Politecnico di Torino

1859

AML

Exercises

Advanced Machine Learning

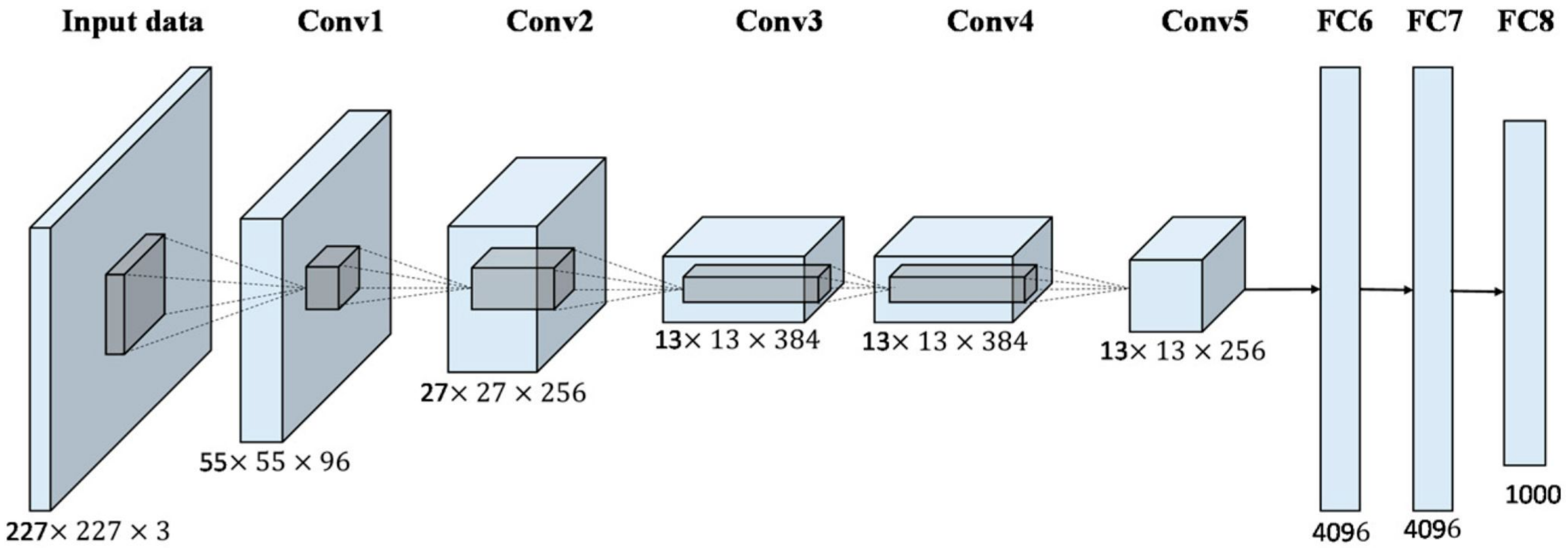
Teaching Assistant:
Claudia Cuttano
claudia.cuttano@polito.it

Overview

1. Train a **C**onvolution **N**eural **N**etwork for image classification:
    - CNN: **AlexNet**
    - Images: **Caltech-101**

2. Exercise **Steps**:
    - Before you start
    - Data preparation
    - Train from scratch
    - Transfer Learning
    - Data Augmentation
    - Beyond AlexNet

AlexNet (link)



Caltech-101 (link)

- 101 object categories:
  - Chair
  - Elephant
  - …
- Additional background category
- 9146 images
- from 40 to 800 images for category

Training resources

Deep Learning requires **GPU** acceleration ⚠️

Training resources

Deep Learning requires **GPU** acceleration

You can use Google Colab!

Code templates

1. The template of the main code is available here:
    - Save a copy on your own drive: “File” -> “Save a copy in Drive”
    - Switch to GPU acceleration: “Edit” -> “Notebook Settings”

2. Caltech-101 and the dataset class template are available via this GitHub repository.

Step 0: Before you start

Before you start

1. Study code and data:
    a. Read carefully the template code (including the comments) to understand how everything is done. Pay extra attention to: Preprocessing, Datasets, Training, Testing
    b. Explore the data provided on the GitHub repository.

2. Run the code:
    a. training should take less than 10 minutes
    b. you have to stay connected

Step 1: Data Preparation

Step 1: Data Preparation

1. Create your own dataset class for Caltech-101, following the code provided in the GitHub repository (`caltech_dataset.py`):
   a. train-test split is already provided in the github repository: `test.txt` and `train.txt` contain the relative paths for all the images they include
   b. there is also a BACKGROUND folder that you are required to filter out
   c. the class should read and store only the images belonging to the corresponding split (see the `split` parameter of the Caltech class)

Step 2: Train from scratch

Step 2: Train from scratch

During training, we need a **validation set** for hyperparameter tuning and model selection.

In deep learning, it is often **prohibitive** to perform a **K-fold Cross Validation** since training takes a lot of time.

The most common procedure is the hold-out, having <u>**one single validation set**</u> both for hyperparameter tuning and model selection.

Step 2: Train from scratch

1. **Split** the training set in **training** and **validation sets**:
   a. The validation may, for example, be 25% the size of the training set
   b. Be careful not to filter out entire classes from the sets!
      Train and validation must be balanced: aim for half samples of each class in train and the other half in val
      hint: the "train_test_split" function from sklearn may be helpful for stratified sampling
2. Implement **model selection** with the validation set:
   a. After each training epoch, evaluate (= test) the model on the validation set
   b. As you test the model after each epoch, save a snapshot of the best-performing model (=network parameters) so far
   c. Use only the best performing model from the previous step
      for the final evaluation on the test

Step 2: Train from scratch

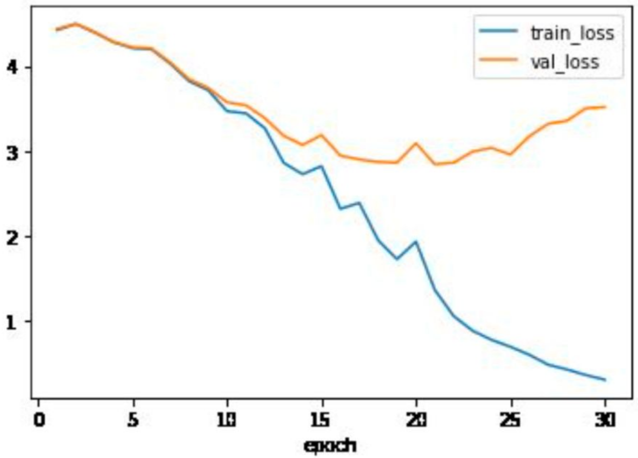
The current implementation trains using SGD with momentum for 30 epochs with an initial learning rate (**LR**) of 0.001 and a decaying policy (STEP_SIZE) after 20 epochs.

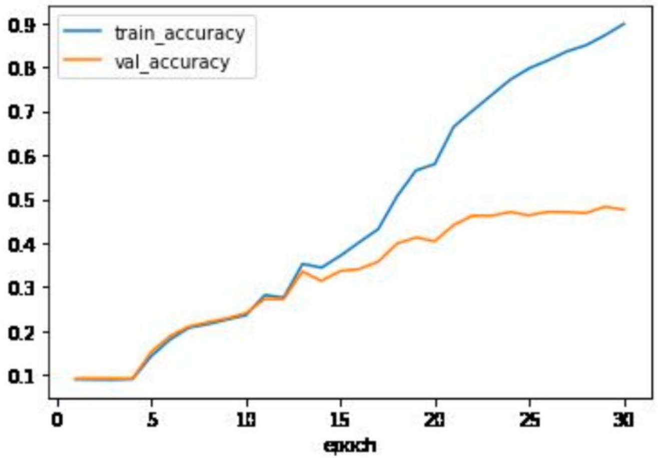
3. Experiment with **at least two different sets of hyperparameters**:

    a. The first hyperparameter to optimize is the learning rate
    b. Low learning rate = the model converges too slowly (loss decreases slowly)
    c. High learning rate = the model converges fast but accuracy is sub-optimal
    d. Too high learning rate = the model diverges (loss increases)

suggestion: experiment changing LR and one among: decaying policy, optimiser, epochs)

Step 2: Train from scratch





Step 3: Transfer Learning

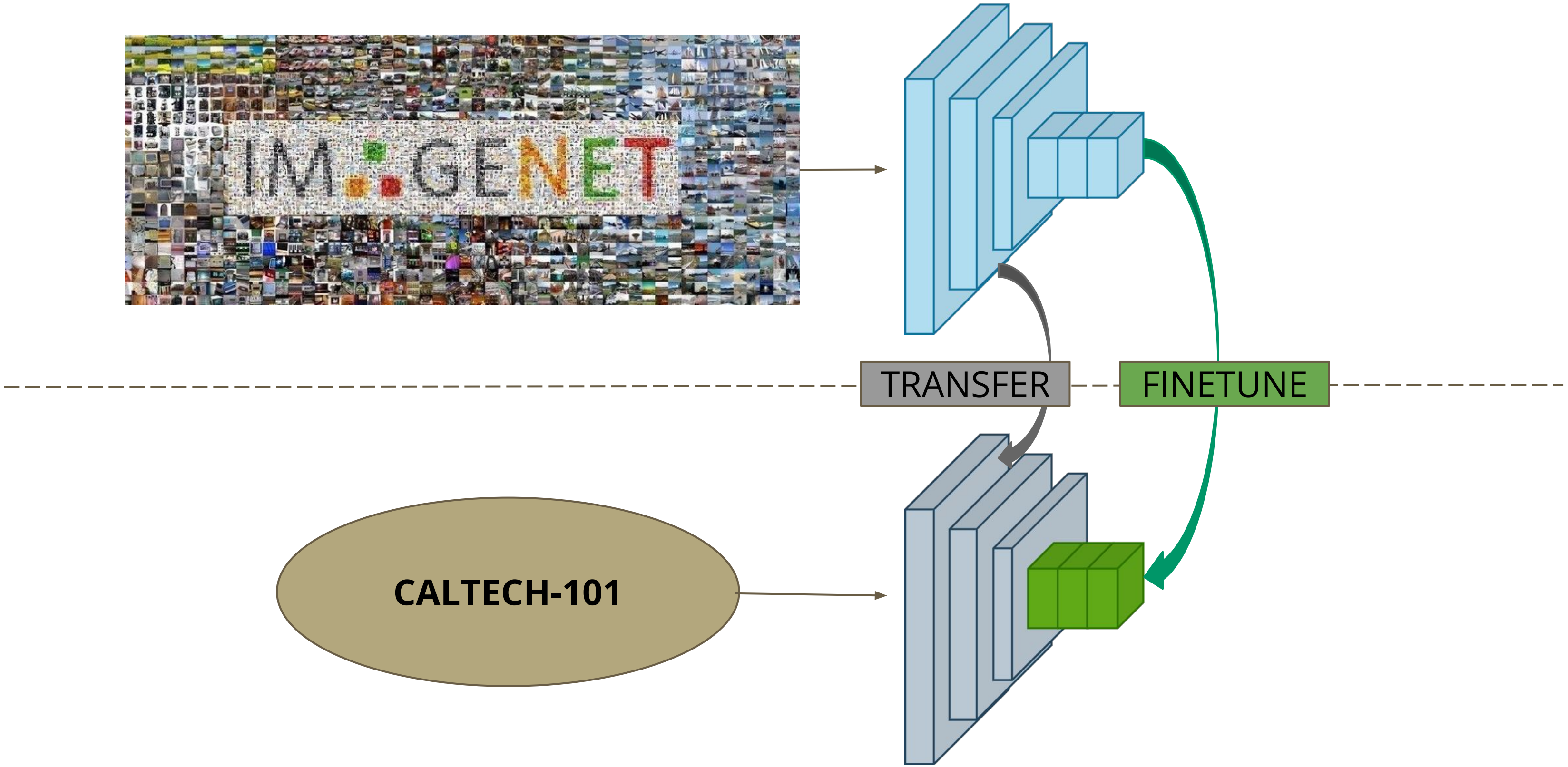
Step 3: Transfer Learning

Deep Learning needs very large datasets to train good features but **Caltech-101 is very small dataset.**

<u>Solution 1</u>: **Transfer Learning**

- Use the weights learned by training on a large related dataset as a starting point for training on the small dataset
- We can also choose to “freeze” part of the network and only train certain layers

Step 3: Transfer Learning



Step 3: Transfer Learning

<u>Solution 1: **Transfer Learning**</u>

1. Load AlexNet with weights trained on the **ImageNet** dataset.
2. Change the **Normalize** function of Data Preprocessing to Normalize using ImageNet’s mean and standard deviation
3. Run experiments with at least **three different sets** of hyperparameters
4. Experiment by training only the fully connected layers (freeze other layers)
5. Experiment by training only the convolutional layers:
    a. Compare all results you obtained, reasoning about what does it mean logically to freeze some layers

Step 4: Data Augmentation

Step 4: Data Augmentation

brightness

horizontal flipping

blur

Step 4: Data Augmentation

Deep Learning needs very large datasets to train good features but **Caltech-101 is very small dataset.**

Solution 2: **Data Augmentation**

- Artificially increase the dataset size by applying some transformations at training time, preserving the label
- Transformations applied should be the ones we are expected to see at test time, otherwise the overall accuracy could be negatively affected

Step 4: Data Augmentation

Solution 2: **Data Augmentation**

1. Apply at least three different sets of preprocessing for training images:
   a. link
   b. no need to use complex transformations, try with the simple ones, such as **flipping** or changing the **brightness** and combine them
   c. if test data is very similar to training data, some data augmentation policies may worsen accuracy
   d. if the loss keeps decreasing, increase training epochs or increase learning rate

2. Compare the results

Step 5: Beyond AlexNet

Step 5: Beyond AlexNet

1. Try with different models (e.g. **VGG**, **ResNet**)
    a. check their requirements in terms of:
        i. input image size
        ii. GPU memory consumed (you may need to significantly reduce batch size)

2. Compare the results

Now it's your turn, try!