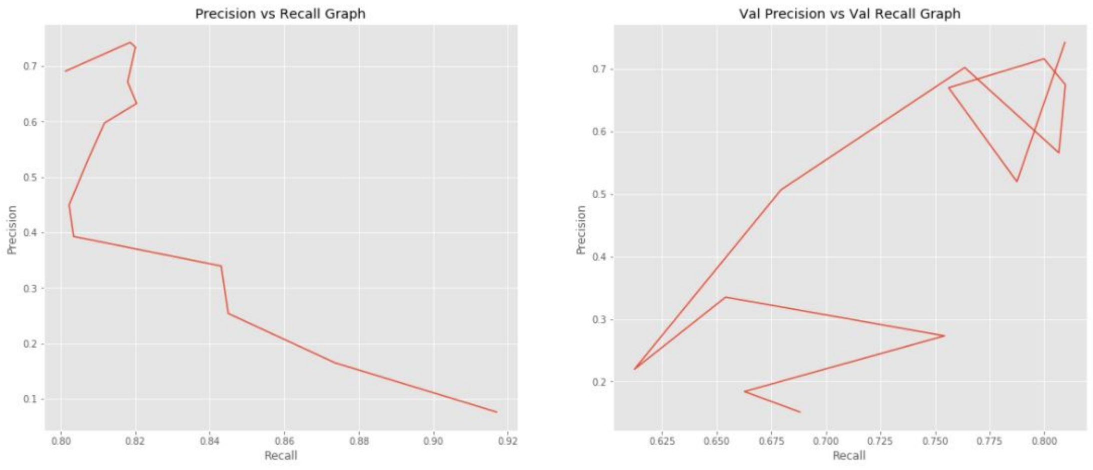


Figure 6: Segnet Precision vs Recall Graph

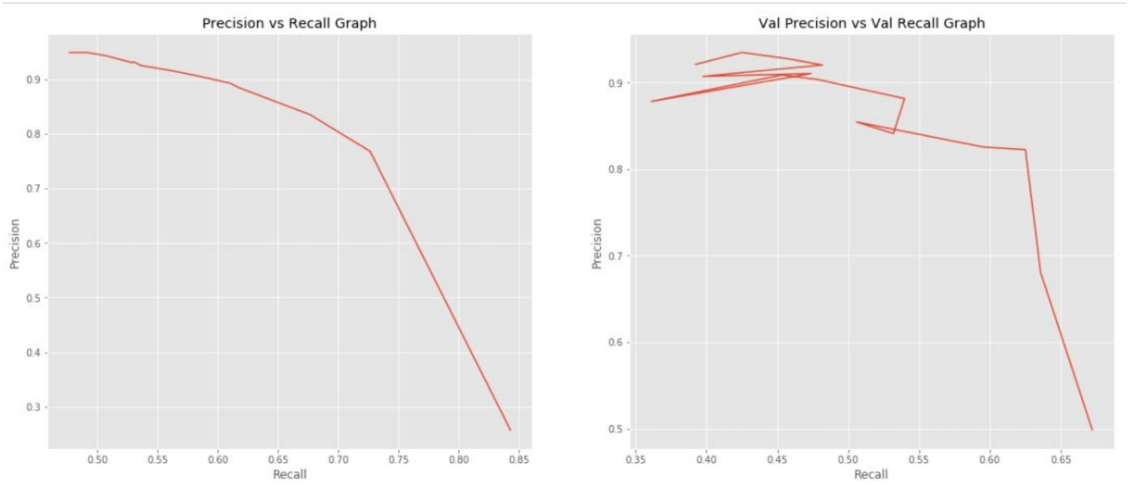


Figure 7: ICnet Precision vs Recall Graph

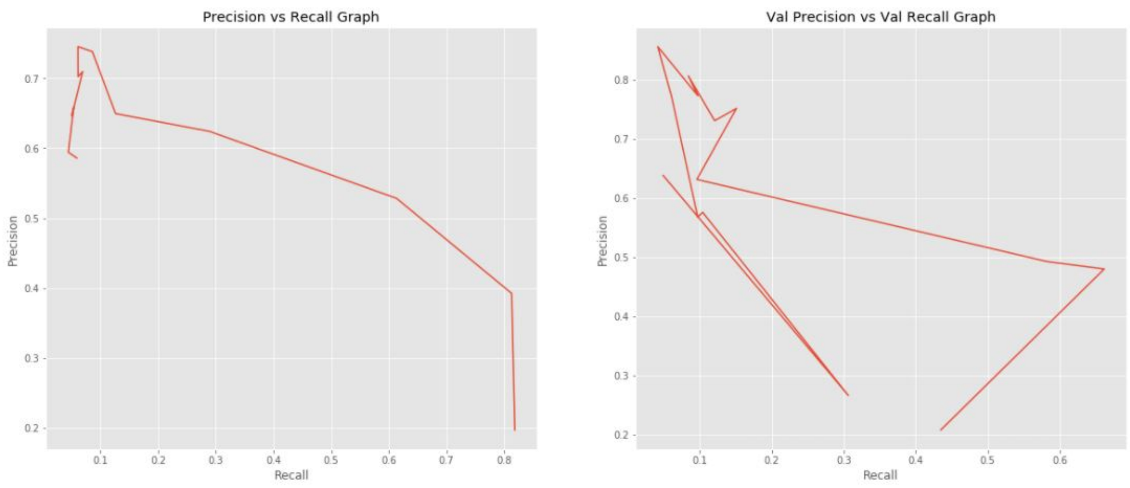
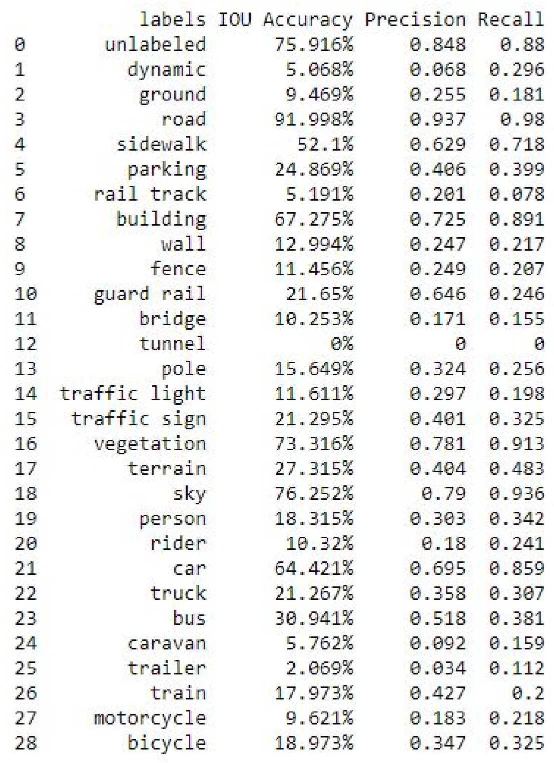


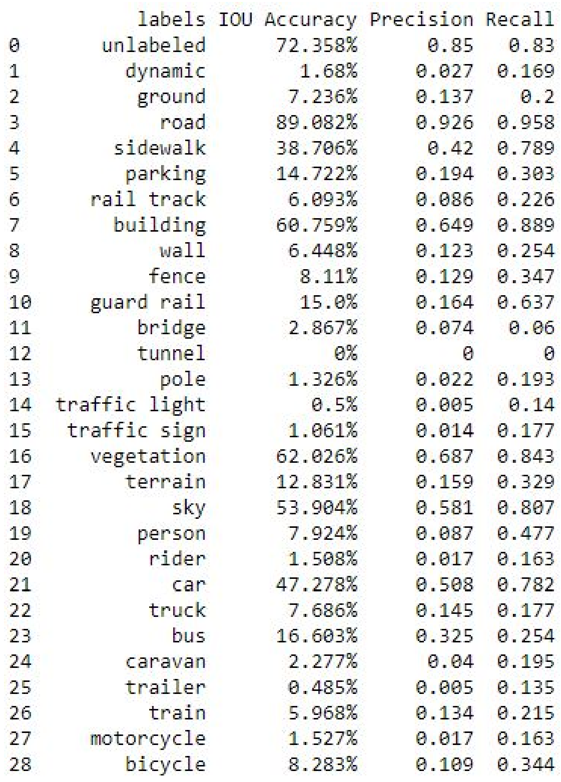
Figure 8: Resnet Precision vs Recall Graph

