The Identification Game - ACSE 4.4 Machine Learning Mini-project Report

Team Gauss 22.05.2020 Instructor name: Kukreja, Navjot

Project Description:

“Going forward, AI algorithms will be incorporated into more and more everyday applications. For example, you might want to include an image classifier in a smartphone app. To do this, you’d use a deep learning model trained on hundreds of thousands of images as part of the overall application architecture. A large part of software development in the future will be using these types of models as common parts of applications.”

Image Recognition is the process of identifying what an image depicts while Image classification analyzes the numerical properties of various image features and organizes data into categories. The aim of this project is to train a classifier capable of taking an RGB input image of size 64*64*3 from a given dataset and output the class corresponding to the image.

Project Structure:

The project is broken down into the following workflows:

- Preprocessing the Image Dataset
- Loading the image dataset
- Building a classifier from scratch
- Utilizing a pre-trained classifier
- Training the classifiers
- Labelling the test set based on trained classifiers

Preprocessing the Image Dataset:

The first stage of preprocessing involved resizing the input images from 64*64 to a suitable size for an ImageNet-scale network in the case of pre-trained model. Other transforms included in preprocessing includes normalizing the images to have a zero mean and doing some random rotations on some of the images to increase diversity without a significant loss in accuracy.

In a bid to further enhance the input data for better accuracy, we experimented on ZCA whitening to decrease the noise in the data. More information on this process can be seen in the short report on the GitHub repository.

Loading the Image Dataset:

A total of 100000 images were provided consisting of 200 classes, each image belonging to one of these classes. This data was initially loaded using the pytorch ImageFolder functionality suitable for the structure of the input images. As no validation set was provided, the input train set had to be split to make provision for validation data. Sklearn functionality, StratifiedShuffleSplit was used to obtain a random sample containing 10% of the train set to be used as validation set. The two data sets were finally loaded into gpu friendly smaller batches using DataLoader to prevent exceeding computation limits.

Building a Classifier from scratch:

The classifier we called SimpleNet was built from 2 convolutional layers with kernel filters of size 3 by 3. The convolution parameters were chosen to detect the predominant features present in the input data. The output of each convolution later was summed with a bias term and passed through activation functions. The Relu activation function was used in this case. This was used to introduce non-linearity into the network. To prevent slowdown and speed up the training process, we introduced max pooling layers into the network. We used a max pooling stride of 2 to improve speed. The thought process preceding this was to pool in the maximum value within the kernel window into the output in a quick and efficient manner. Also, picking a stride of 2 helped to significantly reduce the memory load on the gpu. To prevent out model from overfitting, we introduced drop out. We noticed improvements in validation accuracy when drop out was utilized. The final step of modelling involved adding a fully connected layer where the 3d image is converted into a feature vector.

Utilizing a a Pre-trained Classifier (Transfer Learning):

For transfer learning, the same augmentation as training from scratch was applied with the difference that the image was resized and cropped ([3, 224, 224]) to meet the pre-trained model requirements. A pre-trained ResNet-50 model on the ImageNet dataset was used, consisting of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. ConvNet as fixed feature extractor was applied, where the last fully-connected layer is removed and adjusted to our classes, then treat the rest of the ConvNet as a fixed feature extractor for the Identification Game challenge dataset.

Training the Classifiers:

To begin, the training process, the constant hyper parameters such as learning rate, batch_ , number of epochs and weight decay were set to appropriate known values. The momentum hyper parameter was experimented on using grid search to obtain the momentum value with the best accuracy. We chose a loss criterion that was more intuitive to understand and apply. This was the cross entropy loss which minimizes the distance between the predicted and actual probability distributions. Stochastic gradient descent worked well with both classifiers as the optimizer with the momentum varied and obtained from the best accuracy. Comparing the accuracy results from using a pre-trained model and our “SimpleNet” model built from scratch, the pre-trained model performed better but our SimpleNet model was able to pass the Turing test score.

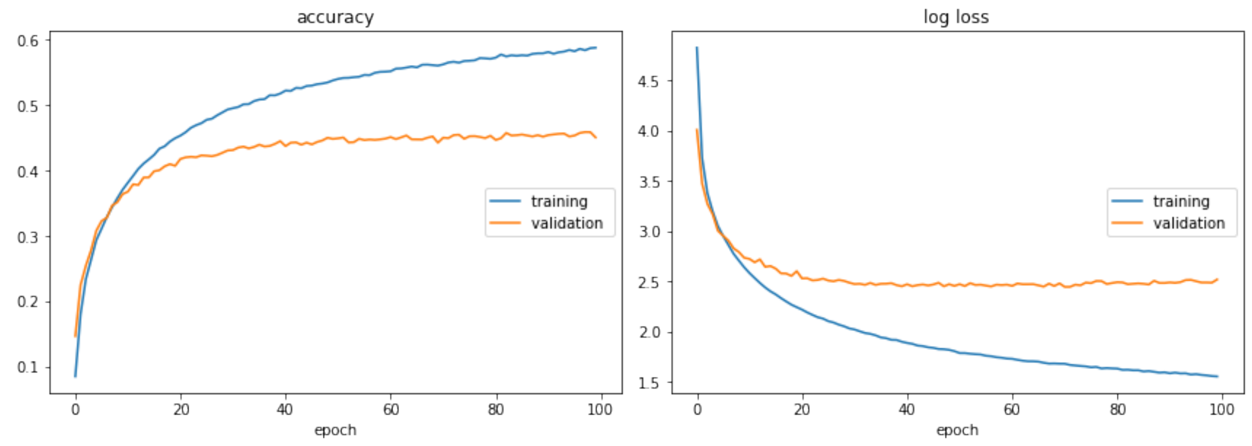
Future Work:

For pretrained-modelling, a second idea was to train the model the from scratch on the down-sampled ImageNet dataset ([3, 64, 64]) and use the pre-trained parameters on the 6on the challenge dataset([3, 64, 64]). Unfortunately, the time where this concept was captured did not let us apply it (training from scratch a 11.73 GiB dataset could take some time)

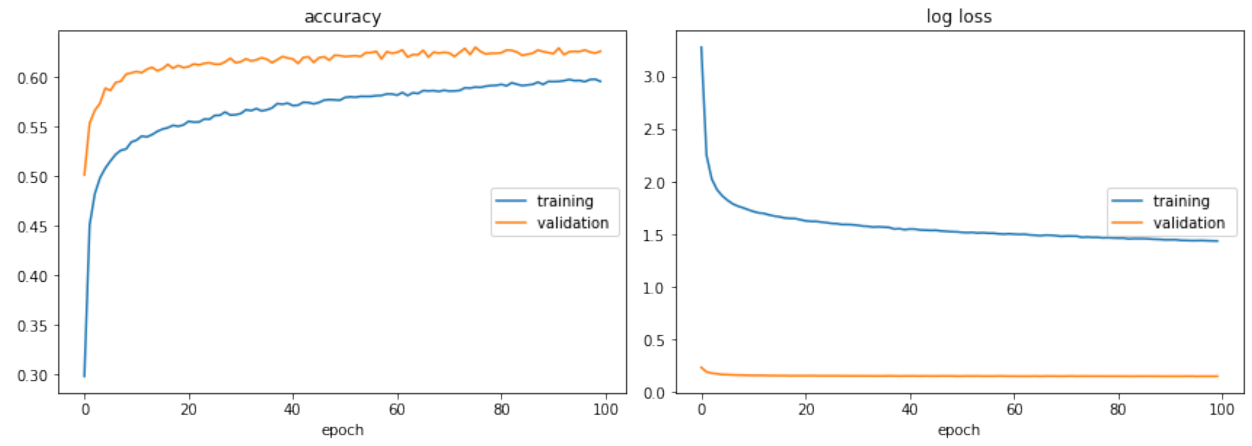
Conclusion:

We were able to pre-process the input data, load the data in batches and build two different models for classifying these input data. Our SImpleNet showed a lot of promise in passing the Turing Test while using a pre-trained model illustrated the advantages of transfer learning especially when the input data size is not adequate. The test data images were successfully labelled with our trained models to F1 scores of 0.4 and 0.6 for our SImpleNet and Pre-trained models respectively.

SimpleNet output accuracy:



Pre-trained Resnet output accuracy:


accuracy
epoch
training
validation
log loss
epoch
training
validation