To explore **transfer learning** from Week 9 lecture and apply it in pistachio image classification

The dataset used in this notebook is the *Pistachio Image dataset* from Ozkan IA., Koklu M. and Saracogli R. (2021). Classification of Pistachio Species Using Improved K-NN Classifier. Progress in Nutrition, Vol. 23, N. 2.

The dataload comes as a folder with 3 subfolders

Pistachio_Image_Dataset folder has two subfolders named Siirt_Pistachio and Kirmizi_Pistachio

Folder name represents the labels and the files within are jpg images

Pistachio_16_Features_Dataset contains 16 handcrafted features represented as a feature vector and labels (An array of length 16 + label)

Pistachio_28_Features_Dataset has 28 handcrafted features which is assumed to include the 16 above

Data is downloaded as a zip and it is the zipped file that is read through the notebook

Once the folder is unzipped we are loading in the 28 feature dataset

The dataset, which is held as rows of vectors, is split into features and labels

Labels are just the corresponding final entry of each vector

Labels are recoded at 0 or 1

Next the images from *Pistachio_Image_Dataset* are read in

In this setup, lots of information (labels, instance id) is held in the folder and files names so this needs to be extracted

The goal is to match the images with the handcrafted vectors to which they are derived

The order of images is matched with the order of vectors in each original dataset

Data is once again split in to test, train, validate

A plot is set up to visualize the class frequencies for each set

The handcrafted vector features are then scaled/normalized

Section 4 focuses on extracting new, learned features. This is the transfer learning component

A pre-trained VGG-16 model is used which is a 16 Layer CNN

The mode architecture is loaded in initialized with a particular available set of weights

A transforms function is used which does some pre-processing to ready the data for the model

Rescaling, image resize and image center cropping

The pertained model has some layers which are called the ‘feature extraction’ layers

These layers specifically refer to the Convolution and MaxPooling layers

Or in other words, it excludes the fully connected layer

The features learned in this section are highly generalizable and are what represent the transfer learning component

The learned features are then passed onto the fully connected layer to execute the task at hand. This part is less transferable

A very simple looping function is set up to pass the images into the layers.

A class is then set up to handle the MLP classification. This does not include anything specific to the CNN code. It is just MLP stuff from previous weeks

Feedforward, evaluation metrics, SGD optimization, training, epochs,

The 3 hyperparams set up are epochs, learning rate and batch size

A class called AutoFeatsDataset is set up. This holds the pre-trained model w/ weights, takes in an image and outputs a learned featured vector of 512 params.

Note, it does not do any training. The model is already trained, we are using transfer learning to create a condensed, high-level representation of the input image, capturing the most important visual features that VGG-16 learned during its training on ImageNet.

This vector is the passed into the MLP, along with the label, and training takes places on the MLP so it can execute the task

The MLP class has a train_model function which has 5 inputs:

- The init version of the model to be trained
- Hyperparams: (Learning rate and Epochs)
- Training set auto features (CNN outputs)
- Validation set auto features (CNN) outputs)

The batch size hyperparam is used in the auto feature stage

Remember the validation set is used for evaluation

Training takes place on train set, validation evaluates the output, this metric is used to decide the best model

Section 6 works with the hand-crafted features data

Given these are already a vector, there is no CNN needed and they plug straight into the MLP

Section 7 fine-tuned a pre-trained VGG. Previously we just used the ‘offline’ weights as they were

The pre-trained VGG and the MLP will be connected as one model and trained today, in turn updating the VGG weights

In the feedforward code, self.pretrained = models.vgg16(weights="IMAGENET1K_FEATURES").features is added as a layer