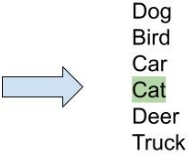
Deep Learning for Computer Vision

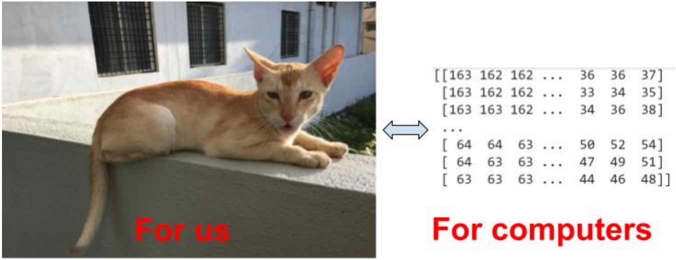
Dr. Konda Reddy Mopuri
Mehta Family School of Data Science and Artificial Intelligence
IIT Guwahati
Aug-Dec 2022

Image Classification

- Familiar CV task: i/p is an image and o/p is a category label



Challenge: Semantic gap



Challenge: View point variation

```
[[161 160 160 ...  46  46  47]
 [161 160 160 ...  43  44  45]
 [161 161 160 ...  42  44  46]
 ...
 [ 69  69  68 ...  62  64  66]
 [ 69  68  68 ...  59  61  63]
 [ 68  68  68 ...  56  58  60]]
```

```
[[160 159 159 ...  40  40  41]
 [160 159 159 ...  37  38  39]
 [160 160 159 ...  35  37  39]
 ...
 [ 68  68  67 ...  66  68  70]
 [ 68  67  67 ...  63  65  67]
 [ 67  67  67 ...  60  62  64]]
```

Challenge: intra-class-variation

Challenge: lighting-variation

Other Challenges

- Occlusion
- Deformation
- Clutter
- ...

Image classification: elementary task for other CV tasks

- Object detection

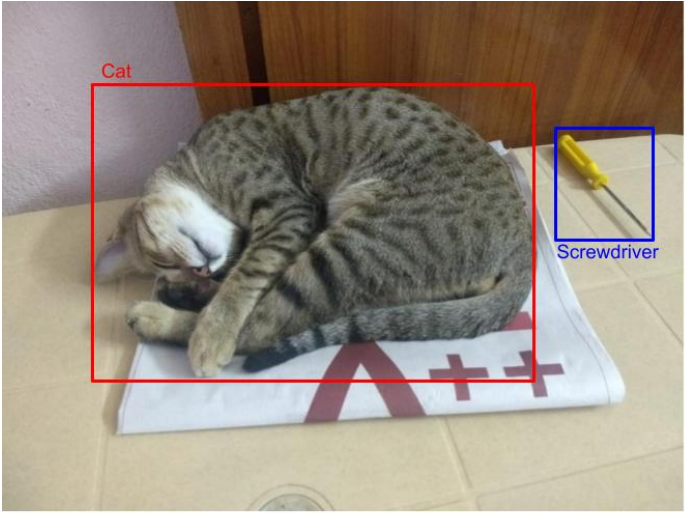
Image classification: elementary task for other CV tasks

- Object detection
- Caption Generation

Image classification: elementary task for other CV tasks

- Object detection
- Caption Generation
- Playing Chess/Go
- ...

Object Detection



Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word → ?

Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word **Cat**

Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word

Cat Sleeping

Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word

Cat Sleeping on

Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word **Cat Sleeping on the**

Caption Generation

Dog
Paper
Cat
Table
Screwdriver
Truck
Pen
Sleeping
.....
<EoS>

Predict the next word

Cat Sleeping on the table

How to build an image classifier?

```
def my_image_classifier():
  # some craftsmanship goes here
  return predicted_class_label
```

How to build an image classifier?

```python
def my_image_classifier():
  # some craftsmanship goes here
  return predicted_class_label
```

- Are there any rules that can we can hard-code? (like writing a program for addition of two numbers)

How to build an image classifier?

```
def my_image_classifier():
  # some craftsmanship goes here
  return predicted_class_label
```

- Are there any rules that can we can hard-code? (like writing a program for addition of two numbers)
- One can see that such an algorithm is not (i) gonna be robust, and (ii) transferable across categories

Here comes Machine Learning!

