Anirudh Krishnan Komaralingam

UNIVERSITY OF MARYLAND

ENEE633 Project 2

Part 1: Handwritten digit recognition

General data setup procedures

1. The MNIST Dataset was collected from **https://deepai.org/dataset/mnist** because the current maintainer of the dataset, Yann LeCun has protected. The data was verified to match an identical collection of digit images which can be collected from the Pytorch python machine learning framework.
2. The data was used in NumPy arrays of SVM and Logistic Regression, but was replicated using Pytorch for the Deep learning portion of the project. This was done because the version made available by Pytorch was formatted into tensors from the start. This is stated because there are do sources of the data in the code and it might be downloaded twice when running.
3. The data was standard normalized before any processing was done as part of classification steps.
4. In situations where training took too long (longer than 2 minutes), a batch of a smaller size is used for training. This was able to achieve similar results to using the complete number of training samples (60000, whereas a batch would be 3000-10000).

SVM

Kernel SVM was performed with the following steps. Python and python libraries were used to make the classifier.

1. Collection and standard scaling of the data.
2. PCA/LDA or no dimensionality reduction. This was performed using established python libraries for efficient implementation.
3. The native LIBSVM library was used to create the SVM classifier.
4. With the C-SVC classifier type from LIBSVM, the linear, Polynomial and RBF kernel were tested. The variation in performance is summarized in the table below.

Results and Analysis

Time is in seconds, accuracy is in percent of correct predictions to total predictions

Important Note: After PCA, the datatype of the training data becomes fp64. This causes catastrophic slow downs and reduces the benefits of reducing the dimensions. As a policy fp64 is changed to fp16 to recover some performance

Linear Kernel – No Dimensionality Reduction			
Train Batch	Accuracy	Train Time	Test Time
100	66.32	0.1	0.2
200	76.73	0.1	0.2
500	84.47	0.1	0.4
1000	86.96	1.5	5.4
3000	89.99	2.6	10.9
10000	91.23	25.7	45.4
60000	93	747	116

The training time scales non-linearly with the batch size. The benefit of more a bigger batch size quickly diminishes and the tradeoff for time becomes more expensive. It can also be observed that the test time increases when more training data is used to train the classifier.

Linear Kernel – PCA 50 Components			
Train Batch	**Acc**	**Train Time**	**Test Time**
100	61.88	0.1	0.2
200	71.25	0.1	0.5
500	79.27	0.1	0.3
1000	84.22	0.2	0.4
3000	88.88	0.3	0.6
10000	92.22	1.5	1.2
60000	93.35	53.9	10.5

Reducing the number of features through PCA dramatically reduces training time, though the full batch still relatively slow to train for the benefit of the larger number of samples to train on. The testing time is also significantly reduced which suggests that the dimension of the data is the cause of increased complexity and not the number of samples. The accuracy of the model with 90% fewer features is within the margin of error and therefore appears to also exceed the performance of using raw data.

Polynomial Kernel – Varying Degree – Full Train Set – PCA 50			
Degree	**Accuracy**	**Train Time**	**Test Time**
2	96.91	25.9	5.6
3	96.25	121	23
5	84.42	201	54
10	91.05	91	12.3
20	Failed To Finish	>20	Unknown

The degree of the kernel has an optimum value, simply making the kernel a complex function does not work well. The training time becomes exponentially high with the degree. The test prediction time also increases with the degree of the classifier. Smaller degrees work best, close to the default value of 3 from the library author. Achieves a high accuracy, comparable to CNN method which is nearly perfect.

RBF Kernel – Pca 50 – 10000 Train Set				
C	**Gamma**	**Accuracy**	**Train Time**	**Test Time**
1000	1	25.24	10.5	6.5
100000	1	25.24	10.5	6.3
100	10	11.35	13.1	8.9
1000	100	11.35	14.2	9.3
1000	0.1	89.04	7.2	4.7
1000	0.01	95	1.5	0.7
1000	10	11.35	13.1	8.8

Highly dependent on parameters and has a very narrow range of optimal values in comparison to the other kernel types. The training speed is the best in the range of values tested compared with the other kernel types. Given the shrinking of the gaussian that we are fitting on to the data, we risk overfitting. Considering the nature of the samples (a small number of classes and small space of variations – handwritten digits) this is most likely occurring. The RBF is the default choice of SVM kernel, but I do not suggest it for this application.

Anirudh Krishnan Komaralingam

UNIVERSITY OF MARYLAND

Logistic Regression

1. Setup data, normalization identically to SVM implementation.
2. The PCA method is also shared with all the classifiers.
3. The probability function and cross entropy loss is taken from the *sklearn* python library.
4. I force the use of cross entropy loss calculation by using the multinomial option.
5. The default solver is the pseudo second order optimizer – L-BFGS.
6. Note that this is the fastest to execute of all the classifiers for highly acceptable classification performance, elaborated in the next section.

Results and Analysis

Logistic Regression				
Batch Size	**PCA**	**Accuracy**	**Train Time**	**Test Time**
500	50	81.48	0.1	0.6
5000	50	89.44	0.5	0.6
60000	50	90.51	2.1	0.4
10000	5	67.8	2.2	0.5
10000	20	86.9	2.2	0.6
10000	150	91.1	1.1	0.4

MNIST Convolutional Neural Network

1. The Convolution neural network contains 3 trainable layers. The first 2 perform 2D convolutions along with the ReLU non-linearity and MaxPool operations. The classification layer is a affine/fully connected layer that produces 10 score values, one for each digit class.
2. Layer 1: 5x5 kernel size, 16 feature maps/channels, 2 unit padding and stride 1. Pooling is also 2x2 and stride 2.
3. Layer 2: identical to layer 1, with 32 output feature maps/channels.
4. Loss function is cross entropy loss and the optimizer is Adam.
5. Trained on full training set unlike some of the cases above and uses a minibatch of 100 images.
6. Network is implemented in Pytorch 1.13, on CPU.

Results and analysis

CNN – Varied Epochs			
Epoch	**Accuracy**	**Train Time**	**Test Time**
1	99	13	2.7
2	97	24.9	2.2
5	99	64	2.2
10	98	1020	2.2

The performance of the network is good given that it makes use of techniques that were not developed (Adam, Batch Norm, Dropout) when early versions of LeNet were being conceived. Learning rate of 1e-2 was used on first attempt and allowed the loss to be minimized to 1e-2 in 1 epoch. Some other benefits are fixed prediction time despite any training time. Model can be saved and reloaded. 10 epoch trained model is saved and submitted.

Anirudh Krishnan Komaralingam

UNIVERSITY OF MARYLAND

Part 2: Transfer Learning

General Data Setup Procedures

1. The data images were of varying sizes, resolution and orientation.
2. It was decided to make the images the size needed for input to VGGXX which would need to done in later parts of the Project, i.e. 224x224x3.
3. Since Pytorch was to be used, the images were also converted to GPU Tensors at the start.
4. While there are several valid data augmentations that Pytorch provides to compensate for the small training data set such as random crops, flips, rotation, colour shifts, etc. they were not performed as part of the routine input transform process as the focus was on transfer learning.

Monkey Convolutional Neural Network

1. The Neural Network is a 5 layer network, with 4 Convolutional-ReLU-Pool layers for feature extraction and 1 fully connected layer for classification.
2. Each convolution layer uses a 3x3 kernel.
3. Layer 1: 224x224, 3 channel RGB monkey image and 16 feature maps
4. Layer 2: 112x112, 16 channel activation maps as input and produces 32 output channels
5. Layer 3: 56x56, 32 channel input and produces 64 output channels
6. Layer 4: 28x28, 64 input channels and produces 128 output channels
7. All layers use padding of 1 and stride of 1.
8. The FC layer produces 10 output scores and uses cross entropy loss function
9. Adam is used as the optimizer.
10. Given the limited data for each class, the goal was to make the basic network as robust as possible within reason.
11. CUDA acceleration is used to accelerate training for more epochs ~ 10.
12. 10 epoch trained model is saved and provided with the submission.
13. Random samples in batches of 32 were used for testing and training.
14. The performance of the network is highly variable. For every training scenario, 5 batches were validated. The accuracy fluctuated between 30% and 85%. However, it can be concluded that there was learning progress made as the lowest accuracy was above the range of random guessing.

Monkey CNN + Frozen VGG16 Feature Extractor

1. VGG16 was chosen as the pretrained feature extractor network. This network is well known for its success on the ImageNet database. It is particularly inefficient but for a small scale problem and small version of the network it is not a problem.
2. In Pytorch, the feature extractor is a sequential unit that can be addressed. This made it easy to freeze the weights of the convolutional portion and leave the FC layers pliant to back propagation.
3. The next change was that VGG16 produces 1000 class scores and there are only 10 monkey species in our dataset. The last fully connected layer was changed to become a linear (fully connected) layer that took as input 4096 parameters (output size of second last layer) and itself output 10 values.
4. The loss function was the same as in the simple CNN, cross entropy
5. The optimizer was changed to SGD because it is known that VGG was designed with use of SGD.
6. A learning rate scheduler was implemented to periodically reduce the learning rate every epoch.
7. One point to note was that manual garbage collection and cache clearance was required when experimenting due to the nature of graph retention on the GPU which is a consequence of the Pytorch implementation.

Results and Analysis

The modified VGG16, referred to as monkey_FrozenVGG16 was trained for 1, 2, 3, 5, 10 epochs, accelerated with CUDA. The loss value decreased consistently and the prediction accuracy was able to score similar results with multiple batches unlike the shallower CNN trained on the small dataset.

The accuracy improved from the range between 30-80% to a consistent 93% with this configuration of the network that was trained for a few minutes (~10) on laptop grade hardware.

There was, however, a difference of 20% between training and test accuracy which did not seem to get better with more training.

Monkey CNN + UnFrozen VGG16

1. The last experiment was taking the partially trained transfer learning network and allowing all the layers to accept gradient flow.
2. The was done in the same way that gradients were frozen, by toggling a flag on all the weights of the convolutional layers which are conveniently presented in VGG16 as feature layers.
3. The network was able to produce a higher training accuracy, approaching 97% which was a predicted improvement on using VGG16 as purely a feature extractor.
4. The network training stagnated when training immediately after unfreezing the weights because the learning rate was decayed. Once the learning rate was restored, the training accuracy began rising.
5. However, it should be noted that the testing accuracy failed to improve and the delta between training and testing seemed to widen because the training accuracy improved.
6. This points to a lack of data.