

This project covers the following topics:

- Downloading an image dataset from web URL
- Understanding convolution and pooling layers
- Creating a convolutional neural network (CNN) using PyTorch
- Training a CNN from scratch and monitoring performance
- Underfitting, overfitting and how to overcome them

How to run the code

Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Colab**. [Google Colab](https://colab.research.google.com/) (<https://colab.research.google.com/>), is a free online platform for running Jupyter notebooks using Google's cloud infrastructure. You can also select "Run on Binder" or "Run on Kaggle" if you face issues running the notebook on Google Colab.

Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up [Python](https://www.python.org/) (<https://www.python.org/>), download the notebook and install the required libraries. We recommend using the [Conda](https://docs.conda.io/projects/conda/en/latest/user-guide/install/) (<https://docs.conda.io/projects/conda/en/latest/user-guide/install/>) distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Using a GPU for faster training

You can use a [Graphics Processing Unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU) to train your models faster if your execution platform is connected to a GPU manufactured by NVIDIA. Follow these instructions to use a GPU on the platform of your choice:

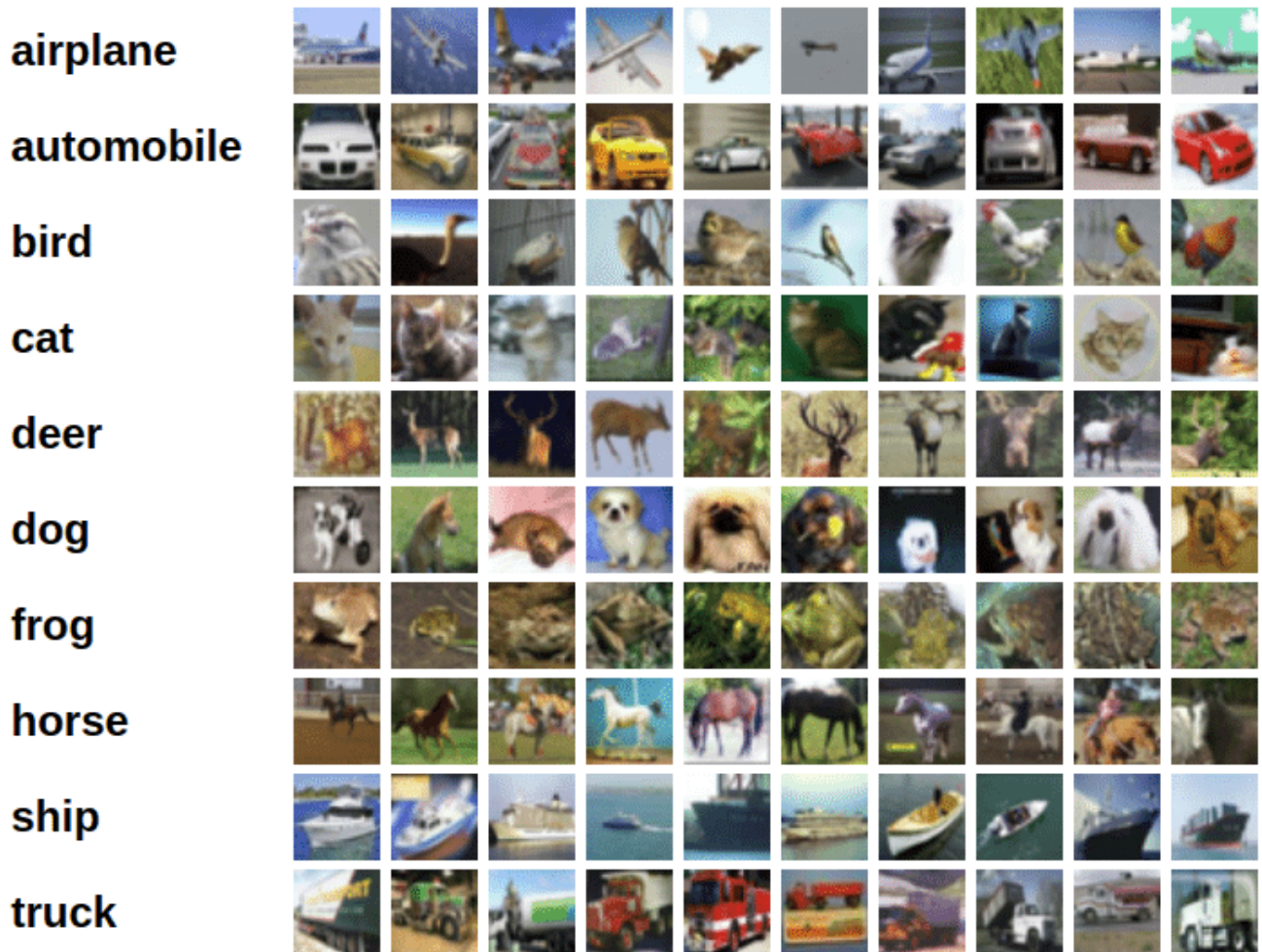
- *Google Colab*: Use the menu option "Runtime > Change Runtime Type" and select "GPU" from the "Hardware Accelerator" dropdown.
- *Kaggle*: In the "Settings" section of the sidebar, select "GPU" from the "Accelerator" dropdown. Use the button on the top-right to open the sidebar.
- *Binder*: Notebooks running on Binder cannot use a GPU, as the machines powering Binder aren't connected to any GPUs.

