Short Answer Problems:

1. The search space is linear because of the epipolar constraint.
2. Instead of looking to find a 1-1 mapping of pixels between the left and right images. We ca examine pixels by defining a patch around each pixel and then finding the corresponding patch in the respective image. The three constraints for examining pixels jointly are the following:

   Uniqueness: any point in one image, there should be at most one matching in the other image3

   Ordering: corresponding points should be in the same order in both views

   Smoothness: expect disparity values to change slowly (for the most part)
3. In the K nearest neighbors algorithm, varying K will have an effect on accuracy. Having K=1 means that we will simply take the closest point's label and assign that to our testing point. There is no K that works against all data, it is dependent on the dataset being used.
4. Parametric learns mapping function from input to output, complexity of function will not change.

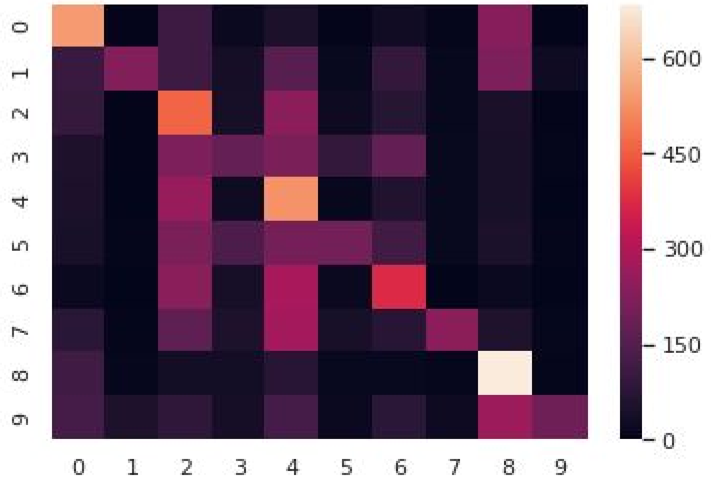
   Non-parametric doesn't learn the mapping function but instead grows in complexity as the dataset grows as well. This is useful because non-parametric methods can be applied on all types of data without a lot of reworking needed.

Image Classification:

Step 0: Dataset downloaded in python script

Step 1:

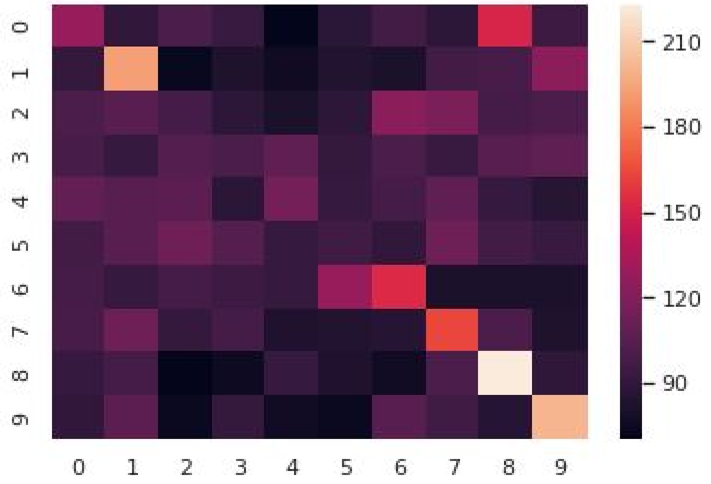
```
Time Taken: 193.5
Accuracy: 0.33
```



Step 2:

`TIME: 658.9`

Accuracy: 0.15



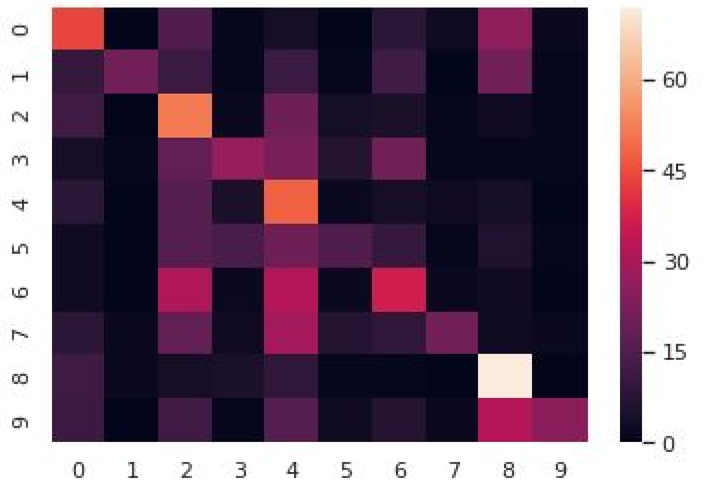
Step 3: KNN outperforms my svm solution in every class. SVM's running time is also three times as large.

Step 4:

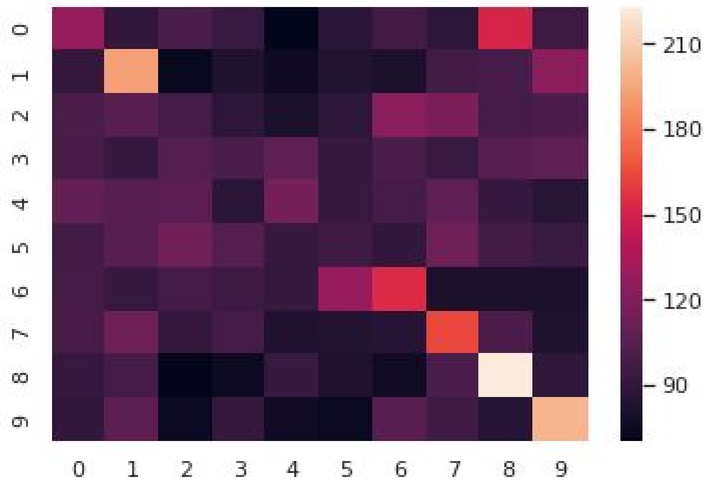
K used:12

Time Taken: 1416.7

Accuracy: 0.37

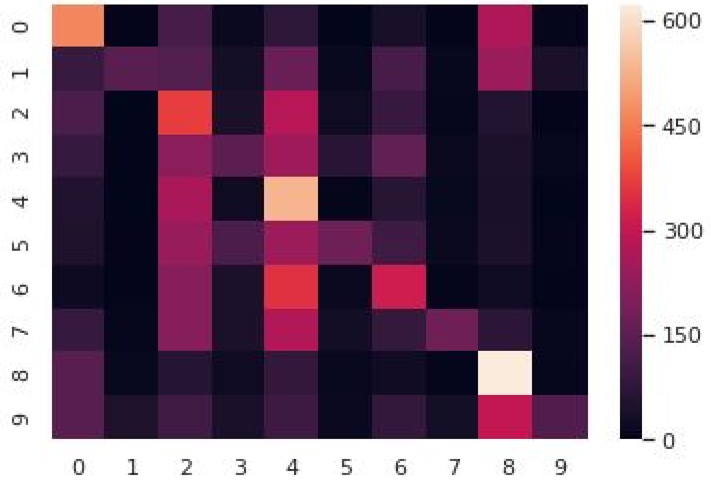


Step 5: After performing multiple runs of cross validation with different values for Cs. I varied them with the largest being at 1 and the lowest being at 1e-8. The cost that returned the best results was the following: 4e-4



