Machine Learning
Week 11
Application Example: Photo OCR

This week will walk through a complex end to end application of machine learning, a photo OCR application. Identifying and recognizing objects, words, and digits in an image is a challenging task. This week will discuss how a pipeline can be built to tackle this problem and how to analyze and improve the performance of such a system.

Photo OCR

Problem Description and Pipeline

There are three reasons we are going to work through a Photo OCR Application:

1. To see an example of how a complex machine learning system can be put together.
2. To talk about the concept of a machine learning pipeline and deciding what to do next.
3. The Photo OCR problem has some interesting ideas to tackle computer vision problems and artificial data synthesis.

Photo OCR stands for Photo Optical Character Recognition

Given a picture, detect where there is text in the image, then look at the text regions and read the text.

The Photo OCR problem

In order to perform Photo OCR:

Photo OCR pipeline

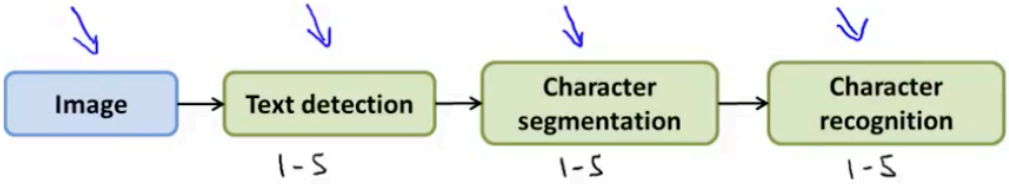
→ 1. Text detection

→ 2. Character segmentation

→ 3. Character classification

A system like this is called a machine learning pipeline.

Photo OCR pipeline

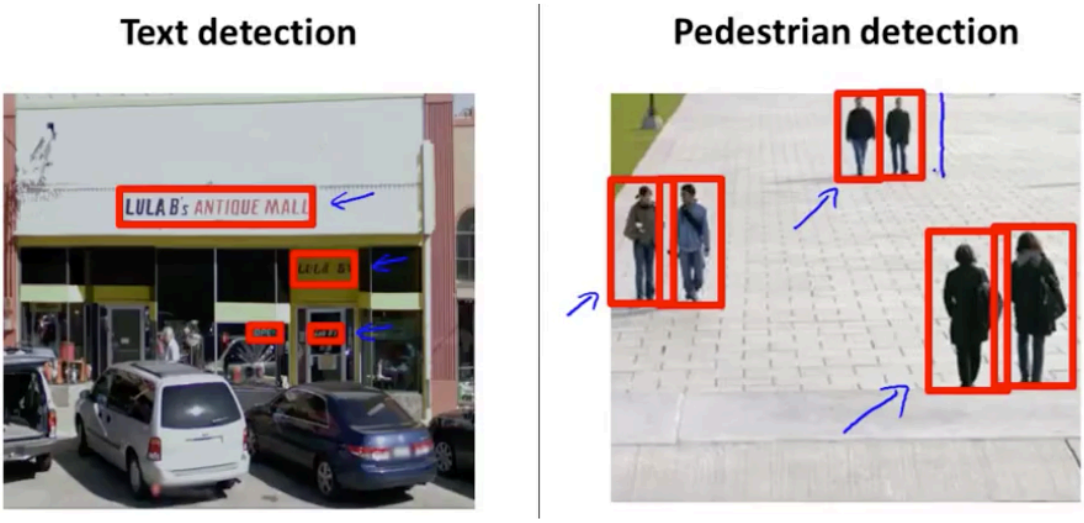


These modules could then be given to separate teams or individuals to divide up the workload.