- *Linux*: If your laptop/desktop has an NVIDIA GPU (graphics card), make sure you have installed the NVIDIA CUDA drivers (<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>).
- *Windows*: If your laptop/desktop has an NVIDIA GPU (graphics card), make sure you have installed the NVIDIA CUDA drivers (<https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>).
- *macOS*: macOS is not compatible with NVIDIA GPUs

If you do not have access to a GPU or aren't sure what it is, don't worry, you can execute all the code in this project just fine without a GPU.

Exploring the CIFAR10 Dataset

For this project, we'll use the CIFAR10 dataset, which consists of 60000 32x32 px colour images in 10 classes. Here are some sample images from the dataset:



In [1]

```
# Uncomment and run the appropriate command for your operating system,
# if required

# Linux / Binder / Windows (No GPU)
# !pip install numpy matplotlib torch==1.7.0+cpu \
# torchvision==0.8.1+cpu torchaudio==0.7.0 -f
# https://download.pytorch.org/whl/torch_stable.html

# Linux / Windows (GPU)
# pip install torch==1.7.1+cu110 torchvision==0.8.2+cu110 \
# torchaudio==0.7.2 -f https://download.pytorch.org/whl/torch_stable.html

# MacOS (NO GPU)
# !pip install numpy matplotlib torch torchvision torchaudio
```

In [2]

```
import os
import torch
import torchvision
import tarfile
from torchvision.datasets.utils import download_url
from torch.utils.data import random_split
```


We'll download the images in PNG format from [this page](https://course.fast.ai/datasets) (<https://course.fast.ai/datasets>), using some helper functions from the `torchvision` and `tarfile` packages.

In [3]

```
# Dowload the dataset
url = "https://s3.amazonaws.com/fast-ai-imageclas/cifar10.tgz"
download_url(url, '.')
```

Downloading <https://s3.amazonaws.com/fast-ai-imageclas/cifar10.tgz>

(<https://s3.amazonaws.com/fast-ai-imageclas/cifar10.tgz>) to `./cifar10.tgz`

96%  129138688/135107811 [00:03<00:00, 47171772.38it/s]

In [4]

```
# Extract from archive
with tarfile.open('./cifar10.tgz', 'r:gz') as tar:
    tar.extractall(path='./data')
```

The dataset is extracted to the directory `data/cifar10`. It contains 2 folders `train` and `test`, containing the training set (50000 images) and test set (10000 images) respectively. Each of them contains 10 folders, one for each class of images. Let's verify this using `os.listdir`.

In [5]

```
data_dir = './data/cifar10'

print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)
```

```
['train', 'test']
['ship', 'bird', 'frog', 'truck', 'automobile', 'deer', 'cat', 'dog', 'horse', 'airplane']
```

Let's look inside a couple of folders, one from the training set and another from the test set. As an exercise, you can verify that there are an equal number of images for each class, 5000 in the training set and 1000 in the test set.

In [6]

```
airplane_files = os.listdir(data_dir + "/train/airplane")
print('No. of training examples for airplanes:', len(airplane_files))
print(airplane_files[:5])
```

```
No. of training examples for airplanes: 5000
['4320.png', '2018.png', '3851.png', '1405.png', '4675.png']
```

In [7]

```
ship_test_files = os.listdir(data_dir + "/test/ship")
print("No. of test examples for ship:", len(ship_test_files))
print(ship_test_files[:5])
```

```
No. of test examples for ship: 1000
['0190.png', '0874.png', '0733.png', '0086.png', '0296.png']
```

The above directory structure (one folder per class) is used by many computer vision datasets, and most deep learning libraries provide utilities for working with such datasets. We can use the `ImageFolder` class from `torchvision` to load the data as PyTorch tensors.

In [8]

```
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
```

In [9]

```
dataset = ImageFolder(data_dir+'/train', transform=ToTensor())
```

Let's look at a sample element from the training dataset. Each element is a tuple, containing a image tensor and a label. Since the data consists of 32x32 px color images with 3 channels (RGB), each image tensor has the shape (3, 32, 32) .

