Scene Recognition using Deep Neural Networks

Tarun Teja Obbina TXO220011 | Manohar Kairuppala MXK220104 | Charan Lokku CXL220029 | Tanusha Nandam TXN220047

***Abstract*—In this report , to evaluate the effectiveness of our proposed solution, we apply ResNet(Residual Neural Network) to the MIT indoor scene recognition dataset. This dataset performs better in representing indoor environments, which makes it a perfect model to build and train. Anticipating that ResNet's versatility will prove beneficial, we expect our approach to provide insights into overcoming the challenges associated with indoor scene recognition.**

Keywords—scene recognition, ResNet, indoor environment

I. INTRODUCTION

Identifying scenes is an essential cognitive function that people use on a regular basis to navigate through a variety of situations with ease. Innovations in artificial intelligence are influenced by this intrinsic sense of place, regardless of the environment. Scene recognition has undergone a revolution thanks to the recent explosion of deep learning techