```
In [ ]: # MNIST dataset
        import torchvision
        import torch
        import torchvision.transforms as transforms
        import numpy as np
```

```
In [ ]: import torch.nn as nn

        torch.manual_seed(42)
```

```
Out[ ]: <torch._C.Generator at 0x248c6a5ea70>
```

```
In [ ]: class ConvNet(nn.Module):
            def __init__(self):
                super(ConvNet, self).__init__()
                # 1 x 28 x 28
                self.layer1 = torch.nn.Sequential(
                    torch.nn.Conv2d(1, 32, kernel_size=5, stride=2, padding=0), # 32 x 12 x
                    torch.nn.ReLU(),
                    torch.nn.MaxPool2d(kernel_size=2, stride=2))  # 32 x 6 x 6
                self.layer2 = torch.nn.Sequential(
                    torch.nn.Conv2d(32, 16, kernel_size=5, stride=1, padding=0), # 64 x 2 x
                    torch.nn.ReLU(),
                    torch.nn.MaxPool2d(kernel_size=2, stride=2)) # 16 x 1 x 1
                self.fc1 = torch.nn.Linear(16, 10)

            def forward(self, x):
                out = self.layer1(x)
                out = self.layer2(out)
                out = out.reshape(out.size(0), -1)
                out = self.fc1(out)
                return out
```

```
In [ ]: dataset_test = torchvision.datasets.MNIST(root='./datasets', train=False, transform
        test_loader = torch.utils.data.DataLoader(dataset=dataset_test, batch_size=10000, s

        dataset_train = torchvision.datasets.MNIST(root='./datasets', train=True, transform
        train_loader = torch.utils.data.DataLoader(dataset=dataset_train, batch_size=100, s
```

```
In [ ]: dataset_train[0][0].shape, len(dataset_train)
```

```
Out[ ]: (torch.Size([1, 28, 28]), 60000)
```

```
In [ ]: class Config:
            lr = 0.001
```

```
In [ ]: config = Config()
```

```
In [ ]: import lightning as L
```

```python
class LightningNN(L.LightningModule):
    def __init__(self, model, config):
        super().__init__()
        self.model = model
        self.loss = torch.nn.CrossEntropyLoss()
        self.config = config

    def forward(self, x):
        return self.model(x)

    def configure_optimizers(self):

        return torch.optim.Adam(self.model.parameters(), lr=self.config.lr)

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.model(x)
        loss = self.loss(y_hat, y)
        return loss

    def predict_step(self, batch, batch_idx, dataloader_idx=0):
        x, y = batch
        y_hat = self.model(x)
        y_pred = torch.argmax(y_hat, dim=-1)
        return y_pred
```

In [ ]:

```python
# model = AlexNet()
model = ConvNet()
```

In [ ]:

```python
lightning_model = LightningNN(model, config)
```

In [ ]:

```python
trainer = L.Trainer(limit_train_batches=100, max_epochs=20)
trainer.fit(model=lightning_model, train_dataloaders=train_loader)
```

```
GPU available: True (cuda), used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs

  | Name  | Type             | Params
-------------------------------------------
0 | model | ConvNet          | 13.8 K
1 | loss  | CrossEntropyLoss | 0
-------------------------------------------
13.8 K    Trainable params
0         Non-trainable params
13.8 K    Total params
0.055     Total estimated model params size (MB)
Epoch 19: 100%|██████████| 100/100 [00:01<00:00, 83.15it/s, loss=0.0482, v_num=19]
`Trainer.fit` stopped: `max_epochs=20` reached.
Epoch 19: 100%|██████████| 100/100 [00:01<00:00, 82.94it/s, loss=0.0482, v_num=19]
```

In [ ]:
```
def test(trainer, lightning_model, dataloader):
    y_pred = trainer.predict(model=lightning_model, dataloaders=dataloader)
    y_pred = torch.cat([y for y in y_pred], dim=0).view(-1)
    y_gt = torch.cat([y for x, y in dataloader], dim=0)
    acc = (y_pred == y_gt).sum().item() / y_gt.size(0)
    return acc
```

In [ ]:
```
test(trainer, lightning_model, train_loader), test(trainer, lightning_model, test_l
```

```
c:\Users\suzy\miniconda3\envs\abbasi\lib\site-packages\lightning\pytorch\trainer\con
nectors\data_connector.py:229: PossibleUserWarning: The dataloader, predict_dataload
er 0, does not have many workers which may be a bottleneck. Consider increasing the
value of the `num_workers` argument` (try 20 which is the number of cpus on this mac
hine) in the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
Predicting DataLoader 0: 100%|██████████| 600/600 [00:03<00:00, 160.27it/s]
Predicting DataLoader 0: 100%|██████████| 1/1 [00:00<00:00,  5.86it/s]
```

Out[ ]:
```
(0.9701166666666666, 0.969)
```

Number of bits per image: 28 * 28 * 8 = 6272 (number of features)

Number of images: 60000

Total number of bits: 6272 * 60000 = 376320000

Architecture 1:

- Layer 1: input channels = 1, output channels = 32, kernel size = 5, stride = 2
- Layer 2 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 3: input channels = 32, output channels = 64, kernel size = 5, stride = 1
- Layer 4 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 5: input channels = 64, output channels = 100
- Layer 6: input channels = 100, output channels = 10

MEC = 4**3 * 28 * 28 / 64 + (64 + 1) * 100 + (100 + 1) * 10 = 3136 + 6500 + 1010 = 10646

Accuracy = 0.9823

Architecture 2:

- Layer 1: input channels = 1, output channels = 32, kernel size = 5, stride = 2
- Layer 2 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 3: input channels = 32, output channels = 64, kernel size = 5, stride = 1
- Layer 4 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 5: input channels = 64, output channels = 10

MEC = 4 ** 3 * 28 * 28 / 64 + (64 + 1) * 10 = 3136 + 650 = 3786

Accuracy = 0.97575

Architecture 3:

- Layer 1: input channels = 1, output channels = 32, kernel size = 5, stride = 2 ()
- Layer 2 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 3: input channels = 32, output channels = 16, kernel size = 5, stride = 1
- Layer 4 (MaxPool): kernel size = 2, stride = 2 (4 : 1)
- Layer 5: input channels = 16, output channels = 10

MEC = 4 ** 3 * 28 * 28 / 16 + (16 + 1) * 10 = 3136 + 170 = 3306

Accuracy = 0.9701

In [ ]:
```
from mec import calculate_mec
```

In [ ]:
```
X = dataset_train.data.numpy()
X = X.reshape(X.shape[0], -1) # flatten
y = dataset_train.targets.numpy()
```

In [ ]:
```
X.shape, y.shape
```

Out[ ]:
```
((60000, 784), (60000,))
```

In [ ]:
```
# Upper bound MEC 3 layer NN
calculate_mec(X, y, task="classification")
```

Out[ ]:
```
3681.371774841067
```

My Method 3 has a MEC of 3306 which is smaller than 3681 (MEC required for a 3 layer NN). This suggests that the use of CNNs is effective.