SYDE 671 Assignment 4

Pascale Walters

Q1: Many traditional computer vision algorithms use convolutional filters to extract feature representations, e.g., in SIFT, to which we then often apply machine learning classification techniques. Convolutional neural networks also use filters within a machine learning algorithm. What is different about the construction of the filters in each of these approaches? Please declare and explain the advantages and disadvantages of these two approaches.

The convolutional filters that are used in traditional computer vision algorithms have been selected during the design of the algorithm. They usually have a specific function, such as the Sobel kernel for edge detection or an averaging filter to perform blurring. In CNNs, the weights that make up the filter are learned through backpropagation. They may not be as easily interpreted by a human.

The advantage of hand-crafted filters in traditional computer vision algorithms is that their function is specifically known. Given enough domain knowledge, the appropriate filter can be selected without the additional computational cost of learning it. However, it can be difficult to select the appropriate filter given that there are many, many possibilities. When learning the filters in a CNN, no specific domain knowledge is known, as the training of the network will give an optimal solution. However, it can be difficult to know whether this optimal solution is the global optimal solution. In addition, the collection of training and testing samples and computational resources are required for CNN training.

Q2: Many CNNs have a fully connected multi-layer perceptron (MLP) after the convolutional layers as a general purpose ‘decision-making’ subnetwork. What effects might a locally-connected MLP have on computer vision applications, and why?
Please give your answer in terms of the learned convolution feature maps, their connections, and the perceptrons in the MLP.

Locally connected MLPs refer to sparsely connected layers where there is not a connection between each input (output from a convolutional layer) and node in the MLP layer. These are generated by removing connections that have a low weight during training.

The advantage of having fewer connections is that the network has fewer parameters, thereby reducing computational costs, especially memory for storage of the trained network. However, the accuracy may be reduced as several values from the convolutional feature maps are not considered in the output of the MLP layer. Also, matrix multiplication is highly optimized on GPUs, which are typically used for deep learning applications. Run time would likely be increased as sparse matrix multiplication is not as optimized on this hardware.

Q3: Given a neural network and the stochastic gradient descent training approach for that classifier, discuss how the learning rate, batch size, and training time hyperparameters might affect the training process and outcome.

Learning rate is a hyperparameter that controls the size of the steps during gradient descent, while the weights are being updated. A large learning rate can mean less time to convergence, but if it is too large, convergence may not be reached as the step size can miss local minimums.

In the training of CNNs, weights are updated after each pass of a batch through the network. The batch size refers to the subset of training samples used for each weight update. A larger batch size means that it takes longer for the weights to be updated, but it is more accurate. Smaller batches are faster, but can be noisier, as there are fewer gradients to include in the average calculation.

Assuming that training time is a proxy for the number of epochs, this refers to the number of times all training samples are passed through the network before training is complete. The more the training samples are passed through the network and the weights updated, the better the model will fit the samples. If the training time is too long, the model can overfit and not generalize well. Shorter training time can give lower accuracy but reduces the risk of overfitting.

Q4: What effects does adding a max pooling layer have for a single convolutional layer, where the output with max pooling is some size larger than 1 x 1 x d?
Notes: ’Global’ here means whole image; ’local’ means only in some image region.

The following are the correct descriptions:

- Decreases computational cost of training (reduces number of parameters that need to be learned)
- Decreases computational cost of testing (fewer parameters means that fewer multiplications need to be performed during inference)
- Decreases overfitting
- Increases underfitting (could occur due to fewer parameters)
- Increases the nonlinearity of the decision function (finding the maximum of a set of values is a nonlinear function)
- Provides local and global rotational invariance (provided that the rotation is small enough to fit within the pooling kernel)
- Provides local scale invariance (provided that the scaling is small enough to fit within the pooling kernel)
- Provides local and global translational invariance (provided that the translation is small enough to fit within the pooling kernel)

Q5: What do these numbers tell us about the capacity of the network, the complexity of the two problems, the value of training, and the value of the two different classification approaches?

