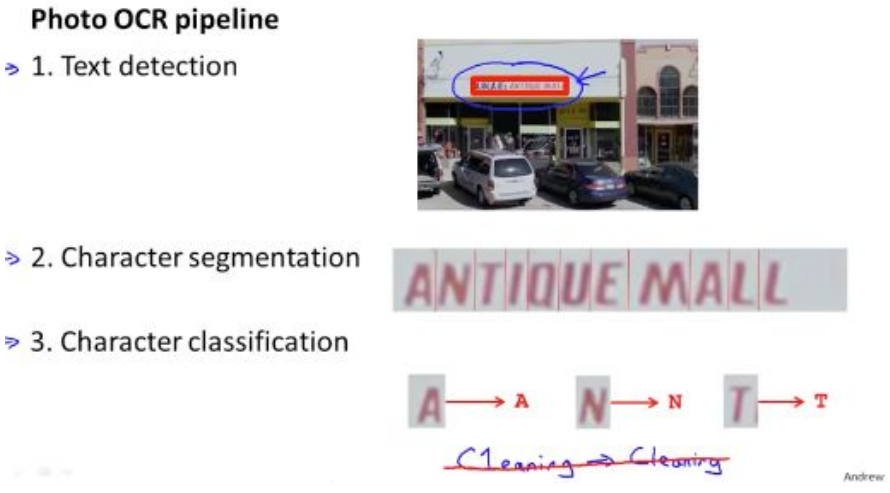
Week 11: Application Example: Photo OCR

Problem Description and Pipeline

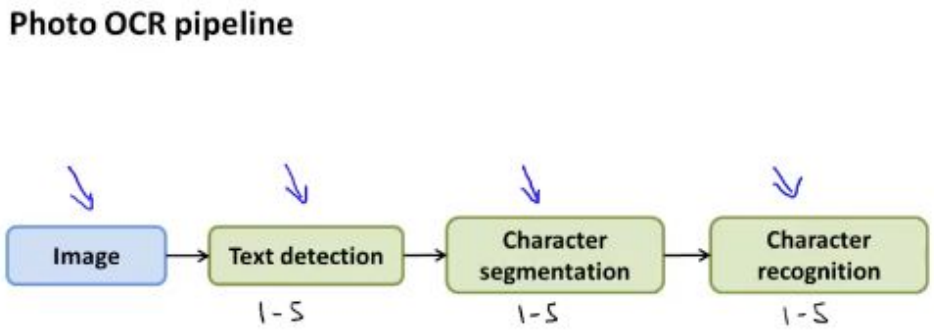
- We will talk about a machine learning application called Photo OCR (Optical Character Recognition). Thanks to cell phone cameras we have tons of digital pictures and Photo OCR focuses on how to get computers to read the text from those pictures. This has many applications such as searching through a photo collection. Photo OCR enables this by first recognizing the text in the pictures and then reading it.

- To perform Photo OCR we follow these steps:
  - Text Detection: detect where is text in the picture
  - Character Segmentation: Separate the characters in the text
  - Character Classification: We run a classifier to recognize each character



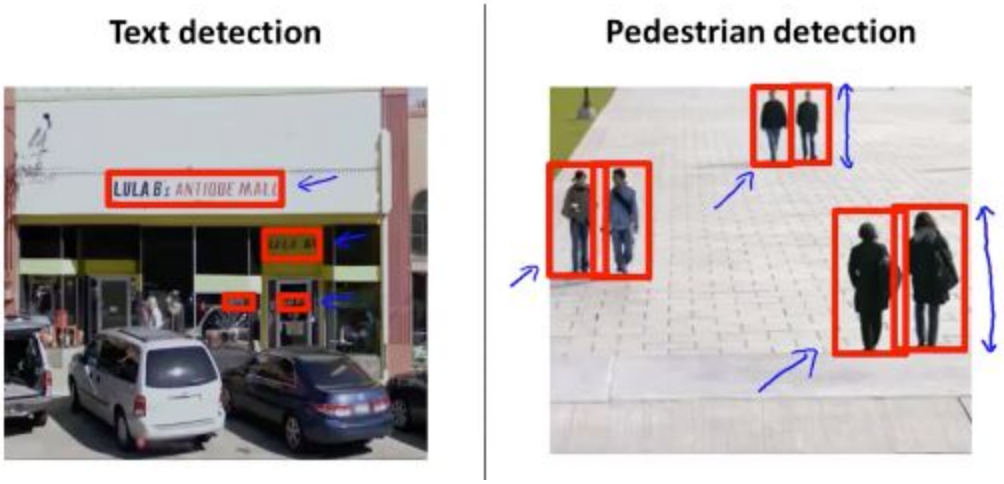
Finally, some algorithms will also do spelling corrections but we are going to ignore that for this example.

- That process is called a machine learning pipeline and, as shown below, it provides a natural way to divide the work among an engineering team. Machine Learning pipelines are very common and can be done by one engineer or a group of engineers. The Photo OCR example below is just one example of a machine learning pipeline.



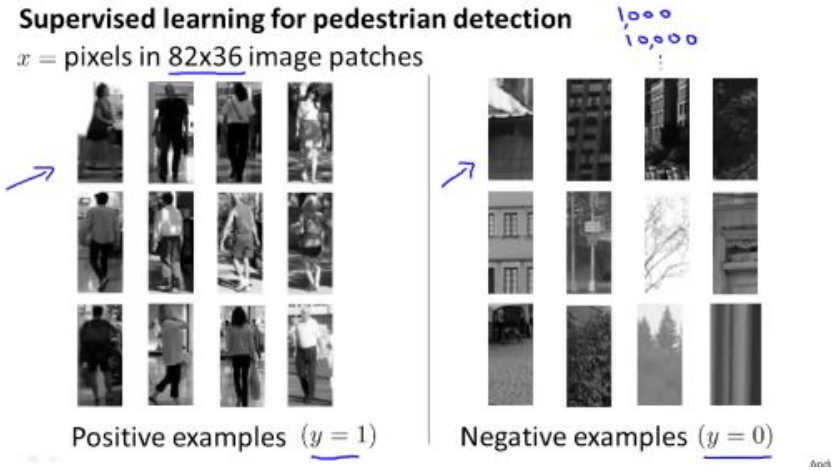
Sliding Windows

- Now we are going to look a little bit more in depth at how the components of the machine learning pipeline work. In this instance we are going to learn about a classifier called Sliding Windows. In order to check text detection first we are going to look at pedestrian detection. Pedestrian detection is a bit easier than text detection because it uses the same aspect ratio whereas text detection uses different aspect ratios. Aspect ratio is the height and the width of the rectangles below. For pedestrians we can use a fixed aspect ratio but we have to use different ones for text detection.



We are then going to see how pedestrian detection works and apply its ideas to text detection.

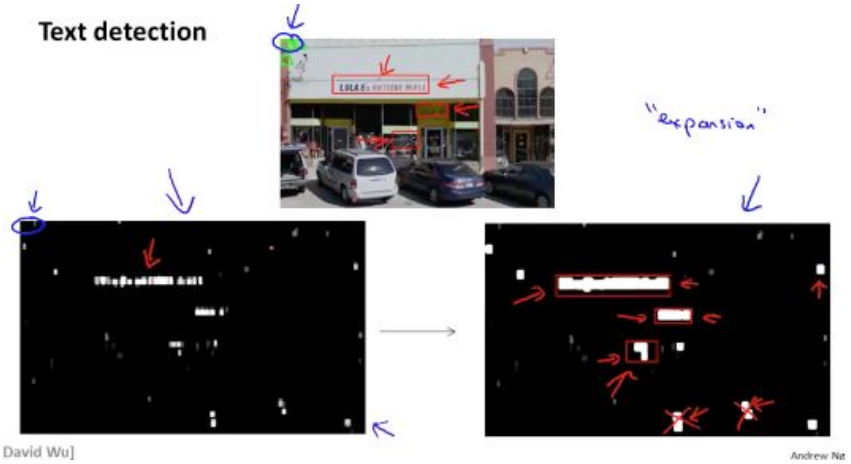
- We can apply supervised learning for pedestrian detection by defining x= pixels in 82x36 image patches. We have a set of positive examples (y=1) and a set of negative examples and we train our supervised learning algorithm (could be a neural network, could be something else) to classify the images presented as pedestrians or not.



