Grace period: the project can be submitted until 11:59 PM of the same day with 30% penalty. Any change in the project after the deadline is considered late submission. One second late is late. The project is graded based on *when it was submitted, not when it was finished.* Homework late days cannot be used for the project.

1. **Identification of Frost in Martian HiRISE Images**[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 Martian terrain with frost. You can find the dataset in `https://dataverse.jpl.nasa.gov/dataset.xhtml?persistentId=doi:10.48577/jpl.QJ9PYA`.

   This dataset was created to study Mars' seasonal frost cycle and its role in the planet's climate and surface evolution over the past 2 billion years. The data helps in identifying low-latitude frosted microclimates and their impact on climate.

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

   i. Images (png files) and labels (json files) are organized in the data directory by "subframes." Subframes are individual 5120x5120 pixel images which are crops of the original HiRISE images (often on the order of 50k x 10k pixels). Individual subframes were annotated by the contributors and then sliced into 299x299 "tiles." Each tile has an associated label for use in training ML algorithms.

   There are 214 subframes and a total of 119920 tiles. Each tile has annotations which have been used to assign labels to the tiles 'frost' or 'background.' Each JSON file contains all the annotation information collected from human annotators.

   The following are relevant to the assignment:

   Image tiles are organized into folders of 'background' and 'frost' classes (binary). For the purpose of the final project, individual tiles shall serve as the data points which need to be classified using binary classification.

   ii. The dataset includes files for splitting the data into train, test and validation. However, you will be provided by an improved version of those files when a repo is created:

   A. `train_source_images.txt`

   B. `test_source_images.txt`

   C. `val_source_images.txt`

   iii. Each of these files contains the IDs of the high rise images (parent folders for the subframes and tiles).

   (c) **Training CNN + MLP**

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

[1]I acknowledge the contribution of my assistants Vibhav Shashank Deshpande, Tarunbir Gambhir, and Kirthika Gurumurthy to preparing this project.

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

ii. Train a three-layer CNN followed by a dense layer on the data. Choose the size of the kernels and depth of the layers and the number of neurons in the dense layer (MLP) on your own. Use ReLU's in all of the layers. Use the softmax function, batch normalization[3] and a dropout rate of 30%, L2 regularization, as well as ADAM optimizer. Use cross entropy loss. Train for at least 20 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.

iii. Report Precision, Recall, and F1 score for your model.

(d) **Transfer Learning**[4]

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. In this project, you will use pre-trained models (`EfficientNetB0`, `ResNet50`, and `VGG16`). For these 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 zoom, 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 [5] and a dropout rate of 30% as well as ADAM optimizer. Use cross entropy loss. You can try any batch size, but a batch size of 8 seems reasonable.

iv. Train using the features calculated by networks (`EfficientNetB0`, `ResNet50`, and `VGG16`) for at least 10 epochs (preferably 20 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.

vi. Compare the results of transfer learning with those of CNN + MLP model and explain them.

[3]`https://en.wikipedia.org/wiki/Batch_normalization`
[4]`https://builtin.com/data-science/transfer-learning`
[5]`https://en.wikipedia.org/wiki/Batch_normalization`