Landon Buell

August 2022

Bilinear Interpolation for Deep Image Classification

**Abstract**

Computer vision is a prominent and ever-growing subfield of Artificial Intelligence born out of digital image processing. As increasingly powerful processors and more disk space becomes available to the commercial, academic, and consumer worlds, the size of image datasets has also increased as well. Storing large masses of input data is a common problem in the world of AI and is routinely revisited. Similarly, most computer vision models are architected to accept images that all have a consistent size within a dataset. These problems combined highlight the need for a more efficient method of storing volumes of image data, without compromising the performance of the models that will process them. In this experiment, we explore a possible solution to this issue where we down-sample images to store them at a fraction of the disk size, and then use various interpolation techniques to rebuild them up to the original size. We explore how the down-sized, then upscaled images compare a baseline, and then offer a discussion for the viability of this as a long term solution.

1. **Introduction**

Deep image classification is type of *Computer Vision* which employs deep neural networks to organize images into categories based on their contents of properties. This practice can be seen widely in the modern word from

1. Related Works
2. Dataset
   1. Case Study 1 – MNIST 784 Dataset
      1. 70,000 Images of single digits 0 through 9
      2. 28 x 28 pixels (single gray-scale channel)
      3. Down-Sample to 7 x 7 (avg of 2 x 2 blocks) to simulate smaller sizes
      4. Allows images to be stored at 1/16 the size in DISK
3. Interpolation Technique – Bilinear Interpolation
4. **Deep Neural Network Architecture**
   1. Convolutional Neural Network
      1. Groups of Layers Conv2D + Conv2D + MaxPool2D
      2. Flatten out activations
   2. **Model Configuration**

At its most fundamental level, a deep neural network is a composite mathematical operation that transforms inputs, called *features* into outputs, called *predictions* using a set of parameters. We can model this operation as:

A neural network *learns* by choosing the elements in the set of parameters such that across a dataset, each input produces and output that matches reasonably close to a predefined ground truth value. A model that obeys this idea is said to be *trained*. We then expect the trained model to be able to produce reasonable predictions on new, unseen, or unlabeled samples similar to the training dataset.

**Loss Function** – For a classification task with *k* unique, categories, the output of the model and te ground truth labels will all be a vector with k elements in it, where each element corresponds to one of the classes. For any output from the model, we apply a *softmax* activation function to scale the vector to have an L1-norm of 1, which allows us to view it as a probability of what class the sample is predicted to belong to. For any input sample, we can quantify the difference between the computed output and the expected output through an *objective function*. In general, an objective function quantifies how similar two vectors are by returning a large value when they have a low projection, and a very small value when they have a high projection. For multicategory classification tasks, we typically employ the *categorical crossentropy* loss function.

**Optimizier** – To choose the parameters in a model that allow for it to produce reasonable predictions across a dataset, we must treat the training process as an optimization process. We start by generating a set of random parameters and passing a sample through the model. We expect this to prediction to be poor compared to the ground truth labels. From this output, an optimizer uses a process called *back propagation* to determine how to adjust each parameter in the model such to slightly decrease the value of the loss function for that sample. For deep neural networks, this process can be repeated hundreds of times across tens-of-thousands of samples until the optimizer algorithm converges on a set of parameters that allows for the model to produce consistently low cost values across a dataset. For this experiment, we choose to employ an *Adaptive-Momentum* (ADAM) algorithm for training.

**Metric Functions** – the loss function is useful for the model’s training/optimization process but offers little human-readable information on how it is performing. For multi-class classification tasks, we typically use a common set of metric functions to characterize the network’s prediction ability over a set of samples.

1. Precision Score
2. Recall Score
3. F1 Score
4. Methodology
   1. For all Images in the full dataset
      1. Sample from (m,n,k) down to (p,q,k)
      2. Scale each pixel to unit variance and zero mean
      3. This simulates the “compacted dataset” (1/16 the size)
5. Results
6. Conclusions