In GOD WE TRUST

Signals and Systems Course

1397-1398 Second Semester

Optional project

Omid Sharafi
96101838
Hamed Nejat
96102578

Instructor:
Dr.Hamid Aghajan

July 1, 2019

Sharif
University
of
Technology

Optional Project

EE

SUT

Contents

1 Ford Fulkerson method **2**
- 1.1 q(A,B) . . . 2
- 1.2 q̃(A,B) . . . 2
- 1.3 Min Cut example . . . 4
- 1.4 Ford-Fulkerson algorithm base . . . 4
  - 1.4.1 Description . . . 4
  - 1.4.2 Simple algorithm . . . 5
  - 1.4.3 Time complexity . . . 6
  - 1.4.4 Common implementation . . . 6
  - 1.4.5 Edmonds-Karp . . . 7
  - 1.4.6 Additional equivalent implementation (Dijkstra min cutset) . . . 7

2 K-Means **8**
- 2.1 K-Means algorithm base . . . 8
- 2.2 NP-Hard! . . . 8
- 2.3 K-Means usage for this problem . . . 8
- 2.4 K-Means algorithm output . . . 9

3 Optional method **10**

4 Methods Comparison **13**

1 Ford Fulkerson method

1.1 q(A,B)

$$q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, i \in A, j \in B} p_{ij}$$

In the above equation, we should maximize it because for the two first terms if part A is for airplane and part B is for sky so that probability of being airplane (a_i) for pixels in A region and probability of being sky (b_i) for pixels in B region should be maximized and for last term, the relativity of two parts should be as small as possible so that because of minus sign, this shows that we should maximize q.

1.2 q̃(A,B)

$$\tilde{q}(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in E, i \in A, j \in B} p_{ij}$$

$$= \sum_{i \in A} (1 - a_i) + \sum_{j \in B} (1 - b_j) + \sum_{(i,j) \in E, i \in A, j \in B} p_{ij}$$

$$= \sum_{i \in A} 1 + \sum_{i \in B} 1 - (\sum_{i \in A} b_i + \sum_{j \in B} a_j - \sum_{(i,j) \in E, i \in A, j \in B} p_{ij})$$

$$= N(cte) - q(A,B)$$

So as we can see above, based on previous part as we showed that q(A,B) should be maximized so q̃(A,B) should be minimized.
Even we could directly say that for first two terms, probability of being sky (b_i) in airplane region (A) and being airplane (a_i) in sky region (B) should be minimized and the relativity of two parts should be as small as possible so that q̃(A,B) should be minimized.