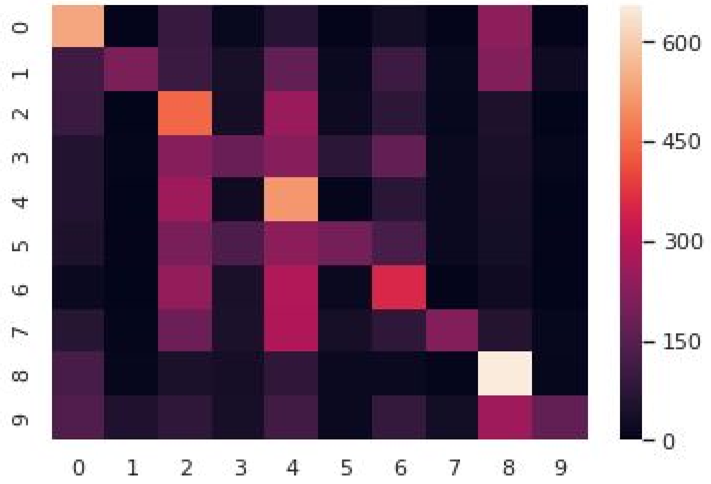
Step 6: After running KNN with 10k images of training data.

Time Taken: 33.148406982421875
Accuracy: 0.3082



Cross validation on 10k gave the following optimized K: 11

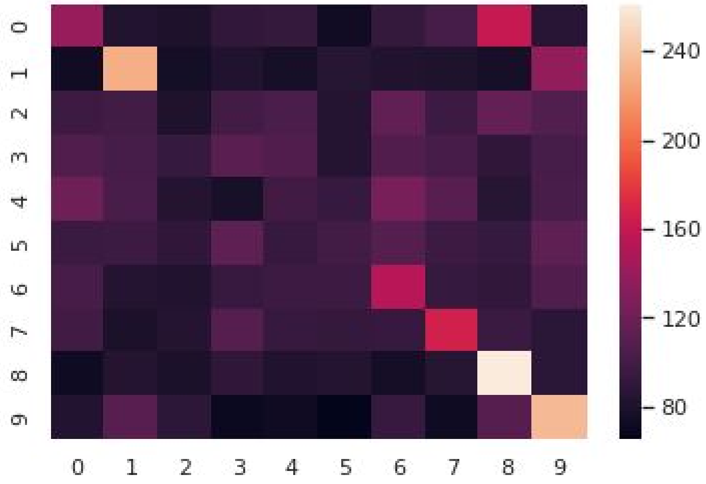
Time Taken: 105.45431709289551
Accuracy: 0.3478



With 30k worth of training data

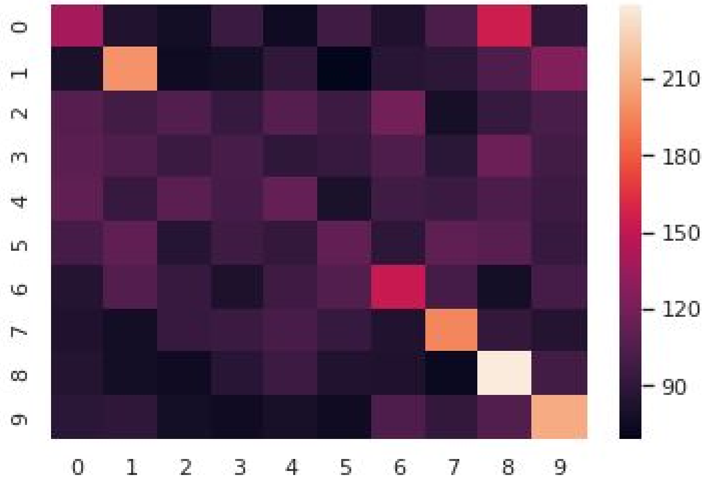
Step 7:
On 10k images:

Accuracy: 0.1578



On 30k images:

Accuracy: 0.157



Step 8:

To use a different feature representation I use Principal Component Analysis. I reduce the dimension size from 3072 to 221 while maintaining 95% of the variance rate. This helps tremendously while performing KNN since KNN suffers from the curse of dimensionality. I am no longer using the pixel values to represent images but PCA to represent the features in a different fashion.