# Assignment 3 (Part I): Learning with Neural Networks

This is part 1 of the assignment. We will release part 2 in coming days and appropriate time will be given for same.

## Introduction

This assignment involves solving a practical machine learning problem. We will gain experience with the iterative process of designing and validating a neural network architecture for a prediction problem using a popular deep learning library (PyTorch). A machine learning task involves making predictions from an input data source. In this assignment we will work with visual data in the form of images. Our focus will primarily be on thoroughly understanding the basic deep learning practices/techniques and a bit less on the modality (image) specific techniques (which can be explored in advanced courses). This assignment will provide few pointers on designing good models. However, additional reading and experimentation from the student's side will be helpful. *Please start the assignment early!*

## Problem Statement

We are given a data set consisting of images of different bird species with a total of K = 10 bird species present in the data set. Each species having between 500 and 1,200 images in the data set and each image containing a type single bird only. Please design a neural network that takes as input a bird image and predicts the class label (one of the K possible labels) corresponding to each image. The next section will provide some hints on designing and validating your model.

## Model Design Guidelines

## Data Prepration

- To prepare the dataset for training, begin by splitting the images into **training** and **validation** sets. A common practice is to use an **80-20 split**, where 80% of the dataset is allocated for training and the remaining 20% is reserved for validation, simulating the model's behaviour during test time. This is not a strict rule and 70-30 or 60-40 splits are also used.
- Create a **data loader** to handle loading the images for training and validation. The data loader will read the images, create batches, and apply any necessary preprocessing steps, such as resizing and normalization, to meet the input requirements of the model. You can start by implementing the split using scikit-learn's `train_test_split`, and then proceed to build the data loader using PyTorch's `DataLoader`, incorporating any preprocessing steps as needed.

## Network Layers

- A basic CNN architecture typically consists of a series of **convolutional layers** that extract features from the image, followed by **non-linearity** (usually ReLU activations) to introduce non-linear behaviour. Use of **pooling layers** introduces spatial invariance and should be explored. Such sequence of layers, help the network learn important features from images that can aid in the classification task.
- After the feature extraction, the output is passed through a **fully connected layer** (also known as a Multi-Layer Perceptron (MLP) denoted as `nn.Linear` in PyTorch.
- Finally, the network needs to predict the class label, using an activation such as *softmax*. Though the final layer may conceptually involve applying **softmax** to generate class probabilities, PyTorch's built-in loss functions, such as `CrossEntropyLoss`, automatically handle the *softmax* internally, so you don't need to apply it explicitly in the model.
- Please keep track of the inputs and outputs resulting from these layers as they help in debugging. Experiments are required for designing the internal layers of your model.

## Loss Function

- Implement a loss function for multi-class classification. See the **categorical cross-entropy**. In PyTorch, you can implement this using `nn.CrossEntropyLoss` for multi-class problems.
- Note that each class may have varied number of examples in the data set and some class fewer samples as compared to others. This is classed as class-imbalance and is commonly encountered in real world applications. To address this, you can give more weight to the underrepresented classes during training, penalizing the model more when it makes errors on these underrepresented classes. This can be done by assigning class weights inversely proportional to the number of samples per class, helping the model focus on minority classes and improve overall performance. You can refer this: link

## Optimisation

- The learning rate has a significant impact on model training. Please try scheduling the learning rate or performing grid search to find good values.
- In class we covered Stochastic Gradient Descent (SGD) for optimisation. Other techniques have also been developed that improve upon basic SGD. For example, a popular optimizer, **Adam** optimizer adjusts the learning rate based on parameter properties (and additionally uses momentum). We suggest starting out with Adam as the optimizer. Other optimizers may also be tried, however, Adam usually provides good results.
- Data Augmentation is process of generating new data samples from the existing dataset to improve model **generalizability**. You can try image rotation, horizontal flip, vertical flip to make model robust. Kindly restrict yourself to using PyTorch based transforms. You can refer PyTorch resources for the available data augmentation methods: here.

## Regularisation

- As the number of parameters in the model increases, the risk of overfitting also rises necessitating the use of regularising the model. Early stopping is one such technique where, one monitors the performance of the model on a validation set and halts training once the performance stops improving,

preventing overfitting by not overtraining the model. Evaluate the impact of early stopping on your model.

- Other regularization techniques that you may try are:
  - **Dropout [2]**: Dropout randomly "drops" (sets to zero) a subset of neurons during training, forcing the network to learn more robust and generalized features.
  - **Batch Normalization** [1]: This technique normalizes the input to each layer in a neural network, helping to stabilize and speed up training. It also acts as a regularizer, reducing the need for other regularization techniques.

## Visualising Class Activation Maps

- To gain insights into which features the model is learning, we can apply **Grad-CAM (Gradient-weighted Class Activation Mapping)**. Grad-CAM helps visualize the important regions in an image that contribute to the model's decision for the predicted class by highlighting areas the model focuses on. Once the model is trained, apply Grad-CAM to see which regions of the image are most influential for each predicted class.
- One of the most popular implementation is : https://github.com/jacobgil/pytorch-grad-cam

# Training and Evaluation process

**Training Code**

- bird.py should contain the all the main code for train and testing the model. We will run bird.py for training and testing in the following manner.
- Do  not change this format.

```
# Running code for training. save the model in the same direc
python bird.py path_to_dataset train bird.pth
```

```
# Running code for inference
python bird.py path_to_dataset test bird.pth
```

**Model saving and loading:**

- After training, your code should save the model in the same directory with name **bird.pth.**
- There is no need to do hyper-parameter search during the final training. You need to submit the model with best hyper-parameters.
- You should follow the following method for loading and saving the model : https://pytorch.org/tutorials/beginner/saving_loading_models.html

```
# save model
PATH = save_model_path
torch.save(model.state_dict(), PATH)

#Load model
PATH =load_model_path
model = ModelClass()
model.load_state_dict(torch.load(PATH))
model.eval()
```

**Testing**

```
# Running code for inference
python bird.py path_to_dataset test model_load_path
```

- **During testing, your code should produce a bird.csv file with predicted label for each image.** You can use the following code to do the same. Keep column head as Predicted_Label.

```
#ensure that shuffle is False
batch_size = 1
test_loader = DataLoader(test_dataset, batch_size=batch_size, sh
```

```
def test_model(model, test_loader, criterion, device, output_csv
    model.eval()
    total = 0
        results = []
    with torch.no_grad():
        for images, _ in test_loader:
            images = images.to(device)

            # Forward pass
            outputs = model(images)
            # Get predictions
            # Save predicted labels
            results.extend(predicted.cpu().numpy())

    # Write only predicted labels to a CSV file
    with open(output_csv, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['Predicted_Label'])
        for label in results:
            writer.writerow([label])
```

- Ensure that you have put shuffle in DataLoader to false for during testing.
- We will evaluate the performance of the model on held-out test set consisting same number of classes but can have different distribution of image samples in each class.
- We will use average of ‘Macro’ and ‘Micro’ F1 score to evaluate the performance of your model. You can look here for reference **Link**

```
from sklearn.metrics import f1_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
f1_macro = f1_score(y_true, y_pred, average='macro')
f1_micro = f1_score(y_true, y_pred, average='micro')
```

```
final = (f1_macro + f1_micro)/2
```

# Implementation Guidelines

- We encourage the use of **Convolutional Neural Networks (CNNs)** for this task. The use of **pre-trained features** from ResNet, AlexNet or Vision Transformers is **not** permitted (since our focus in on basic deep learning in this course). Similarly, direct import / use of existing implementation of common models such as ResNet, AlexNet etc. is **not** permitted.
- Training time : We expect the training time for your model to not exceed 2 hours.
- Model size: You should be able to get a good performance with 30 million parameters.  Feel free to try varying number of layers, but ensure that the model size should not exceed 80 million parameters.

```
# Calculate the number of parameters
num_params = sum(p.numel() for p in model.parameters())
print(f"Total number of parameters: {num_params}")
```

- Set the seed  to following to ensure that your model gets the same performance of our system.

```
import torch
torch.manual_seed(0)
```

- Your submission will include the trained network (Uploaded on your google drive and link shared with us) and the PyTorch implementation. Your model will be evaluated for accuracy on a test set using the network provided and the network that gets trained. The latter is for reproduce-ability, an increasingly important requirement in both industry and research.
- When submitting your work, please include a report (1-3 pages) that includes the following:

- A diagram illustrating your model (or a table that specifies your architecture).
- **Train and Validation Loss vs. Epochs**: Plot the training and validation loss over the epochs to demonstrate how the model's learning progresses over time.
- **Train and Validation Accuracy vs. Epochs**: Plot the training and validation accuracy across epochs to show how well the model is performing on both the training and validation sets.
- **Effect of Model optimisation** : Create a table detailing the **validation accuracy** for data augmentation, regularization technique used (e.g., L2 regularization, dropout), and discuss how these techniques impacted the model's performance.
- **Class Activation Maps (CAM)**: For each class, provide visualizations of the **Class Activation Maps** and explain your observations. Discuss how the model interprets different regions of the image for each class and what insights you can draw from the highlighted areas.