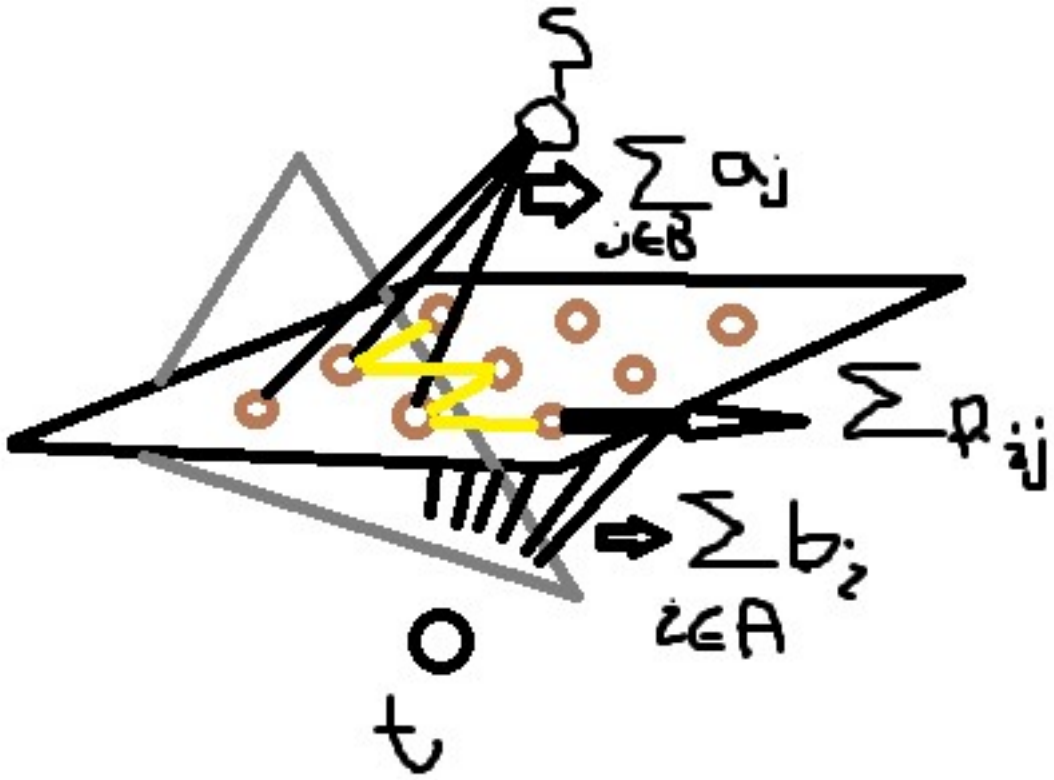

Figure 1: Find min cut of graph

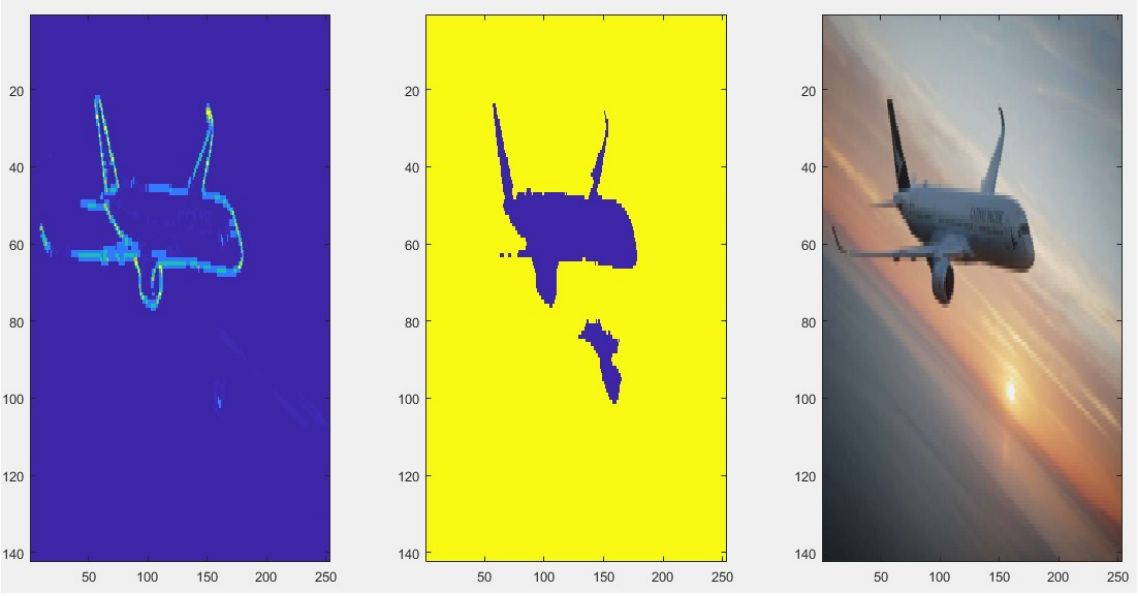

Figure 2: Output for picture 1

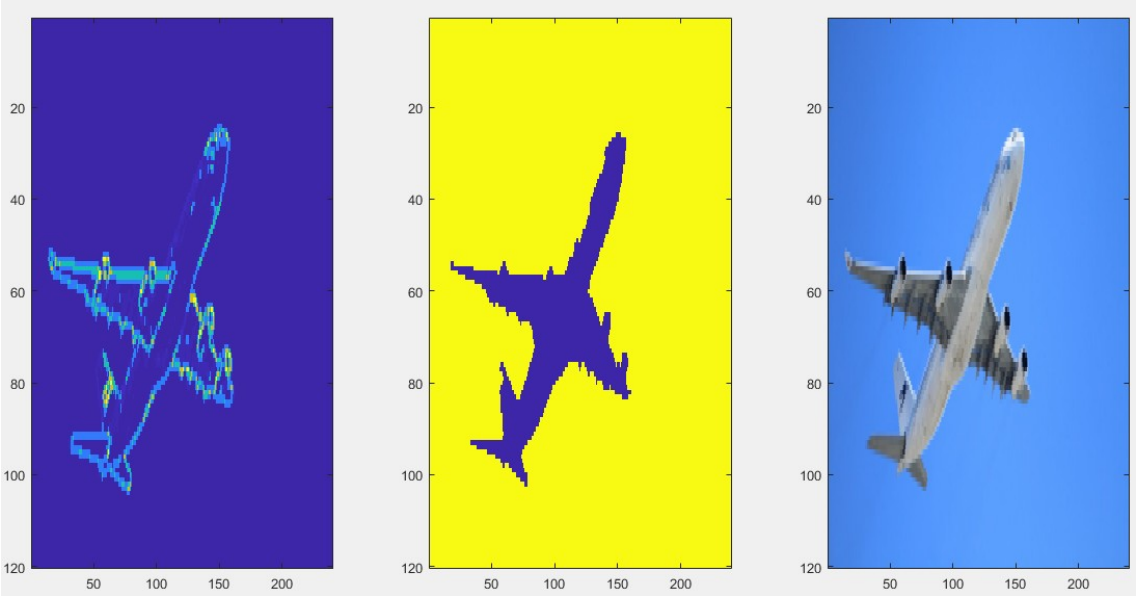

Figure 3: Output for picture 2

1.3 Min Cut example

As we can see in Figure 1 when we find min cut of graph, if weights of s edges is a_i, weights of s edges is b_i and weights between nodes of graph is p_{ij} then it is equivalent of minimizing above function.

1.4 Ford-Fulkerson algorithm base

- credit to Geeks4Geeks

1.4.1 Description

The Ford–Fulkerson method or Ford–Fulkerson algorithm (FFA) is a greedy algorithm that computes the maximum flow in a flow network.

Given a graph which represents a flow network where every edge has a capacity. Also given two vertices source 's' and sink 't' in the graph, find the maximum possible flow from s to t with following constraints:

- a) Flow on an edge doesn't exceed the given capacity of the edge.
- b) Incoming flow is equal to outgoing flow for every vertex except s and t.

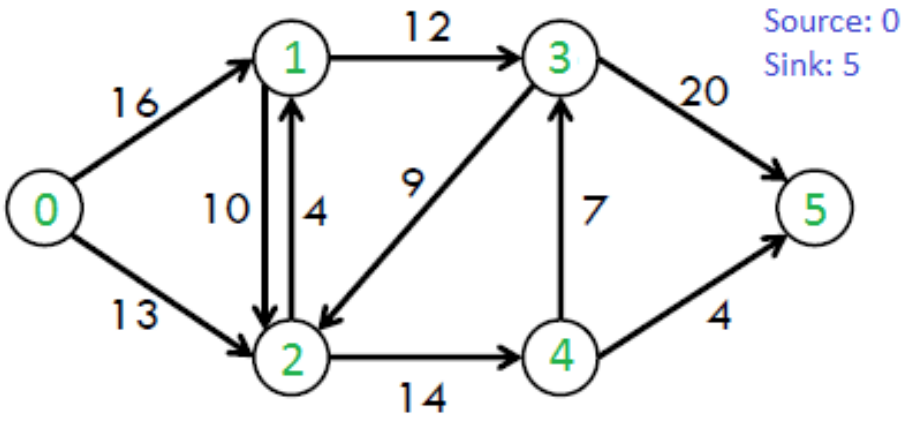

Figure 4: Graph of example 1

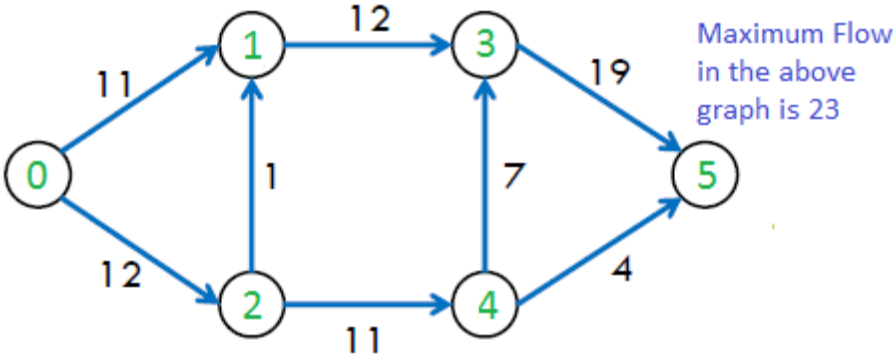

Figure 5: Flow graph of example 1

For example, consider the following graph from CLRS book; (4, 5)

1.4.2 Simple algorithm

The following is simple idea of Ford-Fulkerson algorithm:
1) Start with initial flow as 0.
2) While there is a augmenting path from source to sink. Add

this path-flow to flow.
3) Return flow.

1.4.3 Time complexity

Time Complexity: Time complexity of the above algorithm is O(maxFlow * E). We run a loop while there is an augmenting path. In worst case, we may add 1 unit flow in every iteration. Therefore the time complexity becomes O(maxFlow * E).

1.4.4 Common implementation

Let us first define the concept of Residual Graph which is needed for understanding the implementation. Residual Graph of a flow network is a graph which indicates additional possible flow. If there is a path from source to sink in residual graph, then it is possible to add flow. Every edge of a residual graph has a value called residual capacity which is equal to original capacity of the edge minus current flow. Residual capacity is basically the current capacity of the edge. Let us now talk about implementation details. Residual capacity is 0 if there is no edge between two vertices of residual graph. We can initialize the residual graph as original graph as there is no initial flow and initially residual capacity is equal to original capacity. To find an augmenting path, we can either do a BFS or DFS of the residual graph. We have used BFS in below implementation. Using BFS, we can find out if there is a path from source to sink. BFS also builds $parent[]$ array. Using the $parent[]$ array, we traverse through the found path and find possible flow through this path by finding minimum residual capacity along the path. We later add the found path flow to overall flow. The important thing is, we need to update residual capacities in the residual graph. We subtract path flow from all edges along the path and we add path flow along the reverse edges We need to add path flow along reverse edges because may later need to send flow in reverse direction (See following link for example).

1.4.5 Edmonds-Karp

The above implementation of Ford Fulkerson Algorithm is called Edmonds-Karp Algorithm. The idea of Edmonds-Karp is to use BFS in Ford Fulkerson implementation as BFS always picks a path with minimum number of edges. When BFS is used, the worst case time complexity can be reduced to $O(VE^2)$. The above implementation uses adjacency matrix representation though where BFS takes $O(V^2)$ time, the time complexity of the above implementation is $O(EV3)$ (Refer CLRS book for proof of time complexity)

1.4.6 Additional equivalent implementation (Dijkstra min cut-set)

Beside Edmonds-Karp algorithm that uses BFS for traversing the graph, there is an algorithm for segmentation based on similarity in the graph net. In this algorithm, we have probabilities of similarity between nodes, these probabilities are not assumed as flow capacity, they are edge weights. so this algorithm gives n vertices and n threshold, then starts traversing in the graph using Dijkstra's algorithm from all of them, and based on threshold, it gives us the n-segmentation.