- **IOU accuracy of each class**



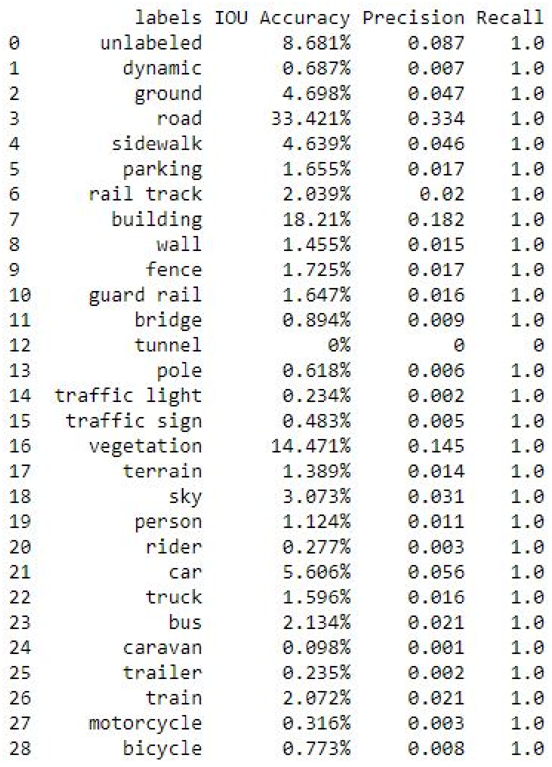
```
           labels IOU Accuracy Precision Recall
0       unlabeled      75.916%     0.848   0.88
1         dynamic       5.068%     0.068  0.296
2          ground       9.469%     0.255  0.181
3            road      91.998%     0.937   0.98
4        sidewalk        52.1%     0.629  0.718
5         parking      24.869%     0.406  0.399
6      rail track       5.191%     0.201  0.078
7        building      67.275%     0.725  0.891
8            wall      12.994%     0.247  0.217
9           fence      11.456%     0.249  0.207
10     guard rail       21.65%     0.646  0.246
11         bridge      10.253%     0.171  0.155
12         tunnel           0%         0      0
13           pole      15.649%     0.324  0.256
14  traffic light      11.611%     0.297  0.198
15   traffic sign      21.295%     0.401  0.325
16     vegetation      73.316%     0.781  0.913
17        terrain      27.315%     0.404  0.483
18            sky      76.252%      0.79  0.936
19         person      18.315%     0.303  0.342
20          rider       10.32%      0.18  0.241
21            car      64.421%     0.695  0.859
22          truck      21.267%     0.358  0.307
23            bus      30.941%     0.518  0.381
24        caravan       5.762%     0.092  0.159
25        trailer       2.069%     0.034  0.112
26          train      17.973%     0.427    0.2
27     motorcycle       9.621%     0.183  0.218
28        bicycle      18.973%     0.347  0.325
```

