# Abstract

# Motivation

Self-driving cars are on the verge of transforming the way we travel. However, there have been hiccups along the way which have derailed the initial hype around this field. But with the Andrew Ng backed Drive.ai initiative, and now with Berkeley’s latest release, the perception that autonomous vehicles are unsafe is giving way to positive developments.

Developing autonomous systems that are able to assist humans in everyday tasks is one of the grand challenges in modern computer science. One example are autonomous driving systems which can help decrease fatalities caused by traffic accidents. While a variety of novel sensors have been used in the past few years for tasks such as recognition, navigation and manipulation of objects, visual sensors are rarely exploited in robotics applications: Autonomous driving systems rely mostly on GPS, laser range finders, radar as well as very accurate maps of the environment

This project's aim is to create a deep neural network trained with the UC Berkeley's dataset which is capable of semantic segmentation of cityscapes. Our main goal is to be able to differentiate vehicles and pedestrians.

Autonomous driving is one of the fastest growing technology areas. From small university-based teams to the big guns like Google and Uber, everyone is determined to be the first to crack the technology that will bring driverless cars to our city streets.

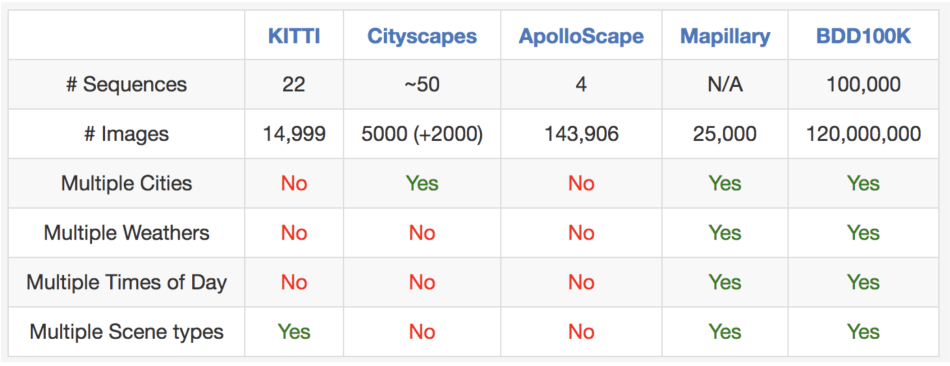
Self-driving cars have got a bad rap recently after an autonomous Uber car hit and killed a pedestrian while traveling in Tempe Arizona. Uber subsequently paused their self-driving development program, but that is not expected to last for long.

# Dataset

UC Berkeley has open sourced the largest and most diverse self-driving dataset for the general public.

# What makes the dataset even more unique and rich to work with is the different weather conditions it has covered, like sunny, overcast, rainy and haze. There is also a good balance between daytime and nighttime scenarios

As you can see in the image below, their claims of this being the largest ever self-driving dataset are not exaggerated in the slightest. Back in March, we saw Baidu release the largest dataset (at that time) in this domain. Berkeley’s release is 800 times larger than that. It’s 4,800 times bigger than Mapillary’s dataset and an incredible 8,000 times bigger than KITTI (let’s not even compare it to the Cityscapes size!).



The demand for these types of datasets has been consistently high and there is no doubt some interesting work will come from the generosity of Berkeley. To coincide with the release of the open source dataset Berkeley has set up three challenges.

The release of this huge dataset means that there is more diversity of data available for researchers and scientists to use in their journey to overcome self-driving car challenges. Berkeley researchers have suggested that they will add to the dataset in the future and expand from only monocular videos to include panorama and stereo videos as well as other types of sensors like LiDAR and radar.

About The Data

This project's aim is to create a deep neural network trained with the UC Berkeley's dataset which is capable of semantic segmentation of cityscapes. Our main goal is to be able to differenciate vehicles and pedestrians.

After considering other datasets as well, our choice fell on the dataset collected and annotated by Berkeley university. It's huge. So we decided to initially only deal with the first 100 samples of segmentated data, but later on we plan to include more. You can download the whole dataset on the following link: http://bdd-data.berkeley.edu/ The licence is included at the end of readme of our github repo.

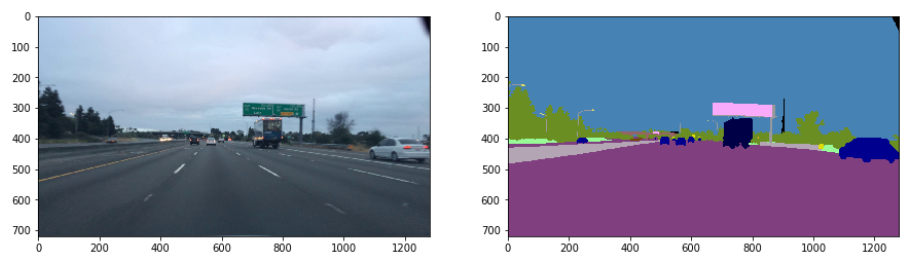
Some of the reasons we chose to work with UC Berkerley's dataset are:

It covers a wide range of driving conditions both regarding daytime and weather

There are over 10,000 samples and corresponding pixel-level annotations for both class-level and instance-level segmentation

After a quick registration, they provide an easy way for downloading the data through google drive

An example of raw data and its annotation:



Preparation of Data

As preparation we split the data into train, validation and test samples, separate them by RGB channels, and standardize it with the help of the StandardScaler.

On a sidenote: Shuffling the data was also intended, but we sadly failed. Later we certainly plan to somehow solve this situation, as we find it necessary, because some of the samples that follow eachother were taken from the same video, and therefore look quite alike, both in terms of the environment and the position of the vehicles.

The .csv file contains metadata about the annotation, like which categories of objects the segmentation differentiates and what color represents each category.

# Data Preprocessing

~~Building an effective neural network model requires careful consideration of the network architecture as well as the input data format.~~

Normalizing image inputs: Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network. Data normalization is done by subtracting the mean from each pixel, and then dividing the result by the standard deviation. The distribution of such data would resemble a Gaussian curve centered at zero. For image inputs we need the pixel numbers to be positive, so we might choose to scale the normalized data in the range [0,1] or [0, 255]. For our data-set example, the following montage represents the normalized data.

Dimensionality reduction: We could choose to collapse the RGB channels into a single gray-scale channel. There are often considerations to reduce other dimensions, when the neural network performance is allowed to be invariant to that dimension, or to make the training problem more tractable.

Data augmentation: Another common pre-processing technique involves augmenting the existing data-set with perturbed versions of the existing images. Scaling, rotations and other affine transformations are typical. This is done to expose the neural network to a wide variety of variations. This makes it less likely that the neural network recognizes unwanted characteristics in the data-set.

# Data augmentation

Training a network on a small dataset generally results in overfitting [48]. To alleviate overfitting, data augmentation is generally applied. It is a technique by which amount of training data is increased using the information within the training data itself.

Think of a person driving a car on a sunny day. If it starts raining, they may initially find it difficult to drive in rain. But slowly they get accustomed to it.

An artificial neural network too finds it confusing to drive in a new environment unless it has seen it earlier. Their are various augmentation techniques like flipping, translating, adding noise, or changing color channel.

# Adam

Adam optimizer: <https://arxiv.org/pdf/1412.6980v8.pdf>

The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

The algorithms leverages the power of adaptive learning rates methods to find individual learning rates for each parameter. It also has advantages of Adagrad [10], which works really well in settings with sparse gradients, but struggles in non-convex optimization of neural networks, and RMSprop [11], which tackles to resolve some of the problems of Adagrad and works really well in on-line settings.

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters.

10 John Duchi, Elad Hazan, and Yoram Singer. [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf). Journal of Machine Learning Research, 12:2121–2159, 2011.

11 Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: [Divide the gradient by a running average of its recent magnitude](https://www.coursera.org/learn/neural-networks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude). COURSERA: neural networks for machine learning, 4(2):26–31, 2012.

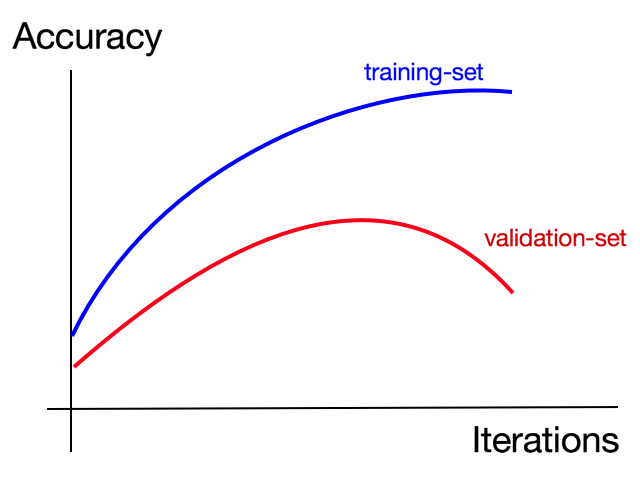
# Early stopping

When training neural networks, numerous decisions need to be made regarding the settings (hyperparameters) used, in order to obtain good performance. Once such hyperparameter is the number of training epochs: that is, how many full passes of the data set (epochs) should be used? If we use too few epochs, we might underfit (i.e., not learn everything we can from the training data); if we use too many epochs, we might overfit (i.e., fit the ‘noise’ in the training data, and not the signal).

Early stopping attempts to remove the need to manually set this value. It can also be considered a type of regularization method (like L1/L2 weight decay and dropout) in that it can stop the network from overfitting.

The idea behind early stopping is relatively simple:

* Split data into training and test sets
* At the end of each epoch (or, every N epochs):
  + evaluate the network performance on the test set
  + if the network outperforms the previous best model: save a copy of the network at the current epoch
* Take as our final model the model that has the best test set performance



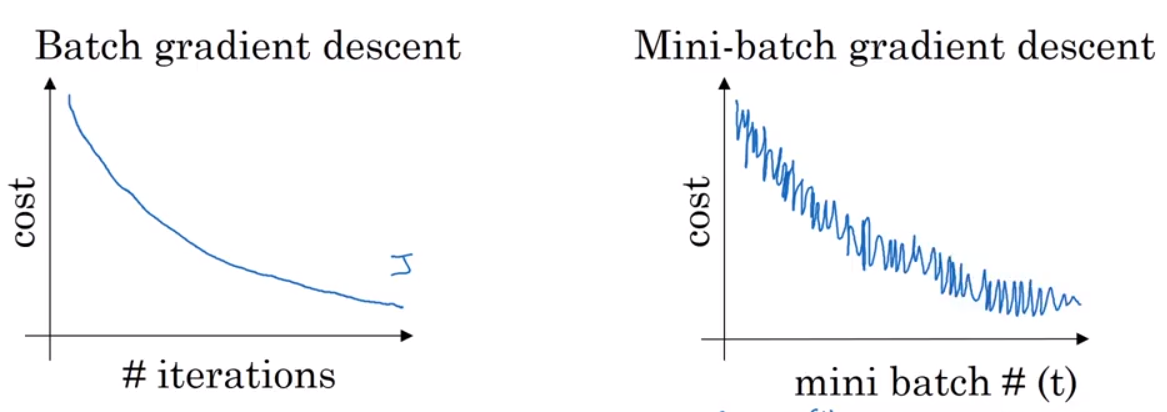
If the training accuracy increases(positive slope) while the validation accuracy steadily decreases(negative slope) then a situation of over fitting may have occurred. Time to stop Training.

# Mini batch

One problem we face when we use vanilla gradient descent (another term for our plain old buddy gradient descent) is the memory problem. Our computers can only handle so much, so what if we’re dealing with millions, if not and more commonly, billions of data? Luckily, as the name implies, mini-batch gradient descent uses the same methods as vanilla gradient descent but only on a smaller scale. We create batches from our training data and train our model on that smaller batch, updating our slope and bias term all the while.

For every epoch, we train our model per batch. This will cause us to create a nested for-loop. A necessary sacrifice in my opinion as opposed to feeding the huge dataset altogether. One problem with mini-batch gradient descent is that our increased number of parameter updates with high variance cause fluctuations in our cost function. There are times when this will cause our gradient to jump to lower local minima because of overshooting, which makes it superior to vanilla gradient descent because mini-batch gradient descent just doesn’t stay within the area of the local minima — it can go beyond that to find other lower local minima.

Instead of implementation of gradient descent on the entire training set, we can split our training set into smaller sets and implement gradient descent on each batch one after the other. In this way, we can get an intuition of gradient descent before finishing entire training set. It makes the algorithm faster and more efficient. It is called mini batch gradient descent.



In contrast with batch gradient descent where cost function smoothly decays, mini-batch gradient has a little bit noisy cost function but still trend downwards. The reason for the noise is that there are some harder mini batches which cause oscillation to cost function.

The most important thing is to select the size of each mini batch. If size is very large (say m) then it will again behave as batch gradient descent (very slow) and if it is very small (say 1) then each example will be a mini batch and it becomes stochastic gradient descent with very large noise. It won’t ever converge and it’ll always just kind of oscillate and wander around the region of the minimum.

Upsides

* The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
* The batched updates provide a computationally more efficient process than stochastic gradient descent.
* The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

Downsides

* Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm.
* Error information must be accumulated across mini-batches of training examples like batch gradient descent.

# Categorical crossentropy

A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model.

We often see categorical\_crossentropy used in multiclass classification tasks.

# Semantic segmentation

~~The problem of semantic segmentation consists of associating a class label to each pixel of a given image, resulting in another image of semantic labels. This problem of image understanding is highly relevant in the context of mobile robotics and autonomous vehicles, for which accurate information of the objects in the scene may be applied for decision making or safe and robust navigation among others.~~

~~Unlike classification where the end result of the very deep network is the only important thing, semantic segmentation not only requires discrimination at pixel level but also a mechanism to project the discriminative features learnt at different stages of the encoder onto the pixel space.~~

~~Semantic segmentation has seen a rapid progress over the past decade. Recent advances achieved by training different types of Convolutional Neural Networks (CNN) have improved notably the accuracy of state-of-the-art techniques~~. ~~Among the many CNN architectures available, convolutional encoder-decoder networks are particularly well adapted to the problem of pixel labeling.~~ ~~The encoder part of the network creates a rich feature map representing the image content and the decoder transforms the feature map into a map of class probabilities for every pixel of the input image.~~ Such operation takes into account the pooling indices to upsample low resolution features into the original image resolution.

*~~Semantic segmentation is one of the key problems in the field of computer vision, as inferring knowledge form images plays a very important role in medical analysis, object detection in satellite images, iris recognition, autonomous vehicles, and many more tasks. It is probably amongst the best approaches towards complete scene understanding.~~*

~~Image segmentation means assigning a semantic annotation label to each pixel in the image, so that each pixel is labeled with the class of its enclosing object. Therefore, image segmentation is also categorized as a dense prediction task. In itself it does not distinguish between object instances, only object categories.~~

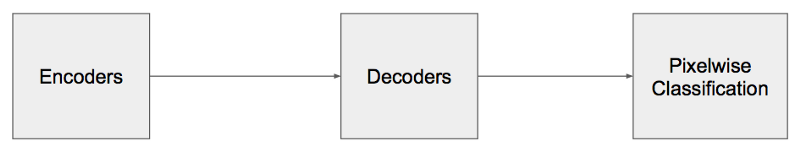
~~The goal of semantic segmentation is to assign each pixel of a photograph to one of several semantic class labels (or to none of them). This is a supervised learning problem which requires training a set of classifiers from data labelled at the pixel level.~~

Semantic segmentation has many potential applications including scene understanding, removing undesired objects from photographs, copy-pasting objects from one photograph to another, or local class-based image enhancement

Networks used for semantic segmentation typically take an RGB image as input data, and have a label which is an n channel image, where n is the number of labels involved. Each channel corresponds to a label, for example cars, road, etc, and each pixel in a certain channel will be 1 or 0 depending on whether that pixel belongs to the label corresponding to that channel.

~~After some research in the field and considering other architectures (like FCNs and U-net) as well, we chose SegNet [paper ref] as our main inspiration for our project.~~

# Segnet:



~~Direct adoption of classification networks for pixel wise segmentation yields poor results mainly because max-pooling and subsampling reduce feature map resolution and hence output resolution is reduced.~~ Even if extrapolated to original resolution, lossy image is generated

~~In this study, we choose the SegNet architecture (Figure 2), which is designed to be an efficient architecture for pixel-wise semantic segmentation. It is primarily motivated by road scene understanding applications which require the capabilities of modellin gappearance, shape and understanding the spatial-relationship (context) between different classes. At the same time, it provides a good balance between accuracy and computational cost.~~ Moreover, SegNet's symmetrical architecture and its use of the pooling/upsampling combination is very **effective for precise re-localisation of features**

~~SegNet, one of the deep convolutional encoder-decoder architectures, consists of two main networks encoder and decoder~~, ~~and~~ **~~some outer layers~~**~~. The two outer layers of the decoder network are responsible for feature extraction task, the results of which are transmitted to the next layer adjacent to the last layer of the decoder network. This layer is responsible for pixel-wise classification (determining which pixel belongs to which class).~~ There is no fully connected layer in between feature extraction layers. In the upsampling layer of decoder, pool indices from encoder are distributed to the decoder where the kernel will be trained in each epoch (training round) at the convolution layer. ~~In the last layer (classification), softmax is used as a classifier for pixel-wise classification~~. ~~The encoder network consists of convolution layer and pooling layer. A technique, called batch normalization (proposed by Ioffe and Szegedy [27]), is used to speed up the learning process of the DCNN by reducing internal covariate shift. In the encoder network, the number of layers is reduced to 13 (VGG16) by removing the last three layers (fully connected layers) [6,8,28,29] for the following two reasons: to maintain the high-resolution feature maps in the encoder network, and to minimize the countless number of parameters from 134 million features to 14.7 million features compared to the traditional deep learning networks such as DCNN [4] and DeCNN [5], where the fully connected layer remains intact. In the activation function of feature extraction, ReLU, max-pooling, and 7 \_ 7 kernels are used in both encoder and decoder networks. For training images, three-channel images (RGB) are used.~~

27 Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv 2015, arXiv:1502.03167.

6 Badrinarayanan, V.; Handa, A.; Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. arXiv 2015, arXiv:1505.07293.

8 Badrinarayanan, V.; Kendall, A.; Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. arXiv 2015, arXiv:1511.00561.

28 Panboonyuen, T.; Vateekul, P.; Jitkajornwanich, K.; Lawawirojwong, S. An Enhanced Deep Convolutional Encoder-Decoder Network for Road Segmentation on Aerial Imagery. In Recent Advances in Information and Communication Technology Series, Proceedings of International Conference on Computing and Information Technology, Tunis, Tunisia, 27–28 April 2017; Springer International Publishing: Cham, Switzerland; Volume 566.

29 Kendall, A.; Badrinarayanan, V.; Cipolla, R. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. arXiv 2015, arXiv:1511.02680.

4 Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 8–10 June 2015; pp. 3431–3440.

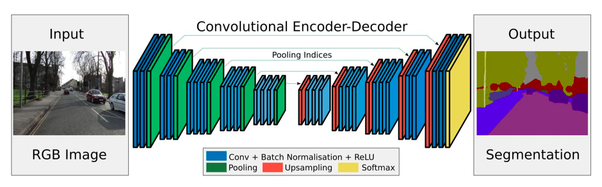
It is a convolutional neural network (CNN) that performs image segmentation. This means that the network learns to assign each pixel a class depending on the object or surface it belongs, e.g a car, highway, tree, building...

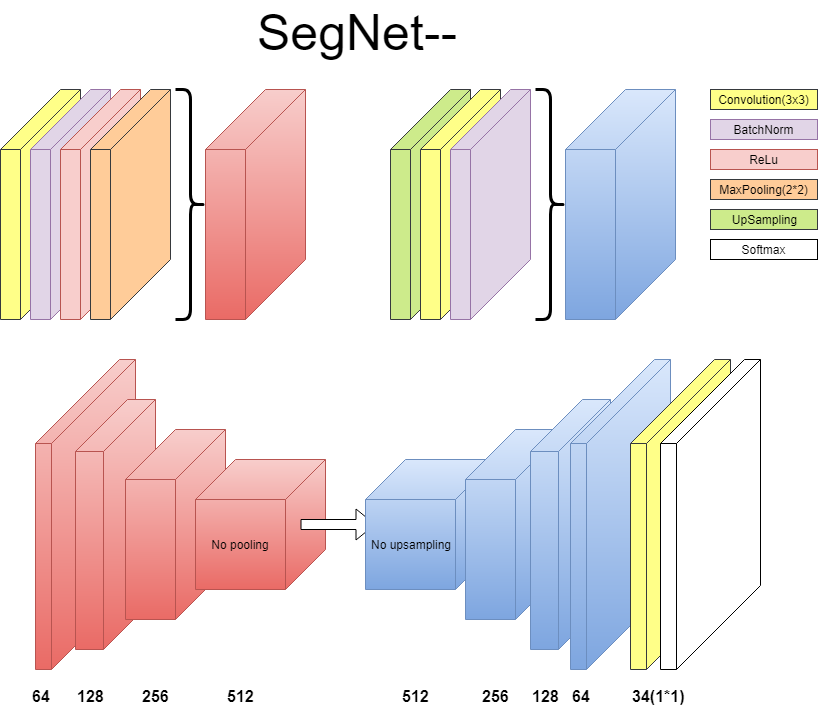
~~It uses an Encoder-Decoder architecture, were the image is first downsampled by an encoder as in a "traditional" CNN like VGG, and then it is upsampled by using a decoder that is like a reversed CNN, with upsampling layers instead of pooling layers.~~

