1. **Transfer Learning for Image Classification**[1]

   It is highly recommended that you complete this project using Keras[2] and Python.

   (a) In this problem, we are trying to build a classifier that distinguishes images of 20 bird species. You are provided with text data in twenty folders.

   (b) **Data Exploration and Pre-processing**

   i. Images in each class are given in separate folders. The file `Classes.xlsx` provides the classes assigned to the bird species images in each folder. Therefore, you encode your classes using one-hot encoding and `Classes.xlsx`.

   ii. Randomly select $\lceil 0.7n_i \rceil$ images from each folder as your training set, $\lceil 0.15n_i \rceil$ as validation set, and the rest as your test set, where n_i is the number of images in folder i and $\lceil x \rceil$ is the ceiling of x.

   iii. In order for all the images to have the same size, zero-pad or resize the images in your dataset. This can be done using various tools, including OpenCV.

   (c) **Transfer Learning**[3]

   i. When dealing with classification of relatively small image datasets, deep networks may not perform very well because of not having enough data to train them. In such cases, one usually uses *transfer learning*, which uses deep learning models that are trained on very large datasets such as `ImageNet` as feature extractors. The idea is that such deep networks have learned to extract meaningful features from an image using their layers, and those features can be used in learning other tasks. In order to do that, usually the last layer or the last few layers of the pre-trained network are removed, and the response of the layer before the removed layers to the images in the new dataset is used as a feature vector to train one more multiple replacement layers. The dataset in this task has only around 50-60 images per class. Given that we have 20 classes, training a deep network with such a small dataset may not yield desirable results. In this project, you will use pre-trained models `EfficientNetB0` and `VGG16`. For both pre-trained networks, you will only train the last fully connected layer, and will *freeze* all layers before them (i.e. we do not change their parameters during training) and use the outputs of the penultimate layer in the original pre-trained model as the features extracted from each image.

   ii. To perform empirical regularization, crop, randomly zoo, rotate, flip, contrast, and translate images in your training set for image augmentation. You can use various tools to do this, including OpenCV.

   iii. Use ReLU activation functions in the last layer and a softmax layer, along with batch normalization [4] and a dropout rate of 20% as well as ADAM

[1] I acknowledge the contribution of my assistants Jerry Alan Akshay and Sunit Ashish Vaidya to preparing this project.

[2] `https://keras.io`

[3] `https://builtin.com/data-science/transfer-learning`

[4] `https://en.wikipedia.org/wiki/Batch_normalization`

optimizer. Use multinomial cross entropy loss. You can try any batch size, but a batch size of 5 seems reasonable.

iv. Train the networks (`EfficientNetB0` and `VGG16`) for at least 50 epochs (preferably 100 epochs) and perform early stopping using the validation set. Keep the network parameters that have the lowest validation error. Plot the training and validation errors vs. epochs.

v. Report Precision, Recall, and F1 score for your model. Remember that this is a multi-class classification problem.