Chethan Kumar R E, M23AIR510
MTech in Augmented and Virtual Reality,
School of Artificial Intelligence and Data Science,
Indian Institute of Technology, Jodhpur.

Project Report: Object Classification using Point Clouds

Introduction:

The aim of this project is to develop a neural network model capable of classifying 3D objects represented as point clouds into different categories. The dataset used for this task is the ModelNet10 dataset, which contains objects from 10 categories. The classification task is essential for various applications in computer vision, robotics, and augmented reality.

Neural Network Architecture:

The neural network architecture used for this project is a convolutional neural network (CNN) designed to process point cloud data. The architecture consists of the following layers:

1. Convolutional Layers: Three convolutional layers with increasing filter sizes and ReLU activation functions. These layers extract features from the input point cloud.

2. Batch Normalization Layers: Batch normalization layers are added after each convolutional layer to improve the stability and speed of training.

3. Global Max Pooling Layer: A global max pooling layer aggregates the feature maps from the convolutional layers into a single feature vector.

4. Dense Layers: Two dense layers with ReLU activation functions are added to the network for further feature processing.

5. Output Layer: The output layer is a dense layer with softmax activation, which produces the final classification probabilities for each object category.

Loss Function:

The loss function used for training the neural network is the sparse categorical cross-entropy loss. This loss function is suitable for multi-class classification tasks where the target labels are integers.

Training Process:

The training process involves optimizing the parameters of the neural network using the Adam optimizer and minimizing the sparse categorical cross-entropy loss. The training is conducted over multiple epochs, with the training and validation data used to monitor the model's performance.

Model Evaluation:

After training, the model is evaluated on a separate test set to assess its performance. The evaluation includes computing the test loss and test accuracy metrics. The test loss indicates how well the model generalizes to unseen data, while the test accuracy measures the proportion of correctly classified objects in the test set.

Results:

The following results were obtained from training and evaluating the model:

- Training Loss vs. Validation Loss Plot: This plot shows the training and validation loss values over epochs. It helps visualize the training process and identify any overfitting or underfitting issues.

- Test Accuracy: The test accuracy indicates the overall performance of the model on unseen data. It represents the percentage of correctly classified objects in the test set.

- Training Accuracy: The training accuracy measures the model's performance on the training data. It indicates how well the model fits the training data.

Conclusion:

In conclusion, the developed neural network model demonstrates promising performance in classifying 3D objects from point cloud data. The architecture, loss function, training process, and evaluation metrics were carefully chosen to achieve satisfactory results. Further optimization and experimentation may be conducted to improve the model's accuracy and robustness for real-world applications.

Code:

Instructions to run code :

1. Create a notebook in Google Collaboratory
2. Copy the below code to the notebook
3. Copy ModelNet10 dataset to your Google drive and update **dataset_dir** variable in the below code appropriately. Example : '/content/drive/MyDrive/DLAssignment2/ModelNet10'
4. Run the notebook

c

```
import numpy as np
import os
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import trimesh

# Define the number of points per mesh
num_points = 1024

def read_off_file(file_path):
    # Load the mesh from the OFF file
    mesh = trimesh.load(file_path)
    print(file_path)
    # Sample or pad/truncate vertices to ensure consistent number of points
    if len(mesh.vertices) >= num_points:
        vertices = mesh.vertices[:num_points]
    else:
        pad_size = num_points - len(mesh.vertices)
        vertices = np.pad(mesh.vertices, ((0, pad_size), (0, 0)), mode='wrap')

    return vertices

from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Path to the directory containing the ModelNet10 dataset in your Google Drive
dataset_dir = '/content/drive/MyDrive/DLAssignment2/ModelNet10'

# List of class names
class_names = os.listdir(dataset_dir)

# Initialize lists to store data and labels
data = []
labels = []

# Load data from OFF files
for class_name in class_names:
    class_dir = os.path.join(dataset_dir, class_name)
    if os.path.isdir(class_dir):
        for split in ['train', 'test']:
            split_dir = os.path.join(class_dir, split)
            if os.path.isdir(split_dir):
                for file_name in os.listdir(split_dir):
                    if file_name.endswith('.off'):
                        file_path = os.path.join(split_dir, file_name)
                        vertices = read_off_file(file_path)
                        data.append(vertices)
                        labels.append(class_name)
```