- NN on MNIST: 92%
    - Epoch 0 loss: 27117.047105130143 Accuracy: 90.915%
    - Epoch 4 loss: 17935.170377465696 Accuracy: 92.40833333333334%
    - Epoch 9 loss: 17091.002916531088 Accuracy: 92.64%

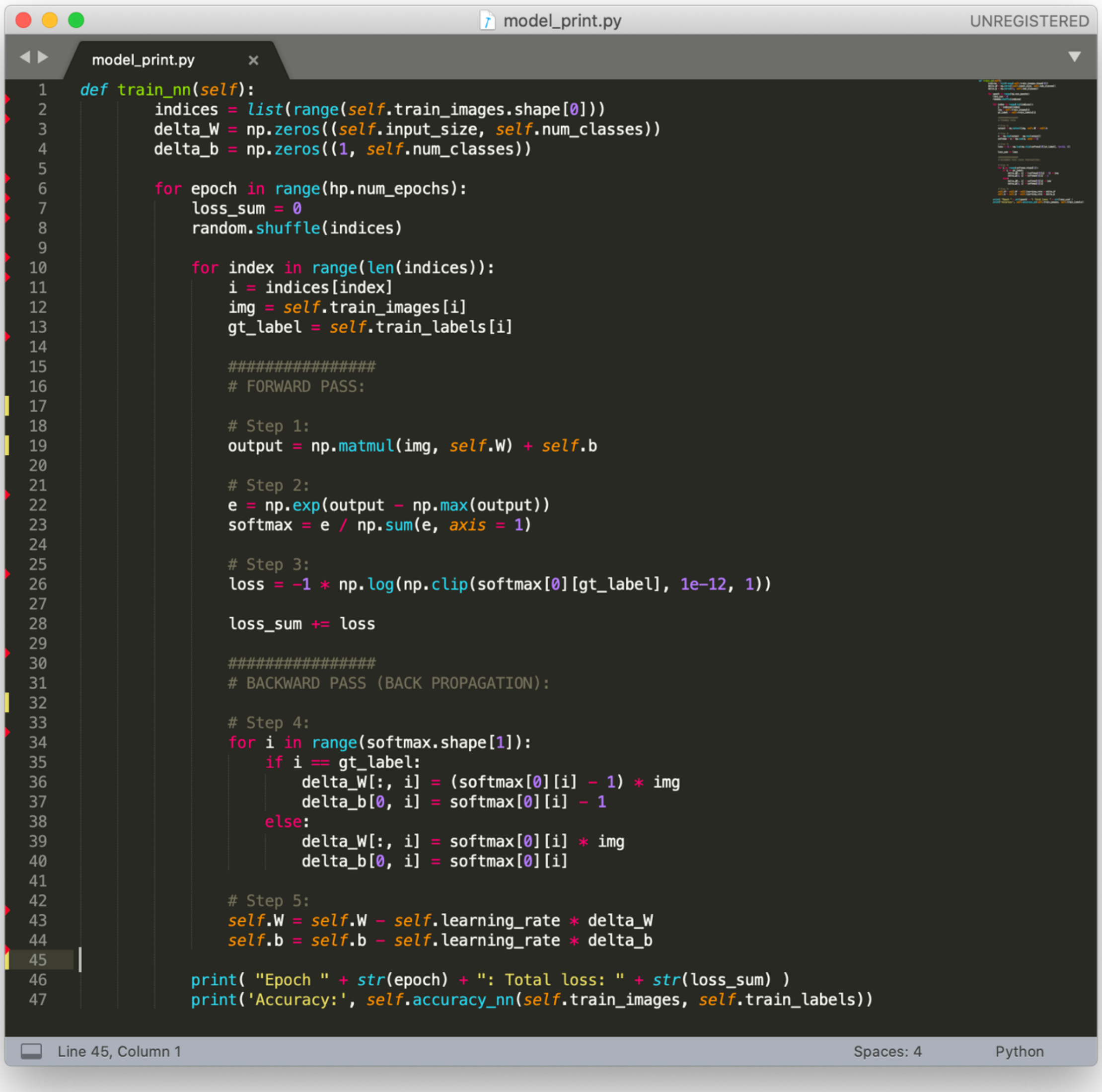
- NN+SVM on MNIST: 91%
    - Epoch 0 loss: 27089.64297008928 Accuracy: 90.68666666666667%
    - Epoch 4 loss: 17819.75302105546 Accuracy: 91.96166666666666%
    - Epoch 9 loss: 17011.798433893237 Accuracy: 92.28333333333333%
- NN on SceneRec: 13%
    - Epoch 0 loss: 14248.747705126762 Accuracy: 13.466666666666666%
    - Epoch 4 loss: 9848.444242039404 Accuracy: 16.4%
    - Epoch 9 loss: 7687.483651236076 Accuracy: 25.4%
- NN+SVM on SceneRec: 22%
    - Epoch 0 loss: 14167.27916792738 Accuracy: 19.733333333333333%
    - Epoch 4 loss: 9760.432397222956 Accuracy: 26.73333333333333%
    - Epoch 9 loss: 7184.5613182953175 Accuracy: 33.86666666666667%

