Image Detection using different Classification models

Team:

- Surbhi Rathore
- Anusha Singamaneni

1. Introduction:

As of recent, the automated vehicles are able to navigate using a form of visual image recognition to differentiate between general objects, developed with the deep neural networks. Image recognition is important for an automated vehicle systems to avoid objects such as trees, animals, and other vehicles, and where the road is. With the continuous development of these systems, we expect image classification to be better, faster and more accurate. To further understand and explore the concept of image recognition, our team has taken an approach to classify a set of images. So, we have selected the CIFAR-10 dataset (Canadian Institute For Advanced Research) collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, which is a collection of images that are commonly used to train machine learning and computer vision algorithms. The motivation behind taking up this project is to understand and implement an efficient image recognition technique with neural networks.

2. Data:

- CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.
- The original one batch data is (10000 x 3072) matrix expressed in numpy array. The number of columns, (10000), indicates the number of sample data. The CIFAR-10/CIFAR-100 dataset, the row vector, (3072) represents an color image of 32x32 pixels.
- The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class.

The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

- The label data is just a list of 10,000 numbers ranging from 0 to 9, which corresponds to each of the 10 classes in CIFAR-10.

  - airplane : 0
  - automobile : 1
  - bird : 2
  - cat : 3
  - deer : 4
  - dog : 5
  - frog : 6
  - horse : 7
  - ship : 8
  - truck : 9

- The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

3. Methodology:

3.1 Data Preparation:

3.1.1 Transforming images into tensors:

- There isn't a way for an algorithm to see an image, we have to convert the image into numbers (tensors) and feed them into algorithm as a feature. We did this by importing PyTorch transform module, which can then transform into tensors for each pixel. Now we have our data consists of numeric values that will be split based on RGB scale.
- At this point we have a transformed version of training dataset where each sample is now a set of three matrices one for each R,G,B color channel being 32x32, with each component being the color of value of that channel for that pixel in that image.

3.1.2 Split data into train and validation sets:

- Pytorch loads the test set independently. So we split our training set into 80/20 ratio between training and cross validation. To do this we implemented a function from scratch ,which creates a random permutations of the numbers that shuffles the indexes of the training set and to split the set into required ratio.

3.1.3 Mini batch Gradient descent:

- Since we're dealing with a large set of features (3072), as well as a large amount of training data (40000), that means regular gradient descent would be extremely costly. We split up both our training and cross validation sets into batches of 100 training examples each. Then, we use SubsetRandomSampler to create an iterable object of randomly shuffled values.
- Then, we can use DataLoader to create the batches. Given the batchSize (100) the sampler (trainSampler), and the transformed image matrix (datasetT) we create a DataLoader object that can be thought of as a list of batches of size (100, 3, 32, 32), or 100 image matrices.

3.2 Machine learning methods:

3.2.1 Multiclass Logistic Regression:

- The first machine learning model we attempted to use was logistic regression, that predicts which of the 10 classes some image from the dataset would be predicted as.
- A logistic regression model is almost identical to a linear regression model i.e. there are weights and bias matrices, and the output is obtained using simple matrix operations. We used nn.Linear to create the model instead of defining and initializing the matrices manually.
- Then we uploaded the images in batches of 100 at a time. Then converted our images into 3x32x32 tensors. This made our batch matrix 100x3x32x32 in dimensions. Then we passed each batch into the model (nn.Linear) which converted the tensors into 100x3072. Then we multiplied by weight matrices, which gave 100x10 result matrix with logits and added the bias.
- Applied SoftMax on the logits and converted them into probabilities. Then performed cross entropy loss function on the probabilities matrix to give a continuous and differentiable cost function. The cross entropy of all different batches is combined which gave us total training loss.
- Calculated the partial derivatives of this cross-entropy function wrt all the weights and biases, then we subtract the gradient from the vector of current weights.

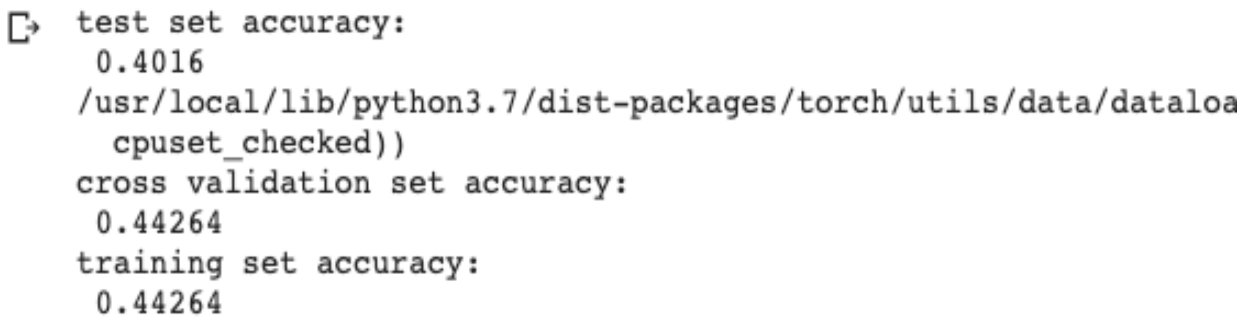
3.2.2 Multi-layer perceptron:

- We used a simple neural network for our second implementation. We created a base model for our neural network where we defined functions for training and validation process. Then defined evaluate function that returns the progress of our model after each epoch and used fit function which updates the weights for each epoch.
- For this model we used 4 hidden layers with input nodes of 1536,768,384 and 128 respectively for each layer. The forward pass function uses ReLu, which is a transition function that applies max(x,0) to an input x.
- Our model is ready to train and evaluate. After initializing random weights, iterate through the neural network and backpropagate depending on our learning rate which is step size and epochs.

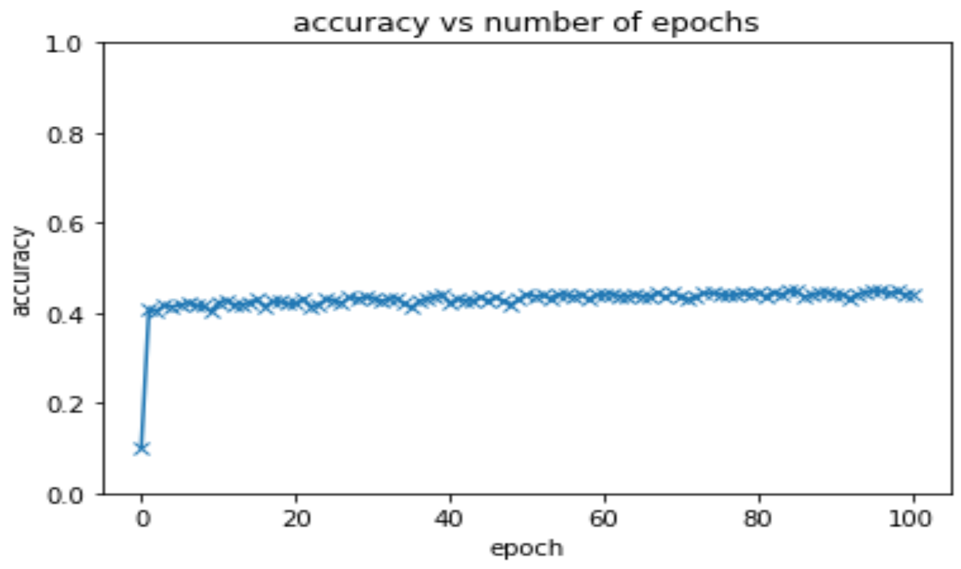
4. **Results:**

4.1 Multiclass logistic Regression:

For multi class logistic regression, after playing around with the learning rate for this model, we found the best to be at 0.009, where the maximum classification accuracy achieved was 41.71%.

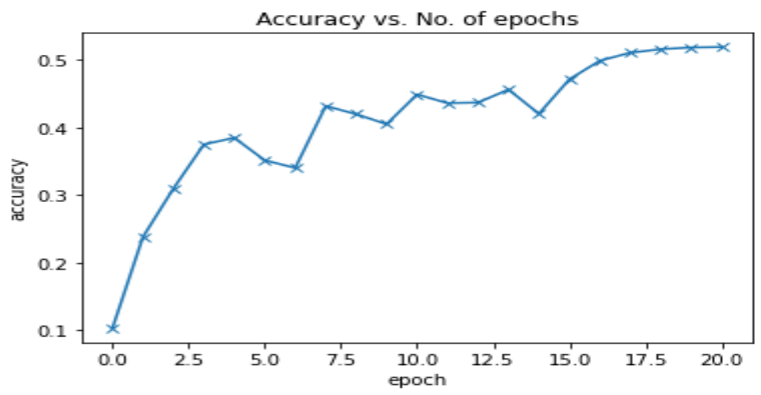


```
test set accuracy:
 0.4016
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloa
  cpuset_checked))
cross validation set accuracy:
 0.44264
training set accuracy:
 0.44264
```



4.2 Multi-layer perceptron:

- Playing around with the learning rate to obtain the minimum loss, we found the best at 0.0001where we achieved the accuracy of 53.09%
- We have converged to the minimum because our accuracy does not improve any further and our loss does not decrease any further. This is the best model our neural network came up with. Reason could be because our model looked at each pixel rather than the whole picture, which is relatively hard to generalize what an image can be.



```
] evaluate(model, test_loader)
  /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.
    cpuset_checked))
  {'val_acc': 0.530957043170929, 'val_loss': 1.3207964897155762}
```

5. Next Steps:

As our two models doesn’t have more than a half accuracy, we want to implement Convolutional filters to process and produce image. Convolution networks are able to look at a portion of a picture, allowing it to retain more information about an image rather than looking at a pixel.