REPORT

- Prajwal Krishna

So since I have been given two set of images - frames and slides and I have to match them, I first made modules for taking input and formatting output, along with a module for comparing each frame with each slide and then take store the slide corresponding to each frame for which we get the largest value of evaluation function, which is supposed to give me the amount of similarity between two images, so essentially the project is reduced to searching best fit for this evaluation function.

First I tried using nor2corr and least square difference between slides to make this eval function but they don't seem to work very well because the images are not perfectly aligned pixel to pixel, nor perfectly scaled and color-contrast balanced. Secondly, these methods do not take any account of objects and features of image and are limited to individual pixel level only.

So looking on internet for appropriate method, I came to know about SIFT in OpenCV. So what SIFT does is basically evaluates and computes keypoints in an image and its corresponding descriptors.The algorithm extracts this keypoints based on certain features of the image which it deems of interest. This algorithm does not take care of color tough but I don't think that matters so much because mostly all slides are black text on white background in the dataset. So to optimize use of resources I had converted each image to black and white before processing.

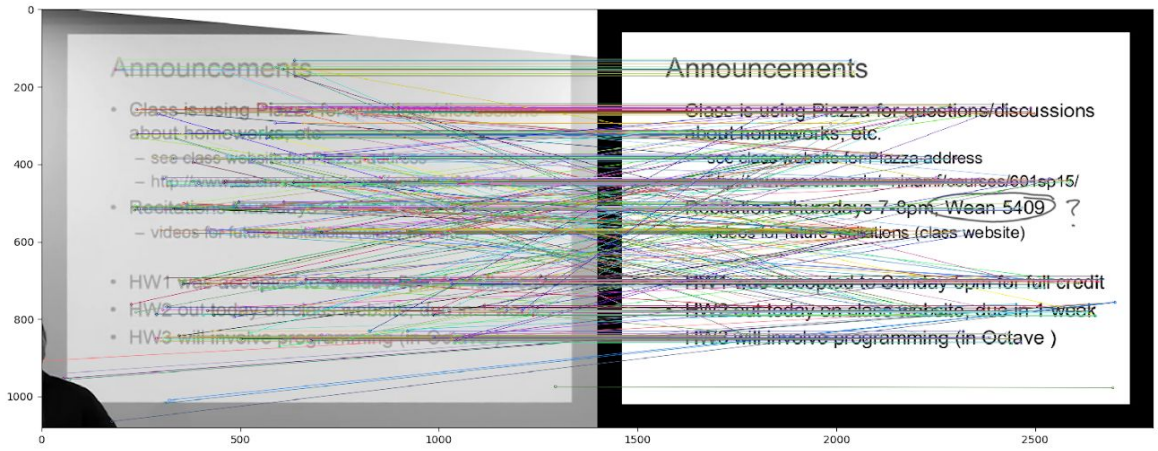
SIFT is a proprietary algorithm patented by David Lowe. Lowe's method for image feature generation transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion(source Wikipedia). So essentially this method overcomes two problems I had listed in my earlier approaches.

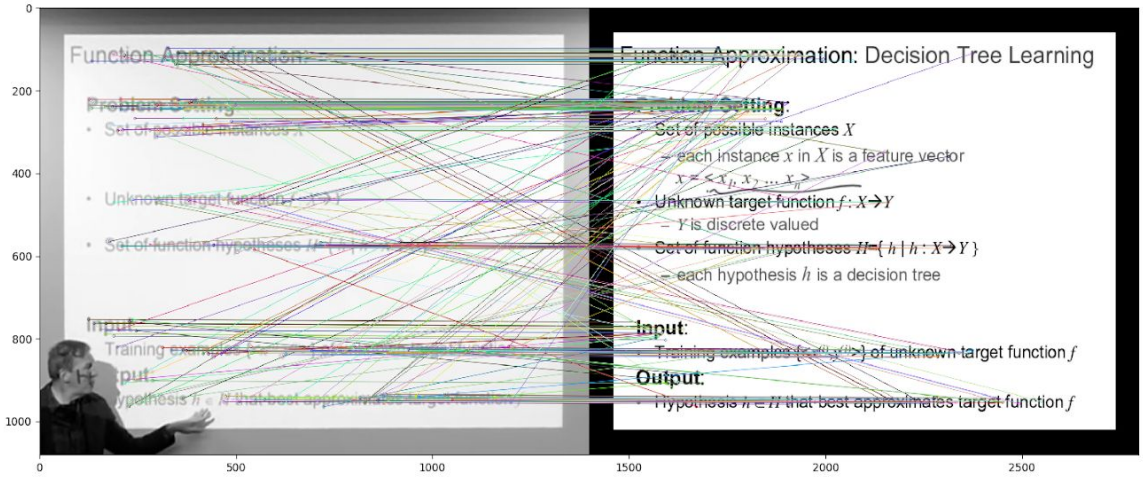
Now finding key points also helps me to reduce complexity of function as I can find key points of frame and slide only once and then while comparing there would not be need to compare at pixel level which is far greater in number than keypoints. This reusability of keypoints helps in faster implementation of algorithm, however, if number of slides is very large this would not be viable as we cannot store the slides at once.

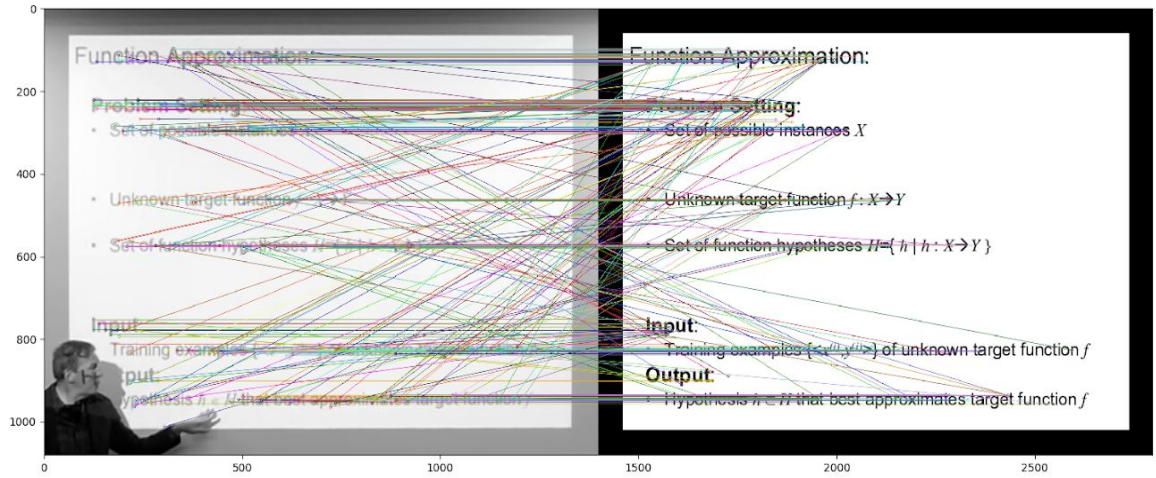
Now for comparing two key points I used simple brute force matcher of OpenCv.It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Which essentially compares every keypoint with every other. I used Knn variety of the brute force matcher, with k=2 . If k=2, it will draw two match-lines for each keypoint. So we have to pass a mask if we want to selectively draw it.

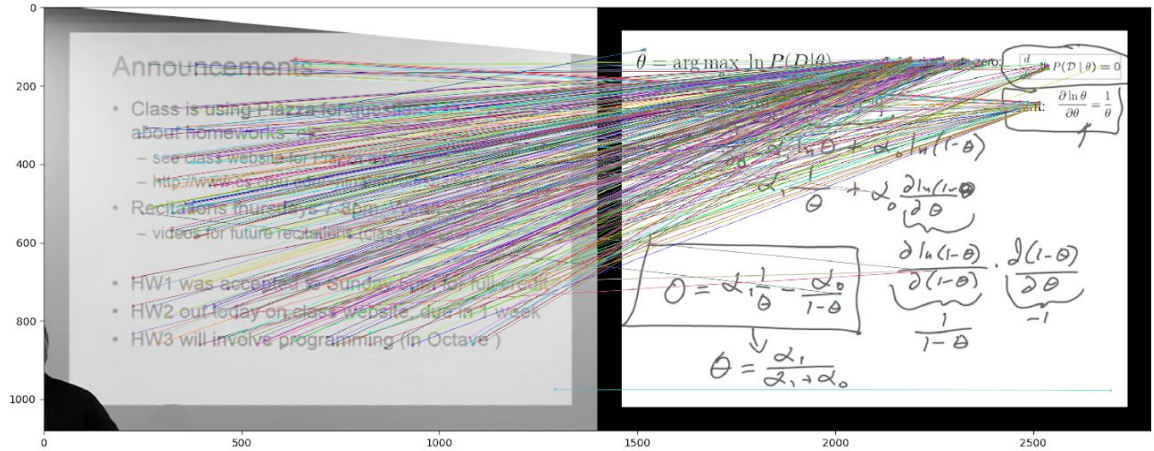
Next iterate over all such matches and accept only those matches with distance ratio is less than 0.75 which is called ratio testing also developed by H.Lowe in the same paper - recommended

