A brief description of the CNN model and the training process.

**The CNN model** used in project 6

Inputs: x

Outputs: z

1. cx = Conv2d(x)
   1. Parameters: weights Wc1 and biases bc1
2. rx = ReLU(cx)
3. y = MaxPool(rx)
4. cy = Conv2d(hx)
   1. Parameters: weights Wc2 and biases bc2
5. ry = ReLU(cy)
6. g = MaxPool(ry)
7. h = Flatten(g)
8. z = Linear(h) = Wf\*h + bf
   1. Parameters: weights Wf and biases bf

**The training process**

1. Initialise theta (weights and biases)
2. Shuffle training images
3. For each epoch
   1. Select batch images and labels based on batch size
   2. Obtain z, h, g, ry, cy, y, rx, cx, and x with forward propagation
   3. Calculate nabla J with backpropagation
   4. Update theta with stochastic gradient descent (SGD)
   5. Obtain new z using new theta with forward propagation
   6. Use z to predict labels for validation images
   7. Compare predictions with actual labels and print accuracy

How the three hyperparameters bound, epsilon, and batch\_size affect the accuracy through validation? You should experiment with different combinations of the three hyperparameters and use tables and figures to explain your discovery.

I had the same results as for project 5 so I will not explain their relation to each other as in-depth, but I’ll provide a general overview.

**Bound**

Smaller bounds (around 0.01) seemed to perform better. However, too small bounds (less than 0.001) had an adverse effect.

Largest bound, lowest accuracy:

Bound = 25, Epsilon = 1e-5, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.093 |
| 2 | 0.092 |
| 3 | 0.091 |
| 4 | 0.089 |
| 5 | 0.091 |

Bound = 1, Epsilon = 1e-5, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.208 |
| 2 | 0.161 |
| 3 | 0.179 |
| 4 | 0.176 |
| 5 | 0.167 |

Smallest bound, highest accuracy:

Bound = 0.01, Epsilon = 1e-5, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.150 |
| 2 | 0.208 |
| 3 | 0.264 |
| 4 | 0.301 |
| 5 | 0.344 |

**Epsilon**

Epsilon seemed to affect the rate of training rather than how high/low the accuracy values were overall. Larger epsilons showed a greater increase in accuracy on each epoch. However, too great an epsilon led to overtraining and no increase in accuracy on subsequent epochs. This could be remedied by decreasing the bound.

Small epsilon, so accuracy values increase slowly:

Bound = 0.01, Epsilon = 1e-6, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.112 |
| 2 | 0.113 |
| 3 | 0.113 |
| 4 | 0.117 |
| 5 | 0.119 |

Epsilon too large for bound (overtrained), so accuracy wavers rather than steadily increasing:

Bound = 1, Epsilon = 1e-5, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.208 |
| 2 | 0.161 |
| 3 | 0.179 |
| 4 | 0.176 |
| 5 | 0.167 |

Bound lowered to account for greater epsilon, so accuracy again steadily increases:

Bound = 0.01, Epsilon = 1e-5, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.150 |
| 2 | 0.208 |
| 3 | 0.264 |
| 4 | 0.301 |
| 5 | 0.344 |

Epsilon further increased for faster training (greater increases in accuracy):

Bound = 0.01, Epsilon = 1e-4, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.426 |
| 2 | 0.830 |
| 3 | 0.885 |
| 4 | 0.910 |
| 5 | 0.922 |

**Batch Size**

With the exception of a batch size of one, smaller batch sizes performed better.

Bound = 0.01, Epsilon = 1e-4, Batch Size = 20

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.179 |
| 2 | 0.238 |
| 3 | 0.285 |
| 4 | 0.310 |
| 5 | 0.325 |

Bound = 0.01, Epsilon = 1e-4, Batch Size = 4

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.426 |
| 2 | 0.830 |
| 3 | 0.885 |
| 4 | 0.910 |
| 5 | 0.922 |

Bound = 0.01, Epsilon = 1e-4, Batch Size = 2

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.883 |
| 2 | 0.931 |
| 3 | 0.948 |
| 4 | 0.951 |
| 5 | 0.958 |

Taking these patterns into account, here were my final training hyperparameters:

Bound = 0.001, Epsilon = 3e-4, Batch Size = 2

| **Epoch** | **Accuracy** |
| --- | --- |
| 1 | 0.116 |
| 2 | 0.867 |
| 3 | 0.941 |
| 4 | 0.961 |
| 5 | 0.968 |

Discuss the difference in capacities of your MLP model in Project 5 and the CNN model for Project 6.

The MLP implemented in project five was able to be run more quickly for image classification, making it a good choice for simple classification applications. Although the algorithm took more epochs to train, each epoch took less time to run.

Since the CNN implemented in project six doesn’t immediately flatten the data but rather puts it through convolutional layers, it is able to recognise the spatial relation between pixels in an image unlike the MLP, making the CNN the stronger classifier. Training took fewer epochs to achieve similar accuracy. A CNN would be the better choice for images that are more difficult to classify.