Cake classification
Machine Learning assignment
Marija Maneva

Abstract

Image recognition is a crucial application of machine learning, and there are two main approaches to tackle it: using hand-crafted image features processed by a classification model or leveraging deep learning with convolutional neural networks that directly process image pixels. Among these approaches, CNNs have emerged as the most suitable architecture for image recognition tasks.

In this lab activity, the focus is on building classifiers for the classification of cake images. The dataset consists of 15 different kinds of cakes, with 120 images per class. The training set contains 100 images per class, while the test set has 20 images per class. All images have been resized to a resolution of 224 × 224 pixels.

PVMLNet

The mentioned CNN is a simplified version of AlexNet architecture. The network consists of eight convolutions and three fully-connected layers (implemented as convolutions), with ReLU activations applied after each layer.
The final output is processed by a softmax function to compute class probabilities.
This CNN has been trained on the ILSVRC-12 subset of ImageNet.

1.1 Low level features

- *Write a script that computes one of the low-level feature vector implemented in the image_features.py file. Train a classifier and evaluate the test accuracy.*

(Note: What are low-level features?
They are a set of predefined visual descriptors or characteristics extracted from images using specific algorithms or techniques. The main difference about these features and deep learning approaches is that they capture basic and fundamental information about images after typically applying mathematical operations or filters, meanwhile, deep learning techniques learn features automatically from raw image data.)

The script “low-level-features.py” contains a code that reads images from specified directories, extracts features using “image_features.color_histogram” and saves the extracted features and labels to text files for further use.
In this case I used the function “color_histogram” to gain the low-level features needed. In fact ,the color_histogram function takes an image and computes the RGB marginal histograms, which represent the distribution of pixel values in each channel of the image. The histograms are normalized to sum to one, providing a representation of the color distribution in the image.

Output:

Type of low-level feature	Test accuracy (%)
color histogram	20.7

Comment :
Low-level feature extraction techniques focus on extracting basic features so there’s the possibility that they may not capture complex patterns present in the images. This limitation can lead to low accuracy.

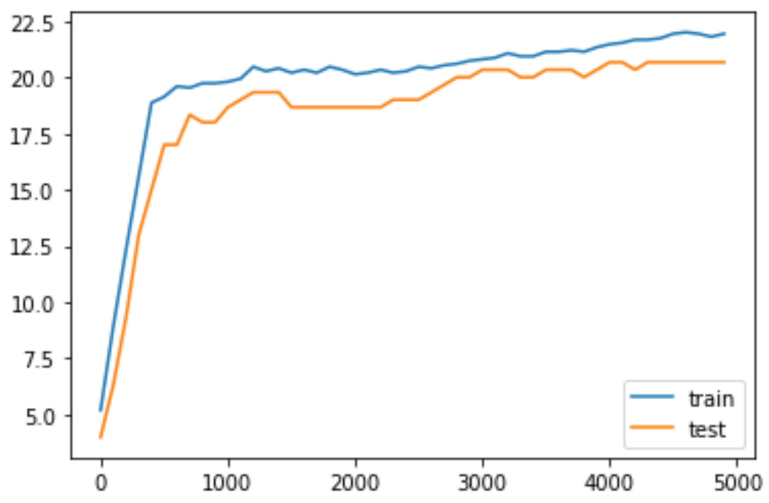


Figure 1: train&test accuracy with color histogram feature

1.2 Neural features

- *Use the pretrained PVMLNet to extract as features the activations of the last hidden layer. Train a perceptron without hidden layers and evaluate the test accuracy.*

The script "neural-features.py" extracts neural features from the images. Training with neural features involves learning representations directly from the data using neural networks.

