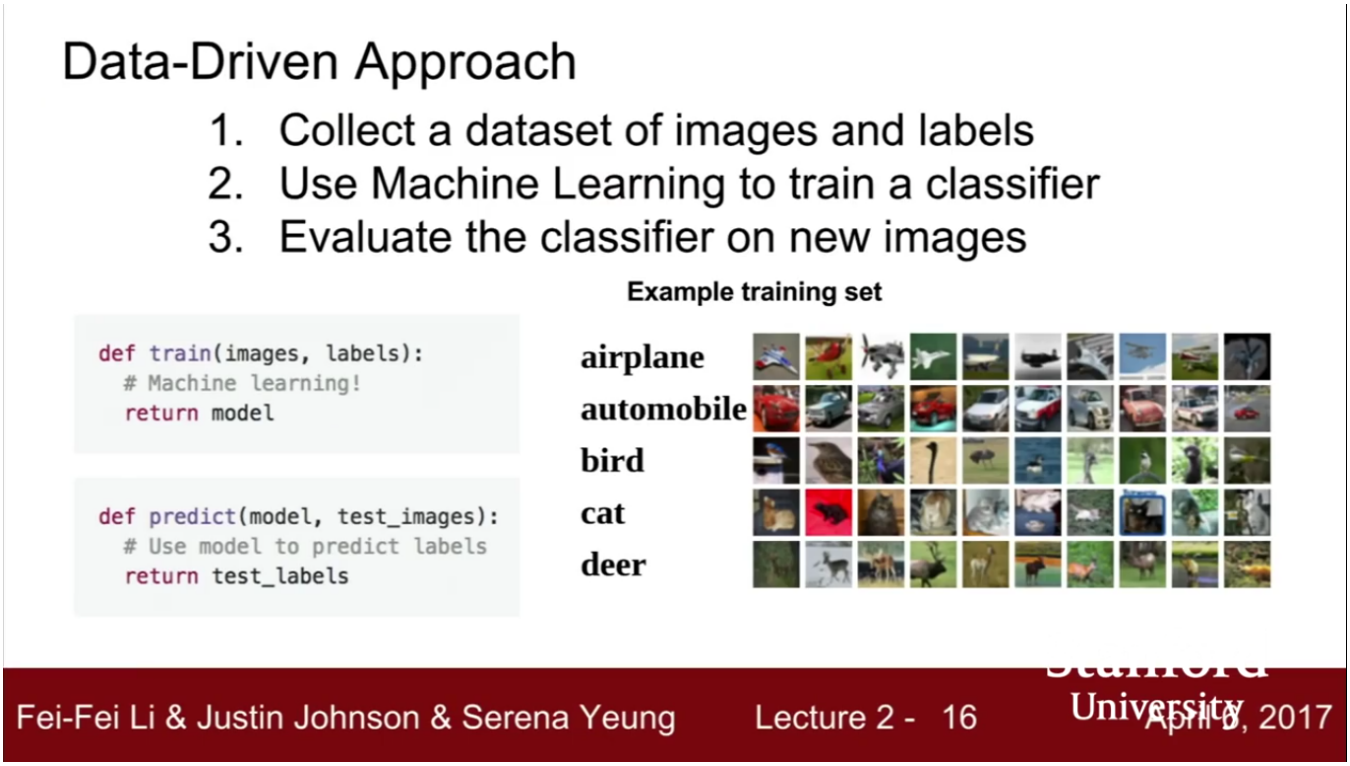
lec2

Lecture 2 notes on 02/11/2019 18:54

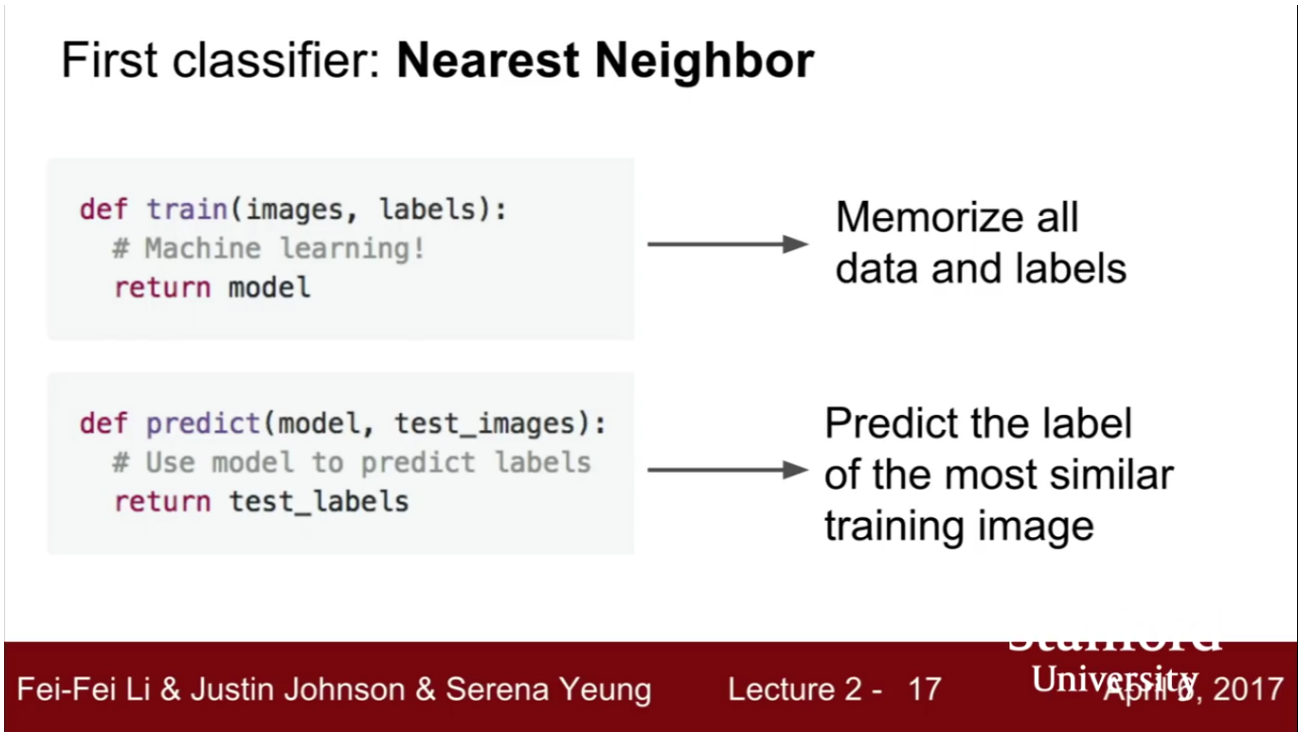
Last Updated: 26/12/2019 22:52

- Image Classification: A core task in CV
- An image is just a big grid of numbers between [0-255], and a computer only sees that matrix and it has to infer the content from it.
- **Semantic Gap** : The semantic idea of an object (cat in a image) and that of the pixel values that the computer sees
- Challenges:
    - Viewpoint variation: All pixels change when the camera moves!
    - Illumination: Different lighting conditions for the same kind of image content/subject.
    - Deformation: Different, varied poses, and various other transforms.
    - Occlusion: Only part of the object may be visible.
    - Background Clutter: Forground object could actually look quite similar in apperance to the background.
    - Intraclass variation: Different visual appearances for the same object. Eg. Different ages, shape, breed of cats
- There is no obvious way to hard-code the algorithm for recognizing a cat, or other classes unlike in other problems like sorting, encription etc. where we can write the steps to execute.
- But there have been attempts to write an algorithm using rules.
    - Image --> Find Edges --> Find Corners --> Infer the content based on relations of these elements of edges and corners.
        - Disadvantage: **Not generalizable**: The implementation is not flexible, and it is hardcoded for a single class.
        - Not scalable. Since we need to write different approaches for each class.
        - Solution: **Data-Driven Approach**

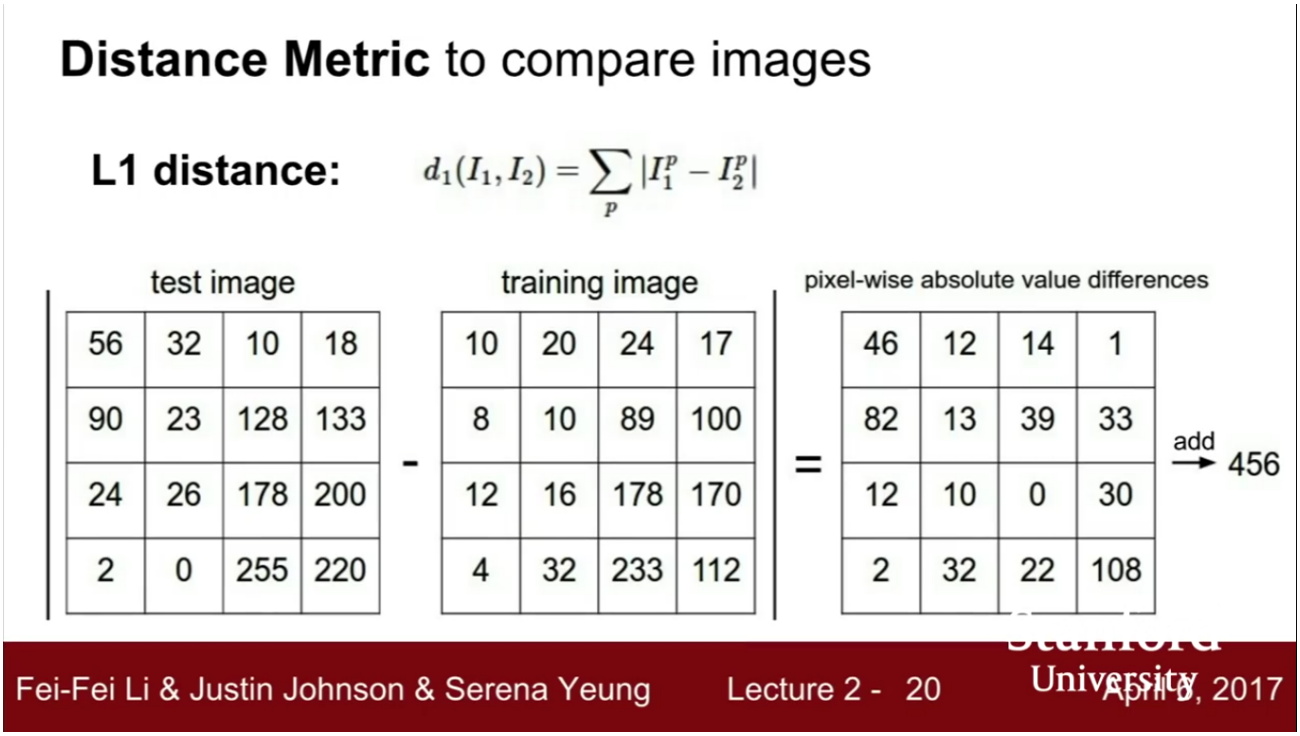
- **Data-Driven Approach**



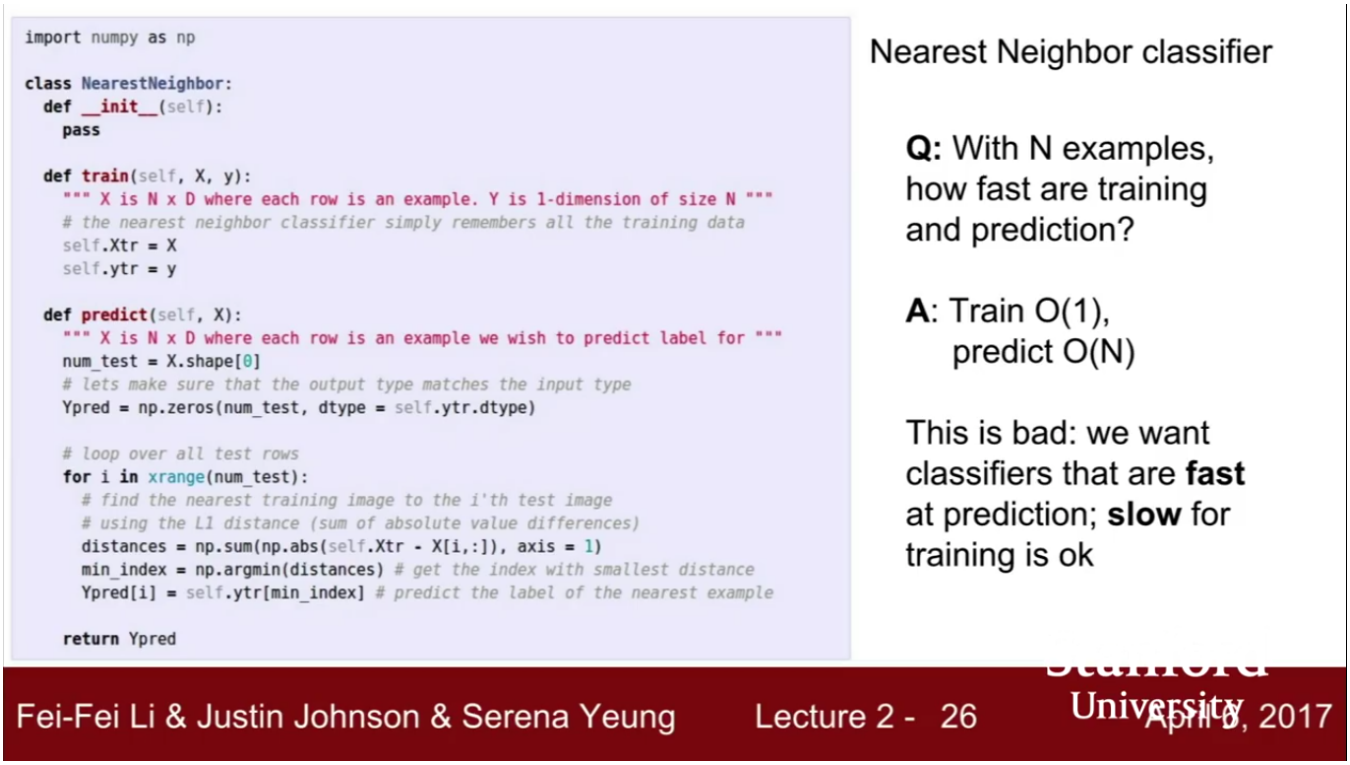
- This Data-Driven approach is not restricted to deeplearning, **we are just utilizing the association of labels to the data.**
- Simplest approach for data-driven:
  - Nearest Neighbour



