2 OCTOBRE

DEEP
LEARNING

THE PROBLEM OF IMAGE CLASSIFICATION

Key Points Overview

Supervised by

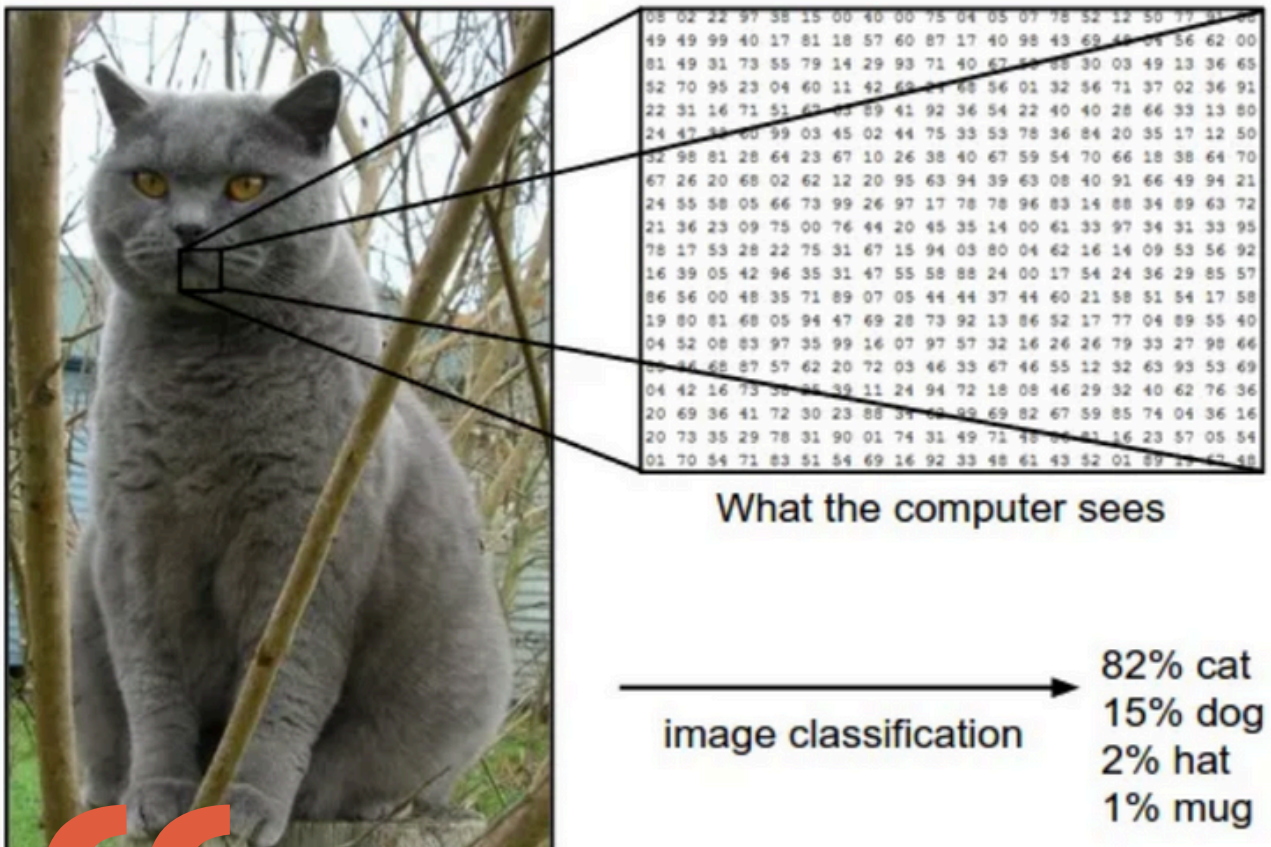
PROF. BELCAID ANASS

Prepared By

BOUHNAS CHAYMAE

SOMMAIRE

Image classification is a fundamental computer vision task where the goal is to assign a label from a set of predefined categories to an image. This simple but significant problem has widespread applications, including medical diagnostics, autonomous driving, and facial recognition. Additionally, tasks like object detection and segmentation are often built on image classification.



For example, a model processing an image of a cat will assign probabilities to possible labels (like "cat" or "dog"). The image is stored as a 3D array of pixel values, where a 248 x 400 pixel image with 3 color channels contains 297,600 values, each representing color intensity(from 0 (black) to 255 (white)). The model's task is to interpret these numbers and output the correct label, such as "cat."

CHANLLENGES

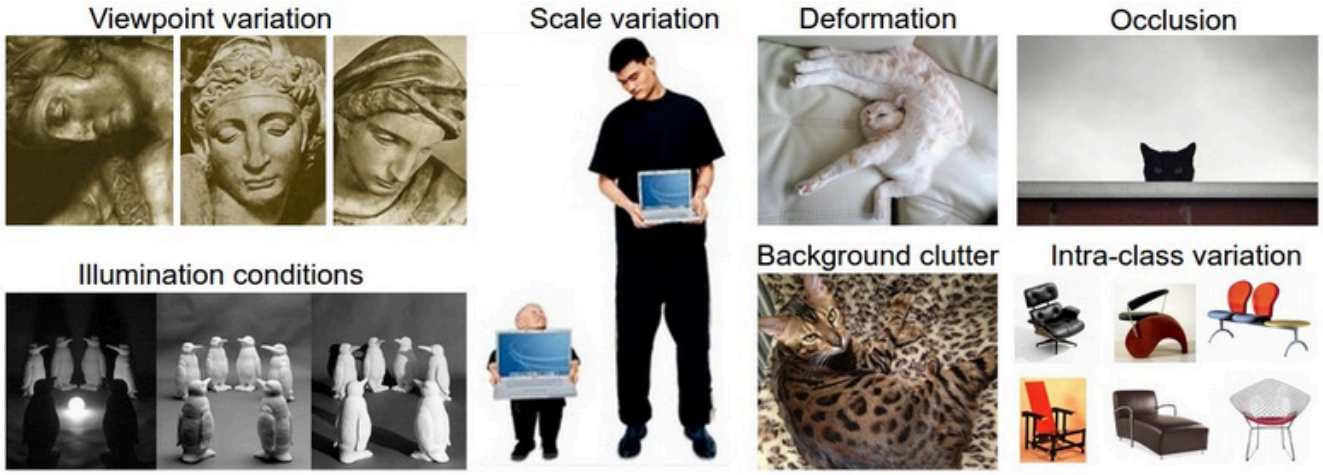


Image classification presents several challenges that make it difficult for computer vision models, despite being easy for humans. Key challenges include:

Viewpoint variation: Objects can appear from different angles relative to the camera.

Scale variation: Objects may vary in size, both in real-world dimensions and in the image.

Deformation: Non-rigid objects can change shape significantly.

Occlusion: Objects may be partially hidden, with only small parts visible.

Illumination conditions: Lighting can drastically change pixel values.

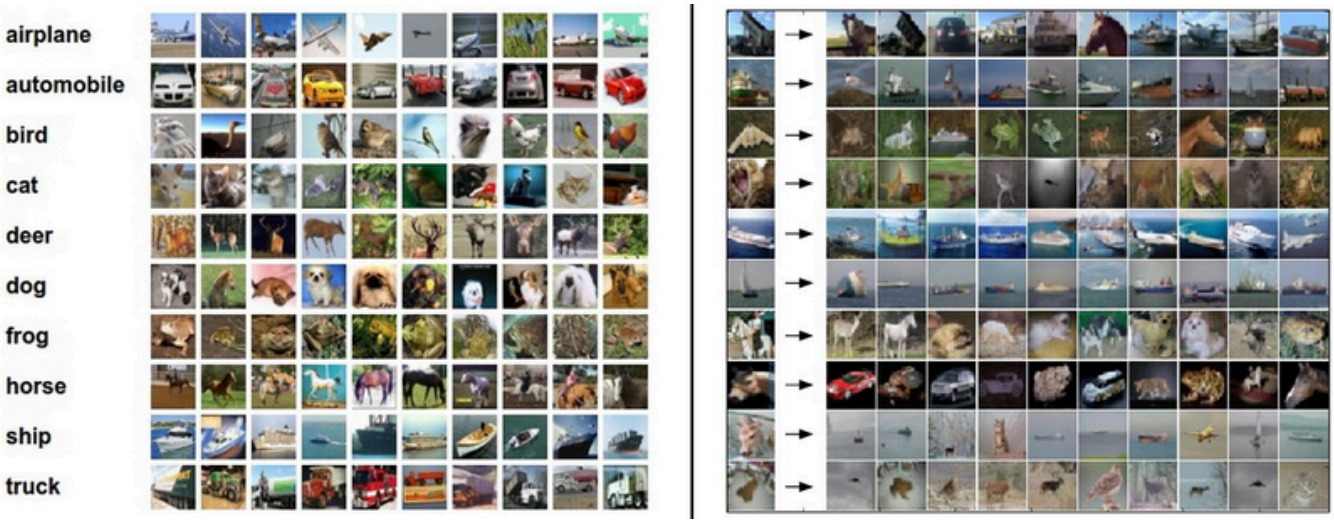
Background clutter: Objects may blend into complex environments, making them hard to distinguish.

Intra-class variation: Objects within the same category, like chairs, can look vastly different, requiring the model to generalize well across variations while distinguishing between classes.

– In a data-driven approach to image classification, algorithms learn to recognize categories, like cats, by analyzing many labeled examples rather than explicit programming. This method relies on building a training dataset, allowing models to discern patterns through exposure, similar to teaching a child.

DEFINITION OF CIFAR-10

CIFAR-10 is a widely used dataset in computer vision for image classification. It consists of 60,000 color images, each measuring 32x32 pixels, divided into 10 classes (such as airplane, automobile, bird, and cat). The dataset is split into 50,000 training images and 10,000 testing images.



Training Data: The dataset used for training models, where each data point (x) is a feature vector (an image) paired with a label (y) that indicates its class (e.g., "cat" or "dog").
n: Represents the dimensionality of the input data (number of pixels in an image).
k: The number of distinct classes in the dataset.
m: The total number of training data points.
Using the CIFAR dataset as an example, each image is 32x32 pixels (n=1024), with 50,000 training images (m) across 10 classes (k). This framework is essential for understanding how image classification models are developed and assessed.

EUCLIDEAN DISTANCE :

Definition: This is the straight-line distance between two points in a multi-dimensional space. It is calculated using the formula:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Where x and y are two points in n-dimensional space.
Usage: In image classification, Euclidean distance is often used to measure similarity between image feature vectors. It works well when the features are on a similar scale.

MANHATTAN DISTANCE

Definition: Also known as "taxicab" or "city block" distance, it calculates the distance between two points by summing the absolute differences of their coordinates:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Usage: This metric is beneficial when the feature space is grid-like and works well in high-dimensional spaces. It can be more robust to outliers compared to Euclidean distance.

EUCLIDEAN DISCHEBYSHEV DISTANCE (INFINITY NORM) :

Definition: This distance metric calculates the maximum absolute difference along any coordinate dimension:

$$d = \max(|x_i - y_i|)$$

Usage: Chebyshev distance is useful in scenarios where the most significant difference in any dimension is of primary interest, such as in some classification tasks where certain features may dominate.