In [10]

```
img, label = dataset[0]
print(img.shape, label)
img
```

torch.Size([3, 32, 32]) 0

Out [10]

```

tensor([[[0.7922, 0.7922, 0.8000, ..., 0.8118, 0.8039, 0.7961],
         [0.8078, 0.8078, 0.8118, ..., 0.8235, 0.8157, 0.8078],
         [0.8235, 0.8275, 0.8314, ..., 0.8392, 0.8314, 0.8235],
         ...,
         [0.8549, 0.8235, 0.7608, ..., 0.9529, 0.9569, 0.9529],
         [0.8588, 0.8510, 0.8471, ..., 0.9451, 0.9451, 0.9451],
         [0.8510, 0.8471, 0.8510, ..., 0.9373, 0.9373, 0.9412]]],

        [[0.8000, 0.8000, 0.8078, ..., 0.8157, 0.8078, 0.8000],
         [0.8157, 0.8157, 0.8196, ..., 0.8275, 0.8196, 0.8118],
         [0.8314, 0.8353, 0.8392, ..., 0.8392, 0.8353, 0.8275],
         ...,
         [0.8510, 0.8196, 0.7608, ..., 0.9490, 0.9490, 0.9529],
         [0.8549, 0.8471, 0.8471, ..., 0.9412, 0.9412, 0.9412],
         [0.8471, 0.8431, 0.8471, ..., 0.9333, 0.9333, 0.9333]]],

        [[0.7804, 0.7804, 0.7882, ..., 0.7843, 0.7804, 0.7765],
         [0.7961, 0.7961, 0.8000, ..., 0.8039, 0.7961, 0.7882],
         [0.8118, 0.8157, 0.8235, ..., 0.8235, 0.8157, 0.8078],
         ...,
         [0.8706, 0.8392, 0.7765, ..., 0.9686, 0.9686, 0.9686],
         [0.8745, 0.8667, 0.8627, ..., 0.9608, 0.9608, 0.9608],
         [0.8667, 0.8627, 0.8667, ..., 0.9529, 0.9529, 0.9529]]])

```

The list of classes is stored in the `.classes` property of the dataset. The numeric label for each element corresponds to index of the element's label in the list of classes.

In [11]

```
print(dataset.classes)
```

```
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

We can view the image using `matplotlib`, but we need to change the tensor dimensions to `(32,32,3)`. Let's create a helper function to display an image and its label.

In [12]

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

In [13]

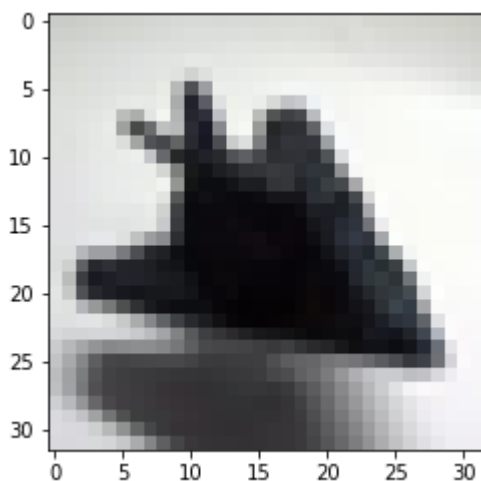
```
def show_example(img, label):
    print('Label: ', dataset.classes[label], "("+str(label)+")")
    plt.imshow(img.permute(1, 2, 0))
```

Let's look at a couple of images from the dataset. As you can tell, the 32x32px images are quite difficult to identify, even for the human eye. Try changing the indices below to view different images.

In [14]

```
show_example(*dataset[0])
```

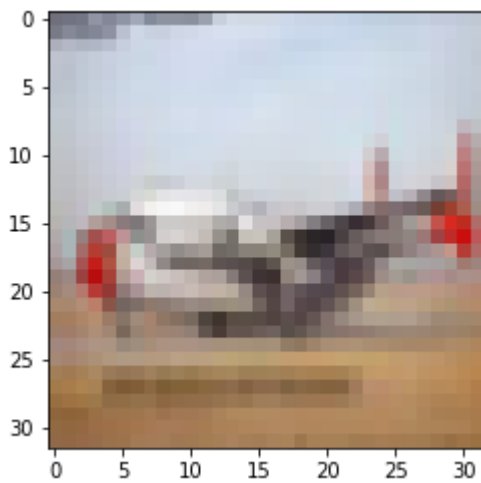
Label: airplane (0)



In [15]

```
show_example(*dataset[1099])
```

Label: airplane (0)



Training and Validation Datasets

While building real world machine learning models, it is quite common to split the dataset into 3 parts:

1. **Training set** - used to train the model i.e. compute the loss and adjust the weights of the model using gradient descent.
2. **Validation set** - used to evaluate the model while training, adjust hyperparameters (learning rate etc.) and pick the best version of the model.
3. **Test set** - used to compare different models, or different types of modeling approaches, and report the final accuracy of the model.

Since there's no predefined validation set, we can set aside a small portion (5000 images) of the training set to be used as the validation set. We'll use the `random_split` helper method from PyTorch to do this. To ensure that we always create the same validation set, we'll also set a seed for the random number generator.

In [16]

