EE 428: Computer Vision

Homework 4: Classification and Regression

Instructor: Jonathan Ventura

In this homework you will analyze aerial imagery of California to detect tree canopy cover and estimate tree height. You will create convolutional neural networks (CNNs) for these tasks to label each pixel in an image as containing tree canopy or not and predict the height off the ground for every pixel.

I have prepared for you a dataset of imagery from San Rafael, CA containing 4,040 256x256 images and associated binary labels (tree/not tree) and height values. (The ground truth labels and heights were created using a combination of aerial LiDAR and multispectral imagery to detect tree canopy and measure the canopy height off the ground, as part of a larger research project at Cal Poly.) Using the techniques described in class, you will create two "fully convolutional" CNNs: one for per-pixel binary classification and one for per-pixel height regression.

Coding environment

Training neural networks is computationally intensive and most of your laptops and the machines on campus don't have the required hardware (GPUs). For this reason I recommend you work on Google Colab, which provides free access to Jupyter Notebooks with GPUs and all of the necessary Python packages installed.

On Google Colab go to `Runtime > Change runtime type` and select "T4 GPU" to enable the GPUs.

Dataset

You can download the dataset from my Dropbox. I included a starter file `hw4.ipynb` that will download and extract the dataset for you.

Code requirements

Load the dataset and prepare train/test splits

1. Load the PNGs from the `image`, `label` and `height` directories into separate Numpy arrays.
2. Convert the `height` array to `float32` and divide by the maximum height value.

Note: In general (and on Google Colab especially) we run the risk of running out of memory by loading the entire dataset. Because of this, do not convert the `image` or `label` arrays to floating point -- leave them as unsigned 8-bit integer (`uint8`). We will handle type conversion and data normalization inside the neural network architecture itself.

3. Show a few examples from the `image`, `label`, and `height` arrays to make sure everything looks okay.
4. Split each array into train and test splits using from `sklearn.model_selection.train_test_split`. Make a 10% test split.

Note: Make sure you split each array in the same way! You can do this by passing all of the arrays to `train_test_split` at once.

Note: To save even more memory, you can `del` the original arrays after splitting.

Create the classification CNN

6. Create a CNN model in Keras for the binary classification problem. You are free to choose the architecture but here is a simple recipe:
   i. Input layer specifing $256 \times 256 \times 3$ input shape.
   ii. Lambda layer for data normalization: `Lambda(lambda x:x/255-0.5)`
   iii. One or more 2D convolution layers with 3x3 filters and ReLU activation; the number of filters per layer can stay constant or double at each layer.
   iiii. Final layer is a single 1x1 2D convolution and `sigmoid` activation for binary classification.
   Note: do not use `Flatten` and `Dense` layers as these are not compatible with a fully convolutional network (a CNN with per-pixel output).
7. Create the Adam optimizer with learning rate of .0004.
8. Compile the model using binary cross-entropy loss function and accuracy metric.
9. Train the model using the `image` array as input and the `label` array for the labels. Use a 10% validation split (argument `validation_split`), batch size of 32, and 10 epochs. Set `verbose=True` to see the training progress.
10. Calculate the test set accuracy for the trained model.

11. Show the result of running the model on a few test set images side-by-side with the ground truth labels.

Create the regression CNN

12. Set up the regression CNN in a similar way, but with the following changes:

- Use `linear` activation on the end of the network, rather than `sigmoid`
- Use `mean_absolute_error` (MAE) loss and no metric
- Train the model with the train split of the `height` array for the labels.

13. Calculate the test set MAE with the trained model.
14. Show the result of running the model on a few test set images side-by-side with the ground truth heights.

Report

Provide a short explanation of your solution. Be sure to document any sources you used in preparing your code, including websites and AI tools.

Discussion questions

1. Describe and analyze the performance of both models. Are you seeing underfitting or overfitting (or both)?
2. What would be your next step to try to improve the performance of your models?

Submission instructions

Submit your Python notebook (.ipynb file) and report (PDF or docx). Please do not put them in a zip file, just submit the files directly.