Output:

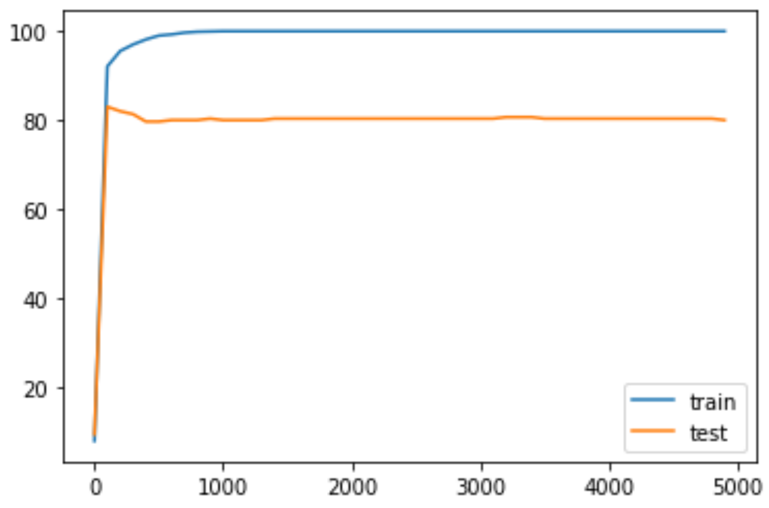


Figure 2 : train&test accuracy with neural features

Test accuracy reached: 80.3 %

Comment :

Neural features have the advantage of capturing more complex and high-level information, allowing the model to potentially achieve better performance and adaptability to different tasks.

1.3 Transfer learning

- *Build a new network by replacing the last layer of PVMLNet with the weights of the trained perceptron*

The script "transfer_learning" contains a code that :

- combines a pre-trained CNN model with an MLP model
- performs image classification on a test image using the combined model
- displays the image along with the top predicted class labels and their probabilities
- computes and displays the confusion matrix