MAHALANOBIS DISTANCE :

Definition: This distance takes into account the correlations of the data set and is defined as:

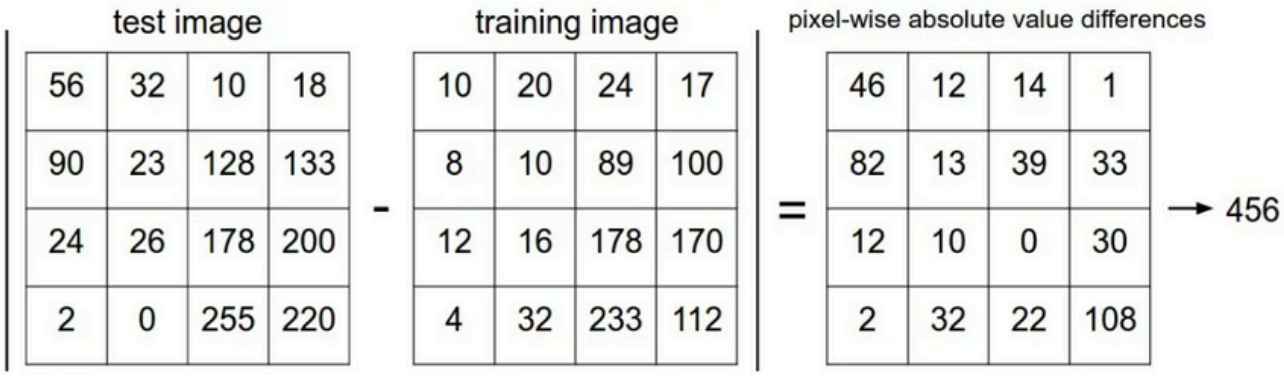
$$d = \sqrt{(x - y)^T S^{-1} (x - y)}$$

Usage: Mahalanobis distance is particularly useful in image classification when dealing with multi-dimensional data where features are correlated. It can help to identify the distance between a point and a distribution, rather than just between points.

DEFINITION:

The Nearest Neighbor Classifier is a straightforward image classification method that compares a test image to all training images to find the closest match, assigning the label of that nearest image. H However, its effectiveness can be limited; the classifier may struggle with noise or complex backgrounds, which can lead to incorrect predictions.
To compare images, one common approach is pixel-wise analysis using the L1 distance, which sums the absolute differences between corresponding pixels:

$$d_1(I_1, I_2) = \sum_p |I_{p1} - I_{p2}|$$



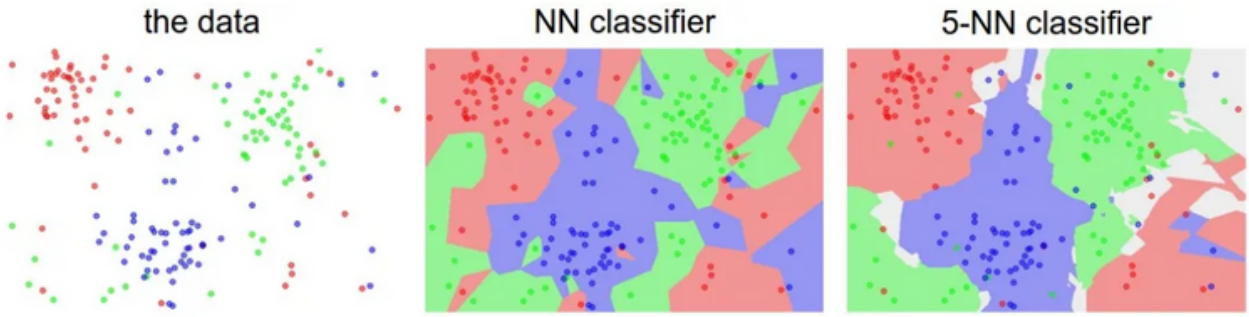
Another option is the L2 distance, representing the Euclidean distance between two vectors:

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_{p1} - I_{p2})^2}$$

The L2 distance is more sensitive to larger discrepancies, preferring multiple small differences over one significant one, while the L1 distance offers a more forgiving metric.