- Instead of trying to encode our knowledge of the objects, we take a data-driven approach

Here comes Machine Learning!

- Instead of trying to encode our knowledge of the objects, we take a data-driven approach
- Build algorithms that can learn from the data

Here comes Machine Learning!

```
def train(data):  # data: (images, labels)
  # Some machine learning!
  return trained_model
```

```
def test(trained_model, test_images):
  # trained_model performs the inference
  # on the input test images
  return predicted_labels
```

Common datasets for image classification: MNIST

- 10-class problem: $\{0, 1, 2, \ldots, 9\}$

Source

Common datasets for image classification: MNIST

- 10-class problem: $\{0, 1, 2, \ldots, 9\}$
- 28×28 gray-scale images

Source

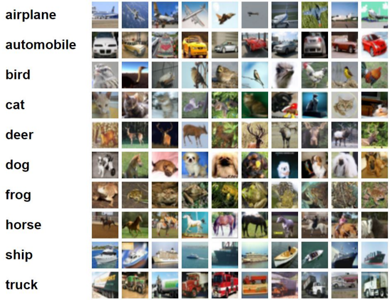
Common datasets for image classification: MNIST

- 10-class problem: $\{0, 1, 2, \ldots, 9\}$
- 28×28 gray-scale images
- $50K$ for training, and $10K$ for testing

Source

Common datasets for image classification: CIFAR-10

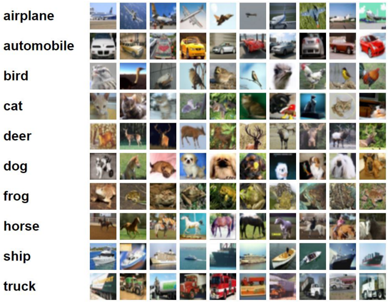
- 10-class problem: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck



Source

Common datasets for image classification: CIFAR-10

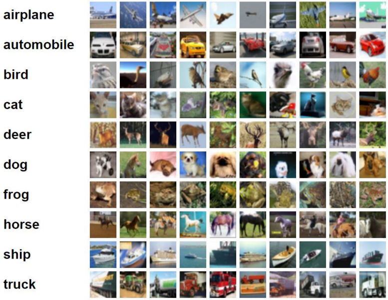
- 10-class problem: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- 32×32 RGB images



Source

Common datasets for image classification: CIFAR-10

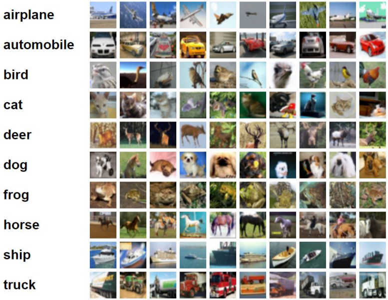
- 10-class problem: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- 32×32 RGB images
- $50K$ for training, and $10K$ for testing



Source

Common datasets for image classification: CIFAR-10

- 10-class problem: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- $32 \times 32 \times 3$ (RGB) images
- $50K$ for training, and $10K$ for testing



Source

We work with CIFAR-10

- We use CIFAR-10 for most of our assignments and experiments
- CIFAR-100 is a related dataset

Common datasets for image classification: ImageNet

- 1000 object categories

Source

Common datasets for image classification: ImageNet

- 1000 object categories
- $1.3M, 50K, 100K$ training, validation and testing images

Source

Common datasets for image classification: ImageNet

- 1000 object categories
- $1.3M, 50K, 100K$ training, validation and testing images
- Considered gold standard (as of 2020s)

Source

Common datasets for image classification

- MIT places

Common datasets for image classification

- MIT places
- Omniglot

Common datasets for image classification

- MIT places
- Omniglot
- iNaturalist
- . . .

