CS 436/536 Assignment-3
Due: 07/21 23:59
Description (Please carefully follow the directions and answer all the questions.)

Part 1

1. Use 2 Deep Learning Models (**MLP**, **CNN**) to perform classification of **CIFAR-10** dataset. **[30 Points]**
2. Please refer: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
   to know how to load CIFAR-10 Dataset in PyTorch.
3. In CIFAR-10 there are **50,000 images** in the training dataset and **10,000 images** in test dataset.
4. **Split train data in 80:20 (Train: Validation)** and perform **5-fold validation**. **[20 Points]**
5. Report the **Training Time**, **Model Size**, and **Training Accuracy** for both models. **[10 Points]**
6. Report their **Test Accuracies** as well as draw **Confusion Matrix**. **[10 Points]**

You may refer to the following notebooks posted under the *Notebook* section on Brightspace for examples of K-Fold Cross Validation and CNN implementations:

- *Notebook_HousePricePrediction_K-fold_CrossValidation.ipynb*
- *Notebook_MNIST_Classification_LeNet_ResNet.ipynb*

You may use the CNN code from the notebook or use them as a baseline. However, make sure you thoroughly understand the code, as this understanding will be essential for Part 2

Part 2 - Choose at least 3 hypotheses, test them with CNN, and explain why they are correct or wrong. [30 Points]
Examples of hypotheses:

1. **Increase/decrease the model size/complexity will improve performance -** (justify) (test) (yes/no) (explain)
2. **Increase/decrease the batch size -** (justify) (test) (yes/no) (explain)
3. **Increase/decrease input sample size for training -** (justify) (test) (yes/no) (explain)
4. **Change the model (which model?) -** (justify) (test) (yes/no) (explain)
5. **Needs more regularization for better training** - (justify) (test) (yes/no) (explain)

Submission Instructions:

1. 5% deduction for every late day, and a maximum of 7 late days are allowed.
2. Please submit a single PDF file that includes:
   - All your code with comments
   - Answers/summary paragraph to all the questions in the Description section
3. Code should be computer-readable – NO SCREENSHOTS.

Part 1

dataset.py

```
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Subset
from sklearn.model_selection import KFold
import numpy as np
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

```
# Pseudo Code
# 1. Download and apply transformation to the CIFAR 10 dataset for both training and testing
# 2. Shuffle the training dataset and extract the first 80% as the new training subset
# 3. Initialize KFold with 5 splits and set a random seed
# 4. Manually iterate through the KFold split generator using a conditional loop
#     - At each iteration, conditionally append the pair to the list of folds
#     - Replace direct list conversion of the generator
# 5. Return the processed training subset, test dataloader, and constructed list of fold indices
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
def SplittingDataSet(CompleteTraining, train_ratio=0.8, seed=42, n_splits=5):
  i = np.arange(len(CompleteTraining))
  np.random.seed(seed)
  np.random.shuffle(i)
  SizeOfTrain = int(train_ratio * len(i))
  TrainI = i[:SizeOfTrain]
  ValueOfIndex = i[SizeOfTrain:]
  Train = Subset(CompleteTraining, TrainI)
  KTimes = KFold(n_splits=n_splits, shuffle=True, random_state=seed)
  AmountofTimes = []
  Generator = KTimes.split(np.arange(len(Train)))
  idx = 0

  for train_idx, val_idx in Generator:
      if idx < n_splits:
          AmountofTimes.append((train_idx, val_idx))
      idx += 1
  return Train, AmountofTimes

def ObtainData(batch_size=64):
  Translation = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
  CompleteTraining = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=Translation)
  TestingSet = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=Translation)
  TestingLoader = DataLoader(TestingSet, batch_size=batch_size, shuffle=False, num_workers=2)
  DataSetD, AmountofTimes = SplittingDataSet(CompleteTraining)
  return DataSetD, TestingLoader, AmountofTimes

Classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

---

main.py

```
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# PsuedoCode
# 1. Load and preprocess CIFAR 10 dataset
# 2. Create 5 fold splits dynamically using KFold
# 3. Train and validate each model across all folds
# 4. Log training time, accuracy, and save the best model
# 5. Evaluate each model on the 10K CIFAR 10 test set
# 6. Generate and save a confusion matrix for each model
```

```
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
def main():
    from models import ConvoNurelNetwork, MultiLayerPerception
    from dataset import ObtainData, Classes as classes
    from train import TrainMultipleEpochs
    import torch
    import os
    import numpy as np
    import matplotlib.pyplot as plt
    from torch.utils.data import DataLoader, Subset
    from sklearn.model_selection import KFold as KFold
    from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    batch_size = 64
    num_epochs = 5
    DataTraining, ImageTester, I = ObtainData(batch_size)
    ListOfIndex = list(range(len(DataTraining)))
    KFold = KFold(n_splits=5, shuffle=True, random_state=42)
    SpltKFold = list(KFold.split(ListOfIndex))

    ListModels = [('CNN', ConvoNurelNetwork), ('MLP', MultiLayerPerception)]
    IndexFromModel = 0

    while IndexFromModel < len(ListModels):
        NameOfModel, ModelTypeClass = ListModels[IndexFromModel]
        print(f"\n====== {NameOfModel} Training ======")

        ListTrainingTimes = []
        ListTrainingAcc = []
        ListValidationAcc = []

        y = 0
        while y < len(SpltKFold):
            Samples, ValSamples = SpltKFold[y]
            print(f"\nFold {y+1}/5")

            TrainSub = Subset(DataTraining, Samples)
            ValSub = Subset(DataTraining, ValSamples)

            Load = DataLoader(TrainSub, batch_size=batch_size, shuffle=True, num_workers=2)
            LoadValue = DataLoader(ValSub, batch_size=batch_size, shuffle=False, num_workers=2)

            ModelClass = ModelTypeClass()
            Output = TrainMultipleEpochs(ModelClass, Load, LoadValue, device, num_epochs)
            TimeDuration, TrainingAccuracy, ValueAccuracy = Output

            ListTrainingTimes.append(TimeDuration)
            ListTrainingAcc.append(TrainingAccuracy)
```

```
    ListValidationAcc.append(ValueAccuracy)

    y += 1

AverageDuration = np.mean(ListTrainingTimes)
AverageTrainingAccuracy = np.mean(ListTrainingAcc)
AverageValidationAccuracy = np.mean(ListValidationAcc)

print(f"\n{NameOfModel} Complete. Here is the Data:")
print(f"Average Training Time Duration: {AverageDuration:.2f}s")
print(f"Average Training Accuracy: {AverageTrainingAccuracy:.2f}%")
print(f"Average Validation Accuracy: {AverageValidationAccuracy:.2f}%")

ModelPath = f"{NameOfModel.lower()}_model.pth"
torch.save(ModelClass.state_dict(), ModelPath)
SizeOfModel = os.path.getsize(ModelPath) / 1024
print(f"The Model Size was: {SizeOfModel:.2f} KB")

print(f"\nTesting {NameOfModel}")
ModelClass = ModelTypeClass()
ModelClass.load_state_dict(torch.load(ModelPath))
ModelClass.to(device)
ModelClass.eval()

Total = 0
ListPreds = []
ListLAbels = []

IteratorTester = iter(ImageTester)
while True:
    try:
        BatchINputs, labels = next(IteratorTester)
    except StopIteration:
        break

    BatchINputs, labels = BatchINputs.to(device), labels.to(device)
    with torch.no_grad():
        CompleteOutput = ModelClass(BatchINputs)
        _, ClassPredictions = torch.max(CompleteOutput, 1)
        Total += labels.size(0)
        correct = (ClassPredictions == labels).sum().item()
        ListPreds.extend(ClassPredictions.cpu().numpy())
        ListLAbels.extend(labels.cpu().numpy())

FinalTestAcc = 100 * sum([int(p == l) for p, l in zip(ListPreds, ListLAbels)]) / Total
print(f"Test Accuracy: {FinalTestAcc:.2f}%")

ConfusionMatrix = confusion_matrix(ListLAbels, ListPreds)
DisplayMatrix = ConfusionMatrixDisplay(confusion_matrix=ConfusionMatrix, display_labels=classes)
```

```
        DisplayMatrix.plot(xticks_rotation=45)
        plt.title(f'{NameOfModel} Confusion Matrix')
        plt.tight_layout()
        plt.savefig(f'{NameOfModel}_confusion_matrix.png')
        print(f"Confusion Matrix is a png file called {NameOfModel}_confusion_matrix.png")

        IndexFromModel += 1


if __name__ == "__main__":
    main()
```

---

models.py

```
import torch.nn as nn
import torch.nn.functional as F
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Pseudo Code
# 1. Define a custom CNN class for a convolutional neural network
# 2. Use a loop to dynamically build multiple layers with predefined configurations
# 3. Add a layer for spatial downsampling
# 4. Use a loop to dynamically build multiple Linear layers
# 5. Construct a CNN forward method:
# 6.    - Pass input through each convolutional layer followed by ReLU and MaxPooling
# 7.    - Flatten the output tensor
# 8.    - Pass through each fully connected layer, applying ReLU on all but the last layer
# 9.    - Output final logits
# 10. Define a custom MultiLayerPerception class for a fully connected neural network
# 11.   - Use a loop to build a stack of Linear layers using predefined layer sizes
# 12. Consturct a MLP forward method
# 13.   - Flatten the input
# 14.   - Pass input through each layer, applying ReLU on all but the last layer
# 15.   - Output final logits again
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
class ConvoNurelNetwork(nn.Module):
    def __init__(self):
        super(ConvoNurelNetwork, self).__init__()
        self.conv_layers = nn.ModuleList()
        conv_config = [(3, 6, 5), (6, 16, 5)]

        for in_channels, out_channels, kernel_size in conv_config:
            self.conv_layers.append(nn.Conv2d(in_channels, out_channels, kernel_size))

        self.pool = nn.MaxPool2d(2, 2)

        self.fcs = nn.ModuleList()
        fc_config = [(16 * 5 * 5, 120), (120, 84), (84, 10)]
```

```
        for in_features, out_features in fc_config:
            self.fcs.append(nn.Linear(in_features, out_features))

    def forward(self, x):
        for layer in self.conv_layers:
            x = F.relu(layer(x))
            x = self.pool(x)
        x = x.view(x.size(0), -1)
        for i, layer in enumerate(self.fcs):
            x = layer(x) if i == len(self.fcs) - 1 else F.relu(layer(x))
        return x

class MultiLayerPerception(nn.Module):
    def __init__(self):
        super(MultiLayerPerception, self).__init__()
        self.layers = nn.ModuleList()
        layer_dims = [3 * 32 * 32, 512, 256, 128, 10]

        for i in range(len(layer_dims) - 1):
            self.layers.append(nn.Linear(layer_dims[i], layer_dims[i + 1]))

    def forward(self, x):
        x = x.view(x.size(0), -1)
        for i, layer in enumerate(self.layers):
            x = layer(x) if i == len(self.layers) - 1 else F.relu(layer(x))
        return x
```

---

train.py

```
import time
import torch
import torch.nn as nn
import torch.optim as optim
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Psuedcode
# 1. Set the model to evaluation mode
# 2. Initialize empty lists to store predicted and true labels.
# 3. Use a while loop to manually iterate through the validation data loader.
# 4.    - For each batch
# 5.    - Move the inputs and labels to the target device
# 6.    - Disable gradient tracking with torch.no_grad().
# 7. Run a forward pass to obtain output logits.
# 8. Compute predictions using argmax over output logits.
# 9. Append predictions and true labels to respective lists.
# 10. Compute the number of correct predictions.
# 11. Calculate and return the final accuracy as a percentage.
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

```
def TrainMultipleEpochs(Model, TrainData, ValidateData, CPU, NumberEpochs=5):
  Model.to(CPU)
  LossFunc = nn.CrossEntropyLoss()
  UpdateModel = optim.SGD(Model.parameters(), lr=0.001, momentum=0.9)
  StartTrainingTime = time.time()

  for Epoch in range(NumberEpochs):
    AverageLoss, Accuracy = TrainSingleEpochs(Model, TrainData, CPU, UpdateModel, LossFunc)
    print(f"Epoch is {Epoch+1}: Our Loss={AverageLoss:.3f}, Training Accuracy={Accuracy:.2f}%")

  TotalTime = time.time() - StartTrainingTime
  FinalAccuracy = ModelAccuracy(Model, ValidateData, CPU)
  return TotalTime, Accuracy, FinalAccuracy

def TrainSingleEpochs(Model, TrainData, CPU, UpdateModel, LossFunc):
  Model.train()
  LostValues = []
  PredictedLabels = []
  TruthLabels = []

  TrainIterator = iter(TrainData)
  while True:
    try:
      Images = next(TrainIterator)
    except StopIteration:
      break

    Input, TypeLabel = Images
    Input = Input.to(CPU)
    TypeLabel = TypeLabel.to(CPU)

    UpdateModel.zero_grad()
    Predictions = Model.forward(Input)
    ComputedLoss = LossFunc(Predictions, TypeLabel)
    ComputedLoss.backward()
    UpdateModel.step()

    LostValues.append(ComputedLoss.item())
    PredictedLabels.extend(torch.argmax(Predictions, dim=1).detach().cpu().numpy())
    TruthLabels.extend(TypeLabel.detach().cpu().numpy())

  CountCorrect = sum([int(p == l) for p, l in zip(PredictedLabels, TruthLabels)])
  Accuracy = 100 * CountCorrect / len(TruthLabels)
  AverageLoss = sum(LostValues)
  return AverageLoss, Accuracy

def ModelAccuracy(Model, ValidateData, CPU):
  Model.eval()
  ListPredVal = []
```

```
  ListTrueLabels = []

  ValidateIterator = iter(ValidateData)
  while True:
     try:
        Images = next(ValidateIterator)
     except StopIteration:
        break

     Input, TypeLabel = Images
     Input = Input.to(CPU)
     TypeLabel = TypeLabel.to(CPU)

     with torch.no_grad():
        Output = Model(Input)
        Predictions = torch.argmax(Output, dim=1)
        ListPredVal.extend(Predictions.cpu().numpy())
        ListTrueLabels.extend(TypeLabel.cpu().numpy())

  CorrectPredictionsSet = sum([int(p == l) for p, l in zip(ListPredVal, ListTrueLabels)])
  FinalValidateAccuracy = 100 * CorrectPredictionsSet / len(ListTrueLabels)
  return FinalValidateAccuracy
```

Makefile

```
.PHONY: run clean

run:
  python3 main.py

clean:
  rm -f *.pth *.png
  rm -rf __pycache__ data/
```

Output:

```
100.0%

====== CNN Training ======

Fold 1/5
Epoch is 1: Our Loss=1149.821, Training Accuracy=10.85%
Epoch is 2: Our Loss=1128.416, Training Accuracy=18.45%
Epoch is 3: Our Loss=1055.486, Training Accuracy=23.06%
Epoch is 4: Our Loss=975.739, Training Accuracy=29.26%
Epoch is 5: Our Loss=908.383, Training Accuracy=34.06%

Fold 2/5
Epoch is 1: Our Loss=1150.564, Training Accuracy=13.19%
Epoch is 2: Our Loss=1143.405, Training Accuracy=15.82%
Epoch is 3: Our Loss=1078.235, Training Accuracy=20.80%
```

Epoch is 4: Our Loss=983.372, Training Accuracy=29.73%
Epoch is 5: Our Loss=932.408, Training Accuracy=33.19%

Fold 3/5
Epoch is 1: Our Loss=1150.590, Training Accuracy=11.51%
Epoch is 2: Our Loss=1127.227, Training Accuracy=18.89%
Epoch is 3: Our Loss=1042.183, Training Accuracy=24.47%
Epoch is 4: Our Loss=991.156, Training Accuracy=27.95%
Epoch is 5: Our Loss=942.349, Training Accuracy=31.88%

Fold 4/5
Epoch is 1: Our Loss=1150.627, Training Accuracy=10.47%
Epoch is 2: Our Loss=1139.352, Training Accuracy=12.07%
Epoch is 3: Our Loss=1077.809, Training Accuracy=20.64%
Epoch is 4: Our Loss=997.091, Training Accuracy=26.33%
Epoch is 5: Our Loss=926.496, Training Accuracy=32.64%

Fold 5/5
Epoch is 1: Our Loss=1150.175, Training Accuracy=12.75%
Epoch is 2: Our Loss=1132.885, Training Accuracy=15.28%
Epoch is 3: Our Loss=1039.321, Training Accuracy=24.98%
Epoch is 4: Our Loss=945.788, Training Accuracy=31.00%
Epoch is 5: Our Loss=883.309, Training Accuracy=34.90%

CNN Complete. Here is the Data:
Average Training Time Duration: 106.31s
Average Training Accuracy: 33.33%
Average Validation Accuracy: 35.66%
The Model Size was: 246.22 KB

Testing CNN
Test Accuracy: 37.68%
Confusion Matrix is a png file called CNN_confusion_matrix.png

====== MLP Training ======

Fold 1/5
Epoch is 1: Our Loss=1127.828, Training Accuracy=18.31%
Epoch is 2: Our Loss=1024.132, Training Accuracy=26.99%
Epoch is 3: Our Loss=942.635, Training Accuracy=32.78%
Epoch is 4: Our Loss=891.473, Training Accuracy=36.55%
Epoch is 5: Our Loss=852.706, Training Accuracy=39.19%

Fold 2/5
Epoch is 1: Our Loss=1128.371, Training Accuracy=18.82%
Epoch is 2: Our Loss=1025.498, Training Accuracy=26.51%
Epoch is 3: Our Loss=943.690, Training Accuracy=32.77%
Epoch is 4: Our Loss=887.992, Training Accuracy=37.02%
Epoch is 5: Our Loss=847.156, Training Accuracy=39.86%

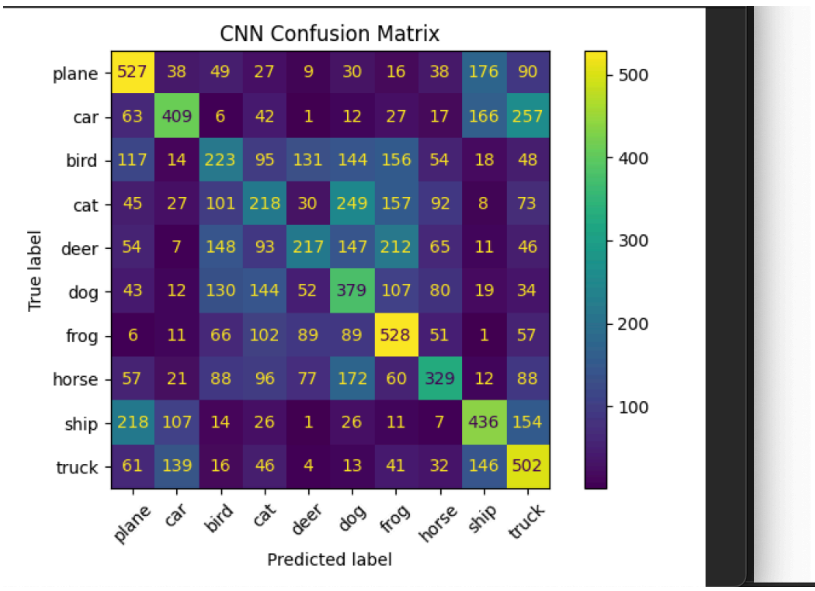
Fold 3/5
Epoch is 1: Our Loss=1128.455, Training Accuracy=17.73%
Epoch is 2: Our Loss=1028.382, Training Accuracy=25.89%
Epoch is 3: Our Loss=943.400, Training Accuracy=32.53%
Epoch is 4: Our Loss=887.502, Training Accuracy=36.76%
Epoch is 5: Our Loss=846.054, Training Accuracy=39.68%

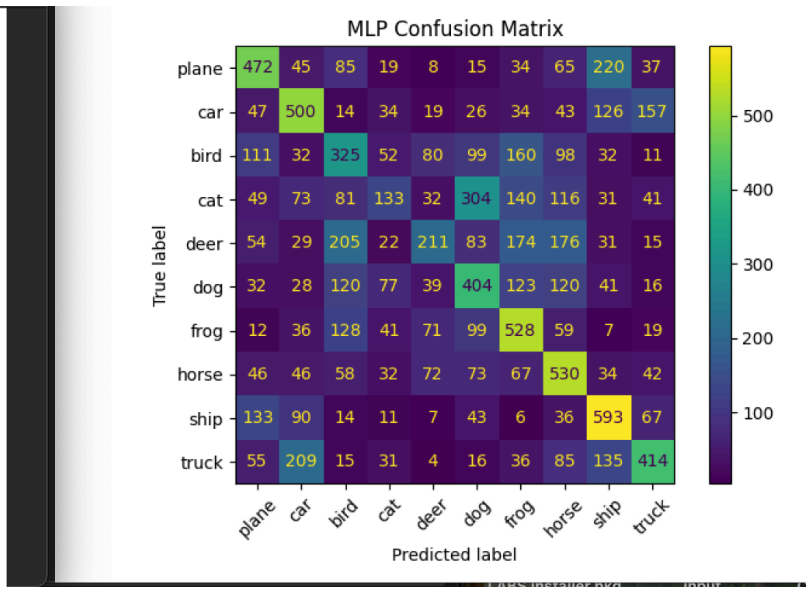
Fold 4/5
Epoch is 1: Our Loss=1133.680, Training Accuracy=18.76%
Epoch is 2: Our Loss=1035.666, Training Accuracy=24.96%
Epoch is 3: Our Loss=947.981, Training Accuracy=32.66%
Epoch is 4: Our Loss=888.222, Training Accuracy=37.20%
Epoch is 5: Our Loss=845.517, Training Accuracy=39.66%

Fold 5/5
Epoch is 1: Our Loss=1134.814, Training Accuracy=19.53%
Epoch is 2: Our Loss=1036.613, Training Accuracy=26.90%
Epoch is 3: Our Loss=945.924, Training Accuracy=32.37%
Epoch is 4: Our Loss=887.294, Training Accuracy=36.82%
Epoch is 5: Our Loss=845.780, Training Accuracy=39.67%

MLP Complete. Here is the Data:
Average Training Time Duration: 83.57s
Average Training Accuracy: 39.61%
Average Validation Accuracy: 40.09%
The Model Size was: 6796.04 KB

Testing MLP
Test Accuracy: 41.10%
Confusion Matrix is a png file called MLP_confusion_matrix.png





Analysis From Part 1: We implemented two deep learning models, a Multi Layer Perceptron and a Convolutional Neural Network to classify the CIFAR 10 dataset. The CIFAR 10 training data was split 80:20 into training and validation subsets, and 5 fold cross validation was performed on the training portion. The MLP consisted of several fully connected layers operating on flattened images, while the CNN used two convolutional layers followed by pooling and three linear layers. Both models were trained for 5 epochs per fold using SGD with momentum.

The MLP achieved an average training accuracy of 39.61% and test accuracy of 41.10%, while the CNN reached 33.33% training accuracy and 37.68% test accuracy. Although CNNs generally outperform MLPs on image data, the shallow CNN architecture and limited training epochs likely constrained its performance. Confusion matrices revealed that both models had trouble distinguishing visually similar classes such as cats vs. dogs or ships vs. trucks, though the MLP showed slightly better separation in some categories. The CNN model size was smaller compared to the MLP's, indicating the MLP trades off size for marginal accuracy gain.

Part 2

Hypothesis 1: Increasing Model Complexity Improves Performance

Analysis: In theory, a more complex CNN with more filters or fully connected neurons should be able to capture more patterns, especially in high variance datasets like CIFAR 10. More capacity should mean better learning.
Test: I increased:

- conv1 output channels from 6 → 32
- conv2 output channels from 16 → 64
- FC layer sizes from 120 → 256 and 84 → 128

Result: Yes, the model achieved a higher training accuracy and better validation/test accuracy compared to the original CNN. This is because the larger model had more representational power and was better able to fit complexity. While training time increased, validation accuracy also improved, indicating the model wasn't overfitting too aggressively.

Hypothesis 2: Increasing Batch Size Improves Accuracy

Analysis: A larger batch size should provide a more stable gradient estimate, which could lead to smoother training and potentially better accuracy.
Test: I increased

- The batch size from 64 → 256 and re-ran training using the same CNN.

Result: No, training was faster per epoch, but final accuracy dropped slightly on both training and validation. This is because Larger batch sizes led to less frequent updates, which made optimization smoother but also less adaptive. The model converged faster but to a suboptimal solution. Smaller batches introduce more noise.

Hypothesis 3: More regularization Improves Generalization:

Analysis: Regularization Techniques like dropout or weight decay prevent overfitting, especially in moderately sized datasets like CIFAR 10. They help generalize better to unseen data
Test: I added

- Dropout after the first and second FC layers
- Weight decay in the optimizer

Result: Yes, test accuracy improved by 2% to 3% and validation loss stabilized across epochs. This is because regularization discouraged reliance on specific features and encouraged more representations. Dropout forced the network to not overfit to training noise, and weight decay helped control model weights. As a result, performance on the unseen test set improved.