value are 0.7-0.8 so I choose 0.75 as middle one. Now see below for the plots -



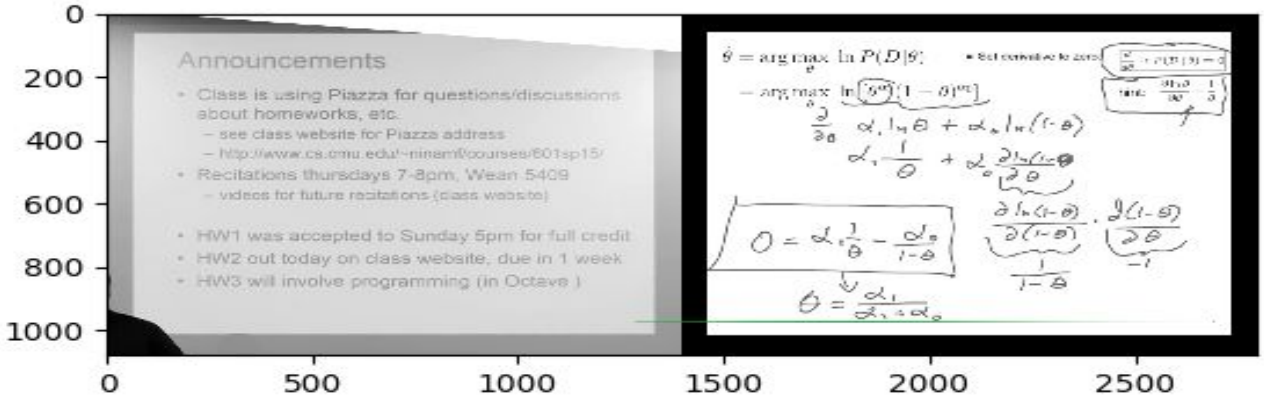


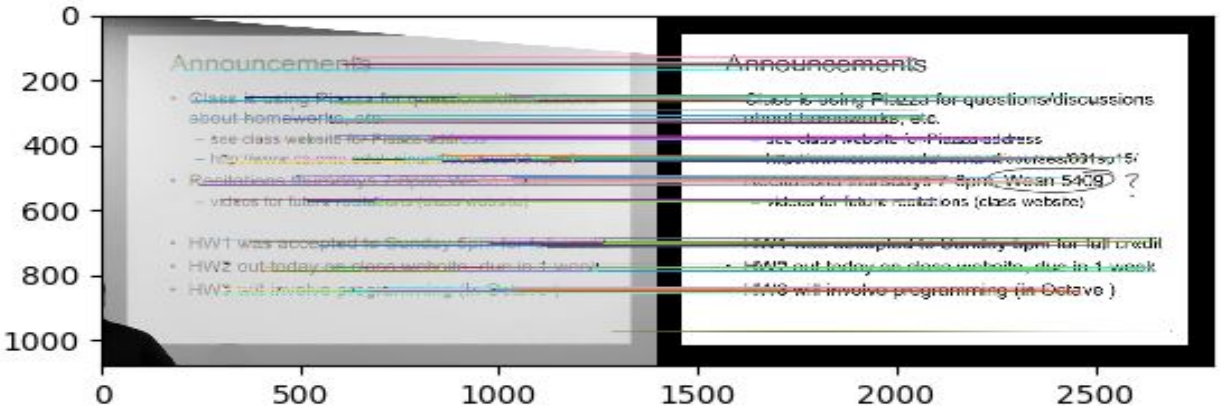

Function Approximation:
Problem Setting:
Set of possible instances X
Unknown target function $f: X \rightarrow Y$
Set of function hypotheses $H = \{ h \mid h: X \rightarrow Y \}$
Input:
Training examples $\{\langle x^{(i)}, y^{(i)} \rangle\}$ of unknown target function f
Output:
Hypothesis $h \in H$ that best approximates target function f

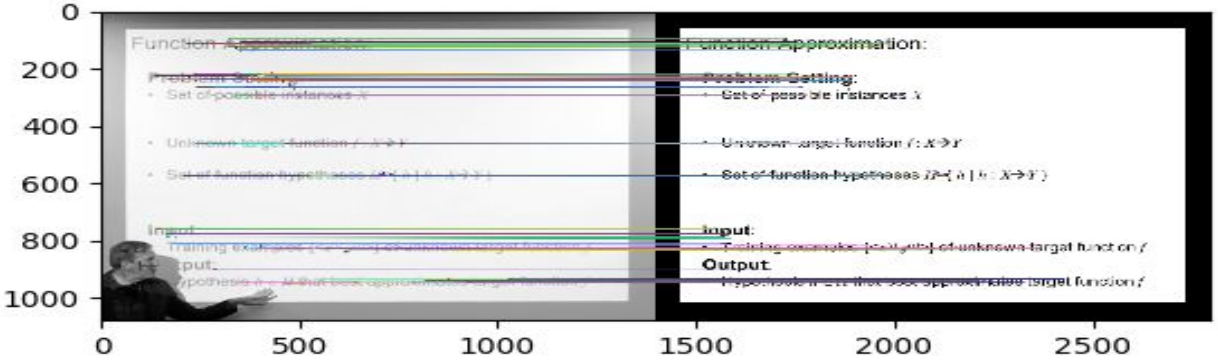

Announcements
Class is using Piazza for questions and discussion about homeworks, etc.
HW1 was accepted w Sunday 9pm for full credit
HW2 out today on class website, due in 1 week
HW3 will involve programming (in Octave)
$\theta = \arg\max_\theta \ln P(D|\theta)$
$\frac{\partial \ln \theta}{\partial \theta} = \frac{1}{\theta}$
$\theta = \frac{\alpha_1}{\alpha_1 + \alpha_0}$

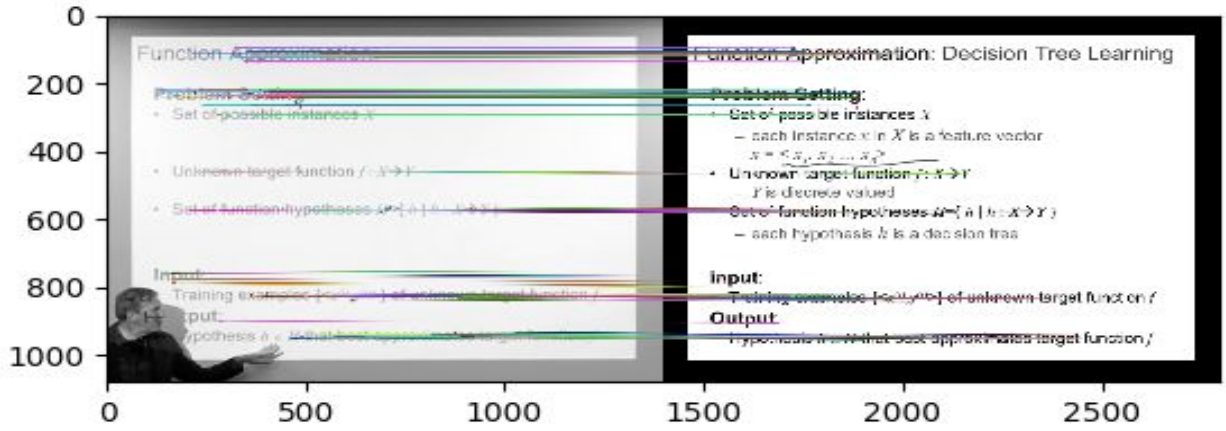
So my measure of goodness of match is based on number of these lines matching features more the number of lines better the match. This works well for most images but not for all, this gave me accuracy of about 80%. Next I deduced that since images are almost aligned the lines must be horizontal or nearly horizontal for my case all else lines are rubbish for my cause. So I find the endpoints of lines and check if they are in D-unit circle of each other if not then I throw away the line, this increased the accuracy to about 95%. Also along with horizontal I do this for vertical also and further improved the result.

This also reduces the matching problem when one slide in subset of other slide like example 2 figure.
Plots after the correction of lines -




Announcements
Announcements
Wean 5409 ?


Function Approximation:
Problem Setting:
Input:
Output:



After making this change the algorithm seems to work fine for most of the cases.

When I ran this on the dataset given I found it to be working correctly for all but 40 slides. So accuracy would be 736/776 which is pretty decent value of 94.84 % .
However, some of the cases where things didn't go well are too difficult for me as well to distinguish example slide in 8_12 and 9_8 are same to my eyes.

Team -
Prajwal Krishna - 20171086

Siddharth Jain - 20171169
Aravind Sai Gadamsetty - 20171087