These results show that the MNIST classification problem is more difficult than that of scene recognition. Accuracy with a one-layer neural network achieves good testing and training results for MNIST (~92%) vs. SceneRec (~20%). The images from MNIST are simpler images that are almost binary, whereas SceneRec has a wide variety of grayscale images and more variation within each class.

The capacity of a network is the range of functions that it can model and is similar to the overfitting and underfitting of the network. With MNIST, the final training and testing accuracies are similar, which means that the network fit the problem well. With SceneRec, the network is likely too simple. The losses and accuracies fluctuate during training, which shows that the problem is to complex for this simple network.

In all cases, there wasn't much change in the accuracy and loss during training for 10 epochs. Furthermore, the SceneRec dataset had fluctuations during training. This shows that training for 10 epochs was more than sufficient for training this network.

There was not much difference in performance between the NN and NN+SVM modes. This makes sense because the decision boundary of a one-layer neural network has almost the same structure of a multiclass SVM.



```python
def train_nn(self):
        indices = list(range(self.train_images.shape[0]))
        delta_W = np.zeros((self.input_size, self.num_classes))
        delta_b = np.zeros((1, self.num_classes))

        for epoch in range(hp.num_epochs):
            loss_sum = 0
            random.shuffle(indices)

            for index in range(len(indices)):
                i = indices[index]
                img = self.train_images[i]
                gt_label = self.train_labels[i]

                ################
                # FORWARD PASS:

                # Step 1:
                output = np.matmul(img, self.W) + self.b

                # Step 2:
                e = np.exp(output - np.max(output))
                softmax = e / np.sum(e, axis = 1)

                # Step 3:
                loss = -1 * np.log(np.clip(softmax[0][gt_label], 1e-12, 1))

                loss_sum += loss

                ################
                # BACKWARD PASS (BACK PROPAGATION):

                # Step 4:
                for i in range(softmax.shape[1]):
                    if i == gt_label:
                        delta_W[:, i] = (softmax[0][i] - 1) * img
                        delta_b[0, i] = softmax[0][i] - 1
                    else:
                        delta_W[:, i] = softmax[0][i] * img
                        delta_b[0, i] = softmax[0][i]

                # Step 5:
                self.W = self.W - self.learning_rate * delta_W
                self.b = self.b - self.learning_rate * delta_b

            print( "Epoch " + str(epoch) + ": Total loss: " + str(loss_sum) )
            print('Accuracy:', self.accuracy_nn(self.train_images, self.train_labels))
```