# Starter Code

## Getting Started

Started code is available on the Moodle and have following structure:

```
A3
|- enviroment.yaml #contains all the required libraries
|- install.sh # install all the dependencies.
|- bird.py # Add all your code here.
|- run.ipynb # Contains instructions to run the code.
|- Dataset # Download from kaggle
```

> Check run.ipynb to setup the environment and dataset.

## Development Environment

You can develop your solution either using your local setup or Kaggle. We recommend using Kaggle as it provide GPUs that can be helpful in model training.

- Local setup: You can create a Python environment on your machine with `conda` using the command `conda env create -f environment.yml` . Here, `environment.yml` file is available with the starter code.
- You can setup kaggle following the instructions here:

Dataset link is provided in the document below. You can directly import the dataset to Kaggle using name **Identify-the-Birds**

instructionsA3.pdf

# Submission Guidelines

- **This assignment is to be done individually or in pairs. The choice is entirely yours.**
- **The assignment is intended for both COL333 and COL671 students.** If the assignment is being done in pairs then the partner is to be selected only within your class. That is COL333 students should be paired within COL333 and similarly masters students in COL671 should pair within their class of COL671.
- You should upload the your model in google drive and make use you make the file accessible to anyone with the link. Create a txt file with name model_path.txt and in first line put the link of your best model. Ensure that there is not modification after the deadline.
- Please include the report (based on the description provided above). The first line of the report should list the name and entry number of student(s) who submit the assignment. Kindly restrict the report to 3 pages. The report also carries some marks.
- The assignment is to be submitted on **Gradescope**. Exactly ONE of the team members needs to make the submission. All registered students have already

been added to Gradescope. The details are given below.

  - You need to upload a single ZIP file named `"submission.zip"`. Upon unzipping, this zip should create a folder containing `bird.py`, `model_path.txt`, `group.txt` and report with name `2020CSZ1234_2020CSZ5678.pdf`. Do not rename the zip file.
  - The group.txt should include the entry numbers of all the group members (one per line). If you are working alone, simply include your own entry numbers. A sample `group.txt` is given below.

    ```
    2020CSZ1234
    2020CSZ5678
    ```

  - If you are working in pairs, you should select the partner using the "**Group Members**" option after uploading the submission in Gradescope.

- **The submission deadline is 6 PM on Wednessday, November 6th , 2024.**
- This assignment (part I + part II) will carry $12(\pm 3)\%$ of the grade.

## Other Instructions

- Late submission deduction of (10% per day) will be awarded. Late submissions will be accepted till 2 days after the submission deadline. There are no buffer days. Hence, please submit by the submission date.
- Please follow the assignment guidelines. Failure to do so would cause exclusion from assessment and award of a reduction (at least 10%). Please strictly **adhere** to the input/output syntax specified in the assignment as the submissions are processed using scripts. Failure to do so stall evaluations and takes away time from other course activities.
- Please do not modify files beyond the files you are supposed to modify. Failure to do so would cause exclusion from assessment/reduction.
- Queries (if any) should be raised on **Piazza**. Please keep track of Piazza posts. Any updates to the assignment statement will be discussed over Piazza. Please do not use email or message on Teams for assignment queries.

- **Please only submit work from your own efforts.** Do not look at or refer to code written by anyone else. You may discuss the problem, however the code implementation must be original. Discussion will not be grounds to justify software plagiarism. Please do not copy existing assignment solutions from the internet or taken from past submission or submissions from other students; your submission will be compared against them using plagiarism detection software. Violations to the honour code will result in deductions as per course and institute policies.

# References

Primary reference are the lecture notes. The Russell and Norvig AIMA text book has related material in Chapter 18 (third edition) and Chapter 22 (fourth edition) respectively.

[1]: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

[2]: Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[3]: Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

PyTorch Tutorials: https://pytorch.org/tutorials/

PyTorch Tutorial slides presented by the TA in class: pytorch_tutorial.pptx

Training an image classifier:
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Among deep learning specific text books, this text book (available online) is a handy reference. Note that in this course we only cover an overview of deep learning.