Figure 1: U-net IOU per class



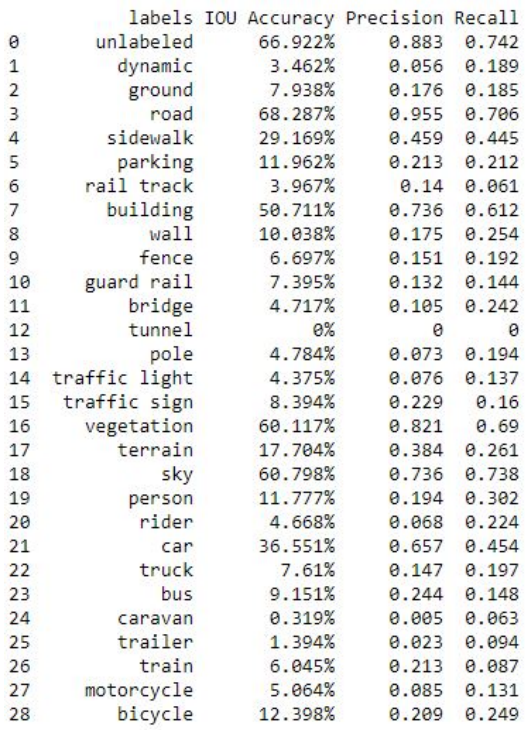
```
           labels IOU Accuracy Precision Recall
0       unlabeled      72.358%      0.85   0.83
1         dynamic        1.68%     0.027  0.169
2          ground       7.236%     0.137    0.2
3            road      89.082%     0.926  0.958
4        sidewalk      38.706%      0.42  0.789
5         parking      14.722%     0.194  0.303
6      rail track       6.093%     0.086  0.226
7        building      60.759%     0.649  0.889
8            wall       6.448%     0.123  0.254
9           fence        8.11%     0.129  0.347
10     guard rail        15.0%     0.164  0.637
11         bridge       2.867%     0.074   0.06
12         tunnel           0%         0      0
13           pole       1.326%     0.022  0.193
14  traffic light        0.5%     0.005   0.14
15   traffic sign       1.061%     0.014  0.177
16     vegetation      62.026%     0.687  0.843
17        terrain      12.831%     0.159  0.329
18            sky      53.904%     0.581  0.807
19         person       7.924%     0.087  0.477
20          rider       1.508%     0.017  0.163
21            car      47.278%     0.508  0.782
22          truck       7.686%     0.145  0.177
23            bus      16.603%     0.325  0.254
24        caravan       2.277%      0.04  0.195
25        trailer       0.485%     0.005  0.135
26          train       5.968%     0.134  0.215
27     motorcycle       1.527%     0.017  0.163
28        bicycle       8.283%     0.109  0.344
```

Figure 2: Segnet IOU per class



```
           labels IOU Accuracy Precision Recall
0       unlabeled       8.681%     0.087    1.0
1         dynamic       0.687%     0.007    1.0
2          ground       4.698%     0.047    1.0
3            road      33.421%     0.334    1.0
4        sidewalk       4.639%     0.046    1.0
5         parking       1.655%     0.017    1.0
6      rail track       2.039%      0.02    1.0
7        building       18.21%     0.182    1.0
8            wall       1.455%     0.015    1.0
9           fence       1.725%     0.017    1.0
10     guard rail       1.647%     0.016    1.0
11         bridge       0.894%     0.009    1.0
12         tunnel           0%         0      0
13           pole       0.618%     0.006    1.0
14  traffic light       0.234%     0.002    1.0
15   traffic sign       0.483%     0.005    1.0
16     vegetation      14.471%     0.145    1.0
17        terrain       1.389%     0.014    1.0
18            sky       3.073%     0.031    1.0
19         person       1.124%     0.011    1.0
20          rider       0.277%     0.003    1.0
21            car       5.606%     0.056    1.0
22          truck       1.596%     0.016    1.0
23            bus       2.134%     0.021    1.0
24        caravan       0.098%     0.001    1.0
25        trailer       0.235%     0.002    1.0
26          train       2.072%     0.021    1.0
27     motorcycle       0.316%     0.003    1.0
28        bicycle       0.773%     0.008    1.0
```