Chethan Kumar R E, M23AIR510
MTech in Augmented and Virtual Reality,
School of Artificial Intelligence and Data Science,
Indian Institute of Technology, Jodhpur.

```
# Convert data and labels to numpy arrays
data = np.array(data)
labels = np.array(labels)

from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode labels to numerical values
labels_encoded = label_encoder.fit_transform(labels)

# Split the dataset into training, validation, and test sets
X_train, X_test, y_train, y_test = train_test_split(data, labels_encoded, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Define the neural network architecture
model = models.Sequential([
    layers.Conv1D(filters=64, kernel_size=1, activation='relu', input_shape=(None, 3)),
    layers.BatchNormalization(),
    layers.Conv1D(filters=128, kernel_size=1, activation='relu'),
    layers.BatchNormalization(),
    layers.Conv1D(filters=1024, kernel_size=1, activation='relu'),
    layers.BatchNormalization(),
    layers.GlobalMaxPooling1D(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(len(class_names), activation='softmax')  # Output layer
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

from tensorflow.keras.callbacks import ModelCheckpoint

# Define callbacks
checkpoint_callback = ModelCheckpoint(filepath='model_checkpoint.h5', save_best_only=True, monitor='val_loss',
mode='min')

# Train the model with callbacks
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val), batch_size=32, verbose=2,
                    callbacks=[checkpoint_callback])

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Chethan Kumar R E, M23AIR510
MTech in Augmented and Virtual Reality,
School of Artificial Intelligence and Data Science,
Indian Institute of Technology, Jodhpur.

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

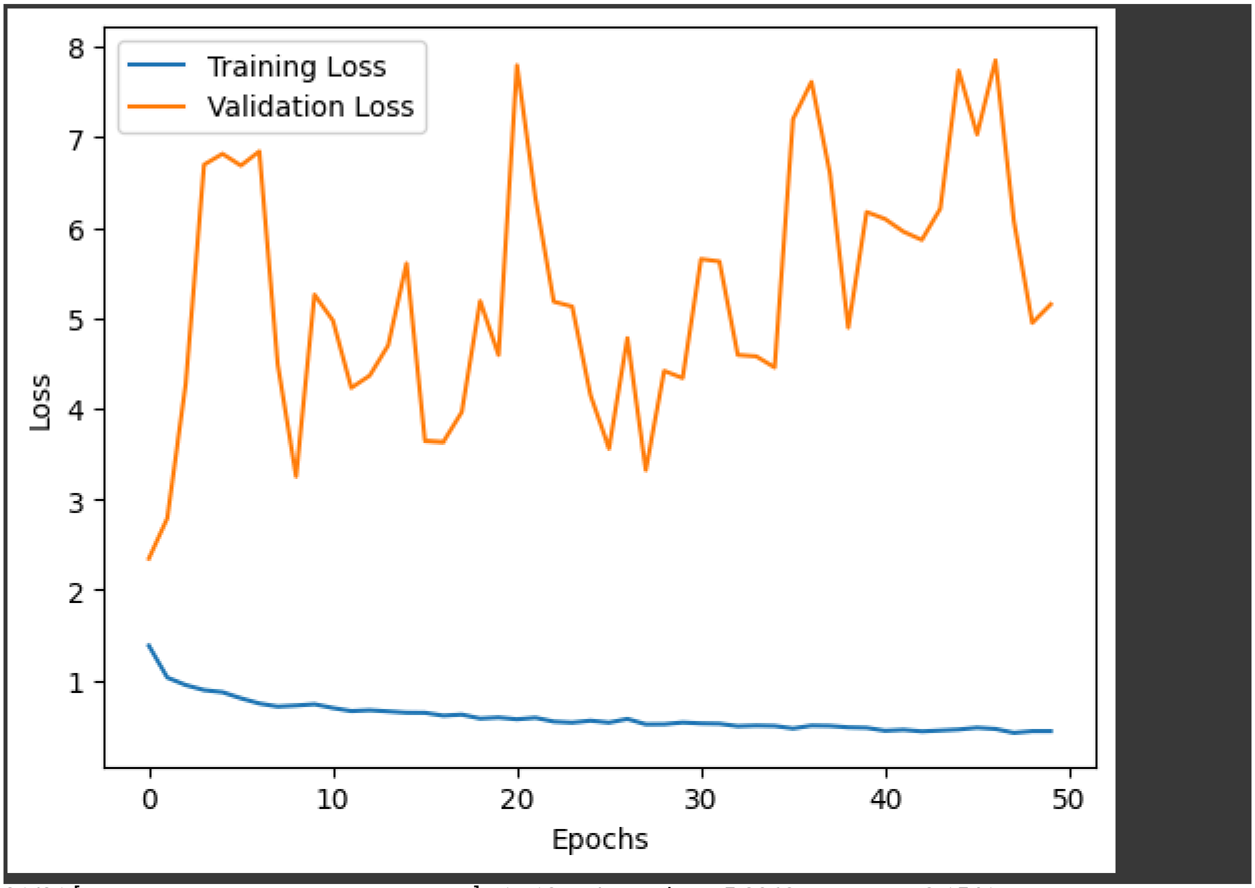
Console output:

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
98/98 - 19s - loss: 1.3836 - accuracy: 0.5691 - val_loss: 2.3469 - val_accuracy: 0.1403 - 19s/epoch - 194ms/step
Epoch 2/50
98/98 - 5s - loss: 1.0312 - accuracy: 0.6683 - val_loss: 2.7925 - val_accuracy: 0.2232 - 5s/epoch - 53ms/step
Epoch 3/50
98/98 - 5s - loss: 0.9484 - accuracy: 0.7008 - val_loss: 4.2699 - val_accuracy: 0.1378 - 5s/epoch - 48ms/step
Epoch 4/50
98/98 - 5s - loss: 0.8925 - accuracy: 0.7136 - val_loss: 6.6952 - val_accuracy: 0.1416 - 5s/epoch - 49ms/step
Epoch 5/50
98/98 - 5s - loss: 0.8698 - accuracy: 0.7113 - val_loss: 6.8150 - val_accuracy: 0.1403 - 5s/epoch - 48ms/step
Epoch 6/50
98/98 - 5s - loss: 0.8023 - accuracy: 0.7317 - val_loss: 6.6853 - val_accuracy: 0.1480 - 5s/epoch - 48ms/step
Epoch 7/50
98/98 - 5s - loss: 0.7448 - accuracy: 0.7566 - val_loss: 6.8417 - val_accuracy: 0.1416 - 5s/epoch - 49ms/step
Epoch 8/50
98/98 - 5s - loss: 0.7108 - accuracy: 0.7659 - val_loss: 4.4884 - val_accuracy: 0.2411 - 5s/epoch - 48ms/step
Epoch 9/50
98/98 - 5s - loss: 0.7222 - accuracy: 0.7595 - val_loss: 3.2537 - val_accuracy: 0.0957 - 5s/epoch - 52ms/step
Epoch 10/50
98/98 - 5s - loss: 0.7366 - accuracy: 0.7611 - val_loss: 5.2583 - val_accuracy: 0.1658 - 5s/epoch - 49ms/step
Epoch 11/50
98/98 - 5s - loss: 0.6948 - accuracy: 0.7675 - val_loss: 4.9723 - val_accuracy: 0.1658 - 5s/epoch - 49ms/step
Epoch 12/50
98/98 - 5s - loss: 0.6608 - accuracy: 0.7898 - val_loss: 4.2283 - val_accuracy: 0.1633 - 5s/epoch - 49ms/step
Epoch 13/50
98/98 - 5s - loss: 0.6704 - accuracy: 0.7777 - val_loss: 4.3637 - val_accuracy: 0.1671 - 5s/epoch - 49ms/step
Epoch 14/50
98/98 - 5s - loss: 0.6570 - accuracy: 0.7841 - val_loss: 4.6973 - val_accuracy: 0.0918 - 5s/epoch - 49ms/step
Epoch 15/50
98/98 - 5s - loss: 0.6432 - accuracy: 0.7869 - val_loss: 5.6030 - val_accuracy: 0.1658 - 5s/epoch - 50ms/step
Epoch 16/50
98/98 - 5s - loss: 0.6420 - accuracy: 0.7933 - val_loss: 3.6447 - val_accuracy: 0.1658 - 5s/epoch - 51ms/step
Epoch 17/50
98/98 - 5s - loss: 0.6099 - accuracy: 0.8003 - val_loss: 3.6311 - val_accuracy: 0.0982 - 5s/epoch - 49ms/step
Epoch 18/50
98/98 - 5s - loss: 0.6221 - accuracy: 0.7847 - val_loss: 3.9600 - val_accuracy: 0.1709 - 5s/epoch - 50ms/step
Epoch 19/50
```

```
98/98 - 5s - loss: 0.5784 - accuracy: 0.8077 - val_loss: 5.1884 - val_accuracy: 0.1645 - 5s/epoch - 49ms/step
Epoch 20/50
98/98 - 5s - loss: 0.5898 - accuracy: 0.7990 - val_loss: 4.5907 - val_accuracy: 0.1671 - 5s/epoch - 49ms/step
Epoch 21/50
98/98 - 5s - loss: 0.5700 - accuracy: 0.8029 - val_loss: 7.7934 - val_accuracy: 0.1594 - 5s/epoch - 49ms/step
Epoch 22/50
98/98 - 5s - loss: 0.5871 - accuracy: 0.8029 - val_loss: 6.3324 - val_accuracy: 0.1684 - 5s/epoch - 49ms/step
Epoch 23/50
98/98 - 5s - loss: 0.5445 - accuracy: 0.8134 - val_loss: 5.1810 - val_accuracy: 0.1671 - 5s/epoch - 53ms/step
Epoch 24/50
98/98 - 5s - loss: 0.5333 - accuracy: 0.8223 - val_loss: 5.1268 - val_accuracy: 0.1658 - 5s/epoch - 50ms/step
Epoch 25/50
98/98 - 5s - loss: 0.5551 - accuracy: 0.8134 - val_loss: 4.1450 - val_accuracy: 0.1684 - 5s/epoch - 49ms/step
Epoch 26/50
98/98 - 5s - loss: 0.5334 - accuracy: 0.8195 - val_loss: 3.5603 - val_accuracy: 0.1314 - 5s/epoch - 50ms/step
Epoch 27/50
98/98 - 5s - loss: 0.5740 - accuracy: 0.8121 - val_loss: 4.7782 - val_accuracy: 0.0510 - 5s/epoch - 49ms/step
Epoch 28/50
98/98 - 5s - loss: 0.5145 - accuracy: 0.8297 - val_loss: 3.3244 - val_accuracy: 0.1390 - 5s/epoch - 50ms/step
Epoch 29/50
98/98 - 5s - loss: 0.5157 - accuracy: 0.8274 - val_loss: 4.4156 - val_accuracy: 0.0816 - 5s/epoch - 50ms/step
Epoch 30/50
98/98 - 5s - loss: 0.5348 - accuracy: 0.8175 - val_loss: 4.3362 - val_accuracy: 0.1645 - 5s/epoch - 49ms/step
Epoch 31/50
98/98 - 5s - loss: 0.5243 - accuracy: 0.8258 - val_loss: 5.6516 - val_accuracy: 0.1645 - 5s/epoch - 53ms/step
Epoch 32/50
98/98 - 5s - loss: 0.5211 - accuracy: 0.8268 - val_loss: 5.6251 - val_accuracy: 0.1620 - 5s/epoch - 49ms/step
Epoch 33/50
98/98 - 5s - loss: 0.4936 - accuracy: 0.8303 - val_loss: 4.5917 - val_accuracy: 0.1671 - 5s/epoch - 49ms/step
Epoch 34/50
98/98 - 5s - loss: 0.5004 - accuracy: 0.8325 - val_loss: 4.5765 - val_accuracy: 0.1671 - 5s/epoch - 50ms/step
Epoch 35/50
98/98 - 5s - loss: 0.4965 - accuracy: 0.8265 - val_loss: 4.4542 - val_accuracy: 0.1378 - 5s/epoch - 49ms/step
Epoch 36/50
98/98 - 5s - loss: 0.4683 - accuracy: 0.8383 - val_loss: 7.1992 - val_accuracy: 0.1416 - 5s/epoch - 53ms/step
Epoch 37/50
98/98 - 5s - loss: 0.5004 - accuracy: 0.8249 - val_loss: 7.6111 - val_accuracy: 0.1658 - 5s/epoch - 50ms/step
Epoch 38/50
98/98 - 5s - loss: 0.4971 - accuracy: 0.8332 - val_loss: 6.6075 - val_accuracy: 0.1658 - 5s/epoch - 49ms/step
Epoch 39/50
98/98 - 5s - loss: 0.4807 - accuracy: 0.8376 - val_loss: 4.8939 - val_accuracy: 0.1735 - 5s/epoch - 50ms/step
Epoch 40/50
98/98 - 5s - loss: 0.4759 - accuracy: 0.8376 - val_loss: 6.1720 - val_accuracy: 0.1684 - 5s/epoch - 49ms/step
Epoch 41/50
98/98 - 5s - loss: 0.4432 - accuracy: 0.8463 - val_loss: 6.0947 - val_accuracy: 0.1645 - 5s/epoch - 49ms/step
Epoch 42/50
98/98 - 5s - loss: 0.4539 - accuracy: 0.8456 - val_loss: 5.9544 - val_accuracy: 0.1658 - 5s/epoch - 50ms/step
Epoch 43/50
98/98 - 5s - loss: 0.4359 - accuracy: 0.8533 - val_loss: 5.8634 - val_accuracy: 0.1684 - 5s/epoch - 49ms/step
Epoch 44/50
98/98 - 5s - loss: 0.4462 - accuracy: 0.8472 - val_loss: 6.2090 - val_accuracy: 0.1684 - 5s/epoch - 53ms/step
```

Chethan Kumar R E, M23AIR510
MTech in Augmented and Virtual Reality,
School of Artificial Intelligence and Data Science,
Indian Institute of Technology, Jodhpur.

Epoch 45/50
98/98 - 5s - loss: 0.4571 - accuracy: 0.8466 - val_loss: 7.7350 - val_accuracy: 0.1633 - 5s/epoch - 50ms/step
Epoch 46/50
98/98 - 5s - loss: 0.4771 - accuracy: 0.8348 - val_loss: 7.0294 - val_accuracy: 0.1645 - 5s/epoch - 49ms/step
Epoch 47/50
98/98 - 5s - loss: 0.4633 - accuracy: 0.8424 - val_loss: 7.8483 - val_accuracy: 0.1173 - 5s/epoch - 53ms/step
Epoch 48/50
98/98 - 5s - loss: 0.4201 - accuracy: 0.8549 - val_loss: 6.0720 - val_accuracy: 0.1671 - 5s/epoch - 49ms/step
Epoch 49/50
98/98 - 5s - loss: 0.4388 - accuracy: 0.8558 - val_loss: 4.9463 - val_accuracy: 0.1684 - 5s/epoch - 49ms/step
Epoch 50/50
98/98 - 5s - loss: 0.4387 - accuracy: 0.8523 - val_loss: 5.1518 - val_accuracy: 0.1684 - 5s/epoch - 50ms/step



31/31 [==============================] - 1s 18ms/step - loss: 5.2060 - accuracy: 0.1541
Test Loss: 5.205972194671631
Test Accuracy: 0.15408162772655487Epoch 1/50