- To compare 2 images we need a metric

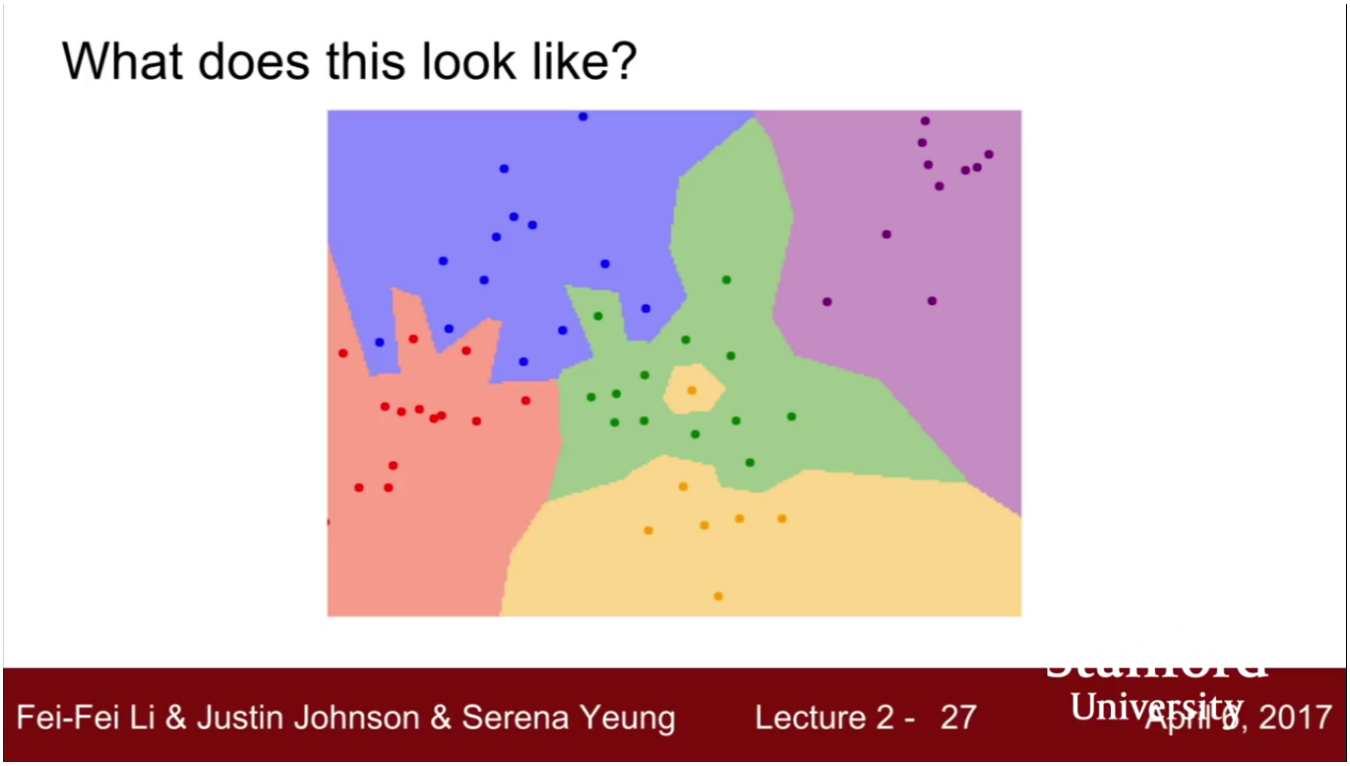


- Algorithm for Nearest Neighbour



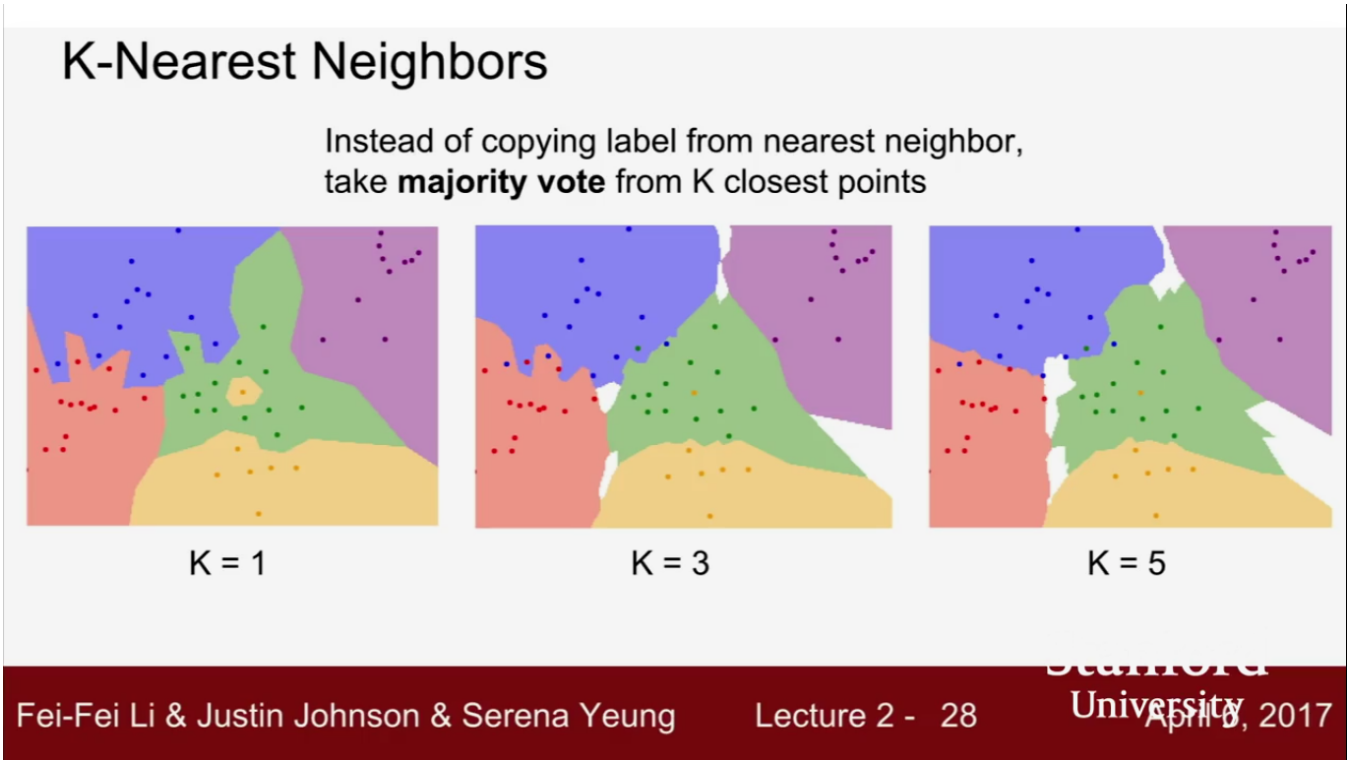
- Disadvantage of the above approach is that, it takes a long time to predict which is not useful in real-life scenarios. We would expect to train a model for long time but need a low prediction time.

- The decision boundary would look something like this:

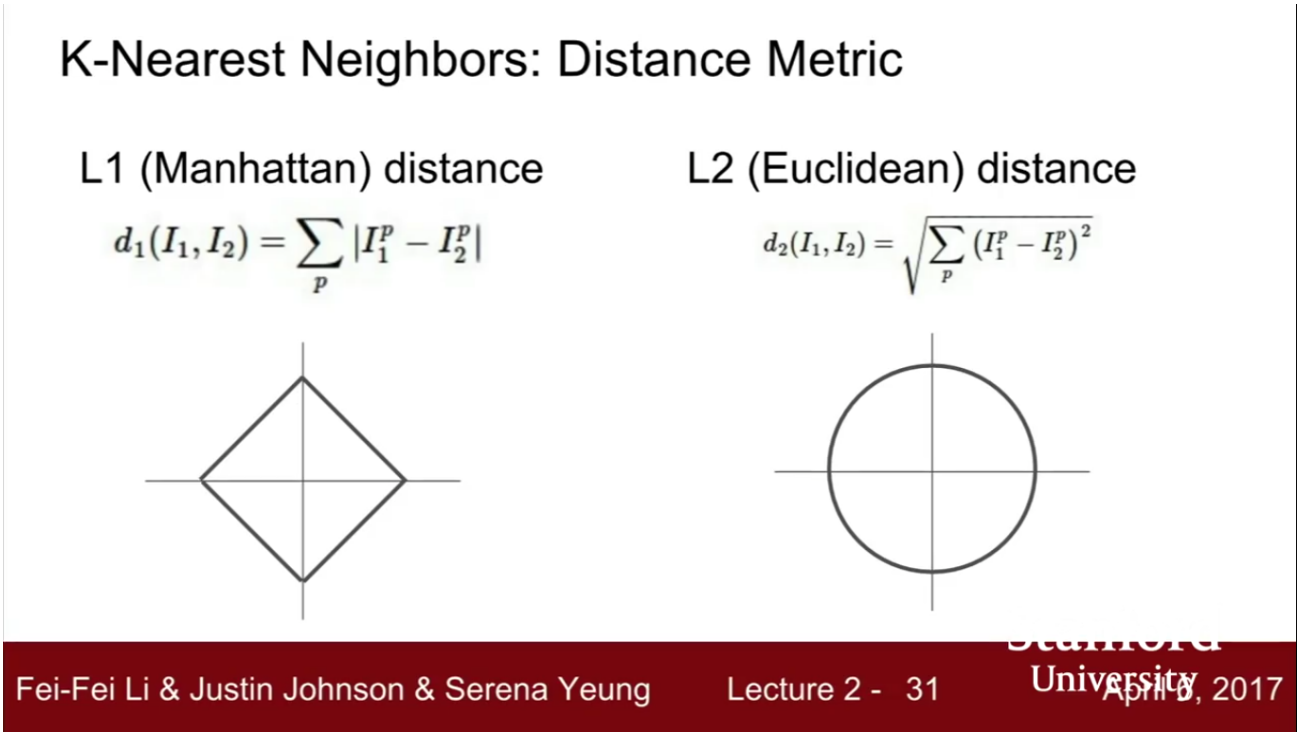


The color of a region represents the class label and the points represent the examples belonging/classified to that class.

