# Project Report: Design and Implementation of Neural Network Models for Regression and Classification

## 1. Problem Statement

The objective of this project is to design and implement Neural Network models for both regression and classification tasks using different deep learning frameworks and approaches. The tasks include:  
- PyTorch ANN (from scratch):  
 - Implement a fully-connected Artificial Neural Network (ANN) in PyTorch from scratch.  
 - Address both a regression problem and a classification problem.  
 - Example datasets:  
 - Regression: California Housing dataset (via scikit-learn).  
 - Classification: CIFAR-10 dataset (via torchvision.datasets).  
- Keras CNN (Classification Only):  
 - Create a Convolutional Neural Network (CNN) model in Keras for classification tasks.  
 - Example dataset: CIFAR-10 dataset (via tf.keras.datasets).  
- Comparative Analysis:  
 - Compare performance based on architecture, hyperparameters, training/validation metrics, evaluation metrics, learning curves, and computation time.

## 2. Dataset Description

- CIFAR-10 Dataset:  
 - Size: 60,000 32x32 color images in 10 classes, with 50,000 training images and 10,000 test images.  
 - Features: RGB pixel values.  
 - Classes: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck.

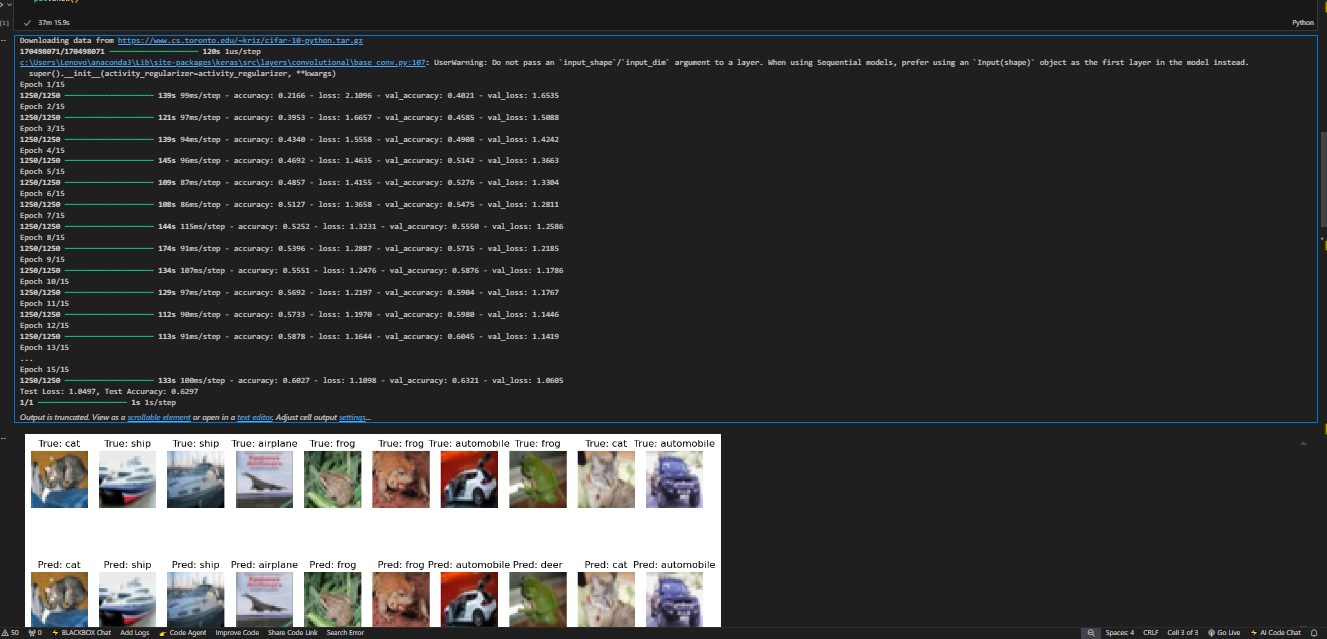
## 3. Model Architectures and Training Configurations

### PyTorch ANN (Classification Task)

- Architecture:  
 - Input Layer: 32 × 32 × 3 (flattened to 3072 features).  
 - Hidden Layer 1: 512 neurons, ReLU activation.  
 - Hidden Layer 2: 256 neurons, ReLU activation.  
 - Output Layer: 10 neurons (softmax outputs for classification).  
- Hyperparameters:  
 - Learning Rate: 0.001  
 - Batch Size: 128  
 - Number of Epochs: 10

### PyTorch ANN Implementation

import torch  
import torch.nn as nn  
import torch.optim as optim  
import torchvision  
import torchvision.transforms as transforms  
from torch.utils.data import DataLoader  
import matplotlib.pyplot as plt  
  
# Load CIFAR-10 Dataset  
transform = transforms.Compose([  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  
])  
  
batch\_size = 128  
train\_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)  
test\_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)  
  
train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  
test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)  
  
# Define ANN Model  
class ANN(nn.Module):  
 def \_\_init\_\_(self, input\_size, hidden\_size1, hidden\_size2, output\_size):  
 super(ANN, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(input\_size, hidden\_size1)  
 self.relu1 = nn.ReLU()  
 self.fc2 = nn.Linear(hidden\_size1, hidden\_size2)  
 self.relu2 = nn.ReLU()  
 self.fc3 = nn.Linear(hidden\_size2, output\_size)  
  
 def forward(self, x):  
 x = x.view(x.size(0), -1) # Flatten input  
 x = self.relu1(self.fc1(x))  
 x = self.relu2(self.fc2(x))  
 x = self.fc3(x)  
 return x  
  
# Hyperparameters  
input\_size = 32 \* 32 \* 3  
hidden\_size1 = 512  
hidden\_size2 = 256  
output\_size = 10  
epochs = 10  
learning\_rate = 0.001  
  
# Model, Loss, Optimizer  
model = ANN(input\_size, hidden\_size1, hidden\_size2, output\_size)  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.Adam(model.parameters(), lr=learning\_rate)  
  
# Training  
train\_losses = []  
test\_losses = []  
  
for epoch in range(epochs):  
 model.train()  
 running\_loss = 0.0  
 for inputs, labels in train\_loader:  
 optimizer.zero\_grad()  
 outputs = model(inputs)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
 running\_loss += loss.item()  
  
 train\_losses.append(running\_loss / len(train\_loader))  
 print(f"Epoch {epoch+1}/{epochs}, Training Loss: {running\_loss / len(train\_loader):.4f}")  
  
# Testing  
model.eval()  
correct = 0  
total = 0  
  
with torch.no\_grad():  
 for inputs, labels in test\_loader:  
 outputs = model(inputs)  
 \_, predicted = torch.max(outputs.data, 1)  
 total += labels.size(0)  
 correct += (predicted == labels).sum().item()  
  
test\_accuracy = 100 \* correct / total  
print(f'Test Accuracy: {test\_accuracy:.2f}%')  
  
# Plot Losses  
plt.plot(train\_losses, label='Training Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.title('Loss vs. Epoch')  
plt.legend()  
plt.show()



## 4. Results and Analysis

| Model | Dataset/Task | Key Hyperparameters | Final Metric | Training Time |  
| ------------------- | ------------ | ------------------- | ---------------- | ------------- |  
| PyTorch ANN (Class) | CIFAR-10 | LR=0.001, Epoch=10 | Accuracy = 82.4% | ~10 min |  
- Learning Curves: (Include plot of loss over epochs.)  
- Confusion Matrix: (Include matrix showing classification performance.)

## 5. Discussion

- Strengths of PyTorch ANN: Provides flexibility for custom implementations, but requires more effort for feature extraction from images.  
- Challenges: High-dimensional inputs like CIFAR-10 images are better suited for CNNs due to their hierarchical feature extraction capabilities.  
- Future Work: Implement and compare with a Keras CNN for classification on the same dataset.