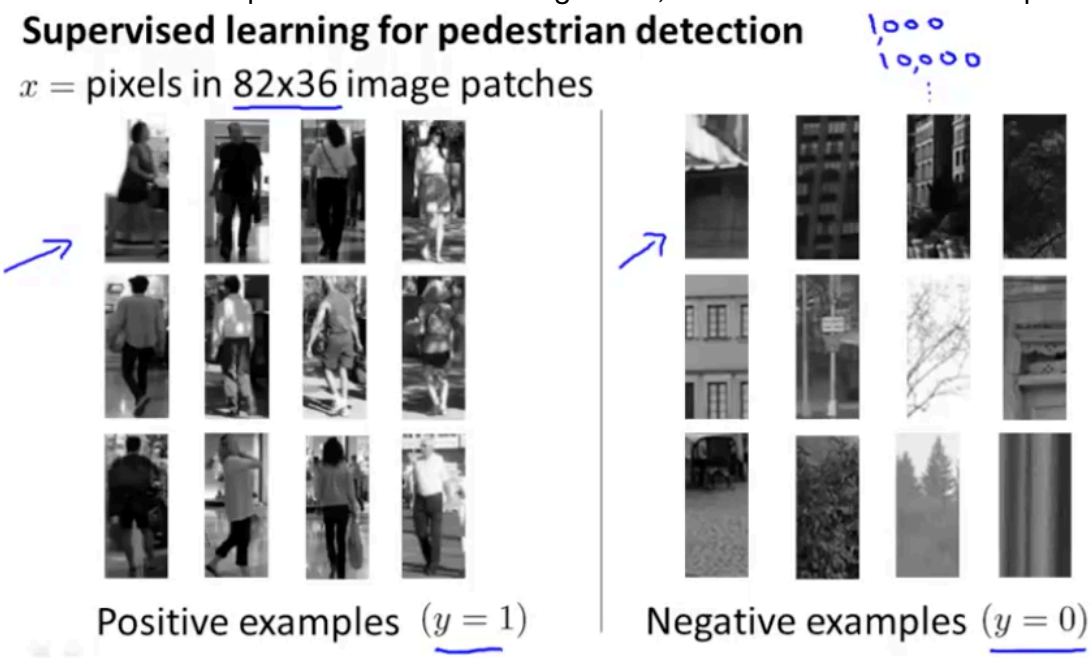
Sliding Windows

This video will look into the individual components of the pipeline.

The first stage of the Photo OCR pipeline was text detection. Object detection is an unusually problem because you need to output many different aspect ratios. As a simpler example lets first look at pedestrian detection where we can use fixed aspect ratios.



In order to build a pedestrian detection algorithm, we can standardize the aspect ratio.



After the algorithm is trained, we can try to use it on our test data. This could work as follows:

We would scan through the image with the green sliding window (we are shifting the window by a 'stride' parameter). Using this process we run each patch through the classifier.

We also start to use large image patches. (Which is then later resized to 82x36 that we trained our algorithm with)

Returning to the text detection example. How do we find the text regions in the image?

We can start with a training set of positive and negative examples. After the algorithm has been trained, we can run a similar sliding to perform detection:

Whiter shades - higher probability of text present, darker - lower probability.

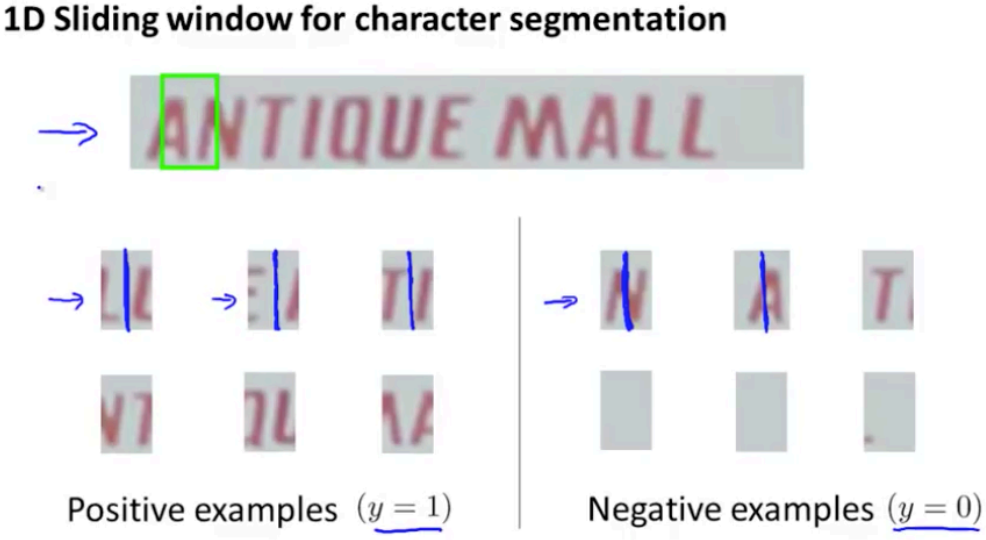
We then apply an expansion operator to expand the areas.

We can now look at the white areas and draw bounding boxes around them.

As a last step we can ignore the vertical aspect ratios and only look for horizontal rectangles.

This completes the first segment of the pipeline.

For the next segment, we need to split the words into characters.



For the last step, we perform character classification using multi-class classification or a neural network.

Summary:

Photo OCR pipeline

1. Text detection

2. Character segmentation

3. Character classification

Getting lots of Data: Artificial Data Synthesis

One of the most reliable ways to get a high performance machine learning system is to take a low bias learning algorithm and train it on a massive training set.

But where do we get so much training data from?

In machine learning, there is an idea called artificial data synthesis. This doesn't apply to every single problem, and to apply it to a specific algorithm takes a lot of thought and innovation.

If this can be applied to your algorithm, it can be an easy way to get a huge training set to give to your machine learning algorithm.

The idea of Artificial Data Synthesis comprises of two parts:
1. Creating new data from scratch.
2. Taking a small labelled training set and amplifying it into a larger training set.

Lets use the idea of OCR again.

We have our real data set, but we also have access to a vast font library. We can take a random letter from a random font library, and paste it onto some random background.

Artificial data synthesis for photo OCR

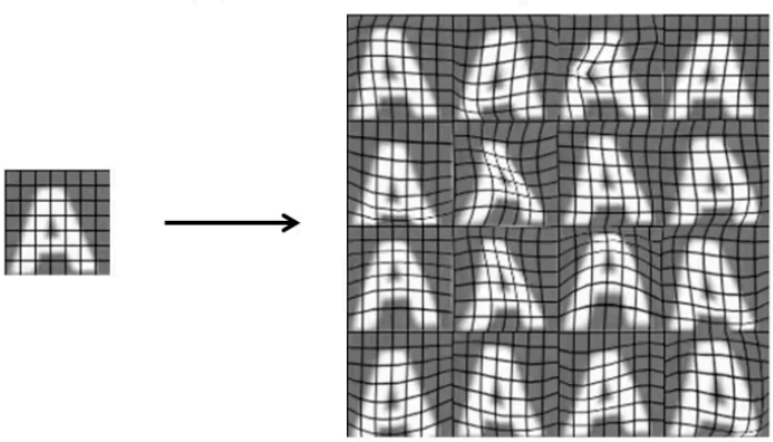
Real data

Synthetic data

We could then apply some blurring, small rotations etc. But we now have some synthetic training set that looks pretty realistic.

Another way to add more synthetic data is to take an image from our real data and then introduce distortions or rotations to turn it into new examples.

Synthesizing data by introducing distortions



Another example of synthesizing data by introducing distortion:

Synthesizing data by introducing distortions: Speech recognition

Original audio:

Audio on bad cellphone connection

Noisy background: Crowd

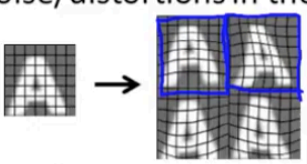
Noisy background: Machinery

We take the original audio and overlay it onto a noisy background of some sort or add distortion and were able to multiple the data set.

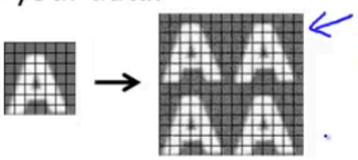
Note that when adding in these distortions, to images or audio, they new instances you are creating should be reflective of the test set / real applications.

Synthesizing data by introducing distortions

- Distortion introduced should be representation of the type of noise/distortions in the test set.



- Audio: Background noise, bad cellphone connection
- Usually does not help to add purely random/meaningless noise to your data.



- x_i = intensity (brightness) of pixel i
- $x_i \leftarrow x_i +$ random noise

The process of artificial data synthesis is really an art in itself.

Some other notes to consider when attempting to get more data:

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 on next

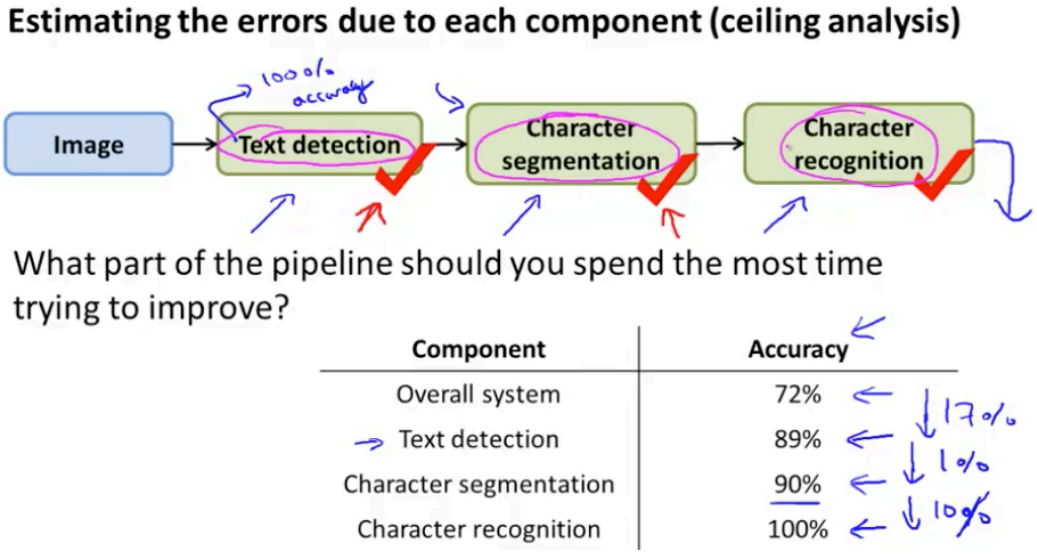
When developing a machine learning system, one of the most valuable resources is your time, and as such you need to careful pick what to work on.

The same applies for a team of engineers, their time allocations are the most valuable resource.

You want to avoid wasting time on a component that won't make a significant difference to the performance of your system.

In this video we will discuss ceiling analysis to guide which parts of the pipeline would be the best use of your time to work on.

Continuing the Photo OCR Example:



We give the text detection module with all of the data labels to provide 100% text section accuracy to the rest of the system and see what the change in the accuracy is of the overall system.

We then do the same for the character segmentation section and see the change in accuracy of the overall system.

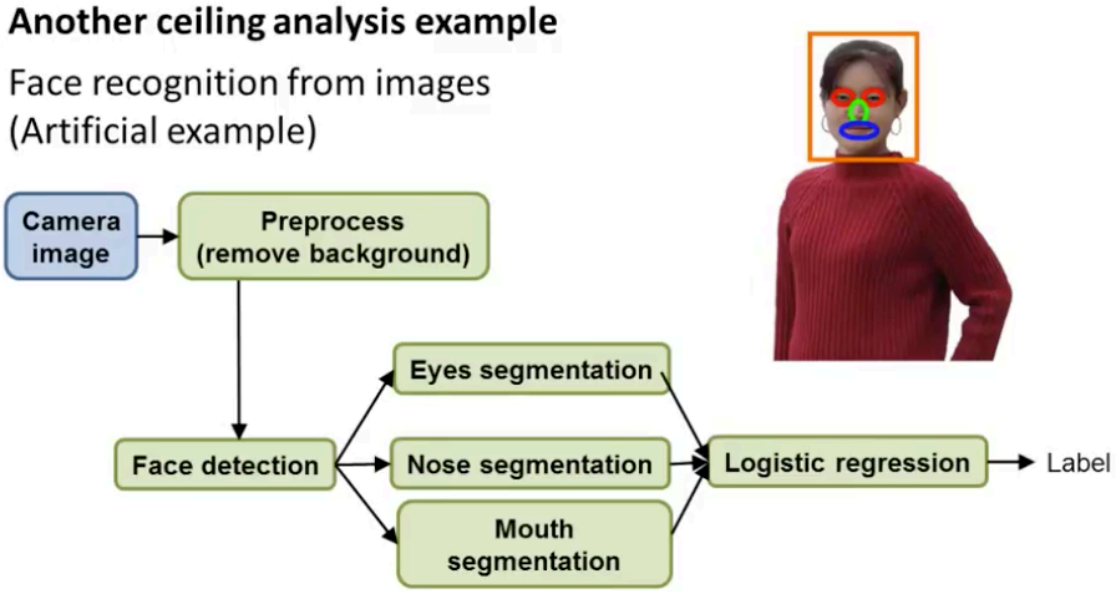
Finally we test the last module and see that the overall system accuracy is now 100% as each of the submodules is outputting at 100% accuracy.

We can then look at where the significant increases in performance came from.

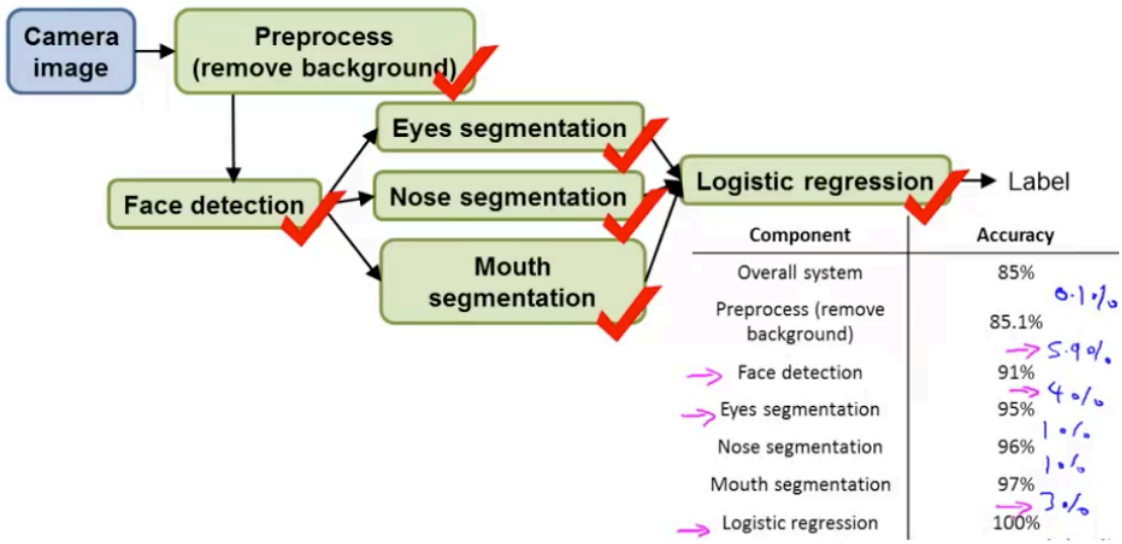
In this example 'text detection' and 'character recognition' seems like the module we should focus on improving to increase system performance while attempting to improve 'character segmentation' isn't going to significantly improve the performance of the system.

This process of measuring the change in overall performance / accuracy of the system as the performance of the submodules change to then determine the upside of improving an individual submodule is called Ceiling Analysis.

Looking at this same idea with a different example:



As we perform Ceiling Analysis:



In this example it appears that face detection, eyes segmentation, and logistic regression are the most worthwhile to work on.

Cautionary tale: Don't put in tons of work trying to improve a module without doing this type of analysis!!! Do not trust your own gut feeling.

Do the analysis.
Do the analysis.
Do the analysis.