EE 519: Deep Learning Theory & Fundamentals **Spring 2023**

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 disctinct classes. After categorizing, the model will be studied using guided backpropogation 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 [1].

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

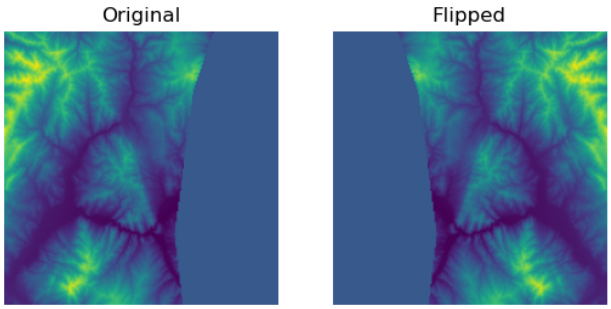
Because of the nature of the data, it was not necessary to perform much data analysis. It was easy to load directly from the files, and straightforward to work with through both the PyTorch and Numpy libraries.

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 aftr training the model. The data ranges are shown below:

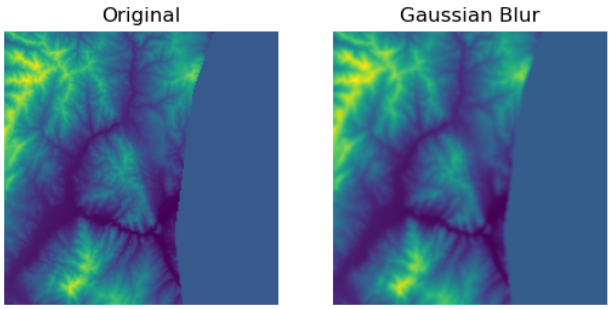
Range	min	max
Before Normalization	-1037.89	2596.36
After Normalization	-0.3997	1

After normalizing all of the image data, a few different image augmentation methods were considered, all from the torchvision.transforms library [2]. 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. Even with prior data management experience, there were some challenges with loading the data properly into the dataloaders. Originally, the goal was

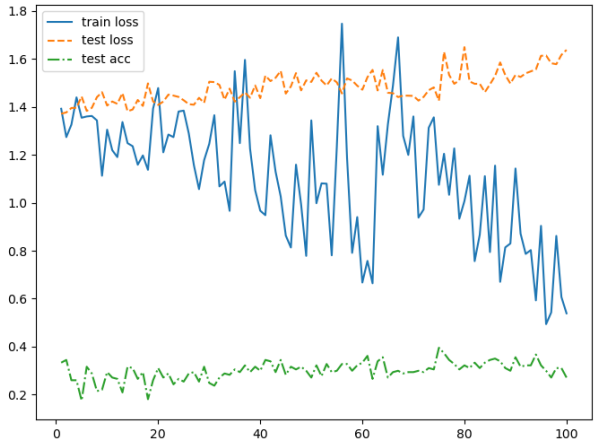
to apply all of the transforms before loading into a dataloader. However, this caused too many data type issues to be worthwhile, so the transform was moved inside of the training loop. Many data types still did not work properly, so everything was converted into a "torch.float64" datatype.

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 backpropogation 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 [3].

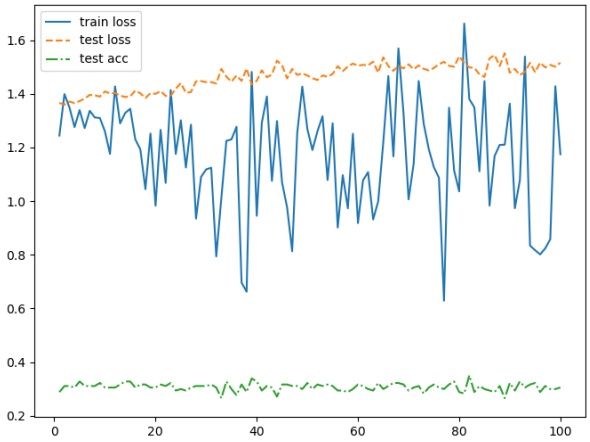
Approach

The initial step in devlopment involved creating a baseline Network in Network model as described in the d2l textbook [4]. This initial model was used to determine whether the data was being processed correctly, and to establish a learning baseline to be built upon. This model was trained for 100 epochs, and overfit the data considerably:

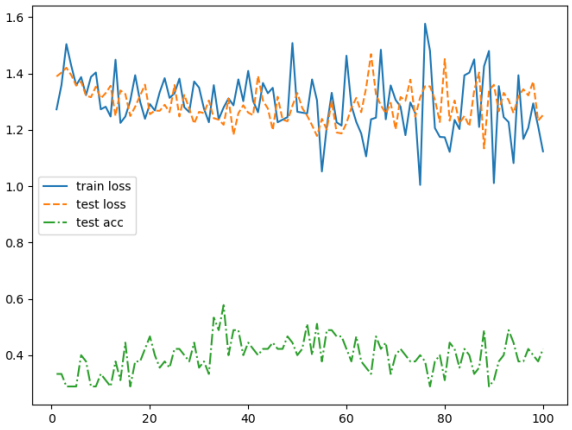


The model also did not appear to be learning very well at all. This was likely due to the small data size.

To artifically increase the data size, image augmentation was utilized via the torchvision.transforms library [2]. 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 helped to reduce the overfitting, however the model did not improve in accuracy much:



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

```
self.net = nn.Sequential(
    nin_block(96, kernel_size=5, strides=3, padding=0),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(256, kernel_size=3, strides=1, padding=2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(384, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_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 backpropogation, following the tutorial [3] and [5]. 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 [3] to calculate the gradients without directly hooking the ReLU layers.

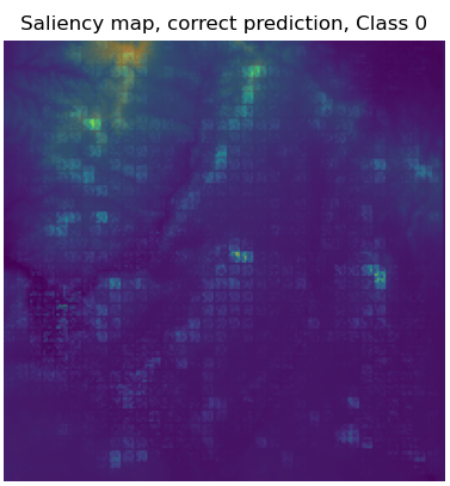
Evaluation and Summary

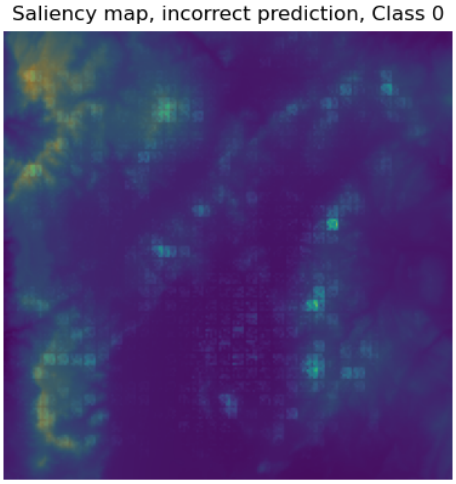
Unfortunately, the trained model did not perform to the level of accuracy desired. The model was able to achieve a top accuracy of 45.2%, with a binary classification accuracy of 69.7%. In addition, the model was not only not accuracy, it was not consistent either. The accuracy varied considerably depending on the training and validation datasets. The final validation loss varied from 1.0 to 1.4, and the accuracy was between 32 and 45%.

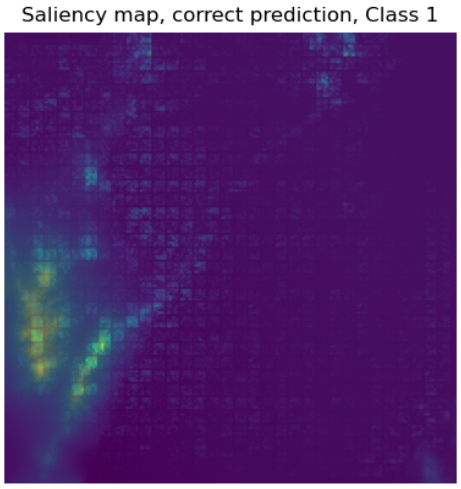
Fortunately, the model was able to stay relatively consistent between both validation and test loss, after incorporating dropout. Both values stayed similar throughout the training period.

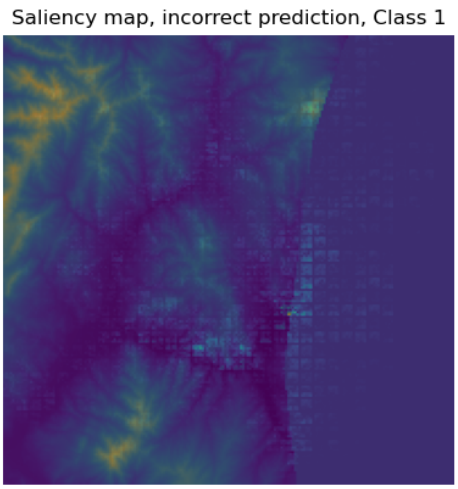
Loss Function	Value, 100 Epochs
Training Loss	1.11
Validation Loss	1.23

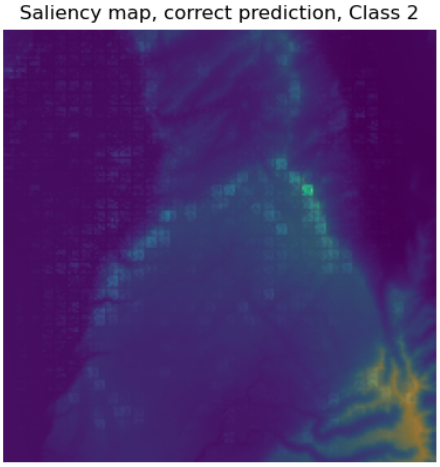
Guided backpropogation was also used to determine which parts of each image the model was considering for its predictions. Unfortunately, because of the lack of accuracy in the model, it was difficult to determine if these regions were actually useful for the model's predictions. However, it was interesting to see that the model was utilizing the topographical features of the images to make its predictions. The saliency maps for a few classes are shown below:

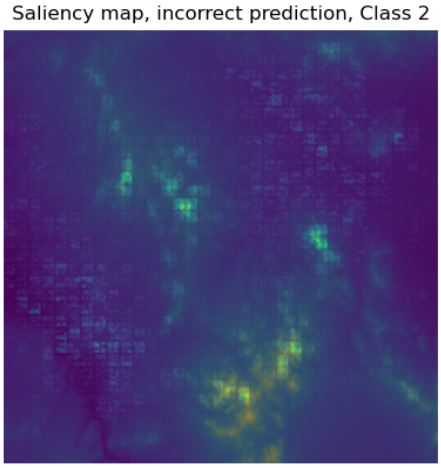


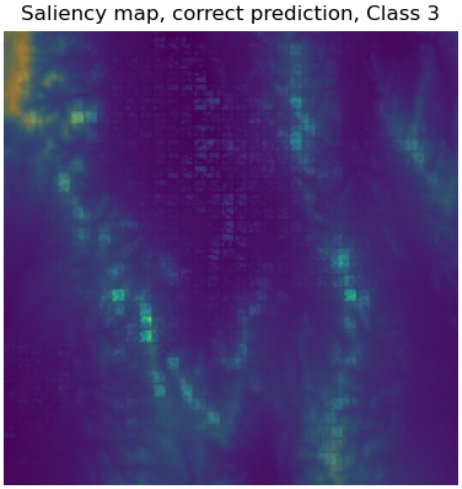


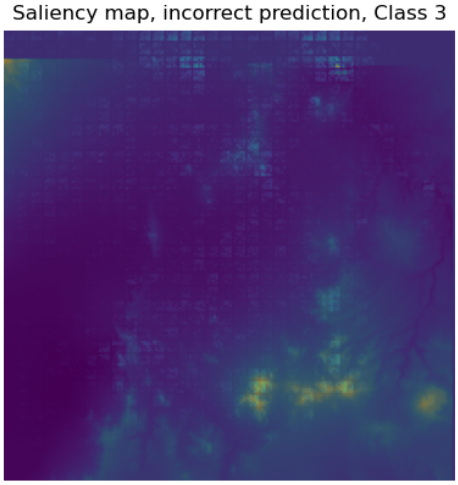








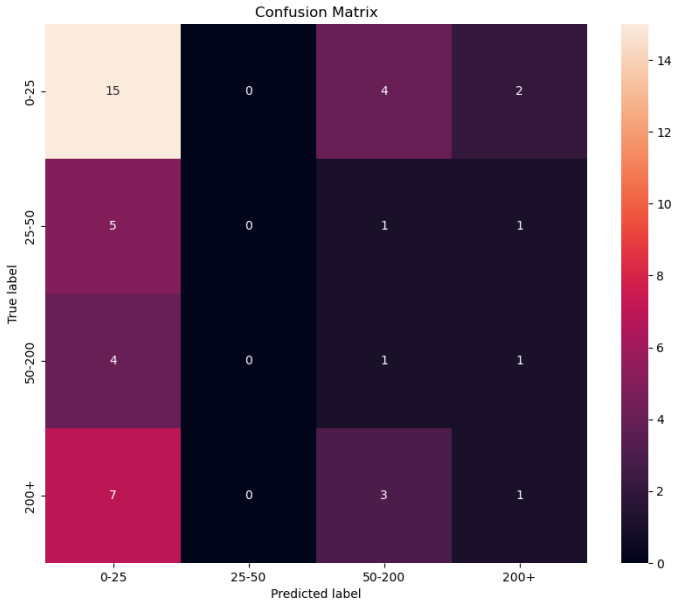




Finally, the following is a confusion matrix fo each classification class. The model seemed to classify the 0 class the best, with the 1 class being by far the worst, with zero correct classifications. The 2 and 3 classes

were very inaccurate as well. The model seemed to guess the 0 class most often. There also appeared to be a few higher residual classifications that were classified in class 0. These images could be ones that contain very little data–so it was difficult to make a prediction in the first place.

It is not clear as to why the model classified the lower values the best. Perhaps the model was overfitting to the lower values, since they were the most common, however there was no evidence of this in the training and validation loss. Perhaps for future development, creating two seperate models for the lower and higher values would be more effective at predicting the labels.



What I Learned

This project was an excellent introduction to image processing using convolutional neural networks. Having already developed basic models to analyze features, this project was a great way to delve deeper into more complex neural network methods. This project also involved more freedoms in regards to choice of neural network design. In the prior project, hyperparameter optimization libraries were used that made most of the experiementation unnecessary.

Additionally, this project served as a great introduction to image specific data modification methods. The torchvision.transforms library [2] 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 backpropogation. 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 [3], [5] 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] https://pytorch.org/vision/stable/transforms.html.

[3] I. A. Khalid, "Saliency map for visualizing deep learning model using pytorch," 2021, https://towardsdatascience.com/saliency-map-using-pytorch-68270fe45e80.

[4] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," 2021, https://d2l.ai/.

[5] K. Chung, "Guided backpropogation with pytorch and tensorflow," 2021, https://www.coderskitchen.com/guided-backpropagation-with-pytorch-and-tensorflow/.