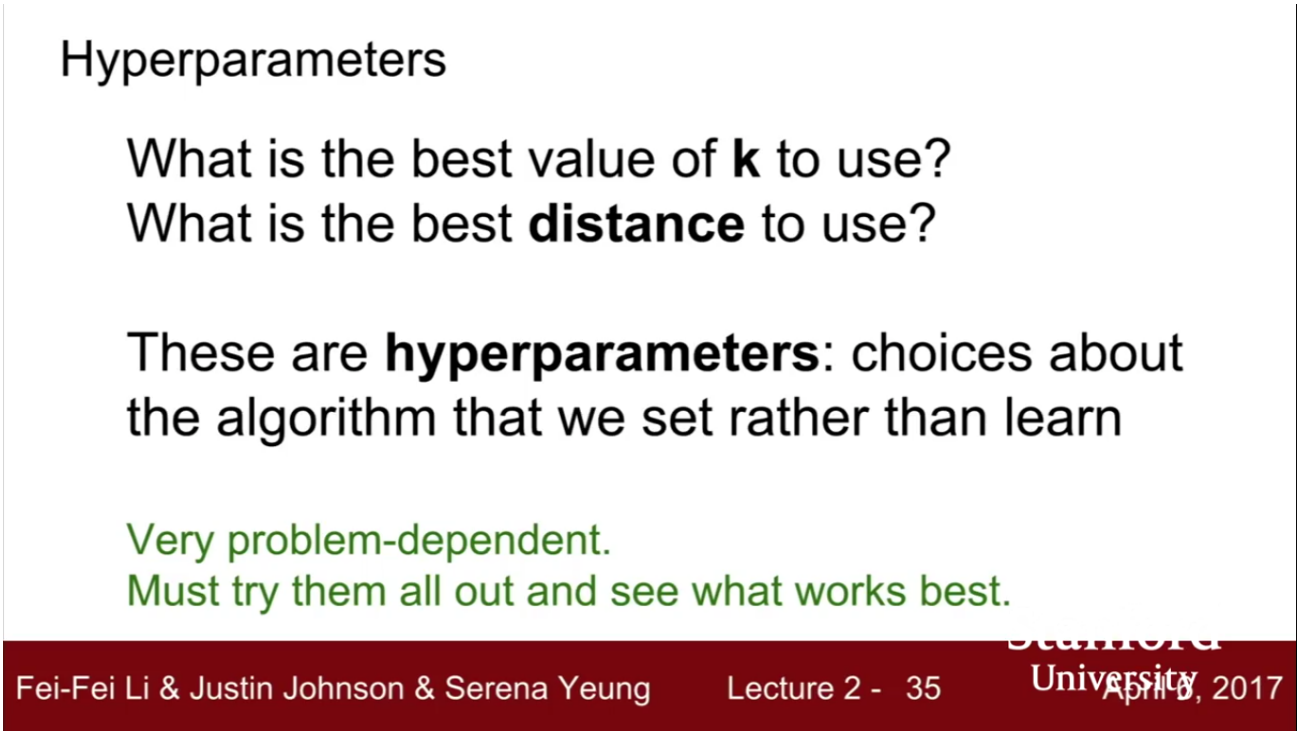
- Disadvantage:
  - Wrong decision boundary in some cases since the decision boundary is calculated based on how near the examples are.
- Solution:
  - Use K-Nearest Neighbour
- K-Nearest Neighbour



- creates smoother boundary regions
- The white region shows where there is no majority. For this case, we can perform a random choice of neighbour.
- Distance Metrics:



- **If the individual entries of your vector has signifance based on their position in the vector, then L1 is the wiser choice compared to L2, because L2 distance is invariant to position of entries in a vector**



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

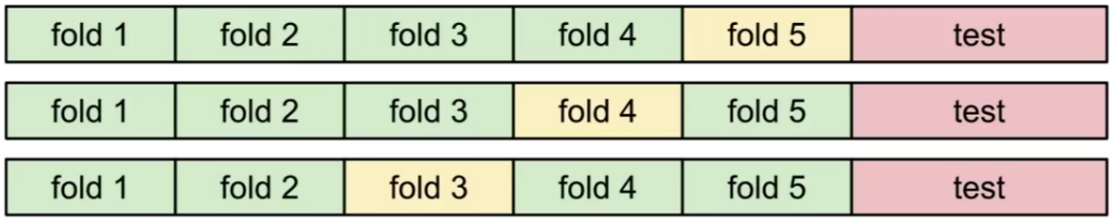
BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

Setting Hyperparameters

Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning

k-Nearest Neighbor on images **never used.**

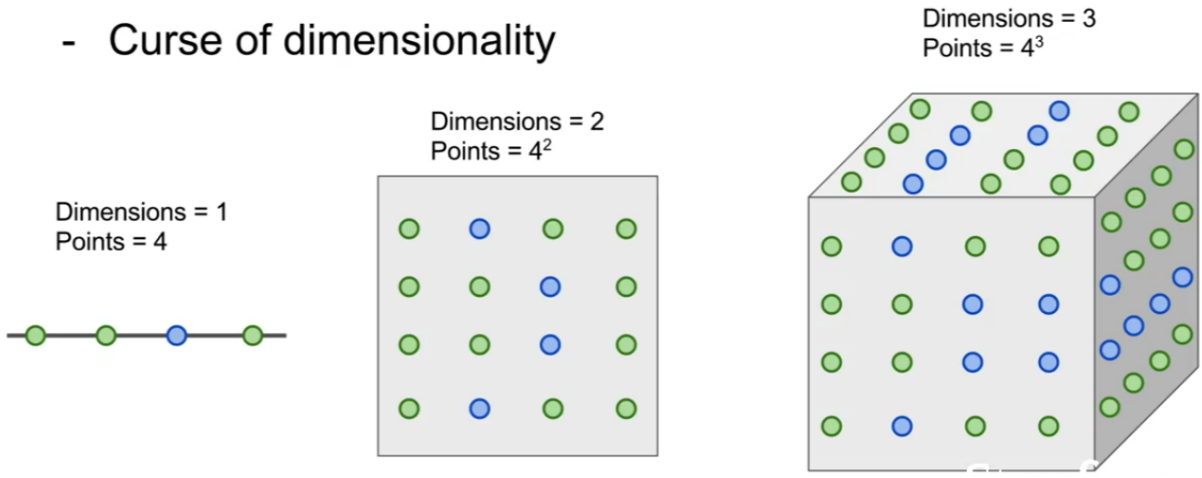
- Very slow at test time
- Distance metrics on pixels are not informative

Original image is CC0 public domain

(all 3 images have same L2 distance to the one on the left)

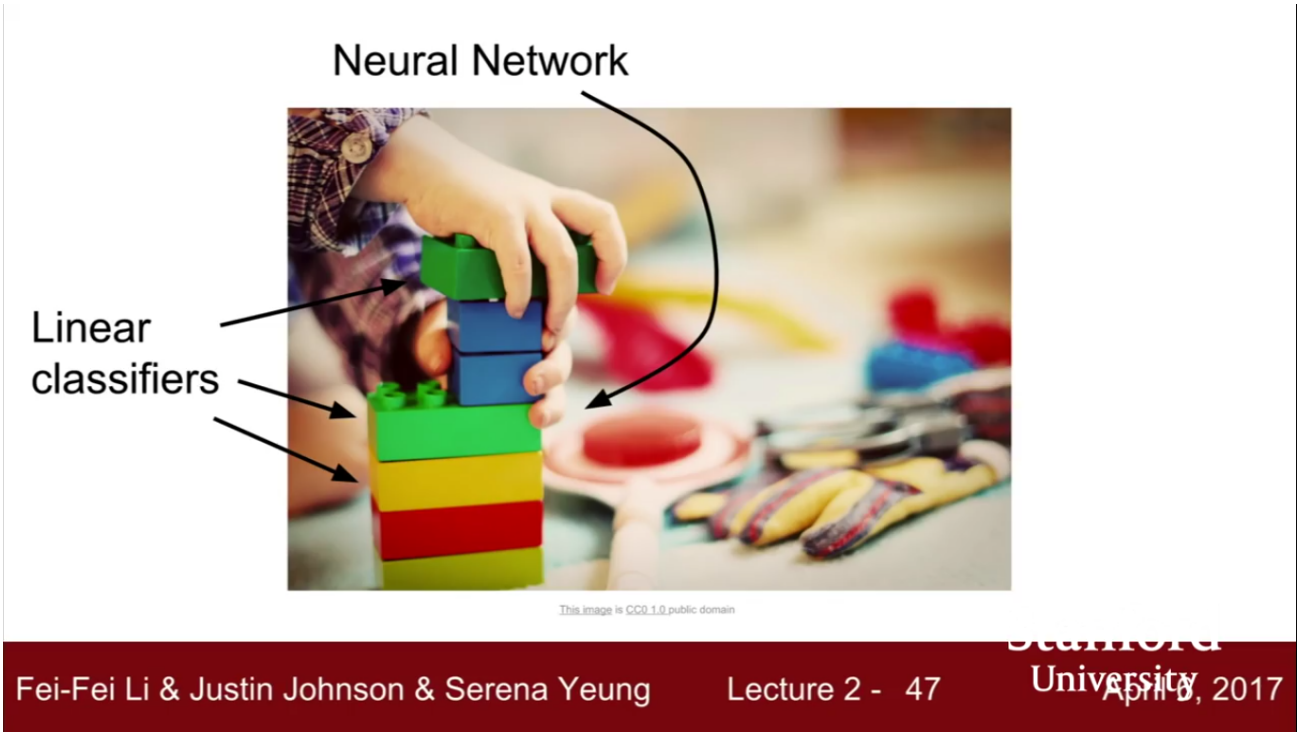
k-Nearest Neighbor on images **never used.**

- Curse of dimensionality

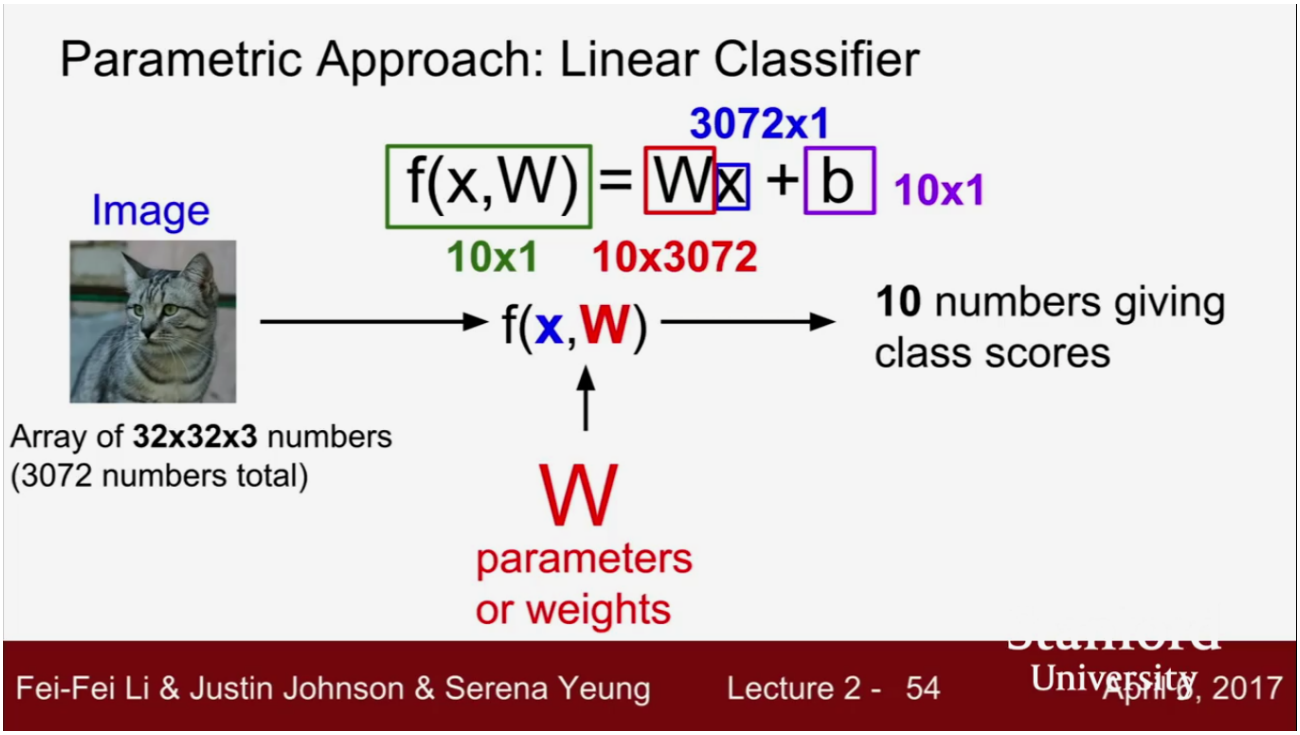


- Linear Classifiers

- One notion of Linear Classifiers in NN is:

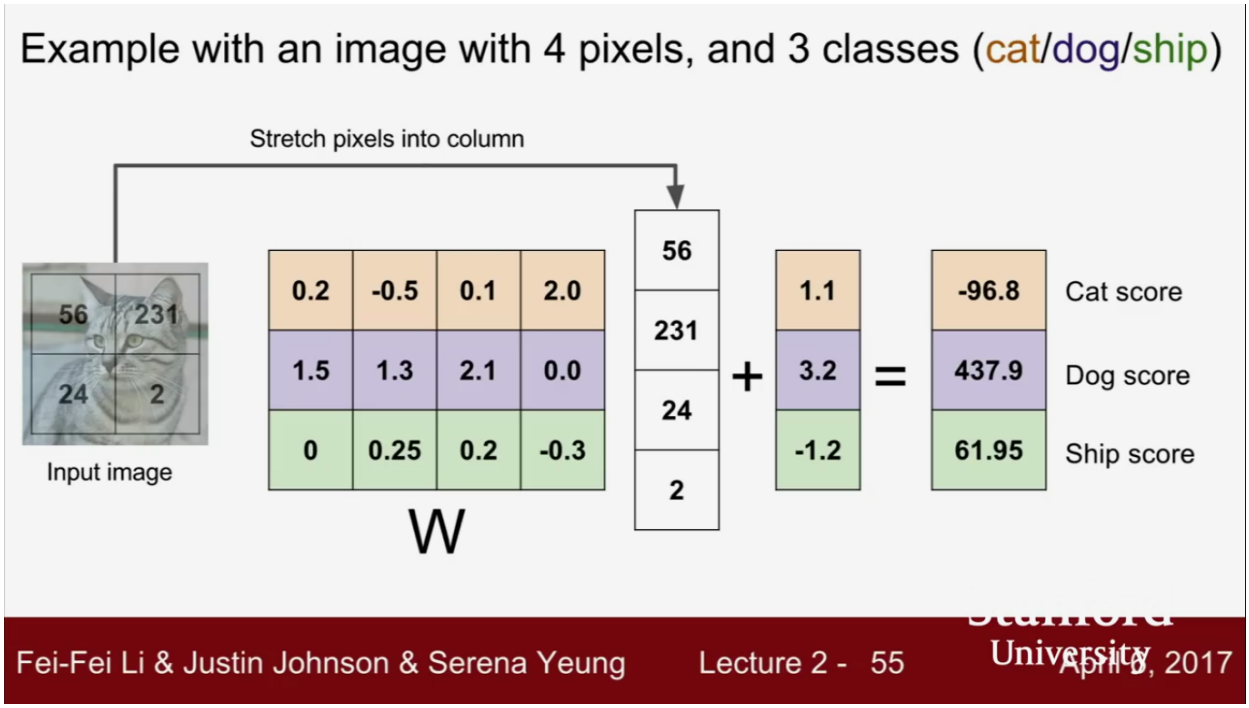


- These deep NN are like LEGO Blocks, we can use many such modules to solve the problem. Eg. Image Captioning: Uses CNN block to understand the image, and RNN to generate the caption.
- **Parametric Approach:**

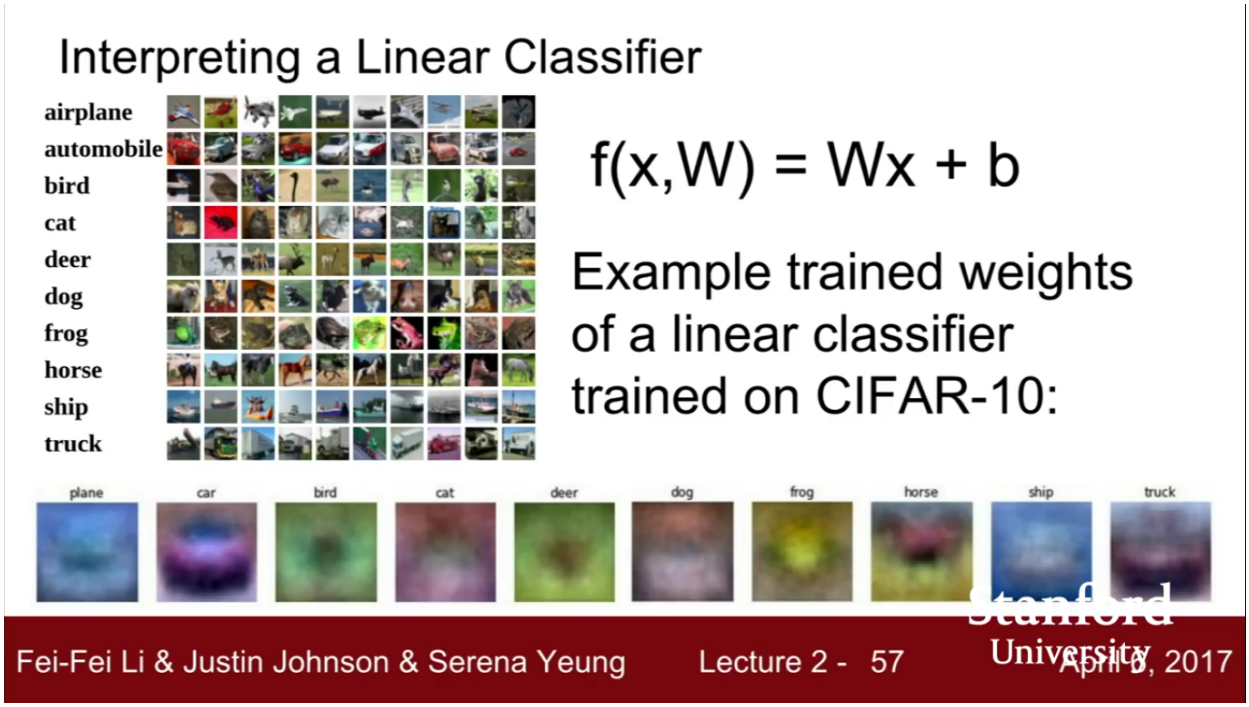


  - The crux is to define the function f which better defines our training data.
  - **In Linear Classifier, the function f = Wx, i.e. it is just a multiplication operation of parameters with the input data.**
  - **The bias term does not interact with the training data.**
  - ***The bias term acts as some sort of data independant preferences for some classes over another. So if our dataset(Cats vs. Dogs) has more cats than dogs then the bias***

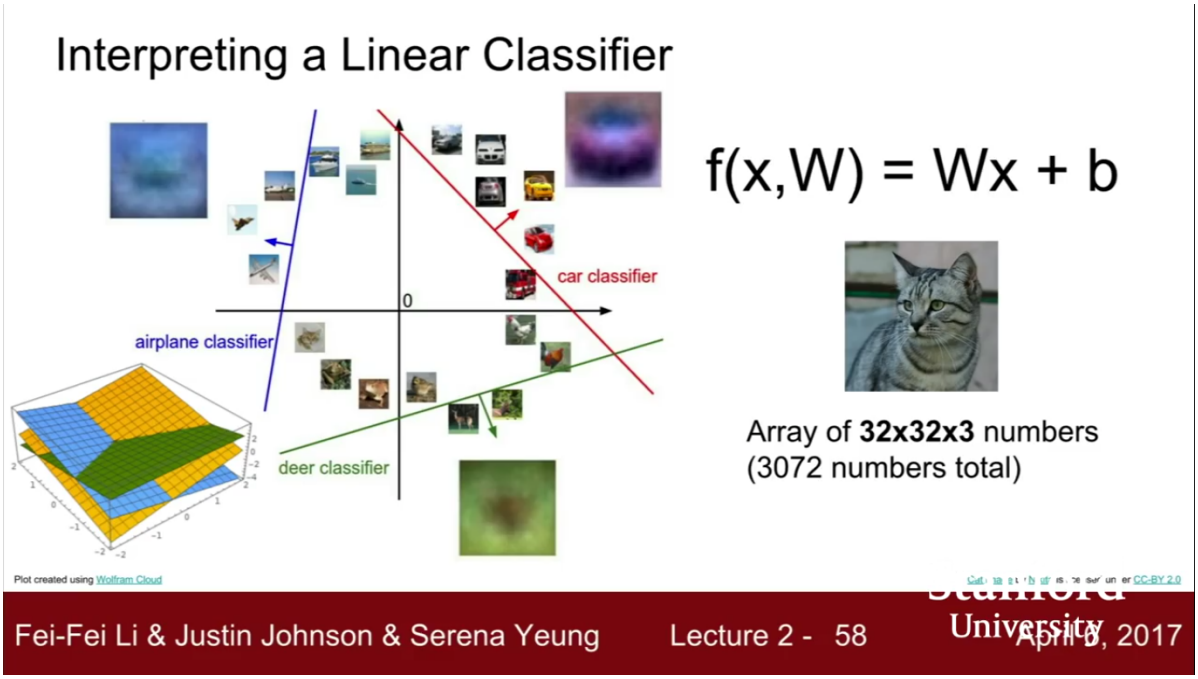
term corresponding to the cat will be relatively higher than for the dog



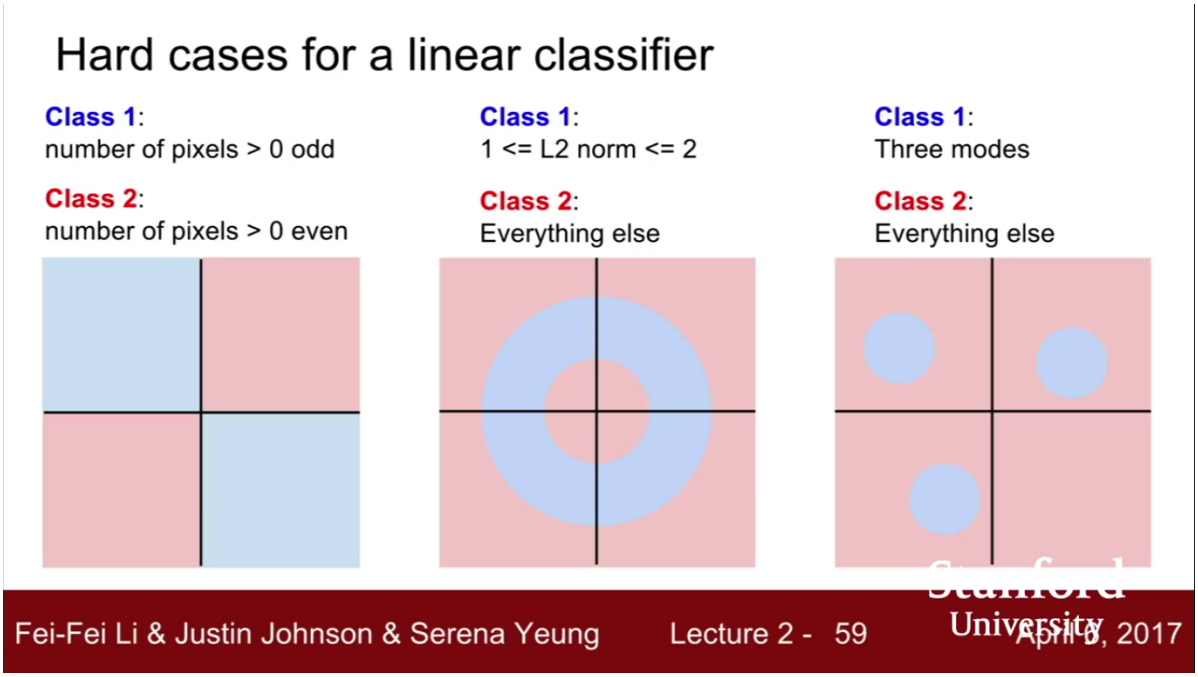
- So this Weight matrix(W) acts as a template matrix where each row corresponds to a template for the corresponding class.
- We can take each row of W and then reshape it to form an image to check what the classifier is learning for the corresponding class.



- We can notice in the second row that for the 'car' class the template is looking for a red blob with a black top blob in the image to classify it as car.
- Disadv:
  - **There is only 1 template for each class**. Eg. For the car class it is looking for a "red" blob with a "black" blob to classify it as "car".
  - Solution: **Neural Networks**. Since they don't have the restriction of learning just 1 template per category.



- During training the Linear Classifier will try to find the best fitting line to separate the given class with the rest of the classes.



$$f(x,W) = Wx + b$$

Coming up:

- Loss function (quantifying what it means to have a "good" W)
- Optimization (start with random W and find a W that minimizes the loss)
- ConvNets! (tweak the functional form of f)