2.1 Combining features

- *Try different combinations of low-level features*

2 features: color histogram & edge detector

Test accuracy reached : 22.3 %

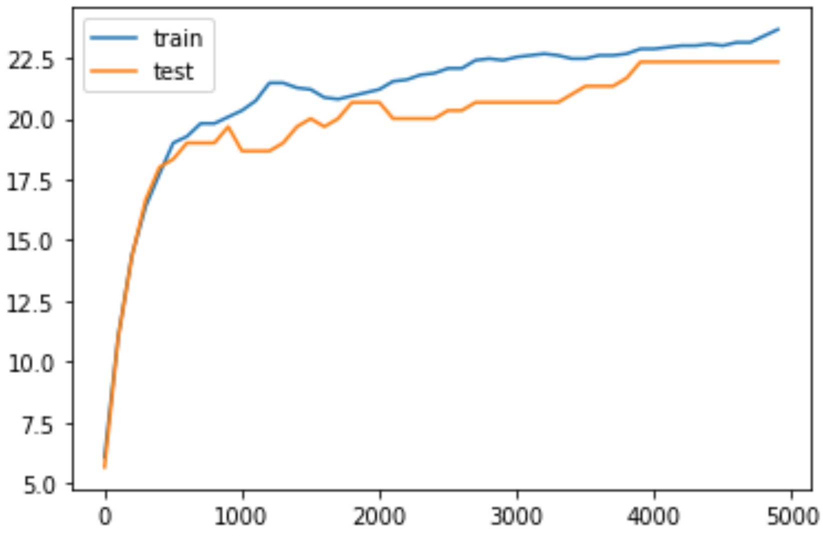


Figure 3: train&test accuracy with two low level features combines

3 features: color histogram & edge detector & co-occurrence matrix

Test accuracy reached : 23.3 %

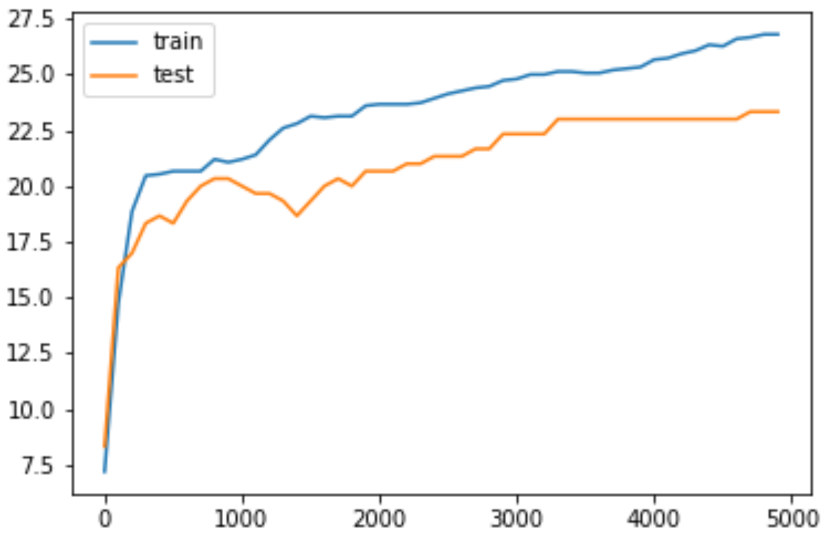


Figure 4: train&test accuracy with three low level features combines

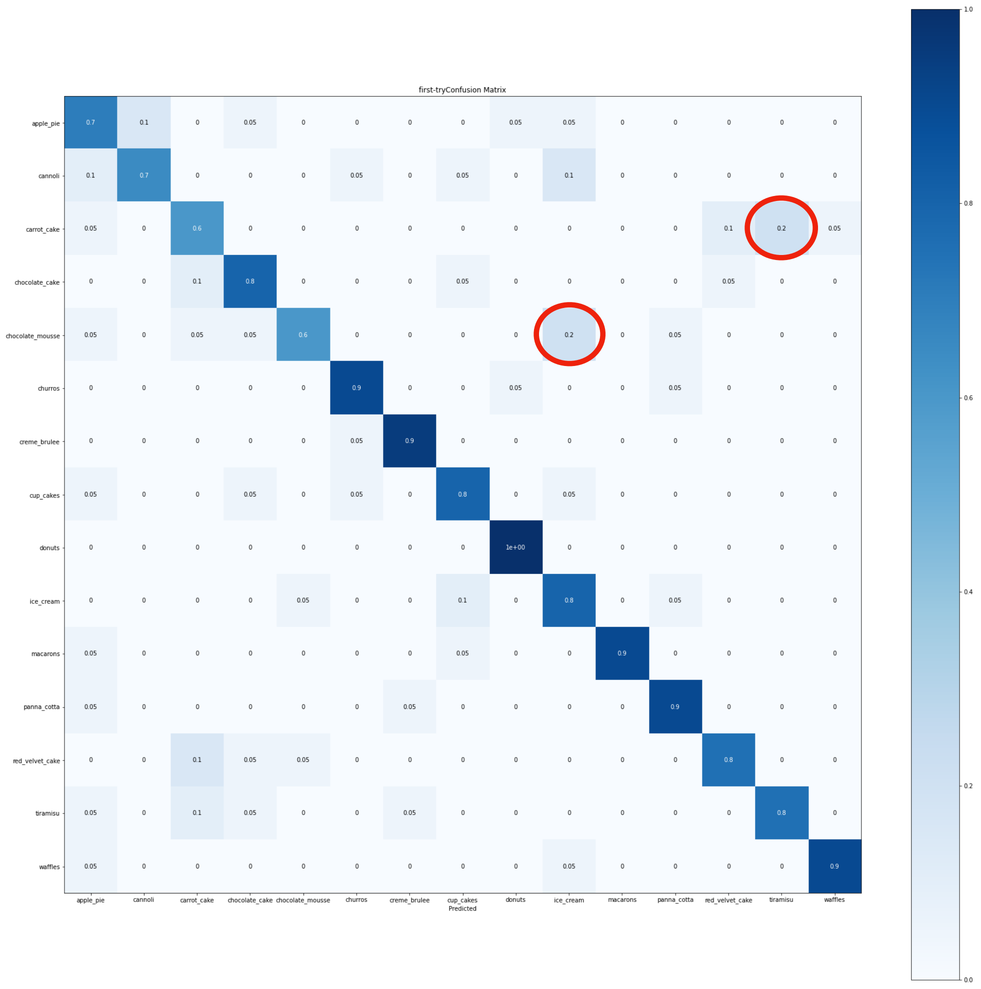
By combining different features, the accuracy of the model increases. When you concatenate more features, multiple feature vectors are combined into a single feature vector. This allows the model to capture different aspects or representations of the data.

2.2 Analysis

- *Identify the pairs of classes that are more likely to be confused with neural features*

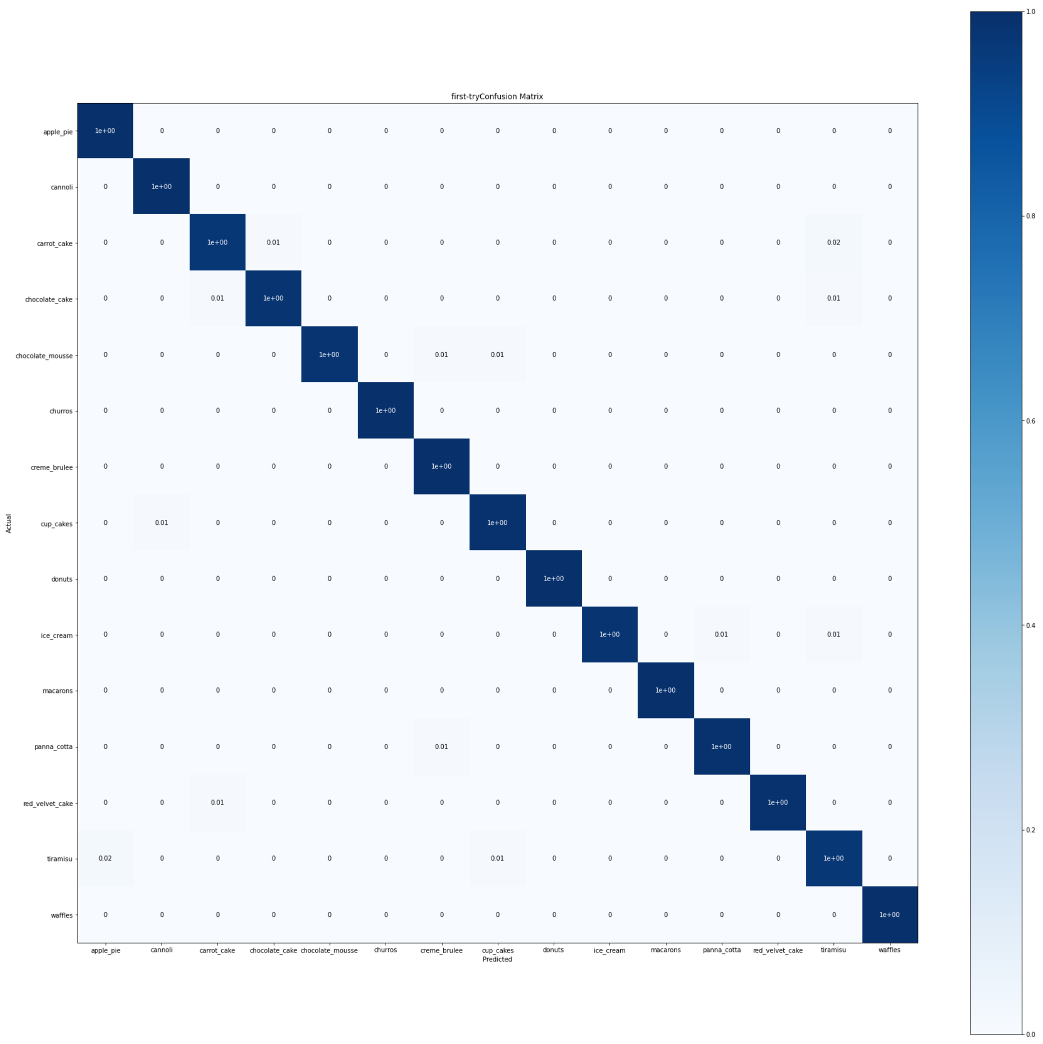
In order to identify the pair of classes that are more likely to be confused the best way is to compute the confusion matrix.
By analyzing the off-diagonal elements in a confusion matrix, it is possible to identify the classes that have a greater tendency to be mistaken for one another. These elements denote instances where the predicted class differs from the actual class. The presence of larger values in these off-diagonal elements indicates a higher level of confusion between specific classes. To determine which classes are more prone to confusion, I examined the off-diagonal elements with the highest values.

Confusion matrix on test set



The test images that are being misclassified with the highest probability are ice cream with chocolate mousse and tiramisu with carrot cake.

Confusion matrix on training set



The training set has a high accuracy so there are classes that are misclassified with a very low probability (0.01) which can be considered as zero.

2.3 Neural features

- *Try to use neural features computed by different hidden layers*

Specifications :

- *Number of batches = 50*
- *Learning rate = 0.0001*
- *Steps = 100*

- ***1 hidden layer with width 100***

Test accuracy reached : 79.7 %

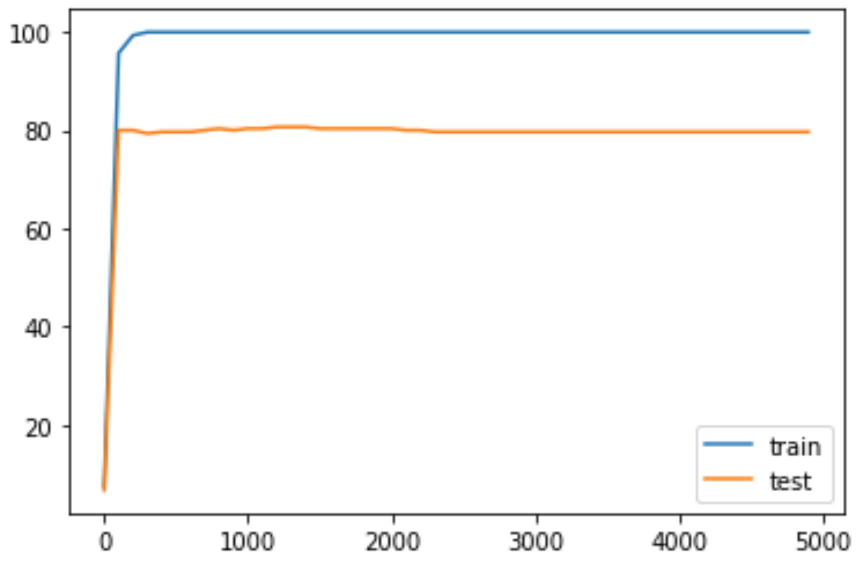


Figure 5: train&test accuracy with neural features & 1 hidden layer (100)

- ***1 hidden layer with width 50***

Test accuracy reached : 78.3 %

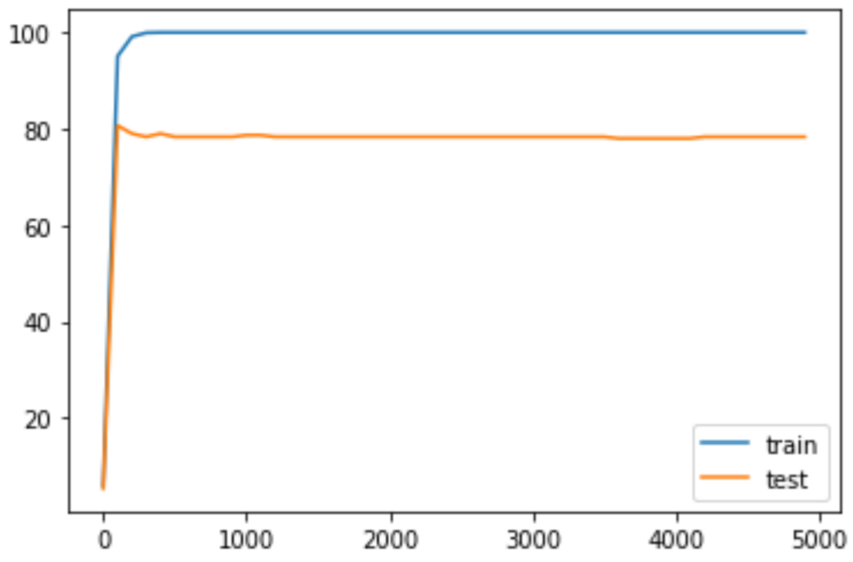


Figure 6: train&test accuracy with neural features & 1 hidden layer (50)