```
random_seed = 42
torch.manual_seed(random_seed);
```

In [17]

```
val_size = 5000
train_size = len(dataset) - val_size

train_ds, val_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(val_ds)
```


Out [17]

```
(45000, 5000)
```

We can now create data loaders for training and validation, to load the data in batches

In [18]

```
from torch.utils.data.data_loader import DataLoader

batch_size=128
```

In [19]

```
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
```

We can look at batches of images from the dataset using the `make_grid` method from `torchvision`. Each time the following code is run, we get a different batch, since the sampler shuffles the indices before creating batches.

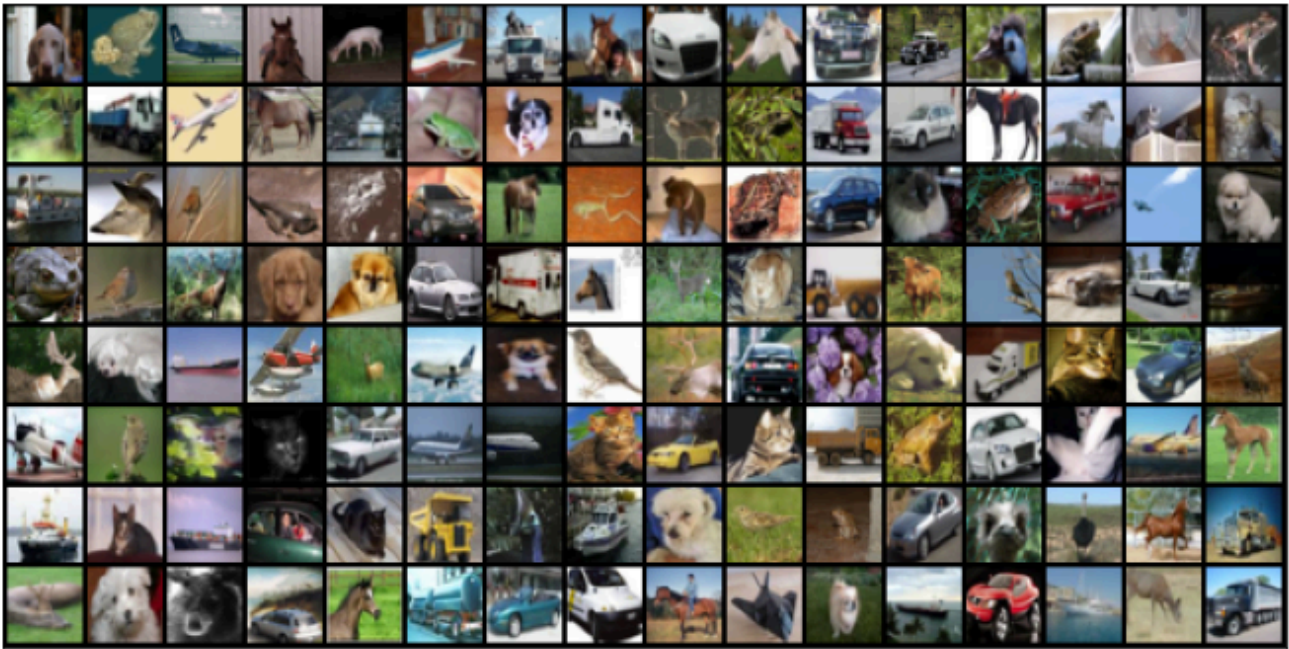
In [20]

```
from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break
```

In [21]

```
show_batch(train_dl)
```



Your Tasks

0. Recommendation is to use PyTorch for this project. You are also allowed to use TensorFlow.
1. Define your image classification model with convolutional layers (using the `nn.Conv2d` class from PyTorch).
2. Choose between GPU (for faster training) or CPU to train your model.
3. Plot the model's validation accuracy curve for each epoch.
4. Plot training and validation losses curve to analyze trends.
5. Test your trained model's performance on three selected images.
6. Save your model and reloading it.
7. Put your model design and all your experimental results to the project report in PDF format.

Submission

The **pdf** version of your project report and the source code are required to be submitted to Canvas. Please package and zip all your files as project-<NetID>.zip. For example, if your NetID is "my390", please name your file as "project-my390.zip". **For your report, please make sure the file format is pdf. Otherwise, the report will not be reviewed.** For the source code, please submit all the completed files.

Collaboration Policy

You are to complete this assignment **individually**. However, you may discuss the general algorithms and ideas with classmates. We require you to:

- not explicitly tell each other your code and report
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied