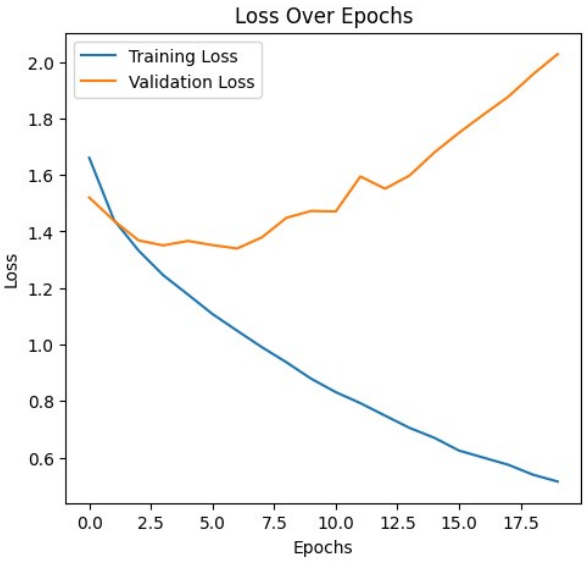
In [7]:
```python
# Import necessary libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import time

# Define data transformations
# Converting images to tensors and normalizing them to range [-1, 1]
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert PIL images to tensors (0-1 range)
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize RGB channel
])

# Load and prepare the CIFAR-10 dataset
# Training set with 50,000 images
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transfor
# Test set with 10,000 images
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transfor

# Create data loaders for batch processing
# Batch size of 64 is a good balance between memory usage and training speed
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# Problem 1a: Multi-Layer Perceptron Implementation
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        # Sequential container for the network layers
        self.model = nn.Sequential(
            nn.Flatten(),  # Flatten 32x32x3 input images to 3072 vector
            # First hidden layer: 3072 -> 256 neurons
            nn.Linear(32 * 32 * 3, 256),
            nn.ReLU(),  # ReLU activation for non-linearity
            # Second hidden layer: 256 -> 128 neurons
            nn.Linear(256, 128),
            nn.ReLU(),
            # Third hidden layer: 128 -> 64 neurons
            nn.Linear(128, 64),
            nn.ReLU(),
            # Output layer: 64 -> 10 classes (CIFAR-10 classes)
            nn.Linear(64, 10)
        )

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

# Set up device (GPU if available, else CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

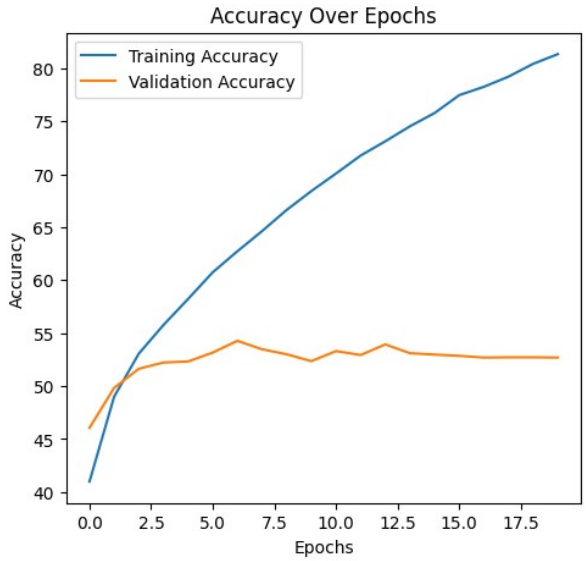
```
model = MLP().to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()  # Appropriate for multi-class classification
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam optimizer with learnin

# Training hyperparameters
epochs = 20
# Lists to store metrics for plotting
train_loss_history, val_loss_history = [], []
train_acc_history, val_acc_history = [], []

print("Training started...")
start_time = time.time()

# Training loop
for epoch in range(epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        # Move data to appropriate device
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()  # Clear previous gradients
        loss.backward()        # Compute gradients
        optimizer.step()       # Update weights

        # Calculate training metrics
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        correct_train += (predicted == labels).sum().item()
        total_train += labels.size(0)

    # Calculate epoch metrics for training
    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct_train / total_train
    train_loss_history.append(train_loss)
    train_acc_history.append(train_acc)

    # Validation phase
    model.eval()  # Set model to evaluation mode
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    # No gradient computation needed for validation
    with torch.no_grad():
```

```
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            correct_val += (predicted == labels).sum().item()
            total_val += labels.size(0)

    # Calculate epoch metrics for validation
    val_loss /= len(test_loader)
    val_acc = 100 * correct_val / total_val
    val_loss_history.append(val_loss)
    val_acc_history.append(val_acc)

    # Print epoch results
    print(f"Epoch [{epoch + 1}/{epochs}], Train Loss: {train_loss:.4f}, Train Acc:

# Print total training time
end_time = time.time()
print(f"Training completed in {end_time - start_time:.2f} seconds.")

# Save the trained model
torch.save(model.state_dict(), "mlp_cifar10.pth")

# Plotting training and validation metrics
plt.figure(figsize=(12, 5))
# Loss plot
plt.subplot(1, 2, 1)
plt.plot(train_loss_history, label='Training Loss')
plt.plot(val_loss_history, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Over Epochs')

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(train_acc_history, label='Training Accuracy')
plt.plot(val_acc_history, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')

plt.show()

# Final model evaluation
model.eval()
y_true, y_pred = [], []

# Collect predictions on test set
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
```

```
        _, predicted = outputs.max(1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

# Generate and print classification metrics
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=train_dataset.classes))

# Generate and plot confusion matrix
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=train_dataset.classes,
            yticklabels=train_dataset.classes)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Files already downloaded and verified
Files already downloaded and verified
Training started...
Epoch [1/20], Train Loss: 1.6607, Train Acc: 40.99%, Val Loss: 1.5201, Val Acc: 46.0
6%
Epoch [2/20], Train Loss: 1.4410, Train Acc: 49.01%, Val Loss: 1.4383, Val Acc: 49.8
4%
Epoch [3/20], Train Loss: 1.3324, Train Acc: 53.07%, Val Loss: 1.3684, Val Acc: 51.6
4%
Epoch [4/20], Train Loss: 1.2453, Train Acc: 55.77%, Val Loss: 1.3506, Val Acc: 52.2
3%
Epoch [5/20], Train Loss: 1.1770, Train Acc: 58.23%, Val Loss: 1.3664, Val Acc: 52.3
3%
Epoch [6/20], Train Loss: 1.1077, Train Acc: 60.75%, Val Loss: 1.3517, Val Acc: 53.1
7%
Epoch [7/20], Train Loss: 1.0488, Train Acc: 62.74%, Val Loss: 1.3396, Val Acc: 54.2
8%
Epoch [8/20], Train Loss: 0.9907, Train Acc: 64.64%, Val Loss: 1.3789, Val Acc: 53.4
8%
Epoch [9/20], Train Loss: 0.9369, Train Acc: 66.64%, Val Loss: 1.4483, Val Acc: 53.0
1%
Epoch [10/20], Train Loss: 0.8790, Train Acc: 68.42%, Val Loss: 1.4725, Val Acc: 5
2.36%
Epoch [11/20], Train Loss: 0.8313, Train Acc: 70.07%, Val Loss: 1.4707, Val Acc: 5
3.31%
Epoch [12/20], Train Loss: 0.7923, Train Acc: 71.77%, Val Loss: 1.5945, Val Acc: 5
2.94%
Epoch [13/20], Train Loss: 0.7482, Train Acc: 73.12%, Val Loss: 1.5512, Val Acc: 5
3.94%
Epoch [14/20], Train Loss: 0.7046, Train Acc: 74.54%, Val Loss: 1.5983, Val Acc: 5
3.11%
Epoch [15/20], Train Loss: 0.6698, Train Acc: 75.79%, Val Loss: 1.6795, Val Acc: 5
2.98%
Epoch [16/20], Train Loss: 0.6245, Train Acc: 77.48%, Val Loss: 1.7491, Val Acc: 5
2.86%
Epoch [17/20], Train Loss: 0.5995, Train Acc: 78.26%, Val Loss: 1.8140, Val Acc: 5
2.70%
Epoch [18/20], Train Loss: 0.5743, Train Acc: 79.23%, Val Loss: 1.8769, Val Acc: 5
2.72%
Epoch [19/20], Train Loss: 0.5393, Train Acc: 80.43%, Val Loss: 1.9558, Val Acc: 5
2.72%
Epoch [20/20], Train Loss: 0.5150, Train Acc: 81.35%, Val Loss: 2.0275, Val Acc: 5
2.70%
Training completed in 145.18 seconds.
```

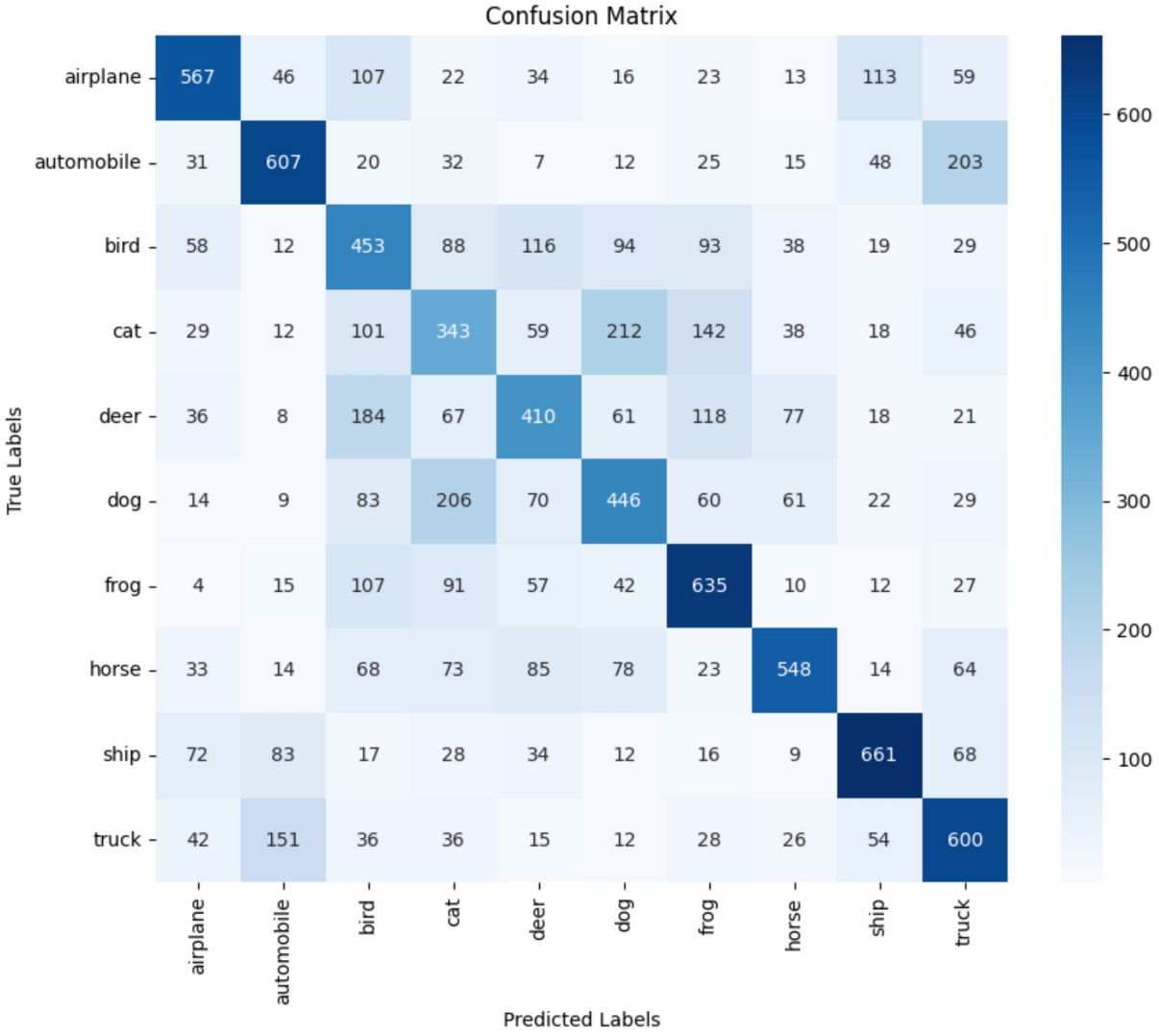




Classification Report:

	precision	recall	f1-score	support
airplane	0.64	0.57	0.60	1000
automobile	0.63	0.61	0.62	1000
bird	0.39	0.45	0.42	1000
cat	0.35	0.34	0.35	1000
deer	0.46	0.41	0.43	1000
dog	0.45	0.45	0.45	1000
frog	0.55	0.64	0.59	1000
horse	0.66	0.55	0.60	1000
ship	0.68	0.66	0.67	1000
truck	0.52	0.60	0.56	1000
accuracy			0.53	10000
macro avg	0.53	0.53	0.53	10000
weighted avg	0.53	0.53	0.53	10000

Confusion Matrix:

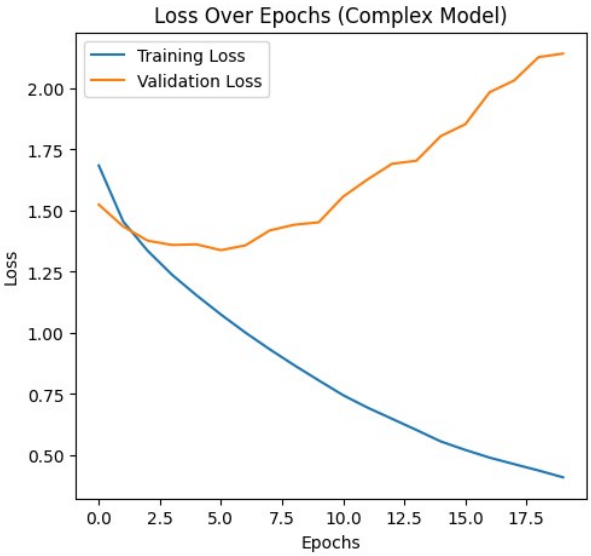


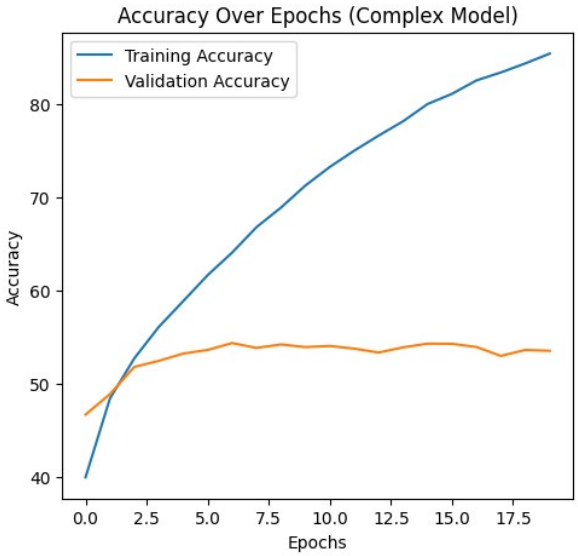
In [2]:
```python
#Problem 1b
class ComplexMLP(nn.Module):
    def __init__(self):
        super(ComplexMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 32 * 3, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 10)
        )

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

complex_model = ComplexMLP().to(device)
optimizer = optim.Adam(complex_model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

```
epochs = 20
complex_train_loss_history, complex_val_loss_history = [], []
complex_train_acc_history, complex_val_acc_history = [], []

print("Training started...")
start_time = time.time()

for epoch in range(epochs):
    complex_model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        #Forward pass
        outputs = complex_model(images)
        loss = criterion(outputs, labels)

        #Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        correct_train += (predicted == labels).sum().item()
        total_train += labels.size(0)

    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct_train / total_train
    complex_train_loss_history.append(train_loss)
    complex_train_acc_history.append(train_acc)

    complex_model.eval()
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = complex_model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            correct_val += (predicted == labels).sum().item()
            total_val += labels.size(0)

    val_loss /= len(test_loader)
    val_acc = 100 * correct_val / total_val
    complex_val_loss_history.append(val_loss)
    complex_val_acc_history.append(val_acc)
```

```
    print(f"Epoch [{epoch + 1}/{epochs}], Train Loss: {train_loss:.4f}, Train Acc:

end_time = time.time()
print(f"Training for Problem 1b completed in {end_time - start_time:.2f} seconds.")

plt.figure(figsize=(12, 5))
#Loss
plt.subplot(1, 2, 1)
plt.plot(complex_train_loss_history, label='Training Loss')
plt.plot(complex_val_loss_history, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Over Epochs (Complex Model)')

#Accuracy
plt.subplot(1, 2, 2)
plt.plot(complex_train_acc_history, label='Training Accuracy')
plt.plot(complex_val_acc_history, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs (Complex Model)')

plt.show()

complex_model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = complex_model(images)
        _, predicted = outputs.max(1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

#confusion matrix
print("\nClassification Report (Problem 1b):")
print(classification_report(y_true, y_pred, target_names=train_dataset.classes))

print("\nConfusion Matrix (Problem 1b):")
conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=train_datas
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix (Problem 1b)')
plt.show()
```

```
Training started...
Epoch [1/20], Train Loss: 1.6847, Train Acc: 39.91%, Val Loss: 1.5249, Val Acc: 46.6
5%
Epoch [2/20], Train Loss: 1.4554, Train Acc: 48.39%, Val Loss: 1.4352, Val Acc: 48.8
5%
Epoch [3/20], Train Loss: 1.3359, Train Acc: 52.71%, Val Loss: 1.3772, Val Acc: 51.7
8%
Epoch [4/20], Train Loss: 1.2378, Train Acc: 56.08%, Val Loss: 1.3596, Val Acc: 52.4
3%
Epoch [5/20], Train Loss: 1.1539, Train Acc: 58.85%, Val Loss: 1.3621, Val Acc: 53.2
1%
Epoch [6/20], Train Loss: 1.0752, Train Acc: 61.64%, Val Loss: 1.3383, Val Acc: 53.6
0%
Epoch [7/20], Train Loss: 1.0014, Train Acc: 64.07%, Val Loss: 1.3580, Val Acc: 54.3
4%
Epoch [8/20], Train Loss: 0.9328, Train Acc: 66.81%, Val Loss: 1.4192, Val Acc: 53.8
2%
Epoch [9/20], Train Loss: 0.8679, Train Acc: 68.90%, Val Loss: 1.4424, Val Acc: 54.1
9%
Epoch [10/20], Train Loss: 0.8060, Train Acc: 71.28%, Val Loss: 1.4523, Val Acc: 5
3.91%
Epoch [11/20], Train Loss: 0.7453, Train Acc: 73.28%, Val Loss: 1.5571, Val Acc: 5
4.03%
Epoch [12/20], Train Loss: 0.6944, Train Acc: 75.03%, Val Loss: 1.6275, Val Acc: 5
3.74%
Epoch [13/20], Train Loss: 0.6489, Train Acc: 76.65%, Val Loss: 1.6915, Val Acc: 5
3.32%
Epoch [14/20], Train Loss: 0.6031, Train Acc: 78.19%, Val Loss: 1.7041, Val Acc: 5
3.88%
Epoch [15/20], Train Loss: 0.5561, Train Acc: 80.04%, Val Loss: 1.8051, Val Acc: 5
4.28%
Epoch [16/20], Train Loss: 0.5215, Train Acc: 81.13%, Val Loss: 1.8531, Val Acc: 5
4.26%
Epoch [17/20], Train Loss: 0.4898, Train Acc: 82.58%, Val Loss: 1.9845, Val Acc: 5
3.92%
Epoch [18/20], Train Loss: 0.4634, Train Acc: 83.43%, Val Loss: 2.0323, Val Acc: 5
2.96%
Epoch [19/20], Train Loss: 0.4373, Train Acc: 84.40%, Val Loss: 2.1279, Val Acc: 5
3.60%
Epoch [20/20], Train Loss: 0.4094, Train Acc: 85.46%, Val Loss: 2.1423, Val Acc: 5
3.51%
Training for Problem 1b completed in 149.73 seconds.
```

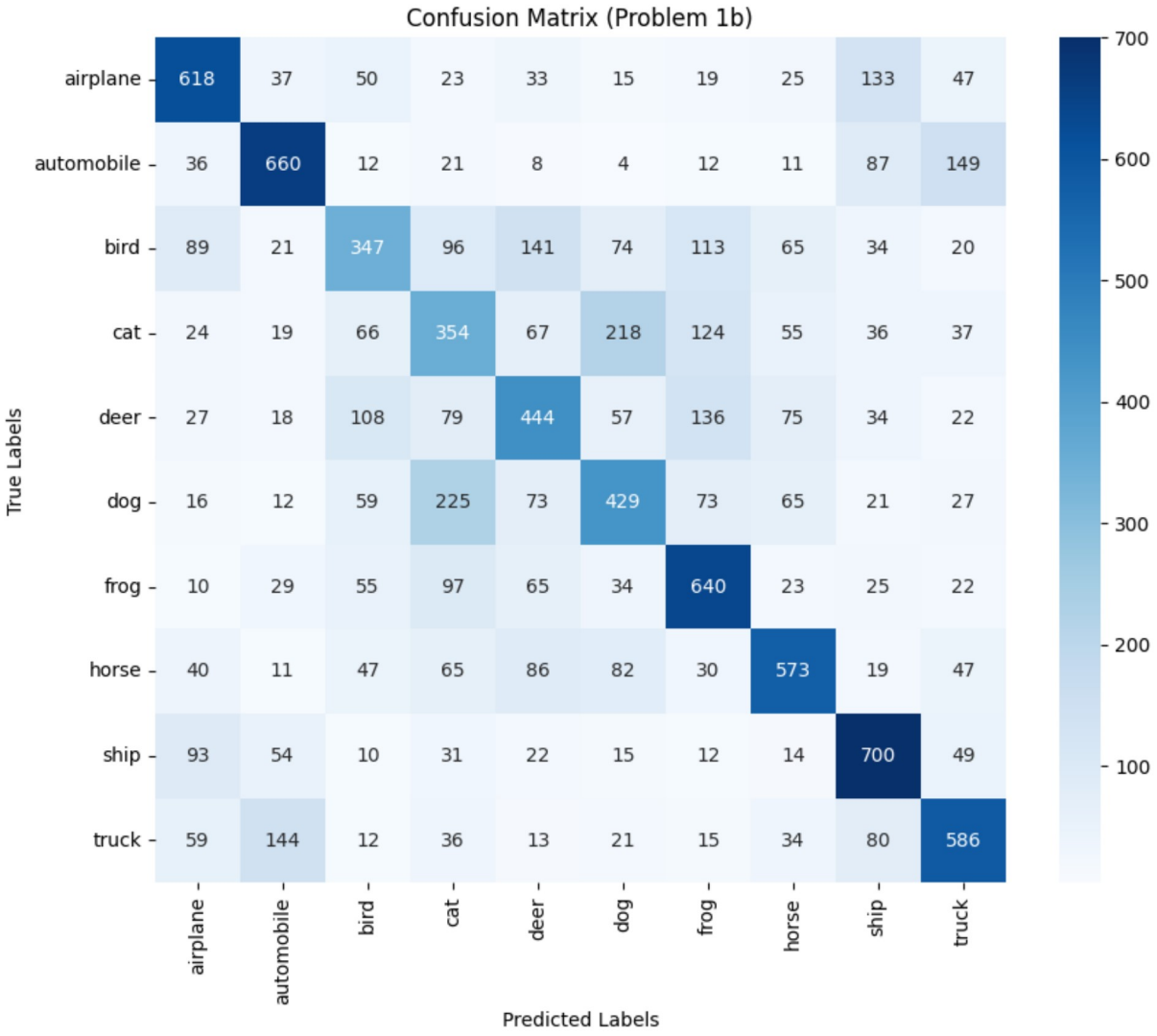




Classification Report (Problem 1b):

	precision	recall	f1-score	support
airplane	0.61	0.62	0.61	1000
automobile	0.66	0.66	0.66	1000
bird	0.45	0.35	0.39	1000
cat	0.34	0.35	0.35	1000
deer	0.47	0.44	0.45	1000
dog	0.45	0.43	0.44	1000
frog	0.55	0.64	0.59	1000
horse	0.61	0.57	0.59	1000
ship	0.60	0.70	0.65	1000
truck	0.58	0.59	0.58	1000
accuracy			0.54	10000
macro avg	0.53	0.54	0.53	10000
weighted avg	0.53	0.54	0.53	10000

Confusion Matrix (Problem 1b):



In [3]:
```
#Problem 2a
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time

#dataset
url = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads/main/Da
data = pd.read_csv(url)

for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].astype('category').cat.codes

target_column = data.columns[-1]
features = data.drop([target_column], axis=1)
targets = data[target_column].values.reshape(-1, 1)
```

```
features = features.apply(pd.to_numeric, errors='coerce').fillna(0)
scaler = StandardScaler()
features = scaler.fit_transform(features)

#split & tensor conversion
X_train, X_val, y_train, y_val = train_test_split(features, targets, test_size=0.2,
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_val_tensor = torch.tensor(X_val, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.float32)

#dataloader
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
val_dataset = TensorDataset(X_val_tensor, y_val_tensor)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)

#MLP model
class MLPHousing(nn.Module):
    def __init__(self, input_size):
        super(MLPHousing, self).__init__()
        self.fc1 = nn.Linear(input_size, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

#model setup
input_size = X_train.shape[1]
model = MLPHousing(input_size)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


#training loop
epochs = 75
train_loss_history, val_loss_history = [], []
start_time = time.time()

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
```

```
        optimizer.step()
        running_loss += loss.item() * X_batch.size(0)

    train_loss = running_loss / len(train_loader.dataset)
    train_loss_history.append(train_loss)

    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            y_pred = model(X_batch)
            loss = criterion(y_pred, y_batch)
            val_loss += loss.item() * X_batch.size(0)

    val_loss /= len(val_loader.dataset)
    val_loss_history.append(val_loss)

    print(f"Epoch [{epoch+1}/{epochs}], Train Loss: {train_loss:.4f}, Val Loss: {va

end_time = time.time()
training_time = end_time - start_time

plt.figure(figsize=(10, 5))
plt.plot(train_loss_history, label='Training Loss')
plt.plot(val_loss_history, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

model.eval()
with torch.no_grad():
    X_val_tensor, y_val_tensor = X_val_tensor.to(device), y_val_tensor.to(device)
    y_val_pred = model(X_val_tensor)
    mse = criterion(y_val_pred, y_val_tensor).item()
    rmse = np.sqrt(mse)

print(f"Final Validation MSE: {mse:.4f}")
print(f"Final Validation RMSE: {rmse:.4f}")
print(f"Model Complexity (Total Parameters): {sum(p.numel() for p in model.paramete
print(f"Total Training Time: {training_time:.2f} seconds")
```

```
Epoch [1/75], Train Loss: 1.1236, Val Loss: 0.8398
Epoch [2/75], Train Loss: 0.6164, Val Loss: 0.6556
Epoch [3/75], Train Loss: 0.5364, Val Loss: 0.6206
Epoch [4/75], Train Loss: 0.5098, Val Loss: 0.6091
Epoch [5/75], Train Loss: 0.4934, Val Loss: 0.6075
Epoch [6/75], Train Loss: 0.4892, Val Loss: 0.6017
Epoch [7/75], Train Loss: 0.4828, Val Loss: 0.6107
Epoch [8/75], Train Loss: 0.4753, Val Loss: 0.6069
Epoch [9/75], Train Loss: 0.4686, Val Loss: 0.6067
Epoch [10/75], Train Loss: 0.4659, Val Loss: 0.6108
Epoch [11/75], Train Loss: 0.4612, Val Loss: 0.6146
Epoch [12/75], Train Loss: 0.4569, Val Loss: 0.6107
Epoch [13/75], Train Loss: 0.4405, Val Loss: 0.6189
Epoch [14/75], Train Loss: 0.4358, Val Loss: 0.6232
Epoch [15/75], Train Loss: 0.4307, Val Loss: 0.6107
Epoch [16/75], Train Loss: 0.4191, Val Loss: 0.6259
Epoch [17/75], Train Loss: 0.4146, Val Loss: 0.6277
Epoch [18/75], Train Loss: 0.4037, Val Loss: 0.6412
Epoch [19/75], Train Loss: 0.4005, Val Loss: 0.6347
Epoch [20/75], Train Loss: 0.3882, Val Loss: 0.6639
Epoch [21/75], Train Loss: 0.3777, Val Loss: 0.6531
Epoch [22/75], Train Loss: 0.3750, Val Loss: 0.6543
Epoch [23/75], Train Loss: 0.3667, Val Loss: 0.6678
Epoch [24/75], Train Loss: 0.3654, Val Loss: 0.6871
Epoch [25/75], Train Loss: 0.3483, Val Loss: 0.6755
Epoch [26/75], Train Loss: 0.3410, Val Loss: 0.6874
Epoch [27/75], Train Loss: 0.3316, Val Loss: 0.6985
Epoch [28/75], Train Loss: 0.3214, Val Loss: 0.7011
Epoch [29/75], Train Loss: 0.3146, Val Loss: 0.6985
Epoch [30/75], Train Loss: 0.3046, Val Loss: 0.7619
Epoch [31/75], Train Loss: 0.3154, Val Loss: 0.6893
Epoch [32/75], Train Loss: 0.3020, Val Loss: 0.7296
Epoch [33/75], Train Loss: 0.2879, Val Loss: 0.7415
Epoch [34/75], Train Loss: 0.2788, Val Loss: 0.7343
Epoch [35/75], Train Loss: 0.2714, Val Loss: 0.7338
Epoch [36/75], Train Loss: 0.2587, Val Loss: 0.7535
Epoch [37/75], Train Loss: 0.2553, Val Loss: 0.7581
Epoch [38/75], Train Loss: 0.2548, Val Loss: 0.7683
Epoch [39/75], Train Loss: 0.2414, Val Loss: 0.7893
Epoch [40/75], Train Loss: 0.2346, Val Loss: 0.7695
Epoch [41/75], Train Loss: 0.2366, Val Loss: 0.7700
Epoch [42/75], Train Loss: 0.2246, Val Loss: 0.8172
Epoch [43/75], Train Loss: 0.2192, Val Loss: 0.8515
Epoch [44/75], Train Loss: 0.2243, Val Loss: 0.7804
Epoch [45/75], Train Loss: 0.2058, Val Loss: 0.8100
Epoch [46/75], Train Loss: 0.2016, Val Loss: 0.8496
Epoch [47/75], Train Loss: 0.1973, Val Loss: 0.8052
Epoch [48/75], Train Loss: 0.1985, Val Loss: 0.8230
Epoch [49/75], Train Loss: 0.1942, Val Loss: 0.8624
Epoch [50/75], Train Loss: 0.1775, Val Loss: 0.8439
Epoch [51/75], Train Loss: 0.1761, Val Loss: 0.8718
Epoch [52/75], Train Loss: 0.1770, Val Loss: 0.8534
Epoch [53/75], Train Loss: 0.1732, Val Loss: 0.8688
Epoch [54/75], Train Loss: 0.1778, Val Loss: 0.8423
Epoch [55/75], Train Loss: 0.1811, Val Loss: 0.8744
Epoch [56/75], Train Loss: 0.1698, Val Loss: 0.8850
```

```
Epoch [57/75], Train Loss: 0.1714, Val Loss: 0.8955
Epoch [58/75], Train Loss: 0.1651, Val Loss: 0.8441
Epoch [59/75], Train Loss: 0.1576, Val Loss: 0.8666
Epoch [60/75], Train Loss: 0.1502, Val Loss: 0.8613
Epoch [61/75], Train Loss: 0.1502, Val Loss: 0.8790
Epoch [62/75], Train Loss: 0.1376, Val Loss: 0.8997
Epoch [63/75], Train Loss: 0.1507, Val Loss: 0.8663
Epoch [64/75], Train Loss: 0.1589, Val Loss: 0.8952
Epoch [65/75], Train Loss: 0.1528, Val Loss: 0.8905
Epoch [66/75], Train Loss: 0.1416, Val Loss: 0.8962
Epoch [67/75], Train Loss: 0.1460, Val Loss: 0.9251
Epoch [68/75], Train Loss: 0.1296, Val Loss: 0.8832
Epoch [69/75], Train Loss: 0.1321, Val Loss: 0.9375
Epoch [70/75], Train Loss: 0.1231, Val Loss: 0.9378
Epoch [71/75], Train Loss: 0.1303, Val Loss: 0.9295
Epoch [72/75], Train Loss: 0.1537, Val Loss: 0.9242
Epoch [73/75], Train Loss: 0.1421, Val Loss: 0.8898
Epoch [74/75], Train Loss: 0.1345, Val Loss: 0.9201
Epoch [75/75], Train Loss: 0.1298, Val Loss: 0.9288
```


```
Final Validation MSE: 0.9288
Final Validation RMSE: 0.9637
Model Complexity (Total Parameters): 12033
Total Training Time: 1.77 seconds
```

In [12]:
```python
# Import required libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```

```
# Load and prepare dataset
data_source = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads
house_data = pd.read_csv(data_source)

# Convert categorical variables to numeric
for column in house_data.select_dtypes(include=['object']).columns:
    house_data[column] = house_data[column].astype('category').cat.codes

# Separate features and target
price_column = house_data.columns[-1]
input_features = house_data.drop([price_column], axis=1)
house_prices = house_data[price_column].values.reshape(-1, 1)

# Clean and scale the data
input_features = input_features.apply(pd.to_numeric, errors='coerce').fillna(0)
feature_scaler = StandardScaler()
input_features = feature_scaler.fit_transform(input_features)

# Split data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(input_features, house_prices,
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_valid_tensor = torch.tensor(X_valid, dtype=torch.float32)
y_valid_tensor = torch.tensor(y_valid, dtype=torch.float32)

# Create data loaders
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
valid_dataset = TensorDataset(X_valid_tensor, y_valid_tensor)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(dataset=valid_dataset, batch_size=32, shuffle=False)

# Define neural network architecture
class HousePriceNet(nn.Module):
    def __init__(self, input_dim):
        super(HousePriceNet, self).__init__()
        self.layer1 = nn.Linear(input_dim, 128)
        self.layer2 = nn.Linear(128, 64)
        self.layer3 = nn.Linear(64, 32)
        self.layer4 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = torch.relu(self.layer3(x))
        x = self.layer4(x)
        return x

# Initialize model and training components
input_dim = X_train.shape[1]
model = HousePriceNet(input_dim)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
```

```
num_epochs = 75
train_losses, valid_losses = [], []
start_time = time.time()

for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        predictions = model(batch_X)
        loss = loss_function(predictions, batch_y)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item() * batch_X.size(0)

    avg_train_loss = epoch_loss / len(train_loader.dataset)
    train_losses.append(avg_train_loss)

    model.eval()
    valid_epoch_loss = 0.0
    with torch.no_grad():
        for batch_X, batch_y in valid_loader:
            batch_X, batch_y = batch_X.to(device), batch_y.to(device)
            predictions = model(batch_X)
            loss = loss_function(predictions, batch_y)
            valid_epoch_loss += loss.item() * batch_X.size(0)

    avg_valid_loss = valid_epoch_loss / len(valid_loader.dataset)
    valid_losses.append(avg_valid_loss)

    print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {avg_train_loss:.4f}, Va

total_time = time.time() - start_time

# Plot training results
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.show()

# Final evaluation
model.eval()
with torch.no_grad():
    X_valid_tensor, y_valid_tensor = X_valid_tensor.to(device), y_valid_tensor.to(d
    final_predictions = model(X_valid_tensor)
    final_mse = loss_function(final_predictions, y_valid_tensor).item()
    final_rmse = np.sqrt(final_mse)

print(f"Final MSE on Validation Set: {final_mse:.4f}")
print(f"Final RMSE on Validation Set: {final_rmse:.4f}")
print(f"Total Parameters in Model: {sum(p.numel() for p in model.parameters())}")
```

```
print(f"Training Duration: {total_time:.2f} seconds")
```

```
Epoch [1/75], Training Loss: 1.0700, Validation Loss: 0.8127
Epoch [2/75], Training Loss: 0.6071, Validation Loss: 0.6170
Epoch [3/75], Training Loss: 0.5154, Validation Loss: 0.6109
Epoch [4/75], Training Loss: 0.5041, Validation Loss: 0.6056
Epoch [5/75], Training Loss: 0.4967, Validation Loss: 0.6134
Epoch [6/75], Training Loss: 0.4867, Validation Loss: 0.6021
Epoch [7/75], Training Loss: 0.4733, Validation Loss: 0.6004
Epoch [8/75], Training Loss: 0.4672, Validation Loss: 0.5995
Epoch [9/75], Training Loss: 0.4644, Validation Loss: 0.6101
Epoch [10/75], Training Loss: 0.4515, Validation Loss: 0.6062
Epoch [11/75], Training Loss: 0.4452, Validation Loss: 0.6068
Epoch [12/75], Training Loss: 0.4337, Validation Loss: 0.6082
Epoch [13/75], Training Loss: 0.4276, Validation Loss: 0.6175
Epoch [14/75], Training Loss: 0.4200, Validation Loss: 0.6301
Epoch [15/75], Training Loss: 0.4112, Validation Loss: 0.6336
Epoch [16/75], Training Loss: 0.4056, Validation Loss: 0.6316
Epoch [17/75], Training Loss: 0.3942, Validation Loss: 0.6388
Epoch [18/75], Training Loss: 0.3874, Validation Loss: 0.6423
Epoch [19/75], Training Loss: 0.3803, Validation Loss: 0.6638
Epoch [20/75], Training Loss: 0.3672, Validation Loss: 0.6603
Epoch [21/75], Training Loss: 0.3617, Validation Loss: 0.6611
Epoch [22/75], Training Loss: 0.3478, Validation Loss: 0.6811
Epoch [23/75], Training Loss: 0.3326, Validation Loss: 0.6748
Epoch [24/75], Training Loss: 0.3243, Validation Loss: 0.6759
Epoch [25/75], Training Loss: 0.3213, Validation Loss: 0.7173
Epoch [26/75], Training Loss: 0.3157, Validation Loss: 0.6890
Epoch [27/75], Training Loss: 0.3174, Validation Loss: 0.6957
Epoch [28/75], Training Loss: 0.2922, Validation Loss: 0.7241
Epoch [29/75], Training Loss: 0.3101, Validation Loss: 0.7307
Epoch [30/75], Training Loss: 0.2808, Validation Loss: 0.7111
Epoch [31/75], Training Loss: 0.2771, Validation Loss: 0.7301
Epoch [32/75], Training Loss: 0.2721, Validation Loss: 0.7403
Epoch [33/75], Training Loss: 0.2591, Validation Loss: 0.7334
Epoch [34/75], Training Loss: 0.2486, Validation Loss: 0.7248
Epoch [35/75], Training Loss: 0.2488, Validation Loss: 0.7624
Epoch [36/75], Training Loss: 0.2441, Validation Loss: 0.7552
Epoch [37/75], Training Loss: 0.2379, Validation Loss: 0.7818
Epoch [38/75], Training Loss: 0.2420, Validation Loss: 0.7634
Epoch [39/75], Training Loss: 0.2305, Validation Loss: 0.7816
Epoch [40/75], Training Loss: 0.2201, Validation Loss: 0.7642
Epoch [41/75], Training Loss: 0.2201, Validation Loss: 0.8012
Epoch [42/75], Training Loss: 0.2248, Validation Loss: 0.7749
Epoch [43/75], Training Loss: 0.2141, Validation Loss: 0.8332
Epoch [44/75], Training Loss: 0.2047, Validation Loss: 0.7821
Epoch [45/75], Training Loss: 0.1956, Validation Loss: 0.8155
Epoch [46/75], Training Loss: 0.2009, Validation Loss: 0.8855
Epoch [47/75], Training Loss: 0.2010, Validation Loss: 0.8590
Epoch [48/75], Training Loss: 0.1944, Validation Loss: 0.8717
Epoch [49/75], Training Loss: 0.1953, Validation Loss: 0.8179
Epoch [50/75], Training Loss: 0.1755, Validation Loss: 0.8570
Epoch [51/75], Training Loss: 0.1735, Validation Loss: 0.8712
Epoch [52/75], Training Loss: 0.1780, Validation Loss: 0.8881
Epoch [53/75], Training Loss: 0.1705, Validation Loss: 0.8757
Epoch [54/75], Training Loss: 0.1552, Validation Loss: 0.8854
Epoch [55/75], Training Loss: 0.1542, Validation Loss: 0.9129
Epoch [56/75], Training Loss: 0.1617, Validation Loss: 0.9158
```

```
Epoch [57/75], Training Loss: 0.1642, Validation Loss: 0.8993
Epoch [58/75], Training Loss: 0.1636, Validation Loss: 0.9402
Epoch [59/75], Training Loss: 0.1674, Validation Loss: 0.8704
Epoch [60/75], Training Loss: 0.1676, Validation Loss: 0.9278
Epoch [61/75], Training Loss: 0.1531, Validation Loss: 0.9271
Epoch [62/75], Training Loss: 0.1393, Validation Loss: 0.9258
Epoch [63/75], Training Loss: 0.1376, Validation Loss: 0.9676
Epoch [64/75], Training Loss: 0.1348, Validation Loss: 0.9617
Epoch [65/75], Training Loss: 0.1286, Validation Loss: 0.9681
Epoch [66/75], Training Loss: 0.1293, Validation Loss: 0.9794
Epoch [67/75], Training Loss: 0.1255, Validation Loss: 0.9728
Epoch [68/75], Training Loss: 0.1225, Validation Loss: 1.0135
Epoch [69/75], Training Loss: 0.1266, Validation Loss: 0.9988
Epoch [70/75], Training Loss: 0.1202, Validation Loss: 0.9914
Epoch [71/75], Training Loss: 0.1223, Validation Loss: 1.0041
Epoch [72/75], Training Loss: 0.1332, Validation Loss: 0.9811
Epoch [73/75], Training Loss: 0.1334, Validation Loss: 0.9820
Epoch [74/75], Training Loss: 0.1311, Validation Loss: 1.0073
Epoch [75/75], Training Loss: 0.1343, Validation Loss: 1.0022
```


```
Final MSE on Validation Set: 1.0022
Final RMSE on Validation Set: 1.0011
Total Parameters in Model: 12033
Training Duration: 1.93 seconds
```

In [11]:
```python
#2a
# Import required libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```

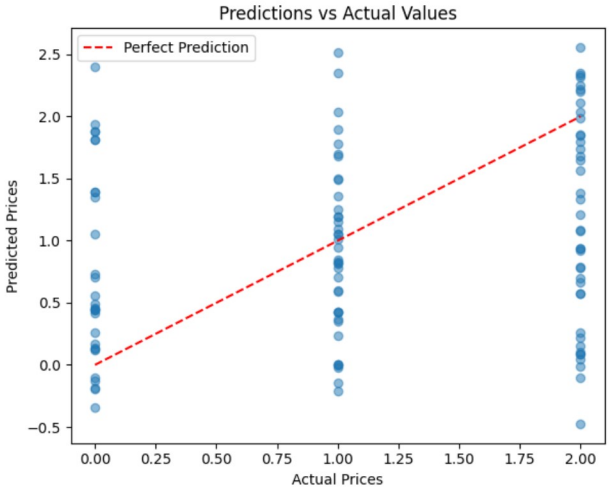
```
# Load and preprocess the housing dataset
url = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads/main/Da
data = pd.read_csv(url)

# Convert categorical variables to numeric
for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].astype('category').cat.codes

# Separate features and target
target_column = data.columns[-1]  # 'price' is the last column
features = data.drop([target_column], axis=1)
targets = data[target_column].values.reshape(-1, 1)

# Handle any missing values and convert to numeric
features = features.apply(pd.to_numeric, errors='coerce').fillna(0)

# Scale the features
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Split data and convert to PyTorch tensors
X_train, X_val, y_train, y_val = train_test_split(features, targets, test_size=0.2,
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_val_tensor = torch.tensor(X_val, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.float32)

# Create DataLoader objects for batch processing
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
val_dataset = TensorDataset(X_val_tensor, y_val_tensor)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)

# Define the MLP model for housing price prediction
class MLPHousing(nn.Module):
    def __init__(self, input_size):
        super(MLPHousing, self).__init__()
        # Modified architecture with gradually decreasing layer sizes
        self.model = nn.Sequential(
            nn.Linear(input_size, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

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

# Initialize model, loss function, and optimizer
input_size = X_train.shape[1]
model = MLPHousing(input_size)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs = 75
train_loss_history = []
val_loss_history = []
start_time = time.time()

for epoch in range(epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        # Forward pass and loss calculation
        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = criterion(y_pred, y_batch)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * X_batch.size(0)

    train_loss = running_loss / len(train_loader.dataset)
    train_loss_history.append(train_loss)

    # Validation phase
    model.eval()
    val_loss = 0.0
    val_predictions = []
    val_targets = []

    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            y_pred = model(X_batch)
            loss = criterion(y_pred, y_batch)
            val_loss += loss.item() * X_batch.size(0)

            # Store predictions and targets for additional metrics
            val_predictions.extend(y_pred.cpu().numpy())
            val_targets.extend(y_batch.cpu().numpy())

    val_loss /= len(val_loader.dataset)
    val_loss_history.append(val_loss)

    print(f"Epoch [{epoch+1}/{epochs}], Train Loss: {train_loss:.4f}, Val Loss: {va

end_time = time.time()
training_time = end_time - start_time
```

```
# Plot training results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_loss_history, label='Training Loss')
plt.plot(val_loss_history, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Training and Validation Loss')
plt.legend()

# Plot predictions vs actual values
plt.subplot(1, 2, 2)
val_predictions = np.array(val_predictions)
val_targets = np.array(val_targets)
plt.scatter(val_targets, val_predictions, alpha=0.5)
plt.plot([val_targets.min(), val_targets.max()],
         [val_targets.min(), val_targets.max()],
         'r--', label='Perfect Prediction')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Predictions vs Actual Values')
plt.legend()
plt.tight_layout()
plt.show()

# Calculate and display final metrics
model.eval()
with torch.no_grad():
    X_val_tensor, y_val_tensor = X_val_tensor.to(device), y_val_tensor.to(device)
    y_val_pred = model(X_val_tensor)
    mse = criterion(y_val_pred, y_val_tensor).item()
    rmse = np.sqrt(mse)

    # Calculate R² score
    y_val_mean = torch.mean(y_val_tensor)
    ss_tot = torch.sum((y_val_tensor - y_val_mean) ** 2)
    ss_res = torch.sum((y_val_tensor - y_val_pred) ** 2)
    r2 = 1 - (ss_res / ss_tot)

print("\nFinal Model Performance:")
print(f"Validation MSE: {mse:.4f}")
print(f"Validation RMSE: {rmse:.4f}")
print(f"R² Score: {r2.item():.4f}")
print(f"Model Parameters: {sum(p.numel() for p in model.parameters())}")
print(f"Training Time: {training_time:.2f} seconds")
```

```
Epoch [1/75], Train Loss: 1.0983, Val Loss: 0.6401
Epoch [2/75], Train Loss: 0.5991, Val Loss: 0.6196
Epoch [3/75], Train Loss: 0.5221, Val Loss: 0.5854
Epoch [4/75], Train Loss: 0.4997, Val Loss: 0.5919
Epoch [5/75], Train Loss: 0.4898, Val Loss: 0.5974
Epoch [6/75], Train Loss: 0.4827, Val Loss: 0.6021
Epoch [7/75], Train Loss: 0.4746, Val Loss: 0.6125
Epoch [8/75], Train Loss: 0.4697, Val Loss: 0.6132
Epoch [9/75], Train Loss: 0.4565, Val Loss: 0.6122
Epoch [10/75], Train Loss: 0.4601, Val Loss: 0.6295
Epoch [11/75], Train Loss: 0.4468, Val Loss: 0.6362
Epoch [12/75], Train Loss: 0.4252, Val Loss: 0.6164
Epoch [13/75], Train Loss: 0.4120, Val Loss: 0.6306
Epoch [14/75], Train Loss: 0.4061, Val Loss: 0.6310
Epoch [15/75], Train Loss: 0.3866, Val Loss: 0.6524
Epoch [16/75], Train Loss: 0.3766, Val Loss: 0.6437
Epoch [17/75], Train Loss: 0.3595, Val Loss: 0.6730
Epoch [18/75], Train Loss: 0.3478, Val Loss: 0.6656
Epoch [19/75], Train Loss: 0.3339, Val Loss: 0.6864
Epoch [20/75], Train Loss: 0.3189, Val Loss: 0.6932
Epoch [21/75], Train Loss: 0.3149, Val Loss: 0.7021
Epoch [22/75], Train Loss: 0.2973, Val Loss: 0.7474
Epoch [23/75], Train Loss: 0.2980, Val Loss: 0.7698
Epoch [24/75], Train Loss: 0.2965, Val Loss: 0.7164
Epoch [25/75], Train Loss: 0.2766, Val Loss: 0.7572
Epoch [26/75], Train Loss: 0.2645, Val Loss: 0.7838
Epoch [27/75], Train Loss: 0.2631, Val Loss: 0.7592
Epoch [28/75], Train Loss: 0.2486, Val Loss: 0.7963
Epoch [29/75], Train Loss: 0.2433, Val Loss: 0.8132
Epoch [30/75], Train Loss: 0.2265, Val Loss: 0.8168
Epoch [31/75], Train Loss: 0.2253, Val Loss: 0.8086
Epoch [32/75], Train Loss: 0.2131, Val Loss: 0.8319
Epoch [33/75], Train Loss: 0.2068, Val Loss: 0.8431
Epoch [34/75], Train Loss: 0.2117, Val Loss: 0.8500
Epoch [35/75], Train Loss: 0.2062, Val Loss: 0.8626
Epoch [36/75], Train Loss: 0.1874, Val Loss: 0.8409
Epoch [37/75], Train Loss: 0.1919, Val Loss: 0.8764
Epoch [38/75], Train Loss: 0.1772, Val Loss: 0.8459
Epoch [39/75], Train Loss: 0.1943, Val Loss: 0.8286
Epoch [40/75], Train Loss: 0.1745, Val Loss: 0.9243
Epoch [41/75], Train Loss: 0.1816, Val Loss: 0.8421
Epoch [42/75], Train Loss: 0.1632, Val Loss: 0.9073
Epoch [43/75], Train Loss: 0.1726, Val Loss: 0.9034
Epoch [44/75], Train Loss: 0.1627, Val Loss: 0.8673
Epoch [45/75], Train Loss: 0.1575, Val Loss: 0.9053
Epoch [46/75], Train Loss: 0.1484, Val Loss: 0.9007
Epoch [47/75], Train Loss: 0.1575, Val Loss: 0.8929
Epoch [48/75], Train Loss: 0.1613, Val Loss: 0.8871
Epoch [49/75], Train Loss: 0.1521, Val Loss: 0.8850
Epoch [50/75], Train Loss: 0.1426, Val Loss: 0.8985
Epoch [51/75], Train Loss: 0.1412, Val Loss: 0.8951
Epoch [52/75], Train Loss: 0.1374, Val Loss: 0.9292
Epoch [53/75], Train Loss: 0.1619, Val Loss: 0.8887
Epoch [54/75], Train Loss: 0.1488, Val Loss: 0.8795
Epoch [55/75], Train Loss: 0.1437, Val Loss: 0.9355
Epoch [56/75], Train Loss: 0.1279, Val Loss: 0.9541
```

```
Epoch [57/75], Train Loss: 0.1310, Val Loss: 0.9138
Epoch [58/75], Train Loss: 0.1348, Val Loss: 0.9119
Epoch [59/75], Train Loss: 0.1261, Val Loss: 0.9303
Epoch [60/75], Train Loss: 0.1170, Val Loss: 0.9379
Epoch [61/75], Train Loss: 0.1161, Val Loss: 0.9362
Epoch [62/75], Train Loss: 0.1198, Val Loss: 0.9533
Epoch [63/75], Train Loss: 0.1238, Val Loss: 0.9461
Epoch [64/75], Train Loss: 0.1240, Val Loss: 0.9543
Epoch [65/75], Train Loss: 0.1189, Val Loss: 0.9078
Epoch [66/75], Train Loss: 0.1162, Val Loss: 0.9699
Epoch [67/75], Train Loss: 0.1166, Val Loss: 0.9084
Epoch [68/75], Train Loss: 0.1104, Val Loss: 0.9952
Epoch [69/75], Train Loss: 0.1109, Val Loss: 0.9142
Epoch [70/75], Train Loss: 0.1101, Val Loss: 1.0209
Epoch [71/75], Train Loss: 0.1228, Val Loss: 0.9299
Epoch [72/75], Train Loss: 0.1258, Val Loss: 0.9079
Epoch [73/75], Train Loss: 0.1241, Val Loss: 0.9751
Epoch [74/75], Train Loss: 0.1228, Val Loss: 0.9000
Epoch [75/75], Train Loss: 0.1122, Val Loss: 0.9943
```




```
Final Model Performance:
Validation MSE: 0.9943
Validation RMSE: 0.9972
R² Score: -0.5781
Model Parameters: 44545
Training Time: 1.79 seconds
```

In [ ]:

```python
#2b
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time

# Load housing dataset from GitHub repository
data_url = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads/ma
housing_dataset = pd.read_csv(data_url)
```

```
# Convert categorical variables to numeric codes
for col in housing_dataset.select_dtypes(include=['object']).columns:
    housing_dataset[col] = housing_dataset[col].astype('category').cat.codes

# Perform one-hot encoding on categorical variables
feature_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
categorical_cols = housing_dataset.select_dtypes(include=['category', 'object']).co
encoded_features = feature_encoder.fit_transform(housing_dataset[categorical_cols])
encoded_features = pd.DataFrame(encoded_features, columns=feature_encoder.get_featu
housing_dataset = pd.concat([housing_dataset.drop(categorical_cols, axis=1), encode

# Define target variable and separate features
target_column = "SalePrice"
if target_column not in housing_dataset.columns:
    target_column = housing_dataset.columns[-1]

input_data = housing_dataset.drop([target_column], axis=1)
price_targets = housing_dataset[target_column].values.reshape(-1, 1)

# Clean and scale the input features
input_data = input_data.apply(pd.to_numeric, errors='coerce').fillna(0)
data_scaler = StandardScaler()
input_data = data_scaler.fit_transform(input_data)

# Split data into training and validation sets
train_features, val_features, train_targets, val_targets = train_test_split(
    input_data, price_targets, test_size=0.2, random_state=42
)

# Convert numpy arrays to PyTorch tensors
train_features_tensor = torch.tensor(train_features, dtype=torch.float32)
train_targets_tensor = torch.tensor(train_targets, dtype=torch.float32)
val_features_tensor = torch.tensor(val_features, dtype=torch.float32)
val_targets_tensor = torch.tensor(val_targets, dtype=torch.float32)

# Create DataLoader objects for batch processing
train_dataset = TensorDataset(train_features_tensor, train_targets_tensor)
val_dataset = TensorDataset(val_features_tensor, val_targets_tensor)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)

# Define neural network architecture for house price prediction
class HousePriceModel(nn.Module):
    def __init__(self, input_size):
        super(HousePriceModel, self).__init__()
        # Define network layers with descriptive names
        self.input_layer = nn.Linear(input_size, 128)
        self.hidden_layer1 = nn.Linear(128, 64)
        self.hidden_layer2 = nn.Linear(64, 32)
        self.output_layer = nn.Linear(32, 1)

    def forward(self, x):
        # Apply ReLU activation after each layer except the output
        x = torch.relu(self.input_layer(x))
        x = torch.relu(self.hidden_layer1(x))
```

```
        x = torch.relu(self.hidden_layer2(x))
        x = self.output_layer(x)
        return x

# Initialize model and training components
input_dim = train_features.shape[1]
model = HousePriceModel(input_dim)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training settings
num_epochs = 75
training_losses = []
validation_losses = []
start_time = time.time()

# Training loop
for epoch in range(num_epochs):
    # Training phase
    model.train()
    batch_loss = 0.0
    for batch_features, batch_targets in train_loader:
        # Move batch data to appropriate device
        batch_features = batch_features.to(device)
        batch_targets = batch_targets.to(device)

        # Forward pass and loss calculation
        optimizer.zero_grad()
        predictions = model(batch_features)
        loss = loss_function(predictions, batch_targets)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()
        batch_loss += loss.item() * batch_features.size(0)

    # Calculate average training loss
    avg_train_loss = batch_loss / len(train_loader.dataset)
    training_losses.append(avg_train_loss)

    # Validation phase
    model.eval()
    val_batch_loss = 0.0
    with torch.no_grad():
        for batch_features, batch_targets in val_loader:
            batch_features = batch_features.to(device)
            batch_targets = batch_targets.to(device)
            predictions = model(batch_features)
            loss = loss_function(predictions, batch_targets)
            val_batch_loss += loss.item() * batch_features.size(0)

    # Calculate average validation loss
    avg_val_loss = val_batch_loss / len(val_loader.dataset)
    validation_losses.append(avg_val_loss)
```

```
    # Print progress
    print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {avg_train_loss:.4f}, Va

# Calculate total training time
total_time = time.time() - start_time

# Visualize training progress
plt.figure(figsize=(10, 5))
plt.plot(training_losses, label='Training Loss')
plt.plot(validation_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.show()

# Final model evaluation
model.eval()
with torch.no_grad():
    val_features_tensor = val_features_tensor.to(device)
    val_targets_tensor = val_targets_tensor.to(device)
    final_predictions = model(val_features_tensor)
    final_mse = loss_function(final_predictions, val_targets_tensor).item()
    final_rmse = np.sqrt(final_mse)

# Print final metrics
print(f"Final Validation MSE: {final_mse:.4f}")
print(f"Final Validation RMSE: {final_rmse:.4f}")
print(f"Model Complexity (Total Parameters): {sum(p.numel() for p in model.paramete
print(f"Total Training Time: {total_time:.2f} seconds")
```

```
Epoch [1/75], Training Loss: 1.3997, Validation Loss: 1.0229
Epoch [2/75], Training Loss: 0.7478, Validation Loss: 0.6702
Epoch [3/75], Training Loss: 0.5750, Validation Loss: 0.6264
Epoch [4/75], Training Loss: 0.5400, Validation Loss: 0.6174
Epoch [5/75], Training Loss: 0.5112, Validation Loss: 0.6120
Epoch [6/75], Training Loss: 0.4996, Validation Loss: 0.6111
Epoch [7/75], Training Loss: 0.4908, Validation Loss: 0.6170
Epoch [8/75], Training Loss: 0.4801, Validation Loss: 0.6145
Epoch [9/75], Training Loss: 0.4748, Validation Loss: 0.6164
Epoch [10/75], Training Loss: 0.4702, Validation Loss: 0.6151
Epoch [11/75], Training Loss: 0.4626, Validation Loss: 0.6218
Epoch [12/75], Training Loss: 0.4617, Validation Loss: 0.6169
Epoch [13/75], Training Loss: 0.4455, Validation Loss: 0.6139
Epoch [14/75], Training Loss: 0.4391, Validation Loss: 0.6176
Epoch [15/75], Training Loss: 0.4288, Validation Loss: 0.6257
Epoch [16/75], Training Loss: 0.4225, Validation Loss: 0.6254
Epoch [17/75], Training Loss: 0.4169, Validation Loss: 0.6274
Epoch [18/75], Training Loss: 0.4083, Validation Loss: 0.6312
Epoch [19/75], Training Loss: 0.4002, Validation Loss: 0.6404
Epoch [20/75], Training Loss: 0.3907, Validation Loss: 0.6332
Epoch [21/75], Training Loss: 0.3824, Validation Loss: 0.6392
Epoch [22/75], Training Loss: 0.3736, Validation Loss: 0.6459
Epoch [23/75], Training Loss: 0.3648, Validation Loss: 0.6582
Epoch [24/75], Training Loss: 0.3581, Validation Loss: 0.6471
Epoch [25/75], Training Loss: 0.3537, Validation Loss: 0.6933
Epoch [26/75], Training Loss: 0.3377, Validation Loss: 0.6531
Epoch [27/75], Training Loss: 0.3313, Validation Loss: 0.6844
Epoch [28/75], Training Loss: 0.3299, Validation Loss: 0.6723
Epoch [29/75], Training Loss: 0.3169, Validation Loss: 0.6785
Epoch [30/75], Training Loss: 0.3074, Validation Loss: 0.6998
Epoch [31/75], Training Loss: 0.3048, Validation Loss: 0.6742
Epoch [32/75], Training Loss: 0.3038, Validation Loss: 0.6931
Epoch [33/75], Training Loss: 0.2917, Validation Loss: 0.6847
Epoch [34/75], Training Loss: 0.2814, Validation Loss: 0.7345
Epoch [35/75], Training Loss: 0.2781, Validation Loss: 0.7737
Epoch [36/75], Training Loss: 0.2956, Validation Loss: 0.6843
Epoch [37/75], Training Loss: 0.2609, Validation Loss: 0.7423
Epoch [38/75], Training Loss: 0.2503, Validation Loss: 0.7142
Epoch [39/75], Training Loss: 0.2562, Validation Loss: 0.7305
Epoch [40/75], Training Loss: 0.2491, Validation Loss: 0.7551
Epoch [41/75], Training Loss: 0.2379, Validation Loss: 0.7140
Epoch [42/75], Training Loss: 0.2241, Validation Loss: 0.7484
Epoch [43/75], Training Loss: 0.2293, Validation Loss: 0.7791
Epoch [44/75], Training Loss: 0.2174, Validation Loss: 0.7398
Epoch [45/75], Training Loss: 0.2219, Validation Loss: 0.7924
Epoch [46/75], Training Loss: 0.2121, Validation Loss: 0.7761
Epoch [47/75], Training Loss: 0.2130, Validation Loss: 0.7719
Epoch [48/75], Training Loss: 0.2188, Validation Loss: 0.7695
Epoch [49/75], Training Loss: 0.2175, Validation Loss: 0.7651
Epoch [50/75], Training Loss: 0.2020, Validation Loss: 0.7907
Epoch [51/75], Training Loss: 0.1962, Validation Loss: 0.7844
Epoch [52/75], Training Loss: 0.1938, Validation Loss: 0.7856
Epoch [53/75], Training Loss: 0.1829, Validation Loss: 0.8084
Epoch [54/75], Training Loss: 0.1921, Validation Loss: 0.8435
Epoch [55/75], Training Loss: 0.1856, Validation Loss: 0.8376
Epoch [56/75], Training Loss: 0.1816, Validation Loss: 0.7980
```

```
Epoch [57/75], Training Loss: 0.1752, Validation Loss: 0.7960
Epoch [58/75], Training Loss: 0.1722, Validation Loss: 0.7926
Epoch [59/75], Training Loss: 0.1688, Validation Loss: 0.8316
Epoch [60/75], Training Loss: 0.1700, Validation Loss: 0.8164
Epoch [61/75], Training Loss: 0.1729, Validation Loss: 0.8212
Epoch [62/75], Training Loss: 0.1668, Validation Loss: 0.8493
Epoch [63/75], Training Loss: 0.1658, Validation Loss: 0.8635
Epoch [64/75], Training Loss: 0.1543, Validation Loss: 0.8433
Epoch [65/75], Training Loss: 0.1567, Validation Loss: 0.8254
Epoch [66/75], Training Loss: 0.1546, Validation Loss: 0.8731
Epoch [67/75], Training Loss: 0.1520, Validation Loss: 0.8600
Epoch [68/75], Training Loss: 0.1507, Validation Loss: 0.8495
Epoch [69/75], Training Loss: 0.1504, Validation Loss: 0.8276
Epoch [70/75], Training Loss: 0.1506, Validation Loss: 0.8634
Epoch [71/75], Training Loss: 0.1438, Validation Loss: 0.8609
Epoch [72/75], Training Loss: 0.1438, Validation Loss: 0.8186
Epoch [73/75], Training Loss: 0.1473, Validation Loss: 0.8702
Epoch [74/75], Training Loss: 0.1670, Validation Loss: 0.8136
Epoch [75/75], Training Loss: 0.1493, Validation Loss: 0.8392
```


```
Final Validation MSE: 0.8392
Final Validation RMSE: 0.9161
Model Complexity (Total Parameters): 12033
Total Training Time: 1.63 seconds
```

In [ ]:
```python
#2c
# Import necessary libraries for the deep learning model
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```

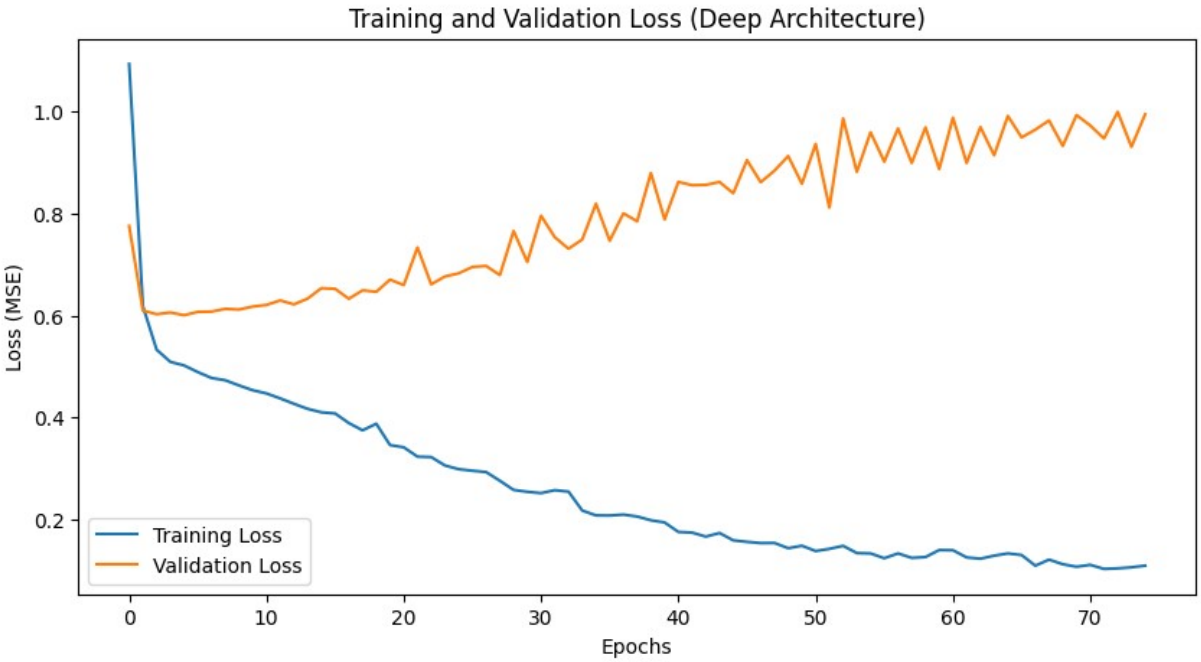
```
# Load the housing dataset
data_source = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads
raw_data = pd.read_csv(data_source)

# Convert categorical variables to numeric codes
for column in raw_data.select_dtypes(include=['object']).columns:
    raw_data[column] = raw_data[column].astype('category').cat.codes

# Initialize and apply one-hot encoding
cat_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
cat_columns = raw_data.select_dtypes(include=['category', 'object']).columns
encoded_cats = cat_encoder.fit_transform(raw_data[cat_columns])
encoded_cats = pd.DataFrame(encoded_cats, columns=cat_encoder.get_feature_names_out
processed_data = pd.concat([raw_data.drop(cat_columns, axis=1), encoded_cats], axis

# Define and extract target variable
price_col = "SalePrice"
if price_col not in processed_data.columns:
    price_col = processed_data.columns[-1]

# Split features and target
house_features = processed_data.drop([price_col], axis=1)
house_prices = processed_data[price_col].values.reshape(-1, 1)

# Clean and scale the feature data
house_features = house_features.apply(pd.to_numeric, errors='coerce').fillna(0)
feature_scaler = StandardScaler()
scaled_features = feature_scaler.fit_transform(house_features)

# Split data into training and validation sets
train_x, val_x, train_y, val_y = train_test_split(scaled_features, house_prices, te

# Convert to PyTorch tensors
train_x_tensor = torch.tensor(train_x, dtype=torch.float32)
train_y_tensor = torch.tensor(train_y, dtype=torch.float32)
val_x_tensor = torch.tensor(val_x, dtype=torch.float32)
val_y_tensor = torch.tensor(val_y, dtype=torch.float32)

# Create data loaders for batch processing
train_data = TensorDataset(train_x_tensor, train_y_tensor)
val_data = TensorDataset(val_x_tensor, val_y_tensor)
train_loader = DataLoader(dataset=train_data, batch_size=32, shuffle=True)
val_loader = DataLoader(dataset=val_data, batch_size=32, shuffle=False)

# Define deep neural network for house price prediction
class DeepHousingNet(nn.Module):
    def __init__(self, input_size):
        super(DeepHousingNet, self).__init__()
        # Deeper architecture with gradually decreasing layer sizes
        self.input_layer = nn.Linear(input_size, 256)
        self.hidden1 = nn.Linear(256, 128)
        self.hidden2 = nn.Linear(128, 64)
        self.hidden3 = nn.Linear(64, 32)
        self.output_layer = nn.Linear(32, 1)
```

```
    def forward(self, x):
        # Forward pass with ReLU activation
        x = torch.relu(self.input_layer(x))
        x = torch.relu(self.hidden1(x))
        x = torch.relu(self.hidden2(x))
        x = torch.relu(self.hidden3(x))
        x = self.output_layer(x)
        return x

# Initialize model and training components
input_dim = train_x.shape[1]
model = DeepHousingNet(input_dim)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training settings
num_epochs = 75
train_losses = []
val_losses = []
start_time = time.time()

# Training loop
for epoch in range(num_epochs):
    # Training phase
    model.train()
    epoch_loss = 0.0
    for batch_x, batch_y in train_loader:
        # Move batch to device and compute predictions
        batch_x = batch_x.to(device)
        batch_y = batch_y.to(device)

        # Forward and backward passes
        optimizer.zero_grad()
        predictions = model(batch_x)
        loss = loss_fn(predictions, batch_y)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item() * batch_x.size(0)

    # Calculate average training loss
    avg_train_loss = epoch_loss / len(train_loader.dataset)
    train_losses.append(avg_train_loss)

    # Validation phase
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for batch_x, batch_y in val_loader:
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device)
            predictions = model(batch_x)
            loss = loss_fn(predictions, batch_y)
            val_loss += loss.item() * batch_x.size(0)
```

```
    # Calculate average validation loss
    avg_val_loss = val_loss / len(val_loader.dataset)
    val_losses.append(avg_val_loss)

    print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {avg_train_loss:.4f}, Va

# Calculate total training time
total_time = time.time() - start_time

# Plot training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Training and Validation Loss (Deep Architecture)')
plt.legend()
plt.show()

# Final model evaluation
model.eval()
with torch.no_grad():
    val_x_tensor = val_x_tensor.to(device)
    val_y_tensor = val_y_tensor.to(device)
    final_preds = model(val_x_tensor)
    final_mse = loss_fn(final_preds, val_y_tensor).item()
    final_rmse = np.sqrt(final_mse)

# Print final metrics
print(f"Final Validation MSE: {final_mse:.4f}")
print(f"Final Validation RMSE: {final_rmse:.4f}")
print(f"Model Complexity (Total Parameters): {sum(p.numel() for p in model.paramete
print(f"Total Training Time: {total_time:.2f} seconds")
```

```
Epoch [1/75], Training Loss: 1.0938, Validation Loss: 0.7758
Epoch [2/75], Training Loss: 0.6183, Validation Loss: 0.6098
Epoch [3/75], Training Loss: 0.5329, Validation Loss: 0.6025
Epoch [4/75], Training Loss: 0.5089, Validation Loss: 0.6058
Epoch [5/75], Training Loss: 0.5019, Validation Loss: 0.6006
Epoch [6/75], Training Loss: 0.4889, Validation Loss: 0.6071
Epoch [7/75], Training Loss: 0.4773, Validation Loss: 0.6076
Epoch [8/75], Training Loss: 0.4728, Validation Loss: 0.6130
Epoch [9/75], Training Loss: 0.4627, Validation Loss: 0.6117
Epoch [10/75], Training Loss: 0.4532, Validation Loss: 0.6175
Epoch [11/75], Training Loss: 0.4470, Validation Loss: 0.6207
Epoch [12/75], Training Loss: 0.4372, Validation Loss: 0.6296
Epoch [13/75], Training Loss: 0.4266, Validation Loss: 0.6218
Epoch [14/75], Training Loss: 0.4167, Validation Loss: 0.6332
Epoch [15/75], Training Loss: 0.4099, Validation Loss: 0.6536
Epoch [16/75], Training Loss: 0.4077, Validation Loss: 0.6524
Epoch [17/75], Training Loss: 0.3887, Validation Loss: 0.6328
Epoch [18/75], Training Loss: 0.3745, Validation Loss: 0.6493
Epoch [19/75], Training Loss: 0.3874, Validation Loss: 0.6465
Epoch [20/75], Training Loss: 0.3456, Validation Loss: 0.6704
Epoch [21/75], Training Loss: 0.3412, Validation Loss: 0.6597
Epoch [22/75], Training Loss: 0.3229, Validation Loss: 0.7334
Epoch [23/75], Training Loss: 0.3220, Validation Loss: 0.6611
Epoch [24/75], Training Loss: 0.3055, Validation Loss: 0.6766
Epoch [25/75], Training Loss: 0.2982, Validation Loss: 0.6828
Epoch [26/75], Training Loss: 0.2952, Validation Loss: 0.6953
Epoch [27/75], Training Loss: 0.2926, Validation Loss: 0.6973
Epoch [28/75], Training Loss: 0.2755, Validation Loss: 0.6793
Epoch [29/75], Training Loss: 0.2575, Validation Loss: 0.7658
Epoch [30/75], Training Loss: 0.2538, Validation Loss: 0.7050
Epoch [31/75], Training Loss: 0.2513, Validation Loss: 0.7958
Epoch [32/75], Training Loss: 0.2570, Validation Loss: 0.7537
Epoch [33/75], Training Loss: 0.2542, Validation Loss: 0.7311
Epoch [34/75], Training Loss: 0.2170, Validation Loss: 0.7492
Epoch [35/75], Training Loss: 0.2075, Validation Loss: 0.8195
Epoch [36/75], Training Loss: 0.2073, Validation Loss: 0.7468
Epoch [37/75], Training Loss: 0.2090, Validation Loss: 0.8005
Epoch [38/75], Training Loss: 0.2053, Validation Loss: 0.7849
Epoch [39/75], Training Loss: 0.1979, Validation Loss: 0.8796
Epoch [40/75], Training Loss: 0.1937, Validation Loss: 0.7886
Epoch [41/75], Training Loss: 0.1751, Validation Loss: 0.8622
Epoch [42/75], Training Loss: 0.1737, Validation Loss: 0.8558
Epoch [43/75], Training Loss: 0.1659, Validation Loss: 0.8564
Epoch [44/75], Training Loss: 0.1729, Validation Loss: 0.8621
Epoch [45/75], Training Loss: 0.1585, Validation Loss: 0.8400
Epoch [46/75], Training Loss: 0.1556, Validation Loss: 0.9050
Epoch [47/75], Training Loss: 0.1532, Validation Loss: 0.8617
Epoch [48/75], Training Loss: 0.1534, Validation Loss: 0.8842
Epoch [49/75], Training Loss: 0.1431, Validation Loss: 0.9130
Epoch [50/75], Training Loss: 0.1479, Validation Loss: 0.8586
Epoch [51/75], Training Loss: 0.1375, Validation Loss: 0.9368
Epoch [52/75], Training Loss: 0.1418, Validation Loss: 0.8119
Epoch [53/75], Training Loss: 0.1477, Validation Loss: 0.9870
Epoch [54/75], Training Loss: 0.1336, Validation Loss: 0.8820
Epoch [55/75], Training Loss: 0.1328, Validation Loss: 0.9596
Epoch [56/75], Training Loss: 0.1235, Validation Loss: 0.9020
```

```
Epoch [57/75], Training Loss: 0.1328, Validation Loss: 0.9672
Epoch [58/75], Training Loss: 0.1241, Validation Loss: 0.8995
Epoch [59/75], Training Loss: 0.1257, Validation Loss: 0.9693
Epoch [60/75], Training Loss: 0.1395, Validation Loss: 0.8876
Epoch [61/75], Training Loss: 0.1391, Validation Loss: 0.9883
Epoch [62/75], Training Loss: 0.1250, Validation Loss: 0.8994
Epoch [63/75], Training Loss: 0.1225, Validation Loss: 0.9699
Epoch [64/75], Training Loss: 0.1284, Validation Loss: 0.9146
Epoch [65/75], Training Loss: 0.1329, Validation Loss: 0.9917
Epoch [66/75], Training Loss: 0.1297, Validation Loss: 0.9496
Epoch [67/75], Training Loss: 0.1088, Validation Loss: 0.9648
Epoch [68/75], Training Loss: 0.1206, Validation Loss: 0.9829
Epoch [69/75], Training Loss: 0.1116, Validation Loss: 0.9329
Epoch [70/75], Training Loss: 0.1067, Validation Loss: 0.9935
Epoch [71/75], Training Loss: 0.1104, Validation Loss: 0.9730
Epoch [72/75], Training Loss: 0.1025, Validation Loss: 0.9476
Epoch [73/75], Training Loss: 0.1035, Validation Loss: 0.9999
Epoch [74/75], Training Loss: 0.1055, Validation Loss: 0.9312
Epoch [75/75], Training Loss: 0.1089, Validation Loss: 0.9950
```




```
Final Validation MSE: 0.9950
Final Validation RMSE: 0.9975
Model Complexity (Total Parameters): 46593
Total Training Time: 1.63 seconds
```