Simple Classifier: Nearest neighbor

Simple Classifier: Nearest neighbor

- Training: Remember the labels of all the training data samples

Simple Classifier: Nearest neighbor

- Training: Remember the labels of all the training data samples
- Testing: Predict the label of the nearest training sample

Nearest neighbor classifier

Remember the labels of training samples

```
def train(data):   # data: (images, labels)
  # ??
  return trained_model
```

Pick the label of the closest training sample

```
def test(trained_model, test_images):
  # ??
  return predicted_labels
```

Nearest neighbor classifier

Remember the labels of training samples

```python
def train(data):  # data: (images, labels)
  # ??
  return trained_model
```

Pick the label of the closest training sample

```python
def test(trained_model, test_images):
  # ??
  return predicted_labels
```

Time for some hands-on!

- Implement the NN classifier and evaluate on MNIST and CIFAR-10

Distance metric between images

- Vectorize (or, flatten) the images, $d = W \times H \times \#channels$

$$d(I_1, I_2) = \left(\sum_{i=1}^{d} \left| I_1(i) - I_2(i) \right|^p \right)^{1/p}$$

Distance metric between images

- Vectorize (or, flatten) the images, $d = W \times H \times \#channels$

$$d(I_1, I_2) = \left(\sum_{i=1}^{d} \left| I_1(i) - I_2(i) \right|^p \right)^{1/p}$$

- Referred to as l_p norm (l_1, and l_2 are commonly used)

Nearest neighbor classifier

- Training and Testing complexity?

Nearest neighbor classifier

- Training and Testing complexity?
- Constant time $O(1)$ and linear $O(N)$ respectively, where N is size of training set

Nearest neighbor classifier

- Training and Testing complexity?
- Constant time $O(1)$ and linear $O(N)$ respectively, where N is size of training set
- This inference complexity is very much undesirable!

Nearest neighbor classifier

- Training and Testing complexity?
- Constant time $O(1)$ and linear $O(N)$ respectively, where N is size of training set
- This inference complexity is very much undesirable!
- Algorithms are there for finding approximate NNs fast

Nearest neighbor classifier

- Training and Testing complexity?
- Constant time $O(1)$ and linear $O(N)$ respectively, where N is size of training set
- This inference complexity is very much undesirable!
- Algorithms are there for finding approximate NNs fast
- Dimensionality reduction can be considered (e.g., PCA)

NN Classification boundaries

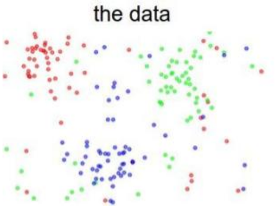


Image Source: CS231n

NN Classification boundaries

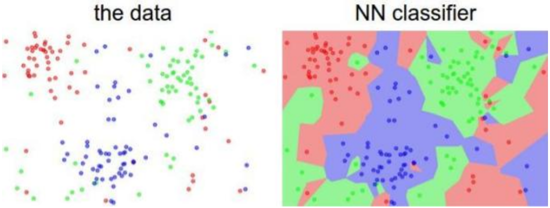
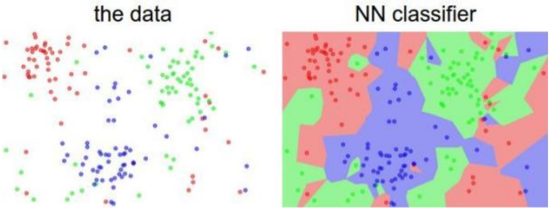


Image Source: CS231n

NN Classification boundaries



Observations

- Boundaries are noisy
- Outliers can affect the decisions seriously

Image Source: CS231n

How to address these issues?

- Instead of relying on the NN, take a majority voting from multiple neighbors

How to address these issues?

- Instead of relying on the NN, take a majority voting from multiple neighbors
- K nearest neighbors (KNN) classifier

K-NN Classification boundaries

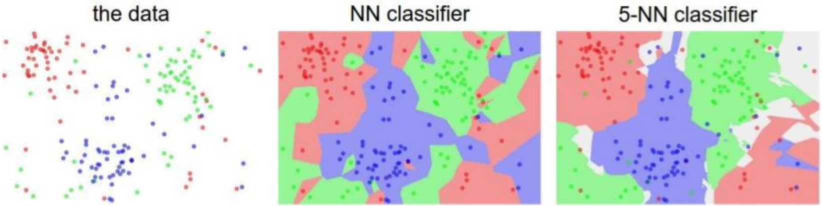


Image Source: CS231n

K-NN Classifier

- With appropriate distance metric one can use KNN classifier for any type of data!