2 K-Means

2.1 K-Means algorithm base

In K-Means algorithm, minimizing that term means that we cluster image in K parts and each pixel is related to it's nearest set by minimizing distance in each cluster so that pixels in each cluster are near and some how the same. So one of the most important things is deciding logical K and defining F function. For example if we want that clusters be physically near to each other (in addition of their color) then we should affect distant in our cost function F.

2.2 NP-Hard!

One of the original ways is greedy method. In this method, first for initial state we can use random nodes and more intelligent for this case usually around of picture is sky and we can use this fact not only for initial state but also in the iteration we can force them to be in cluster of sky.
After starting, in each step we assign each observation to the cluster whose mean has the least squared euclidean distance, this is intuitively the "nearest" mean. Then we update centers of clusters.
For it's convergence, if we could really minimize the term then by invariance law, we would converge to answer and by checking if we have change in clusters or not, we could stop algorithm. But main problem is NP-Hard and we can match this problem with other NP-Hard problems. So with greedy algorithm we do not have any guarantee that the algorithm find the optimum.

2.3 K-Means usage for this problem

For this problem we have 2 clusters (sky and airplane) and we can even boost our convergence by using some facts that we have about sky and plane. For example we can use shape of plane or location of plane and sky or the fact that pixels of plane are together.

2.4 K-Means algorithm output

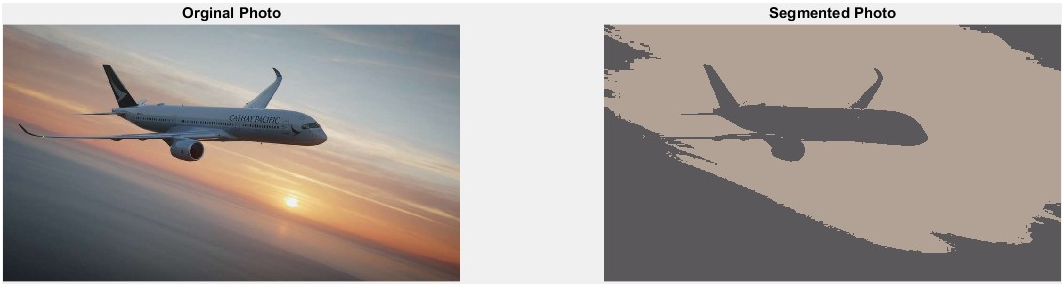

Figure 6: Output (using K-Means algorithm)

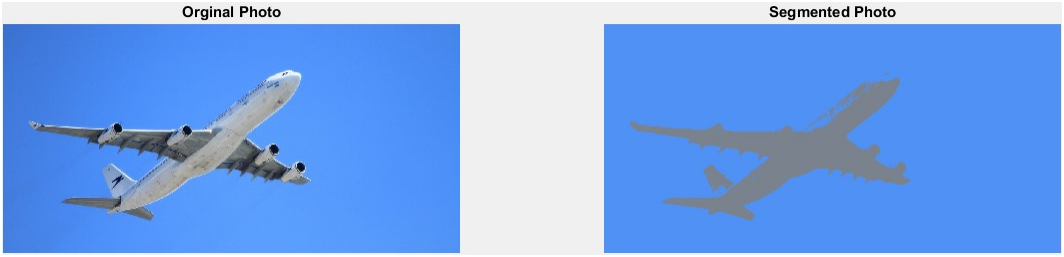

Figure 7: Output (using K-Means algorithm)

The main problem that we have for clustering in figure 6 is that background color is really like airplane and because we do not have any element for distance in cost function, result is not good. One of the best ways to improve output accuracy is to first cluster with high threshold and then fill it's holes or grow it's boarders.

