Stanford ACMLab Fall 2020 Project Report: Team Regressionists

Rohan Sikand
Stanford University

Scott Xu
Stanford University

Priti Rangnekar
Stanford University

Kris Jeong
Stanford University

1 Repository Link

https://github.com/PritiRangnekar/ACMLab

2 Architecture

2.1 Background

Neural network architecture and functionality is largely inspired by the human brain at both the neuronal and network levels. The complex, interwoven paths that connect neurons together in the human brain, have been artificially represented in computers to create artificial neural networks. The breakthrough that is deep learning, has led us to a world in which there are self-driving cars and highly accurate language predictor models. Before this era, machine learning researchers were tasked with the then difficult problem of utilizing image data for these neural network models. However, even classifying digits of handwritten letters was a hard task with just fully-connected layers of neurons. Then, in 2012, Krizhevsky et al. (1), showed that a convolutional neural network can be used to significantly improve model performance on image data. From this, applying deep learning techniques to computer vision has become one of the most exciting topics in academia in the past decade. Specifically, these convolutional structures use additional and different layers to automatically learn spatial representations within images—giving these models much higher accuracy and usability with images.

2.2 Convolutional Layers

For this experiment, we aimed to apply a convolutional neural network to accurately predict the income distribution over a certain area given a satellite image of that area. The model architecture is described as follows. First, a convolutional layer is constructed which takes in three input channels, since the images we worked with are in RGB, and outputs six channels. Furthermore, we defined the kernel to be of size five. We also define a pooling layer to reduce the dimensionality of our output from each convolutional layer. This pooling layer is a two-dimensional max-pooling layer with a kernel size of two and a stride of two. After each convolutional layer, a pooling layer was added

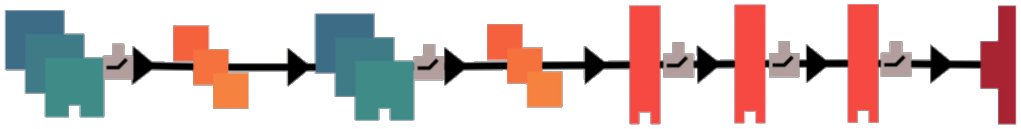

Figure 1: Shown here is a detailed visualization of the model architecture. The blue/green tiles represent the convolutional layers. The ReLU activation function is displayed between each layer where applicable represented by a small gray tile. The orange tiles represent pooling layers. The fully-connected linear layers are represented by the light red rectangular tiles. Finally, the output layer is represented by a maroon rectangular tile.

in the forward method. In building the model architecture, after the first pooling layer, a second convolutional layer was implemented which contains the total number of in channels as the previous layers' out channels: six. This second convolutional layer has twelve out channels. Once again, the same pooling layer was added in the forward after this second convolutional layer.

2.3 Fully-Connected Linear Layers

After the initial convolutional layers, four fully-connected linear layers were implemented in successive order. Since this task is a regression problem, our goal is to have the model predict one output value. Thus, when implementing these fully-connected layers, the last one should output one out feature. To get there, we started with 12 * 61 * 61 in features and 120 out features. The next fully-connected linear layer takes in the 120 out features from the previous layer as its in features. Further, this layer outputted 60 out features. Continuing on, the third fully-connected linear takes in the 60 as its in features and outputs 10 out features. Finally, the last fully-connected linear layer takes in the 10 out features as its in features and outputs the one out feature—our prediction.

2.4 Forward and Activation Functions

In the forward method, activation functions were implemented at the end of each layer. Specifically, a piecewise linear function, the rectified linear unit (ReLU) was used at the end of each layer in the architecture except for the last fully-connected linear layer which is our actual output.

A detailed visualization of the complete convolutional neural network architecture is shown in **Figure 1**.

3 Training

The training process consists of tuning the hyperparameters with the aim of minimizing the loss. Thus, before we trained our model, we designated two variables: the loss and the optimizer. Since this is a regression task, for loss, we used mean squared error which is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

For the optimizer, we used the Adam optimizer to perform stochastic optimization. This optimizer is commonly used and was introduced in (2). The learning rate we used was $lr = 0.01$. We used a batch size of 32 and we trained our model for 200 epochs.

4 Results

Our model functioned correctly and showed that this methodology for predicting the income when given a satellite image is statistically significant. We tested our model on a separate test set and measured the results using $L1$ regularization which represents taking the absolute value of all parameters, then summing them up to the loss function. Thus, values that are smaller, are better. When feeding our trained model the test data, our result was:

$$L1\ regularization = 97.5348 \tag{2}$$

Additionally, we also calculated the average loss over all test data points using the same loss defined in training: mean squared error defined in equation 1. Our result on the test set was:

$$MSE = 21573.2188 \tag{3}$$

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, p. 84–90, May 2017.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.