CEN/CSE 524 Machine Learning Acceleration
Lab Assignment #1

Megan Kuo, Ameya Gurjar

February 9, 2025

Part 1: Workload Analysis, Training, and Inference of an MLP on a CPU

1.a Base Network with Different Batch Sizes

- MAC: Multiply-Accumulate Operations, does not include biases.
- Inference Time: Single batch inference time.

Batch Size	Training Time (s)	Inference Time (μs)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
1	215.4	0.06	73	235,146	234,752	(1, 784)	(1, 10)
64	5.58	0.17	59	235,146	15,024,128	(64, 784)	(64, 10)
128	3.75	0.24	50	235,146	30,048,256	(128, 784)	(128, 10)

Table 1: Performance Metrics for Different Batch Sizes

Questions:

1. What is the shape of weights and activation of each layer of MLP?
2. What is the size of weights and activation of each layer of MLP? (Specify in KB, MB, or GB)
3. What is the effect of changing the batch size on **training time**, **inference time**, **accuracy**, **number of parameters**, and **number of MACs**?
4. List the precision of the inputs of the first layer, weights of each layer, and outputs of the last layer.

Answers: (for single batch)

1. **Shape of weights/activation:**
   - Input layer: Shape = 784
   - Weights (Input $\rightarrow$ Hidden 1): 784×256
   - Hidden Layer 1: Activations = 256
   - Weights (Hidden 1 $\rightarrow$ Hidden 2): 256×128
   - Hidden Layer 2: Activations = 128
   - Weights (Hidden 2 $\rightarrow$ Output): 128×10
   - Output layer: Shape = 10
2. **Size of weights/activation:**
   - Weights (Input $\rightarrow$ Hidden Layer 1): $784 \times 256 \times 4$ bytes = 784 KB
   - Weights (Hidden Layer 1 $\rightarrow$ Hidden Layer 2): $256 \times 128 \times 4$ bytes = 128 KB
   - Weights (Hidden Layer 2 $\rightarrow$ Output): $128 \times 10 \times 4$ bytes = 5 KB
   - Total Weight Size: 917 KB $\approx$ 0.9 MB
   - Activations (per sample): 3.1 KB + 1 KB + 0.5 KB + 0.04 KB $\approx$ 4.64 KB
   - Activations (batch size = 64): $\approx$ 297 KB

3. **Batch size effect: (As batch size grows...)**
   - Training Time: Generally decreases because weights are updated less frequently.
   - Inference Time: Generally increases because bigger batches do more work at once.
   - Accuracy: Can remain the same or slightly decrease with very large batches if hyperparameters (e.g., learning rate) aren't adjusted. Larger batches may lead to less frequent weight updates, which can result in slower convergence or suboptimal solutions.
   - Memory Usage: Increases for larger batches.
   - Parameters and MACs: Parameters (weights and biases) are fixed by the model's architecture. They are determined by the number of layers and neurons, not by the batch size. But number of MACs scale linearly with batch size because they represent the total computations across all samples in a batch.
4. **Precision**
   All inputs, weights, activations, and outputs are stored as 32-bit floating-point numbers (float32).
   - Input layer: float32
   - Weights (Input → Hidden 1): float32
   - Hidden Layer 1: Activations = float32
   - Weights (Hidden 1 → Hidden 2): float32
   - Hidden Layer 2: Activations = float32
   - Weights (Hidden 2 → Output): float32
   - Output layer: float32

Snapshots:

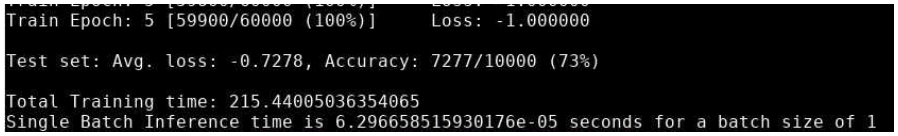

Figure 1: Simple MLP run with bath-size 1

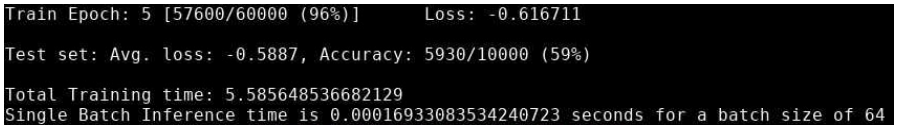

Figure 2: Simple MLP run with bath-size 64

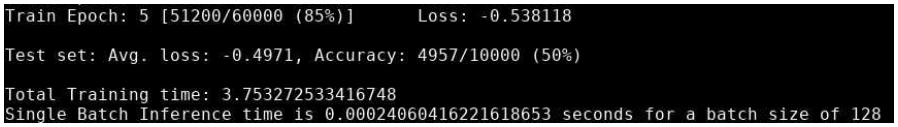

Figure 3: Simple MLP run with bath-size 128

1.b Base Network with Different Neuron Counts

Neuron Count	Training Time (s)	Inference Time (μs)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
Base (Default=256)	6.33	0.17	59	235,146	234,752	(64, 784)	(64, 10)
Smaller (128)	4.97	0.12	67	118,282	118,016	(64, 784)	(64, 10)
Larger (1024)	32.04	0.43	67	936,330	935,168	(64, 784)	(64, 10)

Table 2: Performance Metrics for Different Neuron Counts (Batch Size = 64)

Questions:

- What is the effect of increasing and decreasing neurons in the hidden layers on **training time**, **inference time**, **accuracy**, **total number of parameters**, and **total number of MACs**?

Answers:

- Training Time: Increasing the number of neurons increases training time due to more computations in forward and backward passes. Conversely, decreasing the number of neurons reduces training time.
- Inference Time: More neurons increase inference time due to additional operations required, while fewer neurons decrease inference time.
- Accuracy: Increasing neurons can improve accuracy by capturing more complex patterns, but on simple datasets like MNIST, the improvement may plateau or even decrease due to overfitting. Surprisingly, reducing neurons sometimes improves accuracy by preventing overfitting. (both 128, 1024 leads to better result than 256)
- Total Parameters and MACs: Increasing neurons increases the total number of parameters and MACs linearly, while decreasing neurons reduces them.