- What we do next is take our image and 'loop' through it by using a rectangle to go through it as shown below. We start from the top left corner and work our way through the image. Every 'rectangle' is fed to our classifier which then decides if we have a pedestrian in the image. We can start with relatively small rectangle sizes but we can then increase them to cover more ground. However, when they are increased we default them to our fixed standard ratio before feeding it to the classifier. The process of taking the rectangle around the image is called stepping or striding. You can start with moving one pixel at a time but step sizes of 4 to 8 pixels are more common since they allow us to 'step' through the image faster.

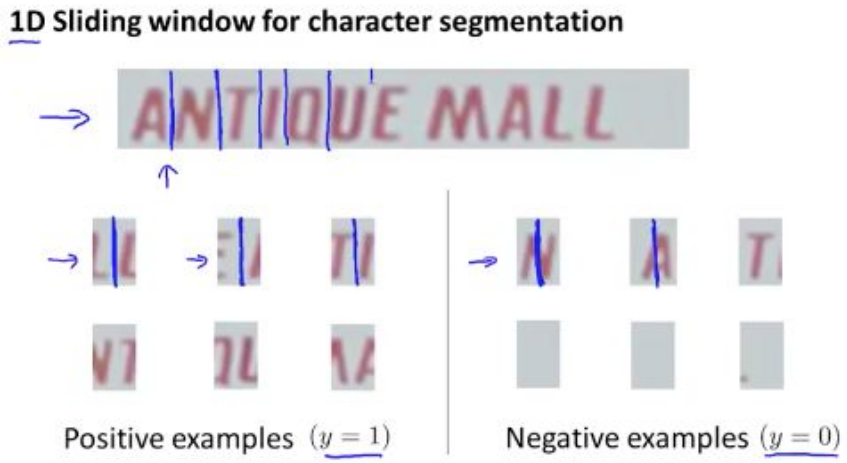
- Now we can come back to text detection and apply the ideas we learn from text detection. We can train our text detection classifier with a labeled training set with

positives (patches of images where there is text) and negative examples (patches of images where there isn't any text). Now that we have a trained classifier we can feed it a new image. Using our previous image below we run sliding windows on it and end up with the image on the left where the white/gray sections of the image are where the classifier detected text. The whiter the image, the higher the probability there is text there.



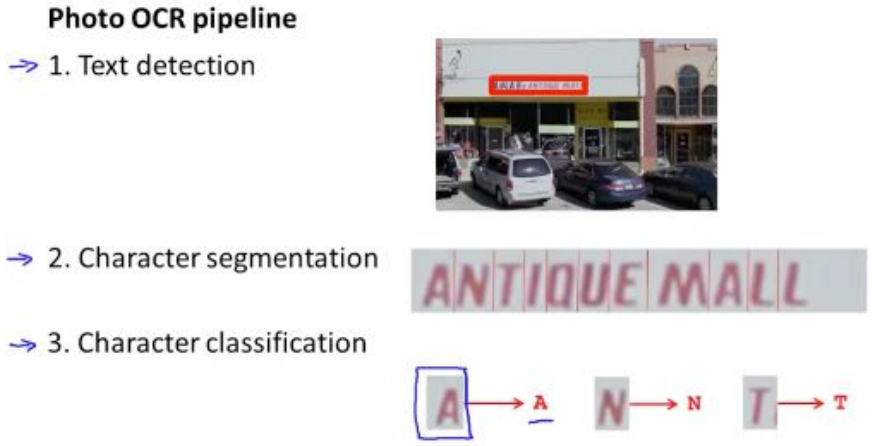
We then take the classifier output and run it through an expansion operator which will draw rectangles around the areas where we think there is text. It will take those regions and expand them. Mathematically speaking, what it does is looks at a pixel and if it is 5, 10 pixels from a white pixel it will also color it white. We end up with the image on the right.

- Next step is to use another supervised learning algorithm, train it with positive and negative examples of character segmentation and then feed it our images. What we do is do a 1D sliding window where we go through the image, feed each window image to the classifier and the classifier will hopefully tell us to add a line separating each character as shown below.



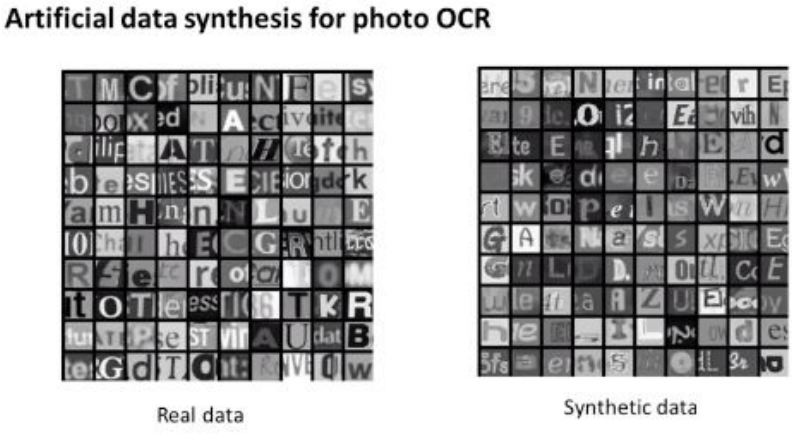
- Summarizing, we first detect the text parts of an image, then run a character segmentation step and finally feed our 'characters' to a character classification

algorithm (yet another supervised learning algorithm) which then recognizes the individual characters (we went over this step in previous weeks of this class)

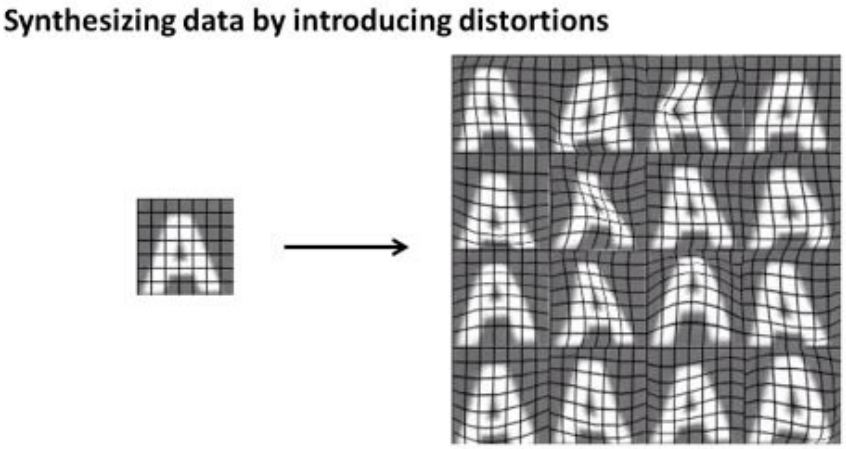


Getting lots of data and artificial data

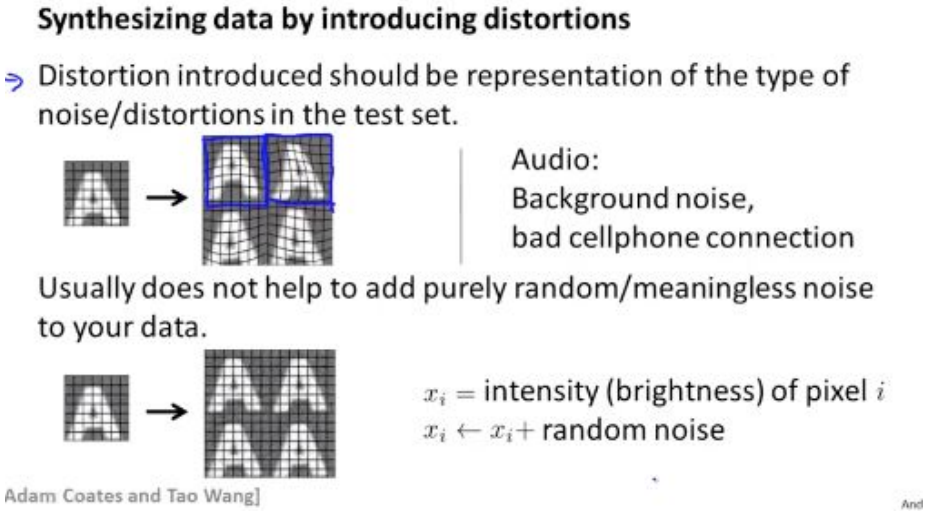
- One of the most reliable ways to get a high performance training set is to take a low bias algorithm and train it with a massive data set. Turns out we have something called artificial data synthesis and it can be a solution to get a massive data set. There are 2 ways to go about this:
    - Create new data from scratch or
    - Amplify an already existing labeled training set
- Let's use the character recognition example to illustrate the 2 methods. Let's start with how we create new data from scratch. The data on the left side is real data. How do we create the data on the right? We use the different fonts available (you can download a library of fonts from the internet) and then stick a character/letter on a random background. We repeat this process for many characters and then perhaps we add a little blurring and/or distortion and we end up with a synthetic dataset like the one on the right.



- The other method is to take an existing example and creating different variations of it as shown below with the character 'A'. We can also apply the same concept to say speech recognition where we can have an audio file and create additional examples by adding relevant distortions ('bad' cell phone connection, machinery noise, crowd noise, etc.)



- The distortions added have to be representative/relevant to the test set. It does not help if we add purely random/meaningless noise to the dataset.



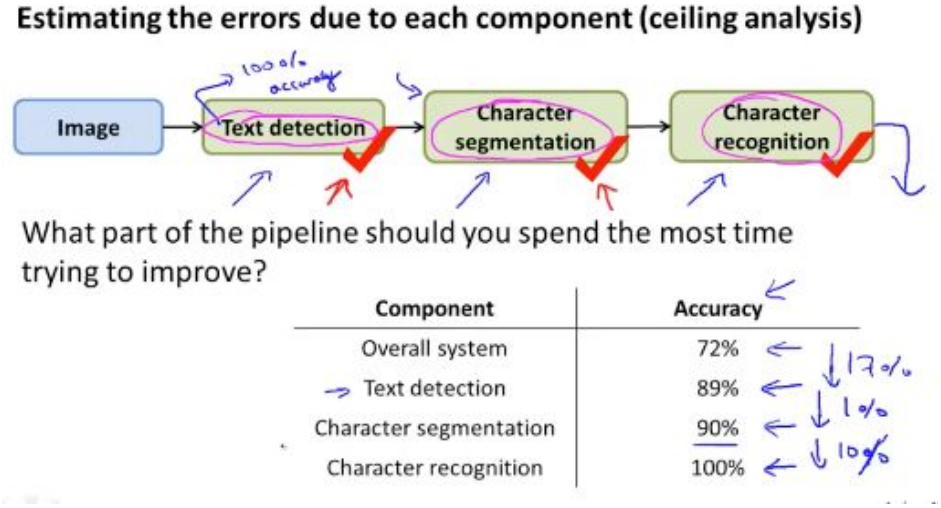
- Before going about creating more data we need to:
  - Make sure we have a low bias classifier
  - Ask ourselves how long would it take to 10x as much data as we have today? Frequently the answer is not long. I can get more data by:
    - Artificial data synthesis (which we examined above)
    - Collect/label it ourselves
    - Crowd-source the work (through say, Amazon Mechanical Turk)

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect/label it yourself
   - "Crowd source" (E.g. Amazon Mechanical Turk)

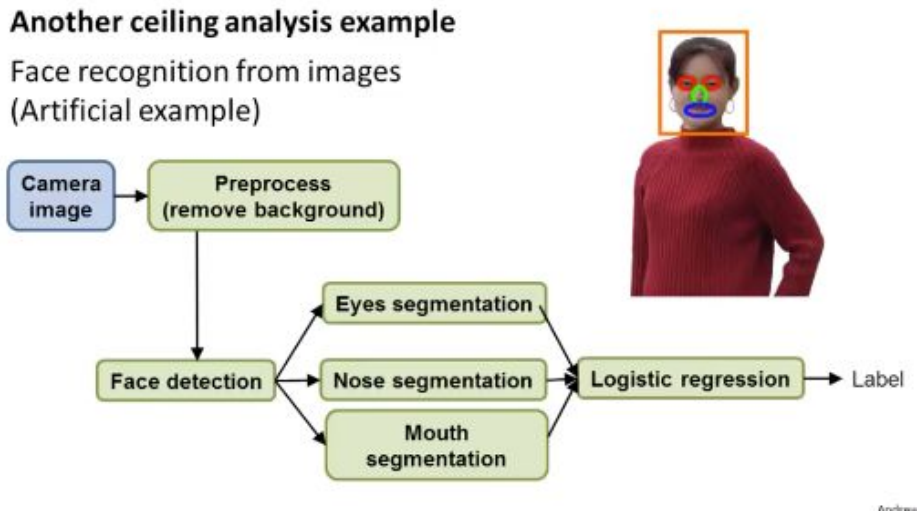
Ceiling Analysis: What part of the pipeline to work next

- Ceiling Analysis will give us signals/guide on us on what parts of the data pipeline should we work next. Let's illustrate it with a couple of examples: Let's continue with our character recognition pipeline we had before. Assume we have a system with some overall accuracy, say 72%. Ceiling Analysis means we go component by component and create one 'perfect' component at a time and see what is the increase in accuracy. For example, we can start with text detection and instead of training our algorithm we are just going to manually tell it where the text is. We then repeat the process for each component. Following our example below we see that a 'perfect' text detection module increased our accuracy 17% (seems like a good place to spend time on). We only got a 1% accuracy increase when we had a 'perfect' character segmentation module so it does not seem like the best use of our time. Finally, a 'perfect' character recognition module increased accuracy by 10% so somewhere in the middle.

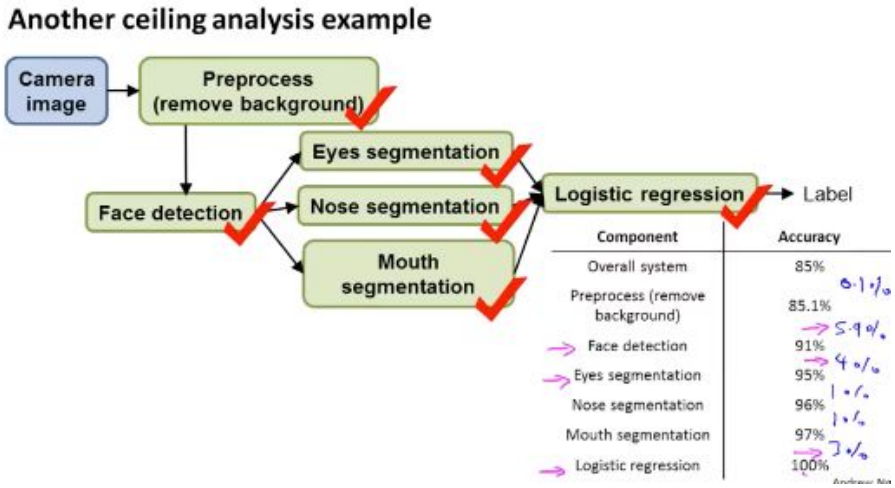


- Let's use another example: face recognition from images. In this case we have the pipeline shown below where we first get an image, remove the background and then we

go about detecting the head by detecting the eyes, the nose and the mouth. All these features are fed to a logistic regression classifier which gives us a label for the image.

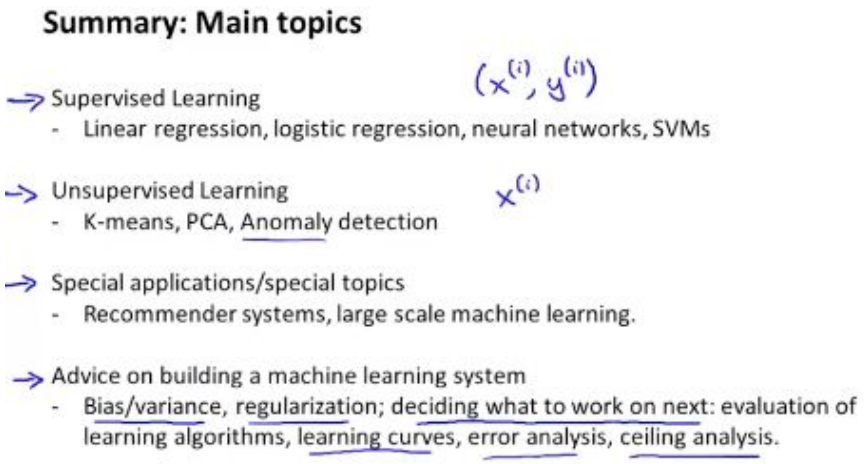


- We then do the same ceiling analysis on each component and end up with the following accuracy improvements on each component. Based on the data below, it seems worth our time to try to improve face detection (5.9% gain), eye segmentation (4% gain) and logistic regression (3%).



Course Summary: Main Topics

- The main topics of the class are shown below



- Finally, Professor Ng thanks us for spending the time to go through this class. Great class, learned a lot, it was well worth the time spent on it and I would do it again in a heartbeat. Thanks Professor Ng!!!