CSC 202: Intro to Machine Intelligence
Image Recognition Project

The last third of the course covered machine learning methods including k-nearest neighbors and neural networks, with a primary focus on image recognition tasks. In this project, you will implement one (or more) classification methods to classify images using a dataset of your choice. You will turn in a Jupyter Notebook that carries out the appropriate analysis, as well as your trained model. Your notebook should begin with your name and a one paragraph summary describing the main goals of your project (e.g., to classify images of animals as cats or dogs). Each code cell should be preceded by a markdown cell that describes what the code does (as in the class notes). Your Notebook should be correctly formatted with an appropriate title, section headers, and subsection headers. [20 points]

1. Find and load a dataset to analyze (required). Several built-in datasets include:

- The fashion-MNIST dataset, https://github.com/zalandoresearch/fashion-mnist
    - This can be loaded from tensor flow using the following code:

```
from tensorflow.keras.datasets import fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

- The CIFAR 10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html
    - This can be loaded from tensor flow using the following code:

```
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

When displaying the images, use a small size to get good resolution. Note that these are color images, and therefore have 3 channels.

- Dogs vs cats, https://www.kaggle.com/chetankv/dogs-cats-images
Here you are downloading the actual images, and should use an *ImageDataGenerator* to read in the images from the appropriate directory (for an example see *https://www.kaggle.com/jsvishnuj/cats-dogs-classification-using-cnn*). Note that these are color images, and therefore have 3 channels. This is also a *binary* classification problem, since there are only 2 classes. Your final layer and loss function should be set appropriately (I can go over this with you). You will need to create an account to get this dataset.

- You may also analyze a dataset of your choice (a good source of image datasets is https://www.kaggle.com/datasets). But note that many datasets are very large (hundreds of GBs) and will be too large to use for this project. If you want help finding your own dataset, let me know!

Note: In all cases, you may choose to analyze 3 classes of images if there are more than 3 in the dataset). You may also limit your training data to include 30,000 images total. Some datasets contain images that are much more difficult to classify correctly than the MNIST dataset used in class. Your grade will not be based on classification accuracy, but you are encouraged to experiment with more complex structures (such as two convolutional layers) than what we have used in order to get good performance. A convolutional layer must always be followed by a max pooling layer; the output layer must be appropriate for your number of classes.

2. For both the training and testing sets, use Python to output the size of the image and the number of samples. This should generally be accomplished by looking at the *shape* of the data. [5 points]

3. Using the *imshow* method, plot the first 15 images, using 3 rows and 10 images per row. [10 points]

4. Output the number of images in each class for both the training and testing sets. How many target values do you have? [10 points]

5. Implement a neural network that predicts the image class from the image, and display the summary of the neural network. Use at least 5 epochs for training. You're your model after training, using the commands discussed previously. You can also *continue* training a model by setting the *initial_epoch* to the epoch to start with (0 by default); the *epoch*s command reflects the *final epoch*. So for example, The following code will train a *cnn* model for 5 epochs. [10 points]

```
# train for 3 epochs
history1 = cnn.fit(X_train_scaled, y_train, epochs=3, batch_size=64,
         validation_split = .1)

# continue training for 2 more epochs, continuing from the 3rd epoch
history2 = cnn.fit(X_train_scaled, y_train, epochs=5, initial_epoch = 3,
   batch_size=64, validation_split = .1)
```

6. Generate a loss curve (showing either loss or accuracy in the training dataset and optionally in the validation dataset). [10 points]

7. Optionally, you may try to optimize your neural network with respect to the structure, number of layers, or number of epochs.

8. Make predictions in the test dataset, and generate the following: [15 points]
    a. A Heatmap of the confusion matrix
    b. A Classification report that shows the recall and precision for each class

9. Create a list containing all of the test images and target values for those that were predicted correctly, and another list for all the test images that were not predicted correctly [10 points]

10. Use *imshow* to plot at least 2 images that were predicted correctly (and their correct predictions), and at least 2 images that were not predicted correctly (and their incorrect predictions). [10 points]