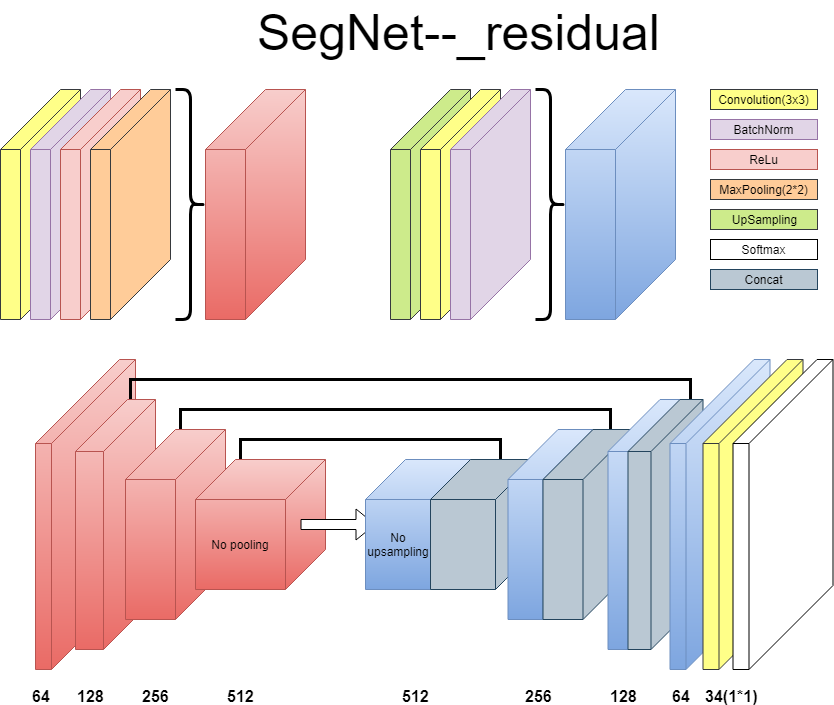
~~The SegNet has an encoder and decoder approach. The encode has various convolution layers and decoder has various deconvolution layers. SegNet improved the coarse outputs produced by FCN. Because of this, it is less intensive on memory. When the features are reduced in dimensions, it is upsampled again to the image size by deconvolution, reversing the convolution effects.~~ Deconvolution learns the parameters for upsampling. The output of such architecture will be coarse due to the loss of information in pooling layers.

----------

~~SegNet consists of layers called encoders and decoders. Each encoder applies convolution, batch normalization and a non linearity, then applies max pooling on the result, while storing the index of the value extracted from each window. Decoders are similar to the encoders, the difference is that they don’t have a non linearity, and they upsample their input, using indices stored from the encoding stage. After the final decoder, the output is fed to a softmax classifier which gives the final prediction.~~ ~~The prediction is going to be an n channel image so we have to write a separate function to convert it into an RGB image where we can look at the results qualitatively.~~

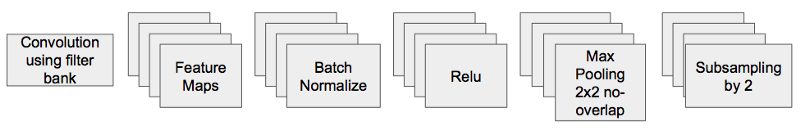






~~The SegNet architecture also follows the encoder-decoder pattern, as most semantic segmentation architectures.~~

# The encoder:



Each encoder is like Fig 3

~~13 VGG16 Conv layers --- The architecture of the encoder network is topologically identical to the 13 convolutional layers in the VGG16 network~~

~~Not fully connected, this reduces parameters from 134M to 14.7M~~

~~The encoder of the network contains a fully convolutional VGG-16. VGG-16 uses a stack of convolutional layers with small receptiove fields in the first layers instead of fewer layers with bigger receptive fields.~~

# The decoder:

~~The decoder has repeated upsampling layers followed by convolutional layers. The decoder network is the inverse of the encoder: it has the same type of convolutions as the encoder network, regarding filter sizes, and the channels of corresponding layers.~~

~~In SegNet upsampling does not take part in the learning process. It is a sort of backward max-pooling operation.~~

# Residual connections:

~~During the forward pass in the encoder, while downsampling, the max-pooling indices are stored – meaning the location of the highest value pixel In the pax-pooling window at each sliding position of the layer. These are then used when the corresponding upsampling layer is working in the decoder. The remaining pixels in the upsampled output are set to zero.~~

“Hence, the decoder network of SegNet consists of a hierarchy of decoders, one corresponding to each encoder and the appropriate decoder uses the max-pooling indices from the corresponding encoder to perform non-linear upsampling of their input feature maps.