Figure 3: Resnet IOU per class



```
           labels IOU Accuracy Precision Recall
0       unlabeled      66.922%     0.883  0.742
1         dynamic       3.462%     0.056  0.189
2          ground       7.938%     0.176  0.185
3            road      68.287%     0.955  0.706
4        sidewalk      29.169%     0.459  0.445
5         parking      11.962%     0.213  0.212
6      rail track       3.967%      0.14  0.061
7        building      50.711%     0.736  0.612
8            wall      10.038%     0.175  0.254
9           fence       6.697%     0.151  0.192
10     guard rail       7.395%     0.132  0.144
11         bridge       4.717%     0.105  0.242
12         tunnel           0%         0      0
13           pole       4.784%     0.073  0.194
14  traffic light       4.375%     0.076  0.137
15   traffic sign       8.394%     0.229   0.16
16     vegetation      60.117%     0.821   0.69
17        terrain      17.704%     0.384  0.261
18            sky      60.798%     0.736  0.738
19         person      11.777%     0.194  0.302
20          rider       4.668%     0.068  0.224
21            car      36.551%     0.657  0.454
22          truck        7.61%     0.147  0.197
23            bus       9.151%     0.244  0.148
24        caravan       0.319%     0.005  0.063
25        trailer       1.394%     0.023  0.094
26          train       6.045%     0.213  0.087
27     motorcycle       5.064%     0.085  0.131
28        bicycle      12.398%     0.209  0.249
```

Figure 4: IC-net IOU per class

In all the architectures, class road was detected most accurately.
1. U-net : 91.998%
2. Segnet : 89.082%
3. IC-net : 68.287%
4. Resnet : 33.421%

XI AMAZON WEB SERVICES

Amazon web services was used to train the deep neural net models as powerful GPUs were required to perform convolution operation on big dataset like cityscrapes. The dataset was uploaded to Amazon cloud storage service called S3 bucket. Then, using one of the service of AWS called Sagemaker jupyter notebook was launched. The dataset stored on the S3 bucket storage device was made public in order to access the dataset from Sagemaker. The below diagram shows how to dataset is handled:

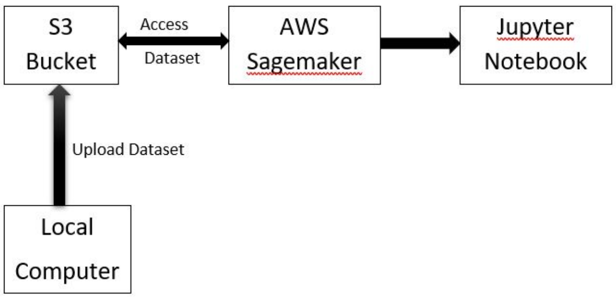


Figure 5: Accesing the dataset from cloud

AWS does not give GPU capabilities to its user by default. Therefore, GPU request need to be sent by the user to the AWS service explaining the need for the GPU. The above mentioned process was followed and the following GPU was allocated:

1) GPU name : Tesla 1xK80
2) Instance type: ml.p2.x.large instance
3) number of vCPUs : 4
4) RAM : 61 GB
5) GPU Memory: 12 GB

XII CONCLUSION

An in depth study of convolutional neural networks used for semantic segmentation is obtained. Different CNN architectures like U-net, Segnet, Resnet and IC-net have been studied and implemented and the results have been compared. By comparing the results, it is clear that U-net performs better than other architectures with a MIOU accuracy of 53.29%. The reason U-net performs better is because in its expansion path, the concatenation of high resolution images from the contraction takes place. This restores the features lost during the upconvolution operation. Hence, it is recommended to use U-net for multiclass semantic segmentation.

XIII ARCHITECTURE SPEED COMPARISON

Architecture speed here refers to the speed at which an architecture can perform semantic segmentation on an image and provide segmented output. In certain real life scenarios, speed matters more than accuracy of the architectures. The following table illustrates the speed of architectures in increasing order:

Architecture Speed Results		
Sno	**Architecture**	**Average speed per sample**
1.	Resnet34	45.42 milli seconds
2.	IC-net	45.5 milli seconds
3.	U-net	121.54 milli seconds
4.	Segnet	128.615 milli seconds

From the table, it can be inferred that Resnet34 architecture performs semantic segmentation quickly than U-net, Segnet and IC-net

