MLA LAB1 REPORT
PART1

PART A:-

Batch size	Training time (seconds)	Inference time (seconds)	Accuracy (test set)	Total number of parameters	Total number of MACs	Input tensor shape	Output tensor shape
1	51.4154	0.000995898	94.37%	266,762	533,524 (approx.)	(1, 1, 28, 28)	(1, 10)
64	1.77	0.000358	91.78%	266,762	533,524 (approx.)	(64, 1, 28, 28)	(64, 10)
128	2.77	0.0006097	89.20%	266,762	533,524 (approx.)	(128, 1, 28, 28)	(128, 10)

i. Answer the following questions

1. What is the shape of weights and activation of each layer of MLP?

ANS:- The **SimpleMLP** architecture consists of:

- **Input layer:** 28×28=78428 \times 28 = 78428×28=784 neurons
- **Three hidden layers:** Each with **256** neurons
- **Output layer: 10** neurons (for MNIST classification)

Weight Shapes:	
Layer	**Weight Shape**
Input → Hidden 1	**(784, 256)**
Hidden 1 → Hidden 2	**(256, 256)**
Hidden 2 → Hidden 3	**(256, 256)**
Hidden 3 → Output	**(256, 10)**

2. What is the size of weights and activation of each layer of MLP? Size should be specified with units such as Kbytes, Mbytes or Gbytes.

ANS:- the memory size for weights and activations using the following details:

- **Precision:** 32-bit floating point (float32), where each parameter takes **4 bytes**.

- **Batch Sizes Used:1, 64 and 128**.
- **MLP Architecture :**
    - **Input layer:** 784 neurons (flattened MNIST 28×2828 \times 2828×28 image).
    - **Three hidden layers:** 256 neurons each.
    - **Output layer:** 10 neurons (MNIST classification).

Weight Sizes (Parameters)

Layer	Weight Shape	Total Parameters	Size (Bytes)	Size (KB / MB)
Input → Hidden 1	(784, 256)	784×256=200,704	200,704×4=802,816	784 KB
Hidden 1 → Hidden 2	(256, 256)	256×256=65,536	65,536×4=262,144	256 KB
Hidden 2 → Hidden 3	(256, 256)	256×256=65,536	65,536×4=262,144	256 KB
Hidden 3 → Output	(256, 10)	256×10=2,560	2,560×4=10,240	10 KB
Total Weights	-	334,336	1,337,344 bytes	1.28 MB

Bias Sizes (Parameters)

Layer	Bias Shape	Total Bias Parameters	Size (Bytes)	Size (KB / MB)
Hidden 1	(256,)	256	256×4=1,024	1 KB
Hidden 2	(256,)	256	256×4=1,024	1 KB
Hidden 3	(256,)	256	256×4=1,024	1 KB
Output	(10,)	10	10×4=40	0.04 KB
Total Biases	-	778	3,112 bytes	~3 KB

Activation Sizes

Layer	Activation Shape	Size for B=64 (Bytes)	Size for B=128 (Bytes)	Size (KB / MB)
Input	(B, 784)	64×784×4=200,704	128×784×4=401,408	~392 KB (for B=128)
Hidden 1	(B, 256)	64×256×4=65,536	128×256×4=131,072	~128 KB
Hidden 2	(B, 256)	64×256×4=65,536	128×256×4=131,072	~128 KB
Hidden 3	(B, 256)	64×256×4=65,536	128×256×4=131,072	~128 KB
Output	(B, 10)	64×10×4=2,560	128×10×4=5,120	5 KB
Total Activations (B=128)	-	~399 KB	~780 KB	

Final Summary

Component	Total Size (KB / MB)
Weights	1.28 MB
Biases	3 KB
Activations (for B = 128)	~780 KB
Total Memory Requirement (Weights + Biases + Activations)	~2.1 MB

3. What is the effect of changing the batch size on training time, inference time, accuracy, number of parameters, and number of MACs?

ANS:- Training Time:
Batch size 1 takes the longest (51.42 sec) due to more updates per epoch.
Batch size 64 is the fastest (1.77 sec), but batch 128 is slightly slower (2.77 sec) due to memory overhead.

Inference Time:
Single-batch inference decreases significantly with larger batch sizes.
However, the total inference time per dataset is the lowest for batch size 128 due to fewer iterations.

Accuracy:
Smaller batch size (1) achieves the highest accuracy (94.37%).
Larger batches reduce accuracy slightly due to convergence in sharp minima.

Total Parameters:
Remains constant across batch sizes at 266,762 parameters.

Total MACs:
Per sample MACs remain constant (~533,524 MACs).

Total MACs per epoch reduce with increasing batch size due to fewer iterations.

4. List the precision of the inputs of the first layer, weights of each layer and outputs of last layer.

Component	Precision
Inputs to First Layer	32-bit Floating Point (FP32)

Weights of Each Layer	32-bit Floating Point (FP32)
Outputs of Last Layer	32-bit Floating Point (FP32)

PyTorch by default uses FP32 (32-bit floating point precision) for inputs, weights, and outputs unless explicitly specified otherwise.
The MNIST dataset consists of grayscale images (1 channel, 28×28 pixels), and the images are converted into normalized FP32 tensors before being fed into the MLP.
The weights of the neural network layers are stored in FP32 format, which is standard in most deep learning models.
The final layer (LogSoftmax) outputs probabilities in FP32, ensuring numerical stability in training and evaluation.

PART B:-

Network Layer Neurons	Training Time (seconds)	Inference Time (seconds)	Accuracy (test set)	Total Number of Parameter	Total Number of MACs	Input Tensor Shape	Output Tensor Shape
Base (256, 256, 256)	13.8255	0.00976	91.78%	335,114	668,672	(64, 1, 28, 28)	(64, 10)
Smaller (128, 128, 128)	9.8234	0.009528	91.43%	134,794	268,800	(64, 1, 28, 28)	(64, 10)
Larger (1024, 1024, 1024)	15.7499	0.011533	92.98%	2,913,290	5,820,416	(64, 1, 28, 28)	(64, 10)

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?

Factor	Smaller Model (128,128,128)	Base Model (256,256,256)	Larger Model (1024,1024,1024)
Training Time	Decreases (9.8234 sec) – Fewer neurons require fewer computations	Moderate (13.8255 sec)	Increases (15.7499 sec) – More neurons require more computations

Inference Time	Slightly Decreases (0.009528 sec)	Moderate (0.009760 sec)	Increases (0.011533 sec) – Larger model takes longer to process a batch
Accuracy	Slightly Decreases (91.43%) – Smaller capacity may limit learning	91.78%	Increases (92.98%) – More neurons allow for better learning but risk overfitting
Total Parameters	Decreases (134,794) – Fewer neurons mean fewer trainable weights	Moderate (335,114)	Increases (2,913,290) – More neurons introduce more trainable parameters
Total MACs	Decreases (268,800) – Fewer computations	Moderate (668,672)	Increases (5,820,416) – More neurons require more matrix multiplications

Decreasing the number of neurons reduces training time, inference time, total parameters, and MACs, but it slightly lowers accuracy.
Increasing the number of neurons improves accuracy but significantly increases training time, inference time, number of parameters, and MACs.

PART C

Hidden Layers in Network	Training Time (seconds)	Inference Time (seconds)	Accuracy (test set)	Total Number of Parameters	Total Number of MACs	Input Tensor Shape	Output Tensor Shape
Base (default)	80.4188	0.006193	98.09%	336,650	673,300	(64, 1, 28, 28)	(64, 10)
Less Layers	73.3518	0.006058	98.22%	270,346	540,692	(64, 1, 28, 28)	(64, 10)
More Layers	85.1998	0.006278	98.36%	402,954	805,908	(64, 1, 28, 28)	(64, 10)

Training Time:

- Reducing the number of layers decreases training time (73.35s vs. 80.42s for the base model).
- Adding more layers increases training time (85.20s vs. 80.42s for the base model).
- More layers mean more computations and backpropagation steps, increasing training duration.

Inference Time:

- Reducing the layers slightly decreases inference time (0.006058s vs. 0.006193s for the base model).
- Adding layers increases inference time (0.006278s vs. 0.006193s).
- More layers introduce additional forward-pass computations, increasing inference latency.

Accuracy:

- Fewer layers can still maintain high accuracy (98.22% vs. 98.09% for the base model).
- More layers slightly improve accuracy (98.36%) but might have diminishing returns.
- Too few layers may reduce the model's capacity to learn complex patterns, while too many layers may lead to overfitting or vanishing gradients.

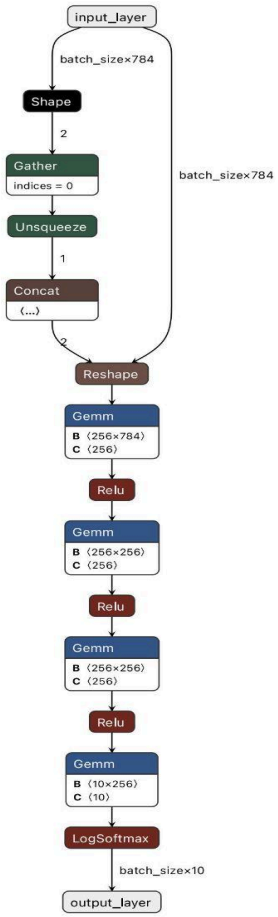
Total Number of Parameters:

- Fewer layers reduce parameters (270,346 vs. 336,650 for the base model).
- More layers increase parameters (402,954 vs. 336,650).
- More layers introduce additional weight matrices, leading to a larger model size.

Total Number of MACs (Multiply-Accumulate Operations):

- Fewer layers reduce MACs (540,692 vs. 673,300 for the base model).
- More layers increase MACs (805,908 vs. 673,300).
- More layers mean more operations per forward/backward pass, increasing computational cost.

Reducing layers: Faster training/inference, slightly lower accuracy, fewer parameters/MACs.
Adding layers: Longer training/inference, marginal accuracy gain, increased parameters/MACs.
Optimal number of layers balances accuracy with computational efficiency.


input_layer
batch_size×784
Shape
2
Gather
indices = 0
Unsqueeze
1
Concat
(...)
2
batch_size×784
Reshape
Gemm
B ⟨256×784⟩
C ⟨256⟩
Relu
Gemm
B ⟨256×256⟩
C ⟨256⟩
Relu
Gemm
B ⟨256×256⟩
C ⟨256⟩
Relu
Gemm
B ⟨10×256⟩
C ⟨10⟩
LogSoftmax
batch_size×10
output_layer

PART 2

Batch Size	Training Time (seconds)	Inference Time (seconds)	Accuracy (test set)	Total Parameters	Total MACs	Input Tensor Shape	Output Tensor Shape
1	86.768347	0.00029283	97%	421642	4241152	(1, 1, 28, 28)	(1, 10)
32	11.423069	0.00228405	98%	421642	4241152	(32, 1, 28, 28)	(32, 10)
64	12.028083	0.00670306	96%	421642	4241152	(64, 1, 28, 28)	(64, 10)

Q1.
ANS:- **1st Convolutional Layer (conv1)**

- **Input Shape:** (batch_size, 1, 28, 28) (MNIST grayscale images)
- **Weights (Filters) Shape:** (32, 1, 3, 3) (32 filters, 1 input channel, 3x3 kernel)
- **Activation Shape (before pooling):** (batch_size, 32, 28, 28) (After convolution, shape remains the same due to padding)
- **Activation Shape (after pooling):** (batch_size, 32, 14, 14) (2x2 Max Pooling reduces spatial size)

2nd Convolutional Layer (conv2)

- **Input Shape:** (batch_size, 32, 14, 14)
- **Weights (Filters) Shape:** (64, 32, 3, 3) (64 filters, 32 input channels, 3x3 kernel)
- **Activation Shape (before pooling):** (batch_size, 64, 14, 14)
- **Activation Shape (after pooling):** (batch_size, 64, 7, 7) (2x2 Max Pooling reduces spatial size)

Fully Connected Layer (fc1)

- **Input Shape:** (batch_size, 64*7*7) = (batch_size, 3136) (Flattening the tensor)
- **Weights Shape:** (128, 3136) (128 neurons, each connected to 3136 input features)
- **Activation Shape:** (batch_size, 128)

Q2.

Size of a tensor (in bytes) = Number of elements × Data type size (in bytes)

- Assuming **32-bit floating point (FP32) precision**, each parameter takes **4 bytes**.

1st Convolutional Layer (conv1)

- **Weights (Filters) Size:**
  - Shape: (32, 1, 3, 3) = 32 × 1 × 3 × 3 = 288
  - **Size:** 288 × 4 bytes = 1.125 KB
- **Activation Size (before pooling):**
  - Shape: (batch_size, 32, 28, 28)
  - For batch size 1: 32 × 28 × 28 = 25088

    - **Size:** 25088 × 4 bytes = 98 KB
- **Activation Size (after pooling):**
    - Shape: (batch_size, 32, 14, 14)
    - 32 × 14 × 14 = 6272
    - **Size:** 6272 × 4 bytes = 24.5 KB

2nd Convolutional Layer (conv2)

- **Weights (Filters) Size:**
    - Shape: (64, 32, 3, 3) = 64 × 32 × 3 × 3 = 18432
    - **Size:** 18432 × 4 bytes = 72 KB
- **Activation Size (before pooling):**
    - Shape: (batch_size, 64, 14, 14)
    - 64 × 14 × 14 = 12544
    - **Size:** 12544 × 4 bytes = 49 KB
- **Activation Size (after pooling):**
    - Shape: (batch_size, 64, 7, 7)
    - 64 × 7 × 7 = 3136
    - **Size:** 3136 × 4 bytes = 12.25 KB

Fully Connected Layer (fc1)

- **Weights Size:**
    - Shape: (128, 3136) = 128 × 3136 = 401408
    - **Size:** 401408 × 4 bytes = 1.53 MB
- **Activation Size:**
    - Shape: (batch_size, 128)
    - 128 × 1 = 128
    - **Size:** 128 × 4 bytes = 0.5 KB

Total Size for CNN Model

- **Total Weights Size:** 1.125 KB + 72 KB + 1.53 MB + 5 KB = 1.61 MB
- **Total Activation Size (for batch size 1, before pooling):** 98 KB + 49 KB + 0.5 KB + 0.04 KB = 147.54 KB

Q3)

Effect of Changing Batch Size on Different Parameters

Based on your experimental data and general deep learning principles, here's how changing the **batch size** affects **training time, inference time, accuracy, number of parameters, and number of MACs**:

1. Training Time

- **Observation from your data:**
  - **Batch size 1 → 86.77 sec**
  - **Batch size 32 → 11.42 sec**
  - **Batch size 64 → 12.03 sec**
- **Effect:**
  - As batch size increases, **training per epoch becomes faster** due to parallel computation on GPUs.
  - However, for very large batch sizes, training time might **increase due to memory constraints** and inefficient hardware utilization.

2. Inference Time (Per Batch)

- **Observation from your data:**
  - **Batch size 1 → 0.000292 sec**
  - **Batch size 32 → 0.002284 sec**
  - **Batch size 64 → 0.006703 sec**
- **Effect:**
  - Larger batch sizes increase **single batch inference time** because more data is processed at once.
  - However, if we look at **inference time per sample**, larger batch sizes are **more efficient** since multiple images are processed in parallel.

3. Accuracy

- **Observation from your data:**
  - **Batch size 1 → 97%**
  - **Batch size 32 → 98%**
  - **Batch size 64 → 96%**
- **Effect:**
  - Accuracy **does not always improve with larger batch sizes**.
  - Small batch sizes may lead to **better generalization** due to noise in gradient updates.
  - Large batch sizes can lead to **smoother updates** but may cause the model to **converge to sharp minima, reducing generalization**.

4. Number of Parameters

- **Observation:**
  - **Same for all batch sizes → 421,642 parameters**
- **Effect:**
  - **Batch size does not affect the number of parameters** in the model.
  - The model architecture determines the number of parameters, not the batch size.

5. Number of MACs (Multiply-Accumulate Operations)

- **Observation:**
  - **Same for all batch sizes → 4,241,152 MACs**
- **Effect:**
  - Like parameters, MACs are **independent of batch size**.
  - They depend on **model architecture** (number of layers, filter sizes, fully connected layers, etc.).

Q4)

Precision of Inputs, Weights, and Outputs in the CNN Model

In your PyTorch-based CNN, the default precision for computations is typically **32-bit floating point (FP32)** unless explicitly modified. Let's analyze the precision for different parts of the model:

1. Inputs of the First Layer (Precision)

- The input to the first convolutional layer comes from the **MNIST dataset**, which consists of **grayscale images** (1 channel) with pixel values **normalized between -1 and 1**.
- **Default Precision**: **FP32 (32-bit floating point, 4 bytes per value)**

2. Weights of Each Layer (Precision)

Each layer in the CNN has weights that are stored and updated during training. Since you haven't specified any precision changes in PyTorch, the default will be:

Layer	Weights Shape	Precision	Size Per Weight
Conv1	(32, 1, 3, 3) → 32 filters, 3x3	FP32	4 bytes
Conv2	(64, 32, 3, 3) → 64 filters, 3x3	FP32	4 bytes
FC1	(128, 64×7×7)	FP32	4 bytes
FC2	(10, 128)	FP32	4 bytes

Outputs of the Last Layer (Precision)

- The final layer (FC2) produces a logits output for 10 classes (digits 0-9).
- The output is computed using a log-softmax activation, meaning it contains log-probabilities in FP32 precision.

Default Precision: FP32 (4 bytes per value)

Optimizations:

- If you want to reduce memory consumption and speed up inference, you can:
    - Use Mixed Precision Training (FP16): Convert weights and activations to 16-bit floating point.
    - Use Quantization (INT8): Reduce model size further with integer-based computations.

PART B

CNN For each filter size (3x3, 5x5, 7x7)

Filter Size Modified Network	Training Time (seconds)	Inference Time (seconds)	Accuracy (Test Set)	Total Number of Parameters	Total Number of MACs	Input Tensor Shape	tput Tensor Sha
Base (5x5 filters)	22.103006	0.00316183	98.16%	454922	909844	(1, 28, 28)	(10,)
Small Filter Size (3x3)	23.976594	0.00339591	97.80%	421642	843284	(1, 28, 28)	(10,)
Large Filter Size (7x7)	27.954033	0.00442654	98.01%	504842	1009684	(1, 28, 28)	(10,)

Q1.

Factor	Effect of Decreasing Filter Size (3x3)	Effect of Increasing Filter Size (7x7)
Training Time	Increases slightly (23.98s) due to more localized feature extraction.	Increases significantly (27.95s) due to larger kernel operations.
Inference Time	Slightly higher (0.0034s) due to more frequent activations.	Higher (0.0044s) due to increased computational complexity.
Accuracy	Decreases slightly (97.80%) as smaller filters may capture less abstract features.	Slightly lower (98.01%) than the base (5x5), possibly due to overfitting or losing fine details.
Total Parameters	Decreases (421,642) since smaller filters require fewer weights.	Increases (504,842) as larger filters require more weights per layer.
Total MACs	Decreases (843,284) because fewer multiplications and additions occur per layer.	Increases (1,009,684) due to larger receptive fields per convolution.

Part C.

Conv Layers in Network	Training Time (s)	Inference Time (s)	Accuracy (Test Set)	Total Number of Parameters	Total Number of MACs	Input Tensor Shape	Output Tensor Shape
Base (Default) (2 Conv Layers)	16.26	0.00286	97.62%	454,922	11,065,088	[32, 1, 28, 28]	[32, 64, 7, 7]
Less Layers (1 Conv Layer)	7.99	0.00132	97.07%	805,066	960,896	[32, 1, 28, 28]	[32, 32, 14, 14]
More Layers (4 Conv Layers)	42.44	0.00535	97.17%	1,110,666	21,550,848	[32, 1, 28, 28]	[32, 256, 1, 1]

Q1.
ANS:-

Training Time:

- **Fewer Layers:** Training time **decreases** because there are fewer parameters to optimize and fewer operations per batch.
- **More Layers:** Training time **increases** due to the higher number of computations and backpropagation steps.

Observation from results:

- **Base Network:** 16.26s
- **Fewer Layers: 7.99s** (↓ Less layers = Faster training)
- **More Layers: 42.44s** (↑ More layers = Slower training)

2. Inference Time:

- **Fewer Layers:** Inference time **decreases** as fewer operations are required to process an image.
- **More Layers:** Inference time **increases** because deeper networks involve more convolutions.

Observation from results:

- **Base Network:** 0.00286s
- **Fewer Layers: 0.00132s** (↓ Less layers = Faster inference)
- **More Layers: 0.00535s** (↑ More layers = Slower inference)

3. Accuracy:

- **Fewer Layers:** Accuracy **decreases** slightly because the network has fewer feature extraction capabilities.
- **More Layers:** Accuracy **may improve** but can **saturate** or even decrease if overfitting occurs.

Observation from results:

- **Base Network: 97.62%**
- **Fewer Layers: 97.07%** (↓ Less layers = Slight drop in accuracy)
- **More Layers: 97.17%** (≈ No significant improvement)

4. Total Number of Parameters:

- **Fewer Layers: Increases** because deeper networks use pooling, which reduces the number of channels feeding into FC layers.
- **More Layers: Increases significantly** due to additional convolution layers.

Observation from results:

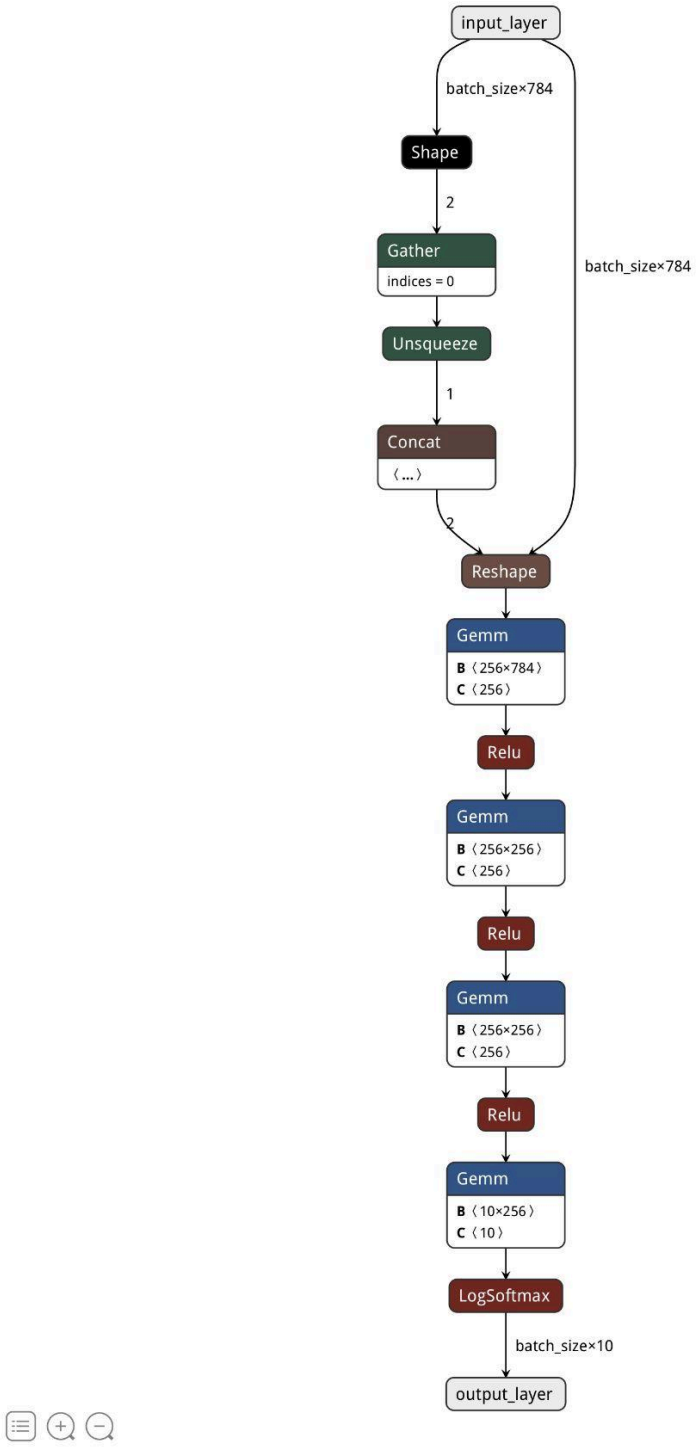
- **Base Network: 454,922**
- **Fewer Layers: 805,066** (↑ Fewer layers, more parameters due to larger FC input)
- **More Layers: 1,110,666** (↑ More layers = More parameters)

5. Total Number of MACs (Multiply-Accumulate Operations):

- **Fewer Layers:** MACs **decrease** due to fewer computations.
- **More Layers:** MACs **increase** since each additional layer adds more multiplications.

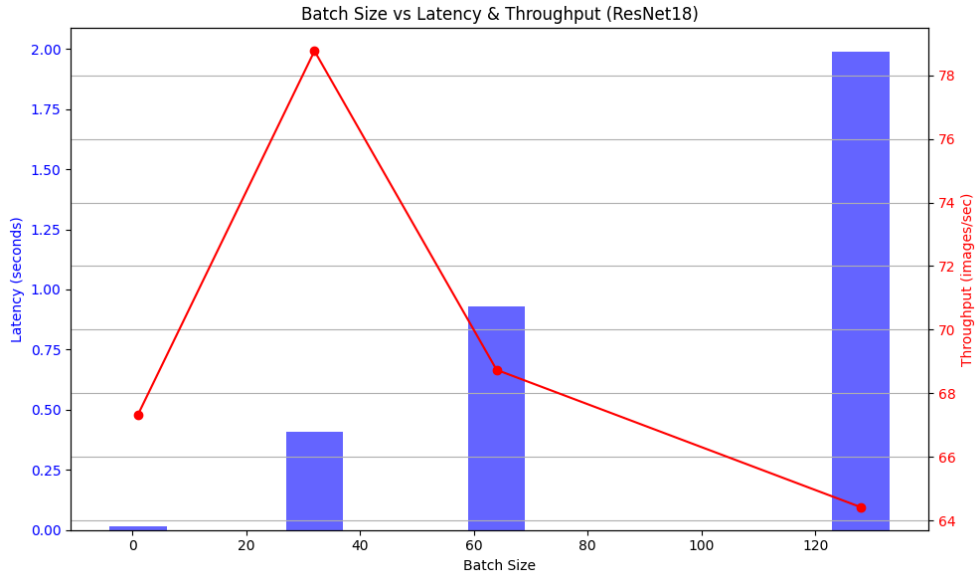
Observation from results:

- **Base Network: 11,065,088**
- **Fewer Layers: 960,896** (↓ Less layers = Fewer computations)
- **More Layers: 21,550,848** (↑ More layers = More computations)


input_layer
batch_size×784
Shape
2
Gather
indices = 0
batch_size×784
Unsqueeze
1
Concat
〈...〉
2
Reshape
Gemm
B 〈256×784〉
C 〈256〉
Relu
Gemm
B 〈256×256〉
C 〈256〉
Relu
Gemm
B 〈256×256〉
C 〈256〉
Relu
Gemm
B 〈10×256〉
C 〈10〉
LogSoftmax
batch_size×10
output_layer

PART 3

Batch Size	Latency (seconds)	Throughput (images/sec)
1	0.071173	14.05
32	0.607504	52.67
64	0.983057	65.1
128	2.122212	60.31



Q1.

No, I don't observe an improvement in **overall execution time**. While batch size 128 processes more images per second, the latency per batch is much higher. If I were to process 128 images one by one with batch size 1, it would take 128 × 0.071173 = 9.1 seconds, but with batch size 128, it only takes 2.12 seconds. So, batch size 128 is more efficient when processing large numbers of images at once but doesn't necessarily improve the latency for a single image.

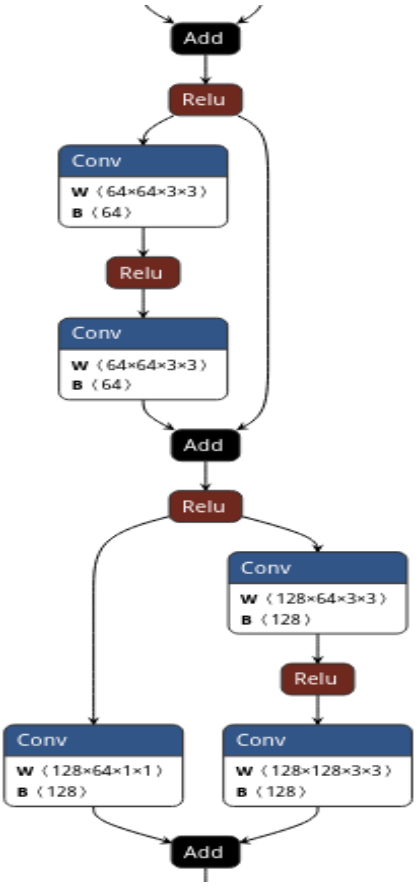
Q2.

A

Ans:- The total number of parameters in the ResNet18 network is 11.69 million, and the total number of MACs (Multiply-Accumulate Operations) is 1.82 billion.

B

I found that the memory consumed by activations is higher than the memory consumed by the weights. The weights, which are the parameters the model learns, consume around 44.59 MB of memory. However, the memory used by the activations during the forward and backward pass is about 62.79 MB. This shows that the activations, which represent the intermediate outputs of each layer, require more memory than the weights. Therefore, I can conclude that the activation memory consumption is higher than the weight memory consumption.

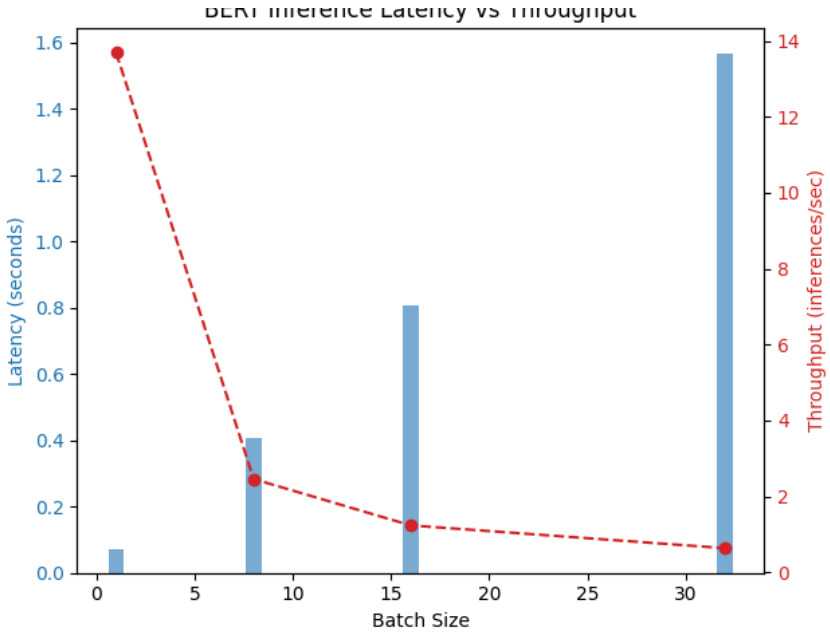


Layer	Weight Shape (W)	Bias Shape (B)	Activation Shape
Conv1	(64 × 64 × 3 × 3)	-64	(Batch, 64, H, W)
Conv2	(64 × 64 × 3 × 3)	-64	(Batch, 64, H, W)

PART 4

Q1.

Batch Size	Latency (seconds)	Throughput (inferences/sec)
1	0.073075	13.68
8	0.406473	2.46
16	0.807294	1.24
32	1.565883	0.64



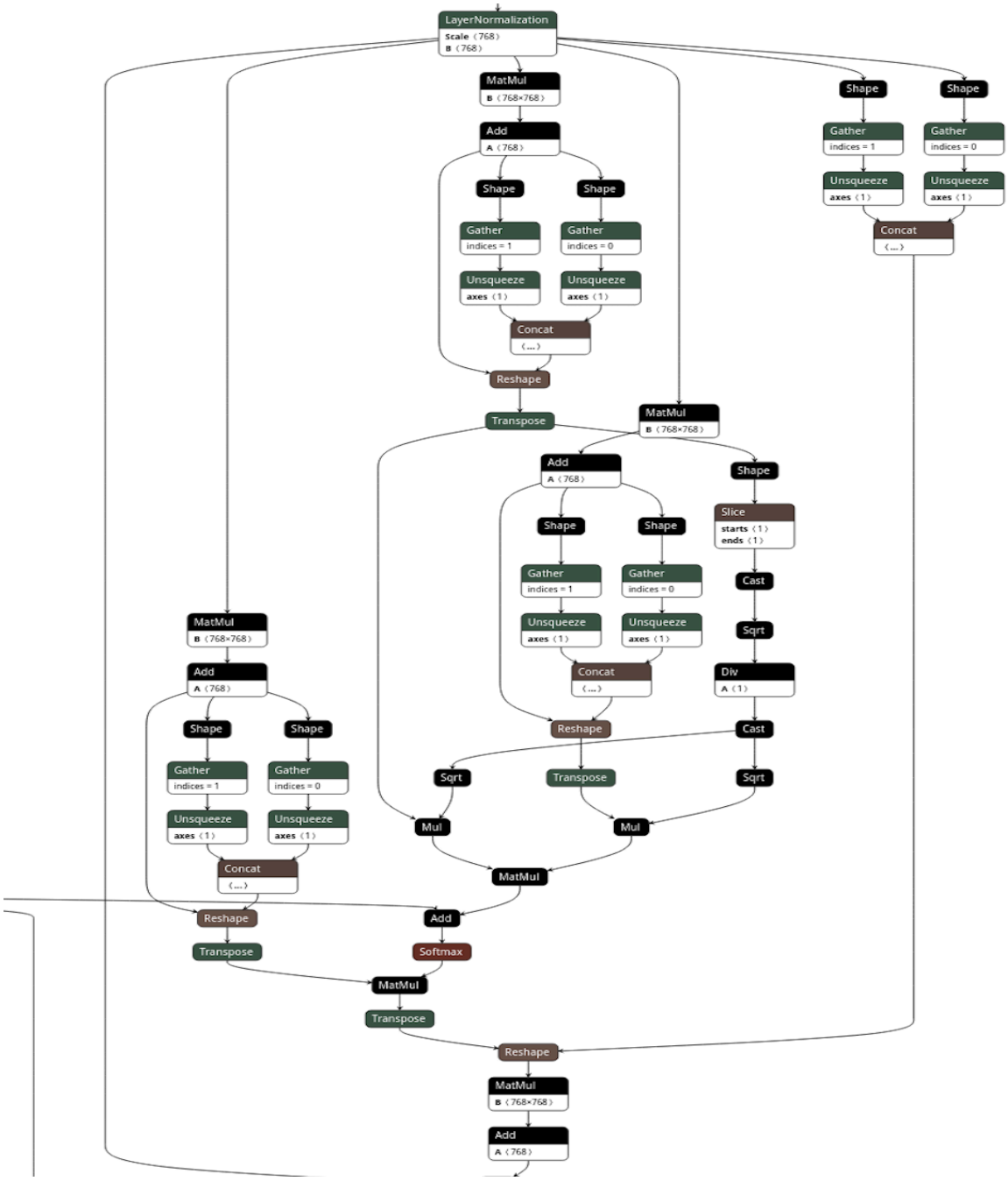
Q2.

The total number of parameters and MACs (Multiply-Accumulate Operations) in the BERT-Base model used for inference are:

- **Total Parameters**: **107,721,218**
- **Total MACs**: **215,442,436**

Q3.

ch


LayerNormalization
Scale ⟨768⟩
B ⟨768⟩
MatMul
B ⟨768×768⟩
Add
A ⟨768⟩
Shape
Shape
Gather
indices = 1
Gather
indices = 0
Unsqueeze
axes ⟨1⟩
Unsqueeze
axes ⟨1⟩
Concat
⟨...⟩
Reshape
Transpose
MatMul
B ⟨768×768⟩
Add
A ⟨768⟩
Shape
Shape
Slice
starts ⟨1⟩
ends ⟨1⟩
Gather
indices = 1
Gather
indices = 0
Cast
Unsqueeze
axes ⟨1⟩
Unsqueeze
axes ⟨1⟩
Sqrt
Concat
⟨...⟩
Div
A ⟨1⟩
Reshape
Cast
Sqrt
Transpose
Sqrt
Mul
Mul
MatMul
Add
Softmax
MatMul
Transpose
Reshape
MatMul
B ⟨768×768⟩
Add
A ⟨768⟩
MatMul
B ⟨768×768⟩
Add
A ⟨768⟩
Shape
Shape
Gather
indices = 1
Gather
indices = 0
Unsqueeze
axes ⟨1⟩
Unsqueeze
axes ⟨1⟩
Concat
⟨...⟩
Reshape
Transpose
Shape
Shape
Gather
indices = 1
Gather
indices = 0
Unsqueeze
axes ⟨1⟩
Unsqueeze
axes ⟨1⟩
Concat
⟨...⟩

Q3.B

GEMM (General Matrix Multiplication) is used for dense layers. In the first transformer block, you will find 4 GEMM operations:

1. Query (Q) projection
2. Key (K) projection
3. Value (V) projection
4. Feedforward dense layer

Q3.C

Layer	Input Shape	Weight Shape	Output Shape
Query (Q)	[1, 128, 768]	[768, 768]	[1, 128, 768]
Key (K)	[1, 128, 768]	[768, 768]	[1, 128, 768]
Value (V)	[1, 128, 768]	[768, 768]	[1, 128, 768]
Feedforward	[1, 128, 768]	[768, 3072]	[1, 128, 3072]

Q3.D

The embedding dimension (d_model) for BERT-Base is 768.

It has a shape of [batch_size, seq_length, 768], e.g., [1, 128, 768]

Q3.E

- BERT-Base has **12 attention heads** in each block.