CMPT 318 Final Project Report: Webcams, Predictions, and Weather Jae Lee 301029487

Introduction:

The aim of this project was to train a model to accurately classify the weather as shown in input images. The accuracy of correctly classified weather will be the model's success metric. The image and the daily weather data sets are both available on the course website.

Convolutional Neural Network (CNN) has recently been most successful in visual imagery analysis and requires least amount of image feature extraction so it has been chosen as the machine learning model. Keras is an open-source Python Neural Network library that runs either a Theano or Tensorflow backend. Keras is easy-to-use and modular API made it the best choice for fast experimentation.

Data:

1. The weather dataset contained several samples missing the 'Weather' labels so so backfill and front-fill methods were used to impute the missing labels.

2. Several samples were also multi-labeled. These were temporarily set aside until the model was modified to handle multi-label classification.

3. There were several unique human-generated labels which were standardized to four labels: [clear, cloudy, rain, snow]. This was done to account for the fact that no or very few images were found for data samples with labels 'ice pellets' and 'fog'.

4. The data and time contained in the image dataset's filenames were extracted and matched to the corresponding date and time of the weather data set.

5. The correctly labeled images were then moved to four label subfolders.

6. The labeled images from the four subfolders were then split into training and validation set such that the validation set was 30% of the original dataset.

7. There was a disproportionately uneven distribution of images across the six classes. The small number of available images to train the classifier on the labels can lead to overfitting and increased generalization error. Keras offers easy-to-use APIs to generate additional input images from existing images via random transformations to fix this problem. Some of the transformations include rotating, changing the width and height,

and shearing the images. In addition, the RGB pixel values were scaled down to 0 -1 range for less computational processing.

Model:
CNNs makes the assumption of locality, where pixels in a region of an image are closely related. Effectively, only a small number of neurons in a layer are connected to the next layer so with more sparse connectivity, the learning rate is much faster than a regular neural network. Generally, a simple CNN architecture can consist of Input layer -> Convolutional layer-> Pooling Layer, and Fully-Connected Layer.

The topology of the CNN used in this project is as follows:

Input ->Convolution ->ReLU->Pooling->Convolutional->ReLU->Pooling->Flatten->FC->ReLU with Dropout ->FC output layer of 4 units with softmax activation function.
In the Keras documentation, it was recommended that the 'sigmoid' activation functions are used.

Dropout is a regularization technique to reduce overfitting. Randomly selected number of neurons are dropped during training to reduce unwanted interactions and dependencies between neurons. The output layer uses the sigmoid activation function as recommended in the Keras documentation.

ReLU is a rectifier activation function to simulate the 'firing' of neurons when the inputs exceed the threshold.

Results:

Stochastic Gradient Descent (SGD) with various learning rates from 0.003 to 0.1 and learning rate decay were tested to find an optimal learning algorithm without overshooting.

Different image sizes were tested from 32,32 pixels to 128,128 pixels.

Sigmoid and Softmax activation functions at the output layer were both tested.

The model given the current topology with the best validation accuracy was 72%.

Future Work:

The model can be improved by further parameter-tuning. The width and depth of the network topologies can be varied. Dropout and other regularization techniques can be tried at different layers. Different gradient descent algorithms like ‘adam’ can also be tested. At each stage of modification, the model’s accuracy can be validated to see if there is any improvements.

Another path could be to build on top of pre-trained network features. Keras documentation features various articles with demonstration.