XIV REFERENCES

1) Layers of a Convolutional Neural Network. (2020, March 02). Retrieved from https://mc.ai/layers-of-a-convolutional-neural-network/
2) MACMAC 51644 silver badges1414 bronze badges. (1969, October 01). Understanding Gaussian Filter and how to plot it in 1-D. Retrieved from https://stackoverflow.com/questions/60462388/understanding-gaussian-filter-and-how-to-plot-it-in-1-d
3) Anh H. Reynolds. (2017, October 15). Convolutional Neural Networks (CNNs). Retrieved from https://anhreynolds.com/blogs/cnn.html
4) Diterbitkan oleh Benny Prijono Lihat semua pos dari Benny Prijono, Prijono, D., Prijono, B., Lihat semua pos dari Benny Prijono, 14, P., Berkata:, P., . . . (wajib), N. (2018, April 03). Student Notes: Convolutional Neural Networks (CNN) Introduction. Retrieved June 26, 2020, from https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/
5) Sakshi TiwariCheck out this Author's contributed articles., Sakshi Tiwari, amp; Check out this Author's contributed articles. (2018, February 06). Activation functions in Neural Networks. Retrieved June 26, 2020, from https://www.geeksforgeeks.org/activation-functions-neural-networks/
6) Intro to Linear vs. Nonlinear Functions. (n.d.). Retrieved June 26, 2020, from https://www.expii.com/t/intro-to-linear-vs-nonlinear-functions-4318
7) V, A. (2017, March 30). Understanding Activation Functions in Neural Networks. Retrieved July 02, 2020, from https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
8) KIDS, T. (2019, July 31). Fully-Connected Layer with dynamic input shape. Retrieved July 03, 2020, from https://medium.com/@tecokids.monastir/fully-connected-layer-with-dynamic-input-shape-70c869ae71af
9) Singh, S. (2019, March 02). Fully Connected Layer: The brute force layer of a Machine Learning model. Retrieved July 03, 2020, from https://iq.opengenus.org/fully-connected-layer/
10) Uniqtech. (2020, April 21). Understand the Softmax Function in Minutes. Retrieved July 03, 2020, from https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

11) (PDF) A Deep Feature Learning Method for Drill Bits Monitoring Using the Spectral Analysis of the Acoustic Signals. (n.d.). Retrieved July 03, 2020, from https://www.researchgate.net/publication/327007086 A Deep Feature Learning Method for Drill Bits Monitoring Using the Spectral Analysis of the Acoustic Signals
12) Daniil's blog. (n.d.). Retrieved July 03, 2020, from http://warmspringwinds.github.io/tensorflow/tf-slim/2016/11/22/upsampling-and-image-segmentation-with-tensorflow-and-tf-slim/
13) 2018 Data Science Bowl. (n.d.). Retrieved July 06, 2020, from https://www.kaggle.com/c/data-science-bowl-2018/data
14) Jcoral. (2019, August 25). CamVid. Retrieved July 06, 2020, from https://www.kaggle.com/jcoral02/camvid
15) Badrinarayanan, Vijay, et al. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." ArXiv.org, 10 Oct. 2016, arxiv.org/abs/1511.00561.
16) Profile, bdd-data.berkeley.edu/portal.html.
17) Ronneberger, O., Fischer, P.; Brox, T. (2015, May 18). U-Net: Convolutional Networks for Biomedical Image Segmentation. Retrieved August 20, 2020, from https://arxiv.org/abs/1505.04597
18) Zhao, H., Qi, X., Shen, X., Shi, J.; Jia, J. (2018, August 20). ICNet for Real-Time Semantic Segmentation on High-Resolution Images. Retrieved August 20, 2020, from https://arxiv.org/abs/1704.08545
19) He, K., Zhang, X., Ren, S.,; Sun, J. (2015, December 10). Deep Residual Learning for Image Recognition. Retrieved August 20, 2020, from https://arxiv.org/abs/1512.03385
20) Zhang, X., Zou, J., He, K., ; Sun, J. (2015, November 18). Accelerating Very Deep Convolutional Networks for Classification and Detection. Retrieved from https://arxiv.org/abs/1505.06798
21) Zhao, H., Shi, J., Qi, X., Wang, X., ; Jia, J. (2017, April 27). Pyramid Scene Parsing Network. Retrieved August 23, 2020, from https://arxiv.org/abs/1612.01105