3 Optional method

- **Region growing and merging :** In all of this family methods such as region growing and merging, super pixel and region competition we first assume some regions and then check a probability tests so that we decide merge them or not. This tests can be act in two ways, in all region or in region's boarders. For example for region growing and merging we check the equation below :

$$\frac{(n_1+n_2)(\mu_1-\mu_2)^2}{n_1\sigma_1^2+n_2\sigma_2^2} < Th?Merge : Nop$$

  In others methods for example in Region competition algorithms, we do not merge regions, instead, regions with better statistical assumptions will win competition and grow, so for energy first we have :

$$E[\Gamma,\alpha_i] = \sum_{i=1}^{M} \frac{\mu}{2} \oint_{\partial Ri} ds - \iint_{Ri} logP(I_{(x,y)} : \alpha_i)dxdy + \lambda$$

  so we have two steps in algorithm, first we fix Γ_i and update α_i and then we fix α_i and update Γ_i.

  - Complexity: for each pixel, the equation is being computed and variance is being computed for each set of pixels in the image, it can be proved that the complexity is $O(N^2K^2)$ memory complexity is $O(N^2Log[k])$ if 2D segment tree algorithm is used for mean an variance computing.

- **Snake and Balloon method:** This method is also like above method but we check borders for grow. This method needs good initial regions.

  - Complexity: again, computation is required for each pixel, but variance update is not necessary with that precision. so the Time is $O(N^2K)$ and memory is $O(N^2)$.

