Algorithm: k-nearest neighbors (KNN) classifier for multi-class or binary image classification

Images are converted to same-size RGB arrays. Distance from any test image to a training image is calculated as the sum of the squares of the absolute value of the difference between each color's intensity for each pixel. Another array stores the k nearest images from the training set, and weighs their significance with an inverse square (1^{st} being 1/(1*1,) 2^{nd} being 1/(2*2), etc.) to avoid high k-values affecting the data. No scaling is performed as all images have values 0-255 for each pixel's 3 color intensities.

Datasets (all are present and organized in the GitHub repo, though have been condensed to reduce runtime):

Rock-Paper-Scissors Images:
https://www.kaggle.com/drgfreeman/rockpaperscissors
Chest X-Ray Images (Pneumonia):
https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
Face Mask Detection ~12K Images Dataset:
https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset

Selecting datasets took a bit of trial and error. Certain datasets were too large to effectively work with or presented insignificant distinguishing features for KNN to effectively classify at the given resolution within a reasonable time. As seen in the pneumonia dataset, the classification was still over 50% accurate but was not able to classify as effectively as something like the rock paper scissors dataset.

Code:

When the program is run, the number of arguments passed and their validity is first checked to make sure a valid amount is supplied, the directories exist, and the k value is an odd number. The main function is then called and stores the training and test images in a dictionary while resizing them as they are opened and added. Next, a KNN object is created since the class stores dictionaries several functions will need to access. In it, the main knn function loops through all the test images checking each one's distance against every image in the training data. The result array stores the count for predicted vs expected key values (where the indices correspond to the keys), and the process is repeated for each image in the test data. During this process, the calculation of distance is chosen to use Euclidean distance without taking the square root to improve runtime performance at the cost of storing a greater number. The k nearest values array is constructed in ascending order of distance and values are added by insertion. Lastly, to get the prediction from the k nearest points, the weights of each predicted key are calculated using the inverse square (1^{st} being 1/(1*1,) 2^{nd} being 1/(2*2), etc.). Once a single key is found as the prediction, the corresponding index for that key and the index for the true key are used to increment the number at the corresponding cell in the prediction array. Once all images are predicted with this process, a new object is created for the Stats class as common functions all make use of the same prediction array within it. Helper functions for true

positive, true negative, false positive, and false negative calculate those values for any size array, then accuracy, precision, and recall make use of those helper functions. Since f-score is not utilizing these functions, it is included outside the class and takes values for precision and recall. The print function is then called printing the values of the prediction array (showing predicted values on the left, expected values on the top, and the list of corresponding keys) and then the corresponding keys for each statistic.

Results:

Run instructions:

git clone https://github.com/onemec/CS2810-Final.git
cd CS2810-Final
pip3 install -r requirements.txt
python3 FinalProject.py rockpaperscissors 7
python3 FinalProject.py pneumonia 7
python3 FinalProject.py facemasks 7

Rock paper scissors:

	Paper (actual)	Rock (actual)	Scissors (actual)
Paper (predicted)	15	0	0
Rock (predicted)	0	14	1
Scissors (predicted)	0	1	14

Statistics:

	Accuracy	Precision	Recall	F-Score
Paper (predicted)	0.955556	1.0	1.0	1.0
Rock (predicted)	0.955556	0.933333	0.933333	0.933333
Scissors (predicted)	0.955556	0.933333	0.933333	0.933333

Pneumonia:

	Normal (actual)	Pneumonia (actual)
Normal (predicted)	11	13
Pneumonia (predicted)	68	178

Statistics:

	Accuracy	Precision	Recall	F-Score
Normal (predicted)	0.7	0.139241	0.458333	0.213592
Pneumonia (predicted)	0.7	0.931937	0.723577	0.814645

Face Masks:

	Without Mask (actual)	With Mask (actual)
Without Mask (predicted)	33	30
With Mask (predicted)	17	20

Statistics:

	Accuracy	Precision	Recall	F-Score
Without Mask (predicted)	0.53	0.66	0.52381	0.584071
With Mask (predicted)	0.53	0.4	0.540541	0.45977

Conclusion:

Overall, I was happy with the implementation of KNN I was able to produce. Due to the difficulty of classifying pneumonia, that dataset had understandably poor performance, but overall other datasets seemed rather effective (and over 50% accuracy in all cases). Had I gotten additional time, I would have hoped to test more datasets with color pertaining more to discerning classes. Additionally, I wondered what impact reducing the resolution would have, converting the image to a single greyscale array, or compressing the color space from 0-255 to something less like 0-63 would do to the statistics. Similarly, although pneumonia remains comparably simple, the shortcomings of KNN on insufficiently preprocessed data could be seen if MRI imaging was processed as benign lesions could identify as cancerous, or similarly structured brains without cancer could cause false positives. Simple issues such as rotation, varied scaling, varied viewpoints, etc. become more pertinent as this measure of similarity becomes less effective.