One thing to note is that the decoder corresponding to the first encoder (closest to the input image) produces a multi-channel feature map, although its encoder input has 3 channels (RGB). This is unlike the other decoders in the network which produce feature maps with the same number channels and size as their encoder inputs. This is done for generating the segmentation masks for each class plus background.”

<https://mohitjain-me.cdn.ampproject.org/c/s/mohitjain.me/2018/09/30/a-look-at-image-segmentation/amp/>

# Evaluation

Evaluating your machine learning algorithm is an essential part of any project. Your model may give you satisfying results when evaluated using a metric say accuracy\_score but may give poor results when evaluated against other metrics such as logarithmic\_loss or any other such metric. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model. When evaluating a standard machine learning model, we usually classify our predictions into four categories: true positives, false positives, true negatives, and false negatives. However, for the dense prediction task of image segmentation, it's not immediately clear what counts as a "true positive" and, more generally, how we can evaluate our predictions.

The most common strategy is to consider segmentation as a pixel-level classification problem and to evaluate it using a confusion matrix at the pixel level.

Overall and per-class accuracies over the entire dataset were the first ones to be reported [18]. The intersection-over-union segmentation measure also known as Jaccard index [4], which counts the total number of mislabeled pixels in the image, is now defacto standard.

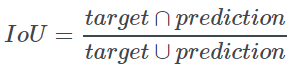
In this paper, we raise the following question: “what is a good semantic segmentation measure?”, and show that the answer is not as trivial as it sounds. Our contribution is threefold. First, we draw the attention of the community to this question that, in our opinion, has been largely overlooked, and review existing semantic segmentation measures. We show that different segmentation algorithms might be optimal for different segmentation measures.

The Jaccard index, also referred to as the intersection-over-union score, is commonly employed in the evaluation of image segmentation results given its perceptual qualities, scale invariance – which lends appropriate relevance to small objects, and appropriate counting of false negatives, in comparison to per-pixel losses.

IoU = TP ⁄ (TP+FP+FN) [[1](https://www.cityscapes-dataset.com/benchmarks/#paperkey_0)], where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set.

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.

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.



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

The IoU score is calculated for each class separately and then averaged over all classes to provide a global, mean IoU score of our semantic segmentation prediction.

https://people.cs.pitt.edu/~kovashka/cs1699/hw4.html

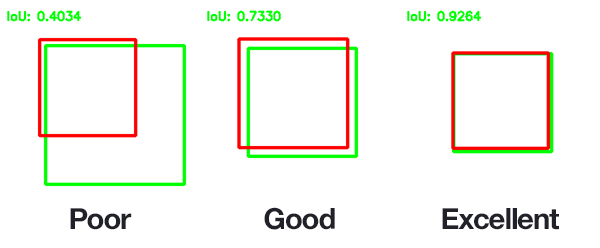
If you have performed any previous machine learning in your career, specifically classification, you’ll likely be used to predicting class labels where your model outputs a single label that is either correct or incorrect.

This type of binary classification makes computing accuracy straightforward; however, for object detection it’s not so simple.

In all reality, it’s extremely unlikely that the (x, y)-coordinates of our predicted bounding box are going to exactly match the (x, y)-coordinates of the ground-truth bounding box.

Due to varying parameters of our model (image pyramid scale, sliding window size, feature extraction method, etc.), a complete and total match between predicted and ground-truth bounding boxes is simply unrealistic.

Because of this, we need to define an evaluation metric that rewards predicted bounding boxes for heavily overlapping with the ground-truth:



n the above figure I have included examples of good and bad Intersection over Union scores.

As you can see, predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap. This makes Intersection over Union an excellent metric for evaluating custom object detectors.

We aren’t concerned with an exact match of (x, y)-coordinates, but we do want to ensure that our predicted bounding boxes match as closely as possible — Intersection over Union is able to take this into account.