Snapshots:

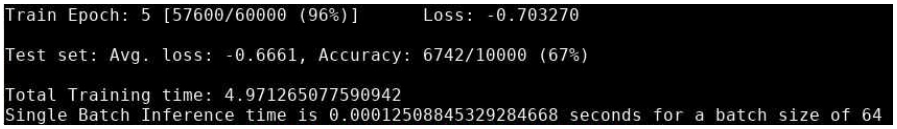

Figure 4: Simple MLP run with hidden neurons 128

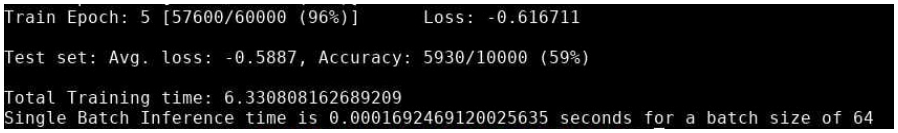

Figure 5: Simple MLP run with hidden neurons 256 (default)

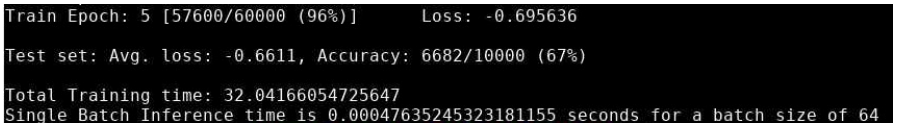

Figure 6: Simple MLP run with hidden neurons 1024

1.c Base Network with Different Layer Counts

Hidden Layers	Training Time (s)	Inference Time (μs)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
Base (default=3)	4.3	0.12	66	118,282	118,016	(64, 784)	(64, 10)
Fewer Layers (2)	3.74	0.09	75	101,770	101,632	(64, 784)	(64, 10)
More Layers (4)	5.52	0.15	65	134,794	134,400	(64, 784)	(64, 10)

Table 3: Performance Metrics for Different Layer Counts (Batch Size = 64)

Model Architecture:

```
class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP, self).__init__()

        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc_add = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.fc1(x))

        if args.layers == 3:
            x = F.relu(self.fc2(x))
        elif args.layers == 4:
            x = F.relu(self.fc2(x))
            x = F.relu(self.fc_add(x))

        x = F.relu(self.fc3(x))

        logits = self.softmax(x)
        return logits
```

Questions:

- What is the effect of increasing and decreasing the number of layers on **training time**, **inference time**, **accuracy**, **total number of parameters**, and **total number of MACs**?

Answers:

- Training Time
  - More Layers: Each extra layer introduces more parameters and operations, which slows down the training process.
  - Fewer Layers: Decreases training time as there are fewer computations and parameters to update during each iteration.
- Inference Time
  - More Layers: Increases inference time because more operations are required to process the input data through the additional layers.
  - Fewer Layers: Decreases inference time, making the model faster at making predictions.
- Accuracy
  - More Layers: Can potentially increase accuracy by capturing more complex relationships in the data. However, on a simple dataset like MNIST, the benefits may be minimal or even negative due to overfitting or training instability. Overfitting occurs when the model becomes too complex and starts memorizing the training data rather than generalizing well to unseen data.

- Fewer Layers: Reduces the model's capacity, which might lower accuracy on more complex tasks. However, for MNIST, a simpler model with fewer layers can still perform well because the dataset is relatively simple and does not require a highly complex model.

- Total Number of Parameters
  - More Layers: Increases the total number of parameters. For example, adding an extra 128×128 layer introduces $128 \times 128 + 128 = 16,512$ additional parameters (weights and biases).
  - Fewer Layers: Decreases the total number of parameters, making the model more compact and easier to store.

- Total Number of MACs (Multiply-Accumulate Operations)
  - More Layers: Increases the number of MACs linearly with each additional layer. For instance, adding a 128×128 layer increases the number of MACs by $128 \times 128 = 16,384$ per input sample.
  - Fewer Layers: Reduces the number of MACs, leading to more efficient computation and lower resource requirements.

Snapshots:

```
Train Epoch: 5 [57600/60000 (96%)]      Loss: -0.768305

Test set: Avg. loss: -0.7415, Accuracy: 7530/10000 (75%)

Total Training time: 3.74177885055542
Single Batch Inference time is 9.56885814666748e-05 seconds for a batch size of 64
```

Figure 7: Simple MLP run with 1 hidden layer

```
Train Epoch: 5 [57600/60000 (96%)]      Loss: -0.757962

Test set: Avg. loss: -0.6531, Accuracy: 6611/10000 (66%)

Total Training time: 4.314209938049316
Single Batch Inference time is 0.00012262511253356934 seconds for a batch size of 64
```

Figure 8: Simple MLP run with 2 hidden layers (default)

```
Train Epoch: 5 [57600/60000 (96%)]      Loss: -0.638324

Test set: Avg. loss: -0.6467, Accuracy: 6542/10000 (65%)

Total Training time: 5.519975423812866
Single Batch Inference time is 0.0001539287567138672 seconds for a batch size of 64
```

Figure 9: Simple MLP run with 3 hidden layers

1.4 Network Visualization

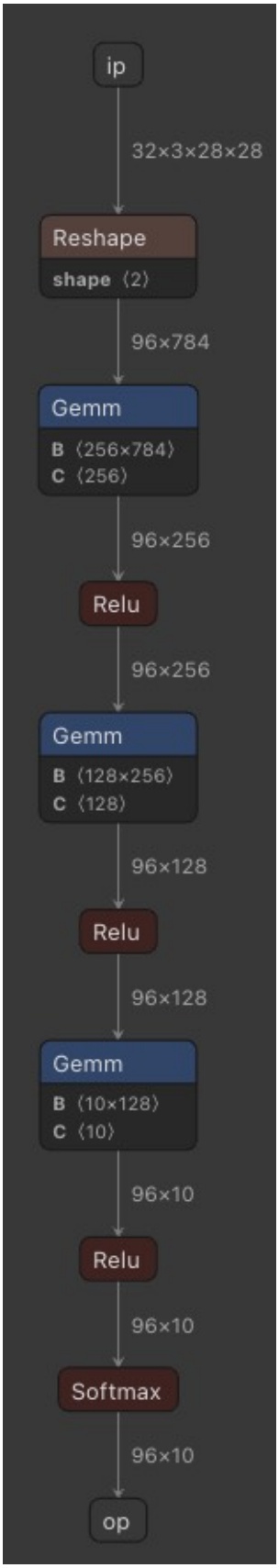

Figure 10: Visualization of the Base MLP Network in Netron

Part 2: Workload Analysis, Training, and Inference of a CNN on a CPU

2.1 Base Network with Different Batch Sizes

Batch Size	Training Time (s)	Inference Time (s)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
1	925.4421	0.0003	10	2762	18600	(1, 1, 28, 28)	(1, 10)
32	158.3924	0.0005	73	2762	595200	(32, 1, 28, 28)	(32, 10)
64	75.3266	0.0009	72	2762	1190400	(64, 1, 28, 28)	(64, 10)

Table 4: Performance Metrics for Different Batch Sizes

Questions:

- What is the shape of weights/filters and activation of each layer of CNN?
- What is the size of weights/filters and activation of each layer of CNN? (Specify in KB, MB, or GB)
- What is the effect of changing the batch size on **training time**, **inference time**, **accuracy**, **total number of parameters**, and **total number of MACs**?
- List the precision of the inputs of the first layer, weights of each layer, and outputs of the last layer.

Answers: (assuming batch_size = 1)

- Batch size does not affect the weights but the activations.

1. **Shape of weights/activation:**
   - Input Image: Shape = $(\text{batch\_size}, \text{channels}, \text{height}, \text{width}) = (1, 1, 28, 28)$
   - Filter for Conv1: Shape = $(\text{batch\_size}, \text{channels}, \text{height}, \text{width}) = (1, 1, 5, 5)$
   - Activations of Conv1: Shape = $(\text{batch\_size}, \text{channels}, \text{height}, \text{width}) = (1, 1, 24, 24)$
   - Filter for Conv2: Shape = $(\text{batch\_size}, \text{channels}, \text{height}, \text{width}) = (1, 1, 5, 5)$
   - Activations of Conv2: Shape = $(\text{batch\_size}, \text{channels}, \text{height}, \text{width}) = (1, 1, 8, 8)$
2. **Size of weights/activation:**
   - Size of weights in Conv1: Size = $(\text{batch\_size} \times \text{channels} \times \text{height} \times \text{width} \times 4bytes) = (1 \times 1 \times 5 \times 5 \times 4\text{bytes}) = 0.1\text{KB}$
   - Size of activations of Conv1: Size = $(\text{batch\_size} \times \text{channels} \times \text{height} \times \text{width} \times 4bytes) = (1 \times 1 \times 24 \times 24 \times 4\text{bytes}) = 2.304\text{KB}$
   - Size of weights in Conv2: Size = $(\text{batch\_size} \times \text{channels} \times \text{height} \times \text{width} \times 4bytes) = (1 \times 1 \times 5 \times 5 \times 4\text{bytes}) = 0.1\text{KB}$
   - Size of activations of Conv2: Size = $(\text{batch\_size} \times \text{channels} \times \text{height} \times \text{width} \times 4bytes) = (1 \times 1 \times 8 \times 8 \times 4\text{bytes}) = 0.256\text{KB}$
3. **Batch size effect:**
   - Training time generally decreases with an increase in batch size as the number of inputs the network processes in a single iteration increases, reducing the number of weight updates required for a dataset.
   - Inference time generally increases because of bigger batches because the network has to process more inputs at once.
   - With large batch sizes, we might observe a stable growth in accuracy or a stable decrease in the loss; however, it might take longer to reach an optimal solution. Conversely, for very small batches, the accuracy will fluctuate a lot, resulting in unstable training and slower convergence as the model weights are updated on highly variable gradients.
   - The total number of parameters depends on the network architecture and not the batch size; hence, it remains constant.
   - The total number of MACs linearly increases with the batch size, as MACs are the total computations performed for all the samples in a batch.
4. **Precision**
   All inputs, weights, activations, and outputs are stored as 32-bit floating-point numbers (fp32).

Snapshots:

```
Test set: Avg. loss: -0.0990, Accuracy: 990/10000 (10%)

Total Training time: 925.4421162605286
Single Batch Inference time is 0.0005103456974029541 seconds for a batch size of 1
Unsupported operator aten::max_pool2d encountered 2 time(s)
Unsupported operator aten::softmax encountered 1 time(s)
----------------------------------------
Total number of parameters: 2762
Total number of MACs: 18600
```

Figure 11: Simple CNN run with batch size 1

```
Test set: Avg. loss: -0.7319, Accuracy: 7344/10000 (73%)

Total Training time: 158.39238739013672
Single Batch Inference time is 0.006645778894424439 seconds for a batch size of 32
Unsupported operator aten::max_pool2d encountered 2 time(s)
Unsupported operator aten::softmax encountered 1 time(s)
----------------------------------------
Total number of parameters: 2762
Total number of MACs: 595200
```

Figure 12: Simple CNN run with batch size 32

```
Test set: Avg. loss: -0.7135, Accuracy: 7159/10000 (72%)

Total Training time: 75.32655668258667
Single Batch Inference time is 0.0009650814533233642 seconds for a batch size of 64
Unsupported operator aten::max_pool2d encountered 2 time(s)
Unsupported operator aten::softmax encountered 1 time(s)
----------------------------------------
Total number of parameters: 2762
Total number of MACs: 1190400
```

Figure 13: Simple CNN run with batch size 64

2.2 Base Network with Different Filter Sizes

Filter Size	Training Time (s)	Inference Time (s)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
Smaller (3x3)	239.4703	0.0027	84	3630	341536	(32, 1, 28, 28)	(32, 10)
Base (5x5)	207.6856	0.0166	73	2762	595200	(32, 1, 28, 28)	(32, 10)
Larger (7x7)	196.3217	0.0013	67	1610	842912	(32, 1, 28, 28)	(32, 10)

Table 5: Performance Metrics for Different Filter Sizes (Batch Size = 32)

Questions:

- What is the effect of increasing and decreasing the filter size on **training time**, **inference time**, **accuracy**, **total number of parameters**, and **total number of MACs**?

Answers:

- Training time decreases with an increase in filter size as bigger filters result in smaller feature maps, which reduce the size of the matrix in the next fully connected layer. Conversely, training time increases with a decrease in filter size as the resulting feature map is bigger.
- Inference time decreases with an increase in filter size as bigger filters result in smaller feature maps, which reduce the size of the fully connected layer. This results in an overall reduction in the number of parameters and hence causes faster inference. Conversely, the inference time grows with a decrease in the filter size as the fully connected layer becomes larger.
- Accuracy drops with an increase in filter size as a larger filter results in a small feature map, meaning fewer features are captured. Conversely, the accuracy increases for smaller filter sizes as more features are captured in the feature map.
- The total number of parameters is inversely dependent on the filter size. Larger filter sizes result in smaller feature maps, which result in the matrix in the fully connected layer being smaller. Conversely, smaller filter sizes result in more parameters.
- The total number of MACs is directly proportional to the filter size. The convolutional layer is a compute-intensive layer, and so the MACs are heavily dependent on the size of the filter, as the greater the filter, the more operations to perform for each output of the feature map.

Snapshots:

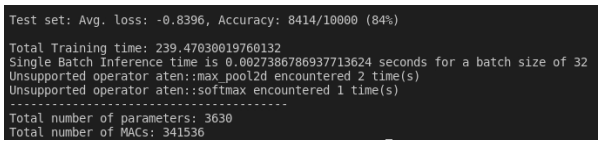

Figure 14: Simple CNN run with 3x3 filter size

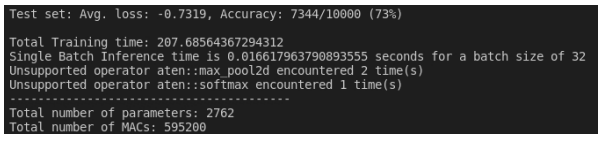

Figure 15: Simple CNN run with 5x5 filter size

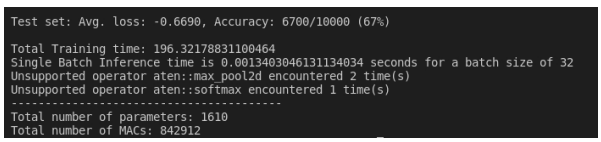

Figure 16: Simple CNN run with 7x7 filter size

2.3 Base Network with Different Layer Counts

Conv Layers	Training Time (s)	Inference Time (s)	Accuracy (%)	Total Params	Total MACs	Input Shape	Output Shape
1 Layer	49.3380	0.0007	95	15536	953600	(32, 1, 28, 28)	(32, 10)
2 Layers (Base)	56.9257	0.0012	73	2762	595200	(32, 1, 28, 28)	(32, 10)
3 Layers	850.9815	0.0025	45	1567	557952	(32, 1, 28, 28)	(32, 10)

Table 6: Performance Metrics for Different Layer Counts (Batch Size = 32)

Questions:

- What is the effect of increasing and decreasing the number of convolution layers on **training time**, **inference time**, **accuracy**, **total number of parameters**, and **total number of MACs**?

Answers:

- Training time increases with an increase in the number of convolution layers, as convolution layers are compute-intensive layers and require a lot of computation to calculate every output of the activations.
- Inference time increases with an increase in the number of convolution layers, as convolution layers are compute-intensive layers and require a lot of computation to calculate every output of the activations.
- Accuracy drops with an increase in the number of convolution layers as the final feature map generated is very small and fails to capture all the features.
- The total number of parameters decreases with an increase in the number of convolution layers. This is because convolution layers are compute-heavy and not parameter-heavy and, hence, don't cause a substantial increase in the number of parameters. However, more layers cause a decrease in feature map size, resulting in a smaller matrix in the fully connected layer, which is the parameter-heavy layer, causing the number of total parameters to go down.
- The total number of MACs decreases as the number of convolution layers increases. The first convolution layer has the highest computational cost, as it convolves with the full image. Each proceeding convolution layer works with a smaller feature map, which is almost half the size of the preceding feature map due to the pooling layer, hence with each proceeding convolution layer, there is a high reduction in MAC operations performed. Finally, the fc layer has a very small input, and hence the overall number of MACs decreases with an increase in the number of convolution layers.

Snapshots:

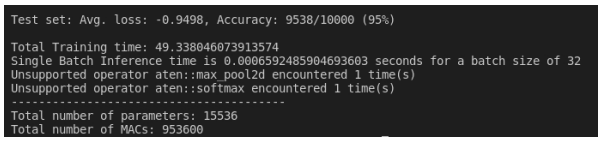

Figure 17: Simple CNN run with 1 Convolution Layer

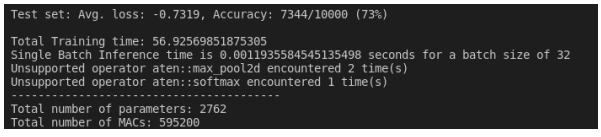

Figure 18: Simple CNN run with 2 Convolution Layers

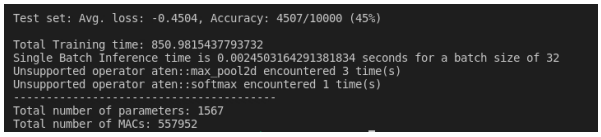

Figure 19: Simple CNN run with 3 Convolution Layers

Model Architecture:

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 1, 5) # input channels = 1, output channels = 1, kernel_size = 5
        self.pool1 = nn.MaxPool2d(2, 2) # kernel_size = 2, stride = 2
        self.conv2 = nn.Conv2d(1, 1, 5)
        self.pool2 = nn.MaxPool2d(2, 2)
        # self.conv3 = nn.Conv2d(1, 1, 2)
        # self.pool3 = nn.MaxPool2d(2, 1)
        self.fc1 = nn.Linear(1 * 4 * 4, 100)
        self.fc2 = nn.Linear(100, 10)


    def forward(self, x):
      x = self.pool1(self.conv1(x))
      x = self.pool2(self.conv2(x))
      # x = self.pool3(self.conv3(x))
      x = torch.flatten(x, 1)
      x = F.relu(self.fc1(x))
      x = F.softmax(self.fc2(x), dim = -1)
      return x
```

2.4 Network Visualization

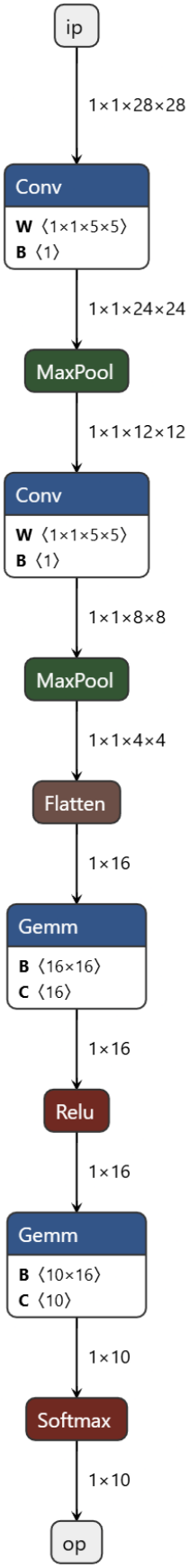

Figure 20: Visualization of the Base CNN Network in Netron

Part 3: Workload Analysis, Inference on Resnet18 CNN on a CPU

3.1 Inference with Different Batch Sizes

- Device: using CPU to inference
- Inference latency: Time to process a given batch.

Batch Size	Latency (s)	Throughput (images/s)
1	0.02	50
32	0.4	80
64	0.95	67
128	1.94	77

Table 7: Performance Metrics for Different Batch Sizes

Bar charts:

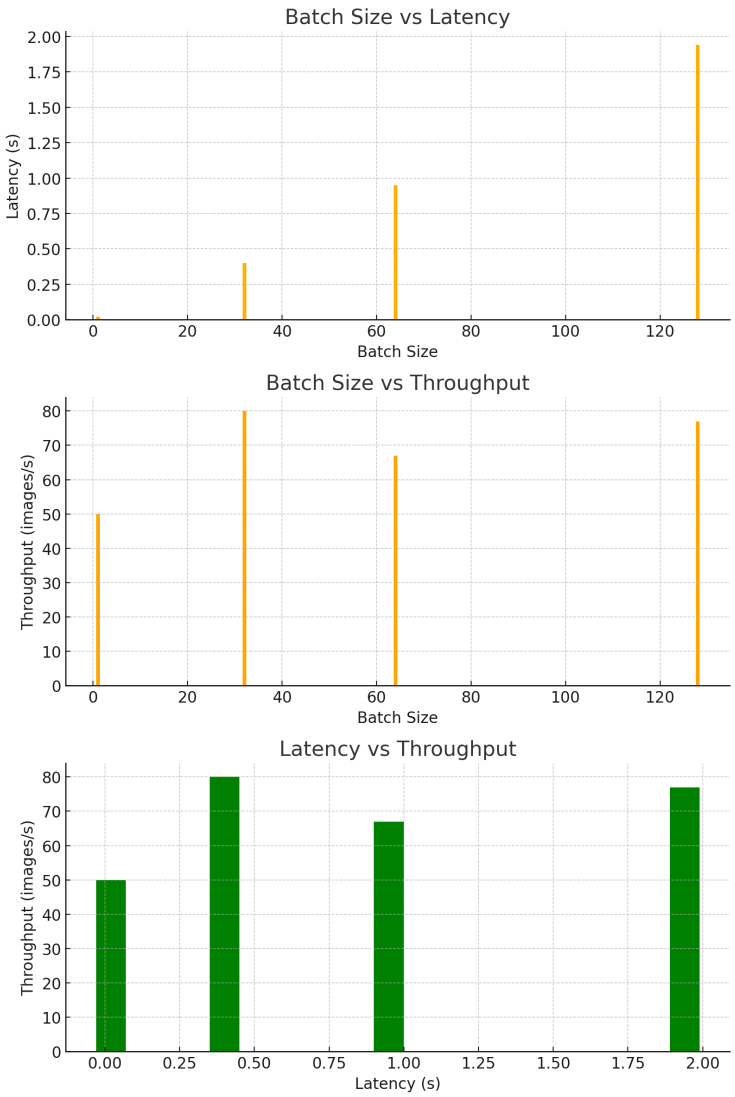

Figure 21: Resnet18 run with different batch size

Questions:

- Do you observe an improvement in the overall execution time by inferring 128 images using batch size = 128, instead of inferring 128 images back-to-back using batch size = 1?
- What are the total number of parameters and total number of MACs in the network?
- Is the memory consumed by activation higher than weights?

Answers:

- Yes, inferring 128 images in a single batch (batch size = 128) is significantly more efficient than processing them one-by-one (batch size = 1) due to improved hardware utilization. This leads to higher throughput because more images are processed per second, resulting in a lower overall execution time for the entire batch.
- Parameters and MACs:
  - Parameters: A standard ResNet18 has approximately 11.7 million parameters. (11,689,512)
  - MACs: For a 224×224 input, ResNet18 performs roughly 1.8 billion MACs per forward pass. Since MNIST images are 28×28 (and often with a single channel), the MAC count will be significantly lower—roughly estimated by scaling down by a factor of $(28/224)^2 \approx 1/64$, which gives around 28 million MACs per inference pass.
- It depends on the scenario:
  - Training: Typically, activations consume more memory than weights because intermediate outputs (activations) for every layer must be stored for backpropagation, and this memory scales with the batch size.
  - Inference: For inference, activations are computed on the fly and may not be stored persistently. However, if you use large batches or if the network is very deep, the temporary activation memory can still be significant. In many deep learning applications, especially during training, the activation memory can exceed that of the weights.

Snapshots:

```
(lab1) [megankuo@sc005:/scratch/megankuo/Project1/lab1_code]$ python part3/part3.py --batch-size 1
Using cpu for inference
Using cache found in /home/megankuo/.cache/torch/hub/NVIDIA_DeepLearningExamples_torchhub
Inference time for batchsize 1 is 0.014925580024719238
```

Figure 22: Resnet18 run with batch size 1

```
(lab1) [megankuo@sc005:/scratch/megankuo/Project1/lab1_code]$ python part3/part3.py --batch-size 32
Using cpu for inference
Using cache found in /home/megankuo/.cache/torch/hub/NVIDIA_DeepLearningExamples_torchhub
Inference time for batchsize 32 is 0.40413406133651736
```

Figure 23: Resnet18 run with batch size 32

```
(lab1) [megankuo@sc005:/scratch/megankuo/Project1/lab1_code]$ python part3/part3.py --batch-size 64
Using cpu for inference
Using cache found in /home/megankuo/.cache/torch/hub/NVIDIA_DeepLearningExamples_torchhub
Inference time for batchsize 64 is 0.9447407460212708
```

Figure 24: Resnet18 run with batch size 64

```
(lab1) [megankuo@sc005:/scratch/megankuo/Project1/lab1_code]$ python part3/part3.py --batch-size 128
Using cpu for inference
Using cache found in /home/megankuo/.cache/torch/hub/NVIDIA_DeepLearningExamples_torchhub
Inference time for batchsize 128 is 1.9479578137397766
```

Figure 25: Resnet18 run with batch size 128

3.2 Network Visualization

Layer	Weight Shape	Activation Shape
Conv1	(64, 3, 7, 7)	(64, 64, 112, 112)
Conv2	(64, 64, 3, 3)	(64, 64, 56, 56)

Table 8: Convolutional Layer Details for Stage 1

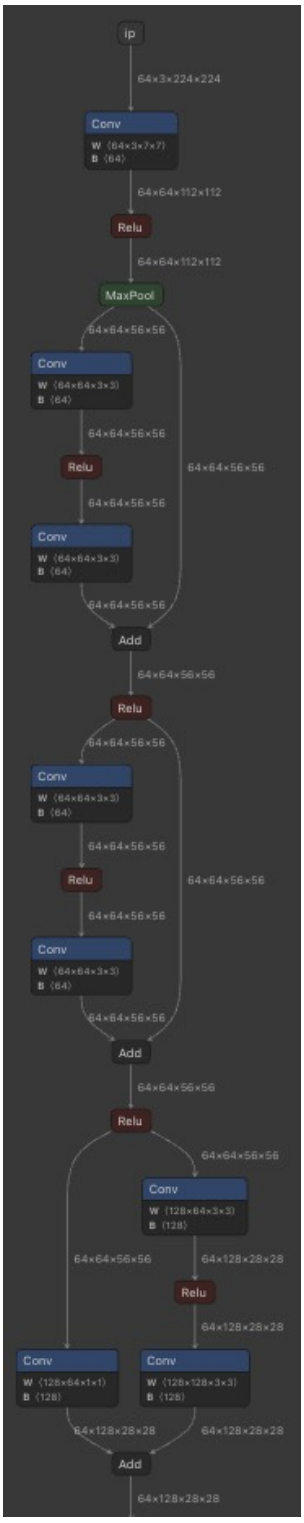

Figure 26: ONNX model visualization

3.3 Convolution layers

Layer Name	Input Shape	Output Shape	Weight Shape	Weight Size (bytes)	Activation Size (bytes)
/conv1/Conv	[64, 3, 224, 224]	[64, 64, 112, 112]	[64, 3, 7, 7]	37632	205520896
/layer1/layer1.0/conv1/Conv	[64, 64, 56, 56]	[64, 64, 56, 56]	[64, 64, 3, 3]	147456	51380224
/layer1/layer1.0/conv2/Conv	[64, 64, 56, 56]	[64, 64, 56, 56]	[64, 64, 3, 3]	147456	51380224
/layer1/layer1.1/conv1/Conv	[64, 64, 56, 56]	[64, 64, 56, 56]	[64, 64, 3, 3]	147456	51380224
/layer1/layer1.1/conv2/Conv	[64, 64, 56, 56]	[64, 64, 56, 56]	[64, 64, 3, 3]	147456	51380224
/layer2/layer2.0/conv1/Conv	[64, 64, 56, 56]	[64, 128, 28, 28]	[128, 64, 3, 3]	294912	25690112
/layer2/layer2.0/conv2/Conv	[64, 128, 28, 28]	[64, 128, 28, 28]	[128, 128, 3, 3]	589824	25690112
/layer2/layer2.0/downsample/downsample.0/Conv	[64, 64, 56, 56]	[64, 128, 28, 28]	[128, 64, 1, 1]	32768	25690112
/layer2/layer2.1/conv1/Conv	[64, 128, 28, 28]	[64, 128, 28, 28]	[128, 128, 3, 3]	589824	25690112
/layer2/layer2.1/conv2/Conv	[64, 128, 28, 28]	[64, 128, 28, 28]	[128, 128, 3, 3]	589824	25690112
/layer3/layer3.0/conv1/Conv	[64, 128, 28, 28]	[64, 256, 14, 14]	[256, 128, 3, 3]	1179648	12845056
/layer3/layer3.0/conv2/Conv	[64, 256, 14, 14]	[64, 256, 14, 14]	[256, 256, 3, 3]	2359296	12845056
/layer3/layer3.0/downsample/downsample.0/Conv	[64, 128, 28, 28]	[64, 256, 14, 14]	[256, 128, 1, 1]	131072	12845056
/layer3/layer3.1/conv1/Conv	[64, 256, 14, 14]	[64, 256, 14, 14]	[256, 256, 3, 3]	2359296	12845056
/layer3/layer3.1/conv2/Conv	[64, 256, 14, 14]	[64, 256, 14, 14]	[256, 256, 3, 3]	2359296	12845056
/layer4/layer4.0/conv1/Conv	[64, 256, 14, 14]	[64, 512, 7, 7]	[512, 256, 3, 3]	4718592	6422528
/layer4/layer4.0/conv2/Conv	[64, 512, 7, 7]	[64, 512, 7, 7]	[512, 512, 3, 3]	9437184	6422528
/layer4/layer4.0/downsample/downsample.0/Conv	[64, 256, 14, 14]	[64, 512, 7, 7]	[512, 256, 1, 1]	524288	6422528
/layer4/layer4.1/conv1/Conv	[64, 512, 7, 7]	[64, 512, 7, 7]	[512, 512, 3, 3]	9437184	6422528
/layer4/layer4.1/conv2/Conv	[64, 512, 7, 7]	[64, 512, 7, 7]	[512, 512, 3, 3]	9437184	6422528

Part 4: Workload Analysis, Inference on a Transformer Network on a CPU

4.1 Inference with Different Batch Sizes

Batch Size	Latency (s)	Throughput (inferences/s)
1	0.0645	15.5039
8	0.3734	21.4247
16	0.6388	25.0469
32	1.2615	25.3666

Table 9: Performance Metrics for Different Batch Sizes

Bar charts:

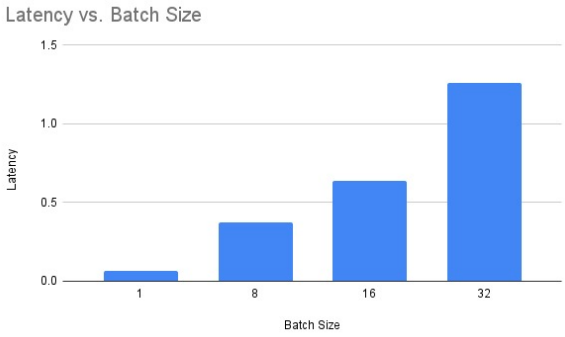

Figure 27: Latency v.s. Batch size

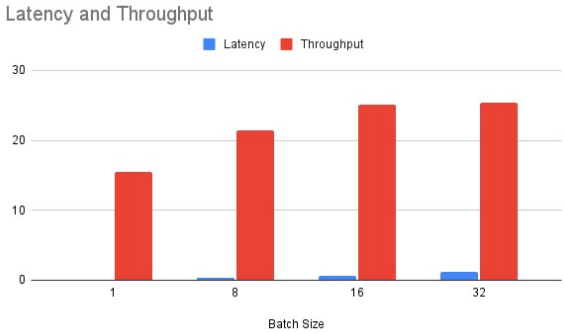

Figure 28: Latency v.s. Throughput

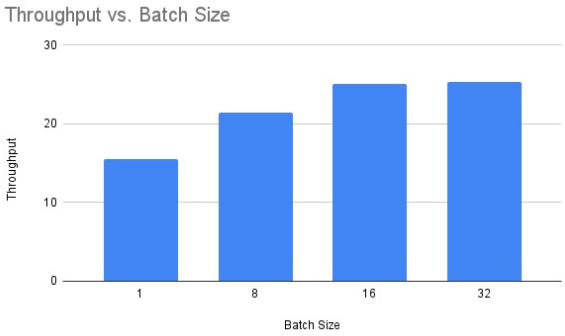

Figure 29: Throughput v.s. Batch size

4.2 Questions:

- What are the total number of parameters and MACs in the network?

Answers:

- The total number of parameters in the "bert-base-cased" model is 107.721M.
- The total number of MACs for batch size 1 is 10,882M; for batch size 8, it is 87,053M; for batch size 16, it is 174,107M; and for batch size 32, it is 348,213M.

4.3 Network Visualization

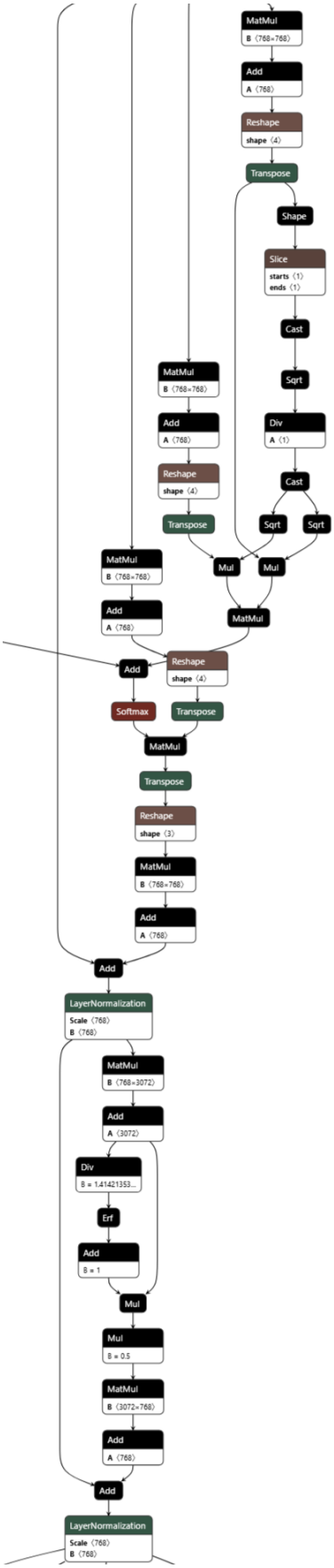

Figure 30: Visualization of the First Block of BERT-Base in Netron

4.4 Questions

- How many GEMM operations are present in the first block?
- List all the dimensions and names of the GEMM operators from the first block.
- What is the d-model (embedding dimensions) size and shape?
- How many heads are present in the network?

Answers:

- There are a total of 8 GEMM operations in the first block of the model.

Sr. No.	Layer Type	GEMM Layer Name	Input Shape	Weight Shape	Output Shape
1.	MatMul	/bert/encoder/layer.0/attention/self/query/MatMul	[32, 128, 768]	[768, 768]	[32, 128, 768]
2.	Add	/bert/encoder/layer.0/attention/self/query/Add	[32, 128, 768]	[768]	[32, 128, 768]
3.	MatMul	/bert/encoder/layer.0/attention/self/key/MatMul	[32, 128, 768]	[768, 768]	[32, 128, 768]
4.	Add	/bert/encoder/layer.0/attention/self/key/Add	[32, 128, 768]	[768]	[32, 128, 768]
5.	MatMul	/bert/encoder/layer.0/attention/self/value/MatMul	[32, 128, 768]	[768, 768]	[32, 128, 768]
6.	Add	/bert/encoder/layer.0/attention/self/value/Add	[32, 128, 768]	[768]	[32, 128, 768]
7.	MatMul	/bert/encoder/layer.0/attention/self/MatMul	[32, 128, 768]	[768, 768]	[32, 128, 128]
8.	Div	/bert/encoder/layer.0/attention/self/Div	[32, 128, 128]	[1]	[32, 128, 128]
9.	Sqrt	/bert/encoder/layer.0/attention/self/Sqrt	[32, 128, 128]	[1]	[32, 128, 128]
10.	Mul	/bert/encoder/layer.0/attention/self/Mul	[32, 128, 128]	[1]	[32, 128, 128]
11.	MatMul	/bert/encoder/layer.0/attention/self/output/MatMul_1	[32, 128, 128]	[32, 128, 768]	[32, 128, 768]
12.	MatMul	/bert/encoder/layer.0/attention/output/dense/MatMul	[32, 128, 768]	[768, 768]	[32, 128, 768]
13.	Add	/bert/encoder/layer.0/attention/output/dense/Add	[32, 128, 768]	[768]	[32, 128, 768]
14.	MatMul	/bert/encoder/layer.0/intermediate/dense/MatMul	[32, 128, 768]	[768, 3072]	[32, 128, 3072]
15.	Add	/bert/encoder/layer.0/intermediate/dense/Add	[32, 128, 3072]	[3072]	[32, 128, 3072]
16.	MatMul	/bert/encoder/layer.0/output/dense/MatMul	[32, 128, 3072]	[3072, 768]	[32, 128, 768]
17.	Add	/bert/encoder/layer.0/output/dense/Add	[32, 128, 768]	[768]	[32, 128, 768]

- The d-model shape for the model is [batch_size, seq_length, hidden_size] = [32, 128, 768]
- The bert-base model has a total of 12 attention heads.

Snapshots:

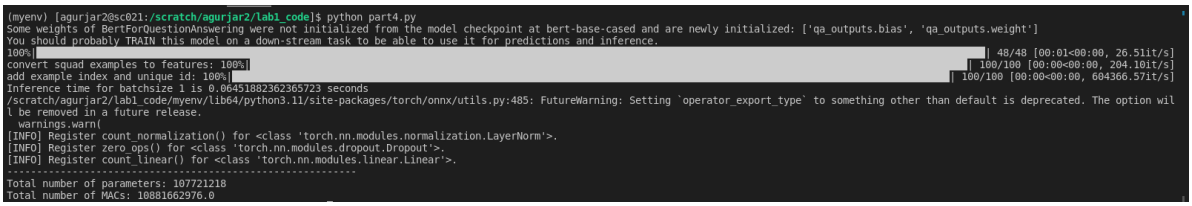

Figure 31: BERT run with batch size 1

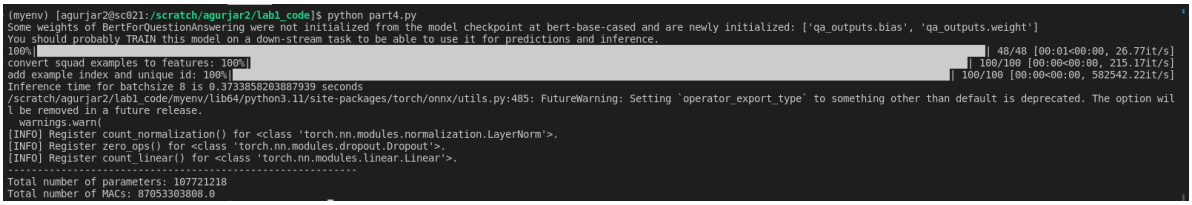

Figure 32: BERT run with batch size 8

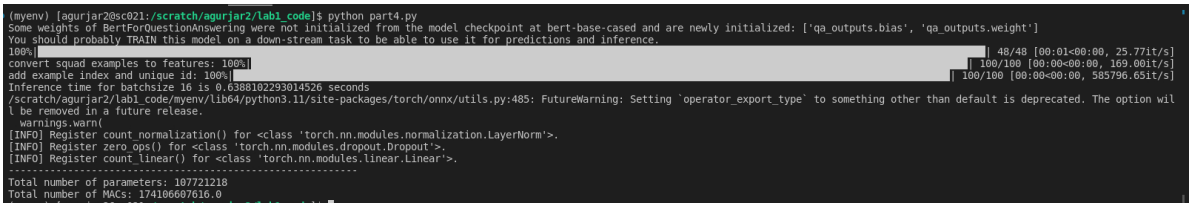

Figure 33: BERT run with batch size 16

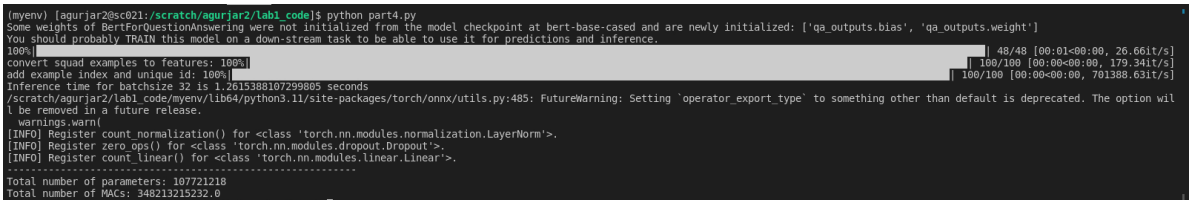

Figure 34: BERT run with batch size 32

Conclusion

In this lab, we explored the impact of batch size, neuron count, and layer count on the performance of MLP and CNN models. We observed that increasing batch size generally improves throughput but may increase latency. Additionally, increasing the number of neurons or layers tends to increase computational complexity and memory usage. These insights will help us design more efficient neural networks in the future.