- ***2 hidden layers with width 100 & 50***

Test accuracy reached : 80.7 %

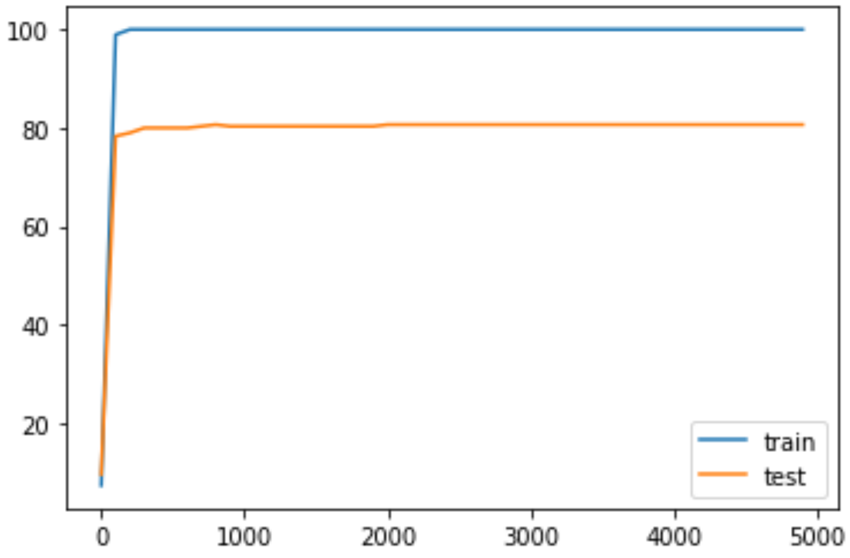


Figure 7: train&test accuracy with neural features & 2 hidden layers (100&50)

The number of hidden layers in a convolutional neural network (CNN) can have an impact on the network's performance and its ability to learn complex features from input data. Adding more hidden layers to a CNN can potentially increase its capacity to capture intricate patterns and representations, but it may also introduce challenges such as overfitting.
As we can see from the plots the accuracy increases by increasing the number of hidden layers and the width of the single layer

- *Reduce the activations to a single feature vector by averaging over the spatial dimensions*

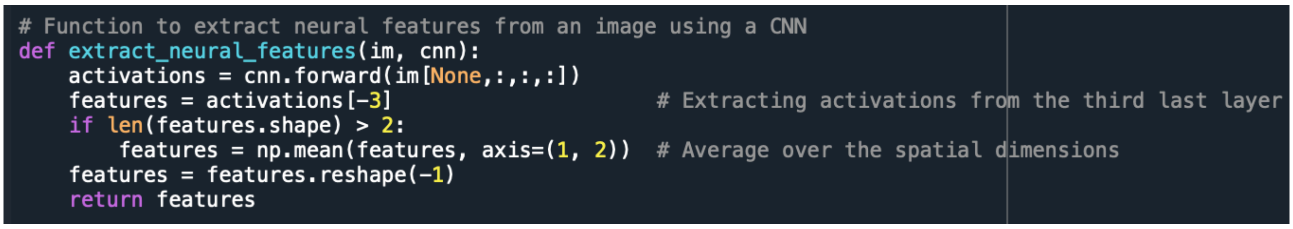
The process of reducing the spatially distributed activations to a single feature vector by averaging over the spatial dimensions is commonly known as global average pooling.
The purpose of global average pooling is to condense the spatial information captured by the activations
The main of applying it is :

- to focus on the most important features within each feature map —> it reduces the risk of overfitting
- to reduce the number of parameters
- to preserve the spatial structure of the activations —> beneficial for image segmentation or localization

Overall, global average pooling provides a way to convert the spatially distributed activations in a CNN into a concise and informative feature vector, facilitating subsequent processing and decision-making tasks.

With no hidden layers and the specifications mentioned in the previous answer, the test accuracy reached is : 79.7 %

Code used in the script "neural-features.py" to obtain the results explained :



```python
# Function to extract neural features from an image using a CNN
def extract_neural_features(im, cnn):
    activations = cnn.forward(im[None,:,:,:])
    features = activations[-3]                      # Extracting activations from the third last layer
    if len(features.shape) > 2:
        features = np.mean(features, axis=(1, 2))  # Average over the spatial dimensions
    features = features.reshape(-1)
    return features
```

“I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.”