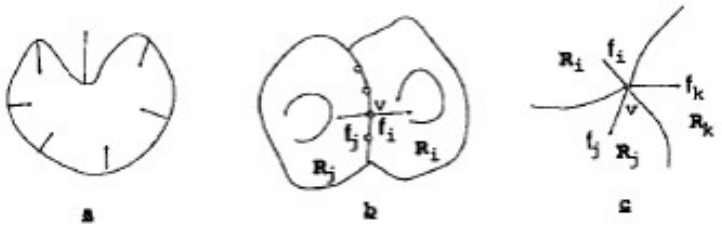

Figure 8: Region growing algorithm example

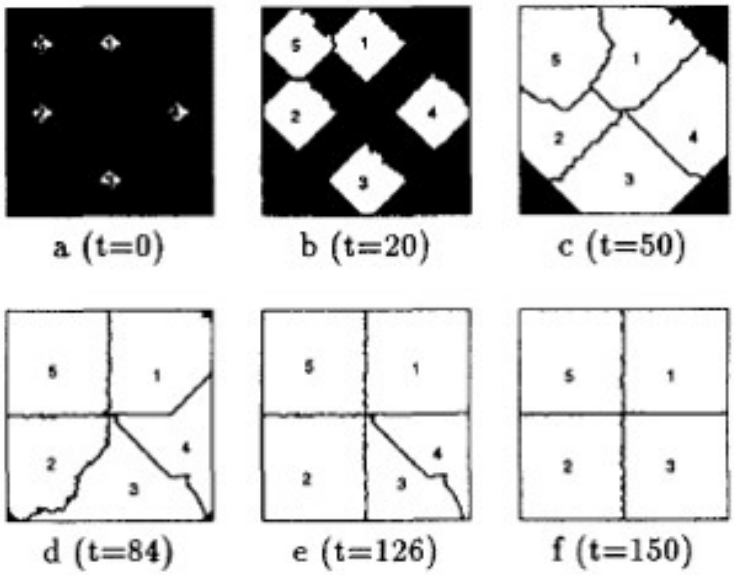

Figure 9: Region growing algorithm example

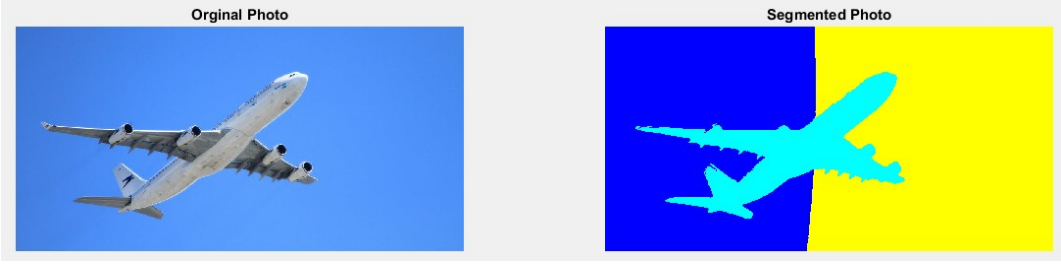

Figure 10: Output (using SuperPixel algorithm)

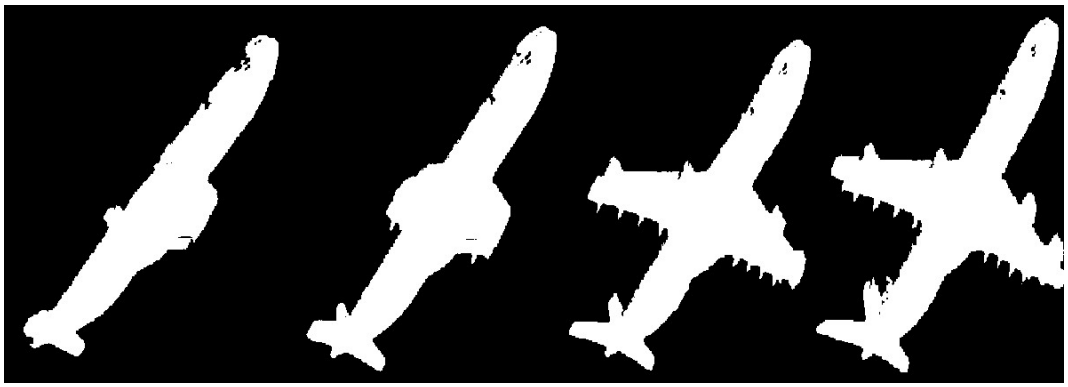

Figure 11: Output (using Snake and Balloon algorithm)

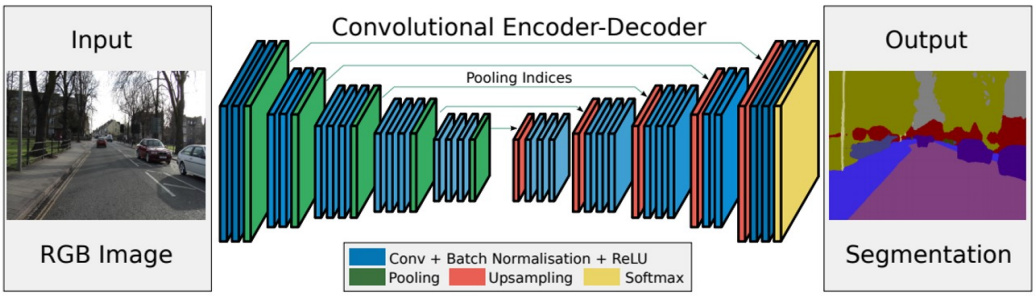

Figure 12: Neural net

Figure 13: Output (using threshold method)

4 Methods Comparison

- **Ford-Fulkerson:**
  - deterministic, if initial condition is acceptable
  - initial condition is required
  - more complex than other methods
- **K-Means algorithm:**
  - non-deterministic
  - initial condition is very important in images, if it was not good, there will be some tiny clusters and some huge clusters.
  - low complexity relative to Cut-set
- **Region growing and merging :**
  - it often generates irregular boundaries and small holes
  - same means distributions problem
  - convergence problem
  - acceptable memory and time complexity
- **Snake and Balloon method:**
  - lower complexity than other methods, memory and time.
  - give us seamlessly clusters
  - require good initial estimates
  - takes to long to converge
  - boundary narrow-sightedness