K-NN Classifier, but

- What is the best value of K?

K-NN Classifier, but

- What is the best value of K?
- What is the suitable distance metric?

K-NN Classifier, but

- What is the best value of K?
- What is the suitable distance metric?
- These are instances of hyper-parameters in learning

K-NN Classifier, but

- What is the best value of K?
- What is the suitable distance metric?
- These are instances of hyper-parameters

Hyper-parameters

- Problem-dependent
- General strategy is to try out different values and see what works best!

Setting hyper-parameters

- Try out on the train set?

Setting hyper-parameters

- Try out on the train set?
- Try out on the test set?

Setting hyper-parameters

- Try out on the train set?
- Try out on the test set?
- Answer is the Validation set!

Pros and Cons of of KNN classifier

- As the size of training data grows to infinity, KNN classifier can represent (almost) any function!

Pros and Cons of of KNN classifier

- As the size of training data grows to infinity, KNN classifier can represent (almost) any function!
- However, succumbs to the curse of dimensionality

Pros and Cons of of KNN classifier

- As the size of training data grows to infinity, KNN classifier can represent (almost) any function!
- However, succumbs to the curse of dimensionality
- Expensive inference

Pros and Cons of of KNN classifier

- As the size of training data grows to infinity, KNN classifier can represent (almost) any function!
- However, succumbs to the curse of dimensionality
- Expensive inference
- Distance metrics in the pixel space is not very informative (but, one may work with more semantic features such as ones from CNNs)

Linear Classifiers

- More powerful than KNN

Linear Classifiers

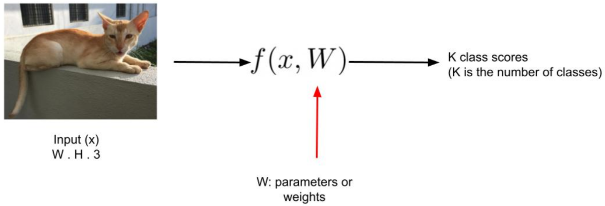
- More powerful than KNN
- Naturally extends to NNs

Linear Classifiers

- More powerful than KNN
- Naturally extends to NNs
- Simple parametric approach for classification

Linear Classifier

- More powerful than KNN
- Naturally extends to NNs
- Simple parametric approach for classification



Linear Classifier: parametric approach

- $f(x, W) = Wx,$
  $x \in \mathcal{R}^d, W \in \mathcal{R}^{K \times d}$

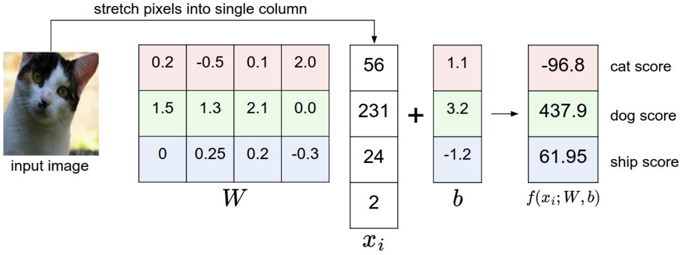
Linear Classifier: parametric approach

- $f(x, W) = Wx$,
  $x \in \mathcal{R}^d, W \in \mathcal{R}^{K \times d}$
- More general form has bias term
  $f(x, W) = Wx + b$, where $b \in \mathcal{R}^K$

Linear Classifier: parametric approach

- $f(x, W) = Wx$,
  $x \in \mathcal{R}^d, W \in \mathcal{R}^{K \times d}$
- More general form has bias term
  $f(x, W) = Wx + b$, where $b \in \mathcal{R}^K$
- Sometimes we may encounter the bias trick (absorbing the bias into the parameter vector)

Linear Classifier: interpretation



Linear classifier predicting the score as weighted sum over the pixel values

Figure Source: CS231n

Linear Classifier: interpretation

Linear classifier as a template matching

Figure Source: CS231n

Linear Classifier: interpretation

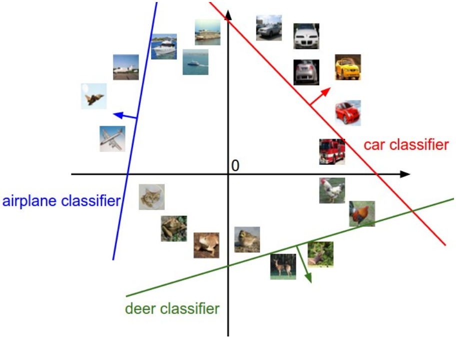
Linear classifier as a template matching

Single template

- Single template may not be sufficient to capture a multi-modal class
- Observe the template for Horse category, heads on both the sides

Figure Source: CS231n

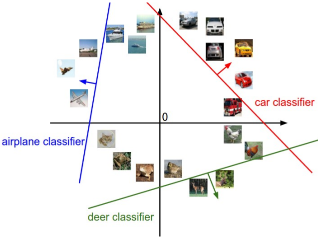
Linear Classifier: interpretation



Linear classifier as a separating hyperplane in the input space

Figure Source: CS231n

Linear Classifier: interpretation



Linear classifier as a separating hyperplane in the input space

Nonlinear boundaries

- Linear classifiers can't learn data that is not linearly separable (e.g., XOR function)

Figure Source: CS231n

The two important functions

- So far, we talked about the function for score prediction (scoring function)

The two important functions

- So far, we talked about the function for score prediction (scoring function)
- But, how do we choose a suitable W?

The two important functions

- So far, we talked about the function for score prediction (scoring function)
- But, how do we choose a suitable W?
- Loss function: measures how much happy we are about the model's predictions

The two important functions

- So far, we talked about the function for score prediction (scoring function)
- But, how do we choose a suitable W?
- Loss function: measures how much happy we are about the model's predictions
- Also known as objective, or cost function

Multi-class SVM loss

- $s_j = f(x_i, W)_j$

Multi-class SVM loss

- $s_j = f(x_i, W)_j$
- $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$

Multi-class SVM loss

- $s_j = f(x_i, W)_j$
- $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$
- Interpretation: it wants the score predicted for the correct class to be higher than that of the incorrect classes by some fixed margin

Multi-class SVM loss

- $s_j = f(x_i, W)_j$
- $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$
- Interpretation: it wants the score predicted for the correct class to be higher than that of the incorrect classes by some fixed margin
- E.g., $s = [13, -7, 11]$, ground-truth is class 0, compute the loss (for a $\Delta value of 10$

Multi-class SVM loss

- $s_j = f(x_i, W)_j$
- $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$
- Interpretation: it wants the score predicted for the correct class to be higher than that of the incorrect classes by some fixed margin
- E.g., $s = [13, -7, 11]$, ground-truth is class 0, compute the loss (for a $\Delta value of 10$
- Also known as 'Hinge loss' Why?

Multi-class SVM loss

- Let's say we found a W that correctly classifies all the training examples

Multi-class SVM loss

- Let's say we found a W that correctly classifies all the training examples
- Is it going to be unique?

Multi-class SVM loss

- Let's say we found a W that correctly classifies all the training examples
- Is it going to be unique?
- If not, how to choose one among them? Can we have preferences?

Regularization

- Common regularization is L_2, $R(W) = \sum_k \sum_l W_{k,l}^2$

Regularization

- Common regularization is L_2, $R(W) = \sum_k \sum_l W_{k,l}^2$
- $L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$

Regularization

- Common regularization is L_2, $R(W) = \sum_k \sum_l W_{k,l}^2$
- $L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$
- $L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$

Softmax loss function

Reading ahead!

- Built on the cross-entropy from the Information theory $H(p,q) = -\sum_x p(x) \log q(x)$
- Generalization of binary Logistic Regression to multiple classes
- Read about it and implement, we will use it for training soon!

Next class

- We are yet to figure out the procedure for identifying a good $W \rightarrow$ Optimization!