

Mini Project 1

Due: April 28, 2023, 11:59PM PT

*Student Name: Andy Franck**Instructor Name: John Lipor*

Problem Description

The objective of this project is to first train a Convolutional Neural Network capable of accurately categorizing geothermal heat flow residuals into one of four distinct classes. After categorizing, the model will be studied using guided backpropagation to determine whether the trained model is utilizing certain structural settings to make its predictions (i.e. changes in elevation and topography).

The model will be trained on a detrended elevation map, partitioned into 222, 200x200 pixel images. For each patch, the label is the maximum residual value among all wells in the patch. The data is provided by the United States Geological Survey [?].

Although it would be convenient for the model to accurately predict a high percentage of labels, it is unlikely to happen with the limited data size. This report aims to develop a model that can accurately predict the labels with a reasonable degree of accuracy, and then analyze the model's behavior to determine whether it is utilizing the desired structural features to make its predictions.

Even without perfect model accuracy, the model can still be useful in determining what topographic features are most useful for predicting geothermal heat flow. This will lead to a better understanding of the relationship between topography and geothermal heat flow, and will help to inform future research.

Exploratory Data Analysis

Because of the nature of the data, it was not necessary to perform much data analysis. It was easy to load directly from the files, and straightforward to work with through both the PyTorch and Numpy libraries.

To begin, the image data values were considered, and it was noted that many did not lie in the range $[0, 255]$. Because of this, the image data was normalized, which resulted in a slight 1.2% increase in accuracy after training the model. The data ranges are shown below:

Range	min	max
Before Normalization	-1037.89	2596.36
After Normalization	-0.3997	1

After normalizing all of the image data, a few different image augmentation methods were considered. Note that all data image transforms are performed inside of the training loop for each dataloader iteration, so it is difficult to determine the exact size increase for the dataset. However, because the augmentation transform was placed inside of the training loop, it was possible to train the model for more epochs without overfitting on the same exact data.

The first method of data augmentation was to randomly apply horizontal and vertical flips to the images. This was done to increase the data size, and to make the model more robust to different orientations of the same topography.

Secondly, the images were rotated a random amount between 0 and 90 degrees, in another effort to increase data size and increase robustness. Random Gaussian image noise was also applied to the images, but was removed after causing performance issues with little to no accuracy benefit.

Finally, the dataset contained some images that contained very little data. These images were negatively impacting the model's ability to learn, since they lacked the topographical structure that the model aimed to study. These images were removed from the dataset. This resulted in an average 2.3% increase in accuracy. Examples are shown below:



Challenges

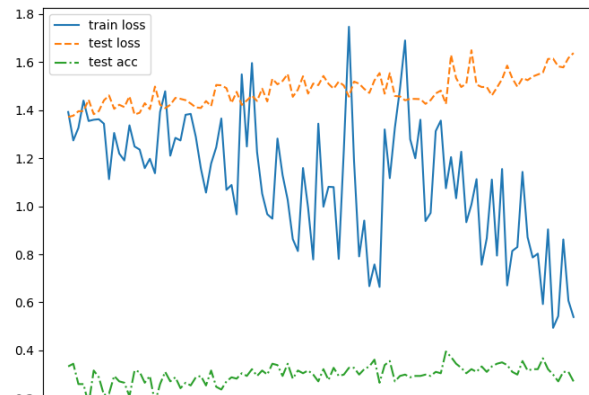
The largest challenge of the project was the limited data size. With only 222 images, it was very difficult to make a model train properly and accurately predict labels.

Additionally, the model was extremely prone to overfitting the data. To combat this, dropout was added after each Network in Network block. This helped to reduce the overfitting, however the more dropout added, the less the model seemed to learn. Unfortunately, this led to a model either overfit the data considerably, or barely learned anything at all.

Finally, the guided backpropagation analysis proved very difficult to implement. After adding hooks for the ReLU layers in the model, the analysis was subject to the "vanishing gradient" issue. This was solved by utilizing a different method to calculate saliency maps, without directly hooking the ReLU functions [?].

Approach

The initial step in development involved developing a baseline Network in Network model as described in the d2l textbook. This initial model was used to determine whether the data was being processed correctly, and to determine whether the model was capable of learning. This model was trained for 100 epochs, and overfit the data considerably:

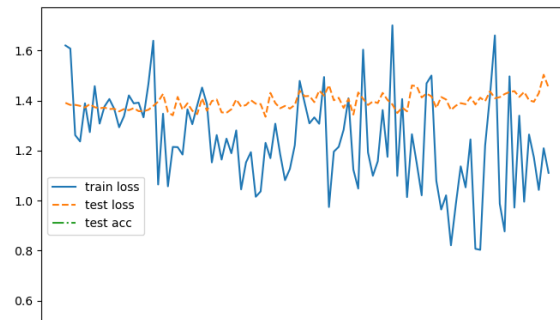


The model also did not appear to be learning very well at all. The loss function was not decreasing, and the accuracy was not increasing. This was likely due to the small data size.

To artificially increase the data size, image augmentation was utilized via the `torchvision.transforms` library. This library allows for the creation of a transform object that can be applied to a dataset. The transform object contained both random flips and rotations at random degrees. The transform was applied after loading each batch at the start of the training loop, so that each batch was augmented differently. After applying the transform, the model appeared to train slightly better, but still did not appear to be learning very well:



The model still appeared to be overfitting, so dropout was added after each Network in Network block. This



Because of the lack of data, it was decided that increasing the complexity of the model would not be beneficial. The final model appears as follows, and was trained for 100 epochs at a 0.01 learning rate:

```
self.net = nn.Sequential(
    nn_block(96, kernel_size=5, strides=3, padding=0),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nn_block(256, kernel_size=3, strides=1, padding=2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nn_block(384, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nn_block(num_classes, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten())
```

Saliency maps of the model were generated using guided backpropagation, following the tutorial [?] and [?]. Originally, a hook was added to every ReLU activation layer in the network, and then the model was run on a single image. The gradients were then calculated at the output layer with respect to the input image. However, this method resulted in many gradient values equal to zero, so a modification was made following [?] to calculate the gradients without directly hooking the ReLU layers.

Evaluation and Summary

Unfortunately, the model did not have a high accuracy. The final accuracy was 37% over all four classes, and 66% for binary classification.

What I Learned

This project was an excellent introduction to image processing using convolutional neural networks. Having

network design. In the prior project, hyperparameter optimization libraries were used that made most of the experimentation unnecessary.

Additionally, this project served as a great introduction to image specific data modification methods. The torchvision.transforms library was very useful in this regard, and it was interesting to see how the data augmentation methods affected the model's ability to learn.

Finally, this project was a great introduction to guided backpropagation. This method was very useful in determining which features the model was utilizing to make its predictions, and a great introduction to more advanced machine learning techniques. Tutorials such as [?], [?] were invaluable for developing a better understanding on hooking the model and calculating the gradients.

References

- [1] O. of Energy Efficiency and R. Energy, "Geothermal basics," 2022, <https://www.energy.gov/eere/geothermal/geothermal-basics>.
- [2] I. A. Khalid, "Saliency map for visualizing deep learning model using pytorch," 2021, <https://towardsdatascience.com/saliency-map-using-pytorch-68270fe45e80>.
- [3] K. Chung, "Guided backpropagation with pytorch and tensorflow," 2021, <https://www.coderskitchen.com/guided-backpropagation-with-pytorch-and-tensorflow/>.