4-NEAREST NEIGHBOR CLASSIFIER

To improve prediction accuracy, the k-Nearest Neighbor (k-NN) Classifier is often employed, where instead of just finding the nearest image, the algorithm identifies the top k closest images and uses their labels to vote on the test image's label. This approach helps mitigate the influence of outliers, making the classifier more robust.
Overall, while the Nearest Neighbor Classifier provides a foundation for understanding image classification, its limitations necessitate the exploration of more advanced techniques.



The example shows the difference between Nearest Neighbor (NN) and 5-Nearest Neighbor (5-NN) classifiers in classifying 2D points across three classes (red, blue, green). The NN classifier may misclassify points due to outliers, while the 5-NN classifier averages votes from multiple neighbors, leading to smoother decision boundaries and improved generalization. Gray areas indicate ties in votes, reflecting the balance between sensitivity to outliers and classification accuracy.

BROADCASTING :

When we talk about broadcasting, we're referring to the way NumPy handles arrays of different shapes in mathematical operations. The rules for broadcasting are as follows:

- Alignment: Broadcasting compares the dimensions of the arrays starting from the rightmost dimension. If dimensions are equal or one of them is 1, they are compatible.

Exemple :

- Array A: Shape (4, 3)This means 4 rows and 3 columns.
- Array B: Shape (3, 5)This means 3 rows and 5 columns.

Aligning from the right:

- A has shape (4, 3), which we can think of as:
  - Dimension 0: 4 (rows)
  - Dimension 1: 3 (columns)
- B has shape (3, 5), which can be viewed as:
  - Dimension 0: 3 (rows)
  - Dimension 1: 5 (columns)

HOW BROADCASTING WORKS :

When performing an operation between A and B, the dimensions will align as follows:

- The second dimension of A (3) matches the first dimension of B (3):
  - A: (4, 3) aligns with B: (3, 5)

This means that for each of the 4 rows in A, the corresponding 3 rows in B can be operated on together, creating a larger resulting array.

FINAL RESULTING SHAPE :

- After broadcasting:
  - A effectively becomes (4, 3, 5) (A is expanded along a new dimension).
  - B effectively becomes (4, 3, 5) by repeating along the first dimension (4 times).

The operation results in an array where each combination of values from A and B can be processed together, producing a new shape of (4, 5).

5-CODE EXAMPLES

Example Arrays :

Array A (Shape: (4, 3)):

```
[[ a11, a12, a13 ],
 [ a21, a22, a23 ],
 [ a31, a32, a33 ],
 [ a41, a42, a43 ]]
```

Array B (Shape: (3, 5)):

```
[[ b11, b12, b13, b14, b15 ],
 [ b21, b22, b23, b24, b25 ],
 [ b31, b32, b33, b34, b35 ]]
```

RESULT OF BROADCASTING :

When performing operations like addition or multiplication, the two arrays will be broadcast together, leading to a resulting shape of (4, 5). Each row of Array A combines with each row of Array B:

```
[[ a11 + b11, a12 + b12, a13 + b13, b14 + b14, b15 + b15 ],
 [ a21 + b21, a22 + b22, a23 + b23, b24 + b24, b25 + b25 ],
 [ a31 + b31, a32 + b32, a33 + b33, b34 + b34, b35 + b35 ],
 [ a41 + b41, a42 + b42, a43 + b43, b44 + b44, b45 + b45 ]]
```

EXPLANATION OF THE RESULT :

- Row-wise Operation: Each element of Array A adds to the corresponding elements of each row in Array B, leading to a new array where:
  - The first row is derived from the first row of A and all rows of B.
  - This continues for all rows in A, ensuring each row combines with all rows in B.

EXAMPLE IN PYTHON (NUMPY):

Broadcasting to (4, 3, 5)

```python
import numpy as np
A = np.random.rand(4, 3)
B = np.random.rand(3, 5)
result = A[:, :, np.newaxis] + B[np.newaxis, :, :]
```

The resulting shape of the broadcasted operation is (4, 3, 5).

CODE :

We would now like to classify the test data with the kNN classifier

Step 1 :

first we must compute the distances between all test examples and all train examples.
we have 3 methodes but in this note we are going to see 2 :

Using two loops :

```python
def compute_distances_two_loops(self, X):

        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        for i in range(num_test):
            for j in range(num_train):

                diff = X[i] - self.X_train[j]

                dists[i, j] = np.sqrt(np.sum(diff ** 2))

                pass
        return dists
```

The compute_distances_two_loops function calculates the Euclidean distance between each test point in the provided dataset X and all training points stored in self.X_train.

5-CODE EXAMPLES

It initializes a distance matrix and uses nested loops to iterate over each test and training example. For every pair, it computes the difference between their feature vectors, squares this difference, sums the squared values, and finally takes the square root to determine the distance. The resulting matrix dists holds the distances between every test and training pair, allowing further processing for classification tasks.

Using one loop :

```python
def compute_distances_one_loop(self, X):
    """
    Compute the distance between each test point in X and each training point
    in self.X_train using a single loop over the test data.

    Input / Output: Same as compute_distances_two_loops
    """
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        dists[i] =  np.sqrt(np.sum((X[i] - self.X_train)**2,axis = 1))

    return dists
```

The compute_distances_one_loop function efficiently calculates the distances between a set of test points and training points using a single loop.

Instead of iterating through both the test and training points (as in the double loop method), this implementation focuses on each test point while leveraging NumPy's broadcasting capabilities. For each test point X[i], the function calculates the squared differences from all training points in self.X_train. By summing these squared differences along the appropriate axis, it forms a single array that represents the distances to all training points.

Finally, it applies the square root to convert the squared distances to Euclidean distances, resulting in a distance matrix where each row corresponds to a test point and each column corresponds to a training point. This approach enhances performance by reducing the time complexity compared to the double loop method.

Step 2 :

Given these distances, for each test example we find the k nearest examples and have them vote for the label

Python Copy Caption ...

```python
def predict_labels(self, dists, k=1):
    num_test = dists.shape[0]
    y_pred = np.zeros(num_test)


    for i in range(num_test):
        # Step 1: Find the k nearest neighbors
        closest_y = list(np.argsort(dists[i])[:k])


        # Step 3: Find the label with the highest count

        y_pred[i] =self.y_train[np.max(closest_y, key = closest_y.count)]

    return y_pred
```

The predict_labels function predicts labels for test samples based on their distances to training samples. It takes a distance matrix (dists) as input, where each entry represents the distance between a test point and a training point. For each test point, it identifies the indices of the k nearest training points using np.argsort(dists[i])[:k]. Then, it retrieves the corresponding labels from self.y_train and determines the most common label among these neighbors. This label is stored in y_pred, which is returned at the end. If there's a tie in label counts, the smallest label is selected.

QUESTION 1 :

Inline Question 1 :

Notice the structured patterns in the distance matrix, where some rows or columns are visible brighter. (Note that with the default color scheme black indicates low distances while white indicates high distances.)

- What in the data is the cause behind the distinctly bright rows?
- What causes the columns?

Bright Rows :

The distinctly bright rows in the distance matrix typically indicate that the corresponding test images are relatively distant from most training images. When a row appears "mostly whitish," it suggests that the pixel values of that test image differ significantly from the pixel values of the training images, leading to higher distances. In contrast, a "mostly blackish" row implies that the test image is similar to many training images, meaning their pixel values are close together.

Bright Columns :

For the columns, the brightness indicates the distance of training images from test images. If a column is bright, it suggests that the corresponding training image is far from the test images, implying it has pixel values that are not closely aligned with those of the test images. A dark column, on the other hand, would indicate that the training image is similar to many test images.

Rows (Test Images): Bright rows signify high distances from training images, indicating low similarity.
Columns (Training Images): Bright columns indicate that a training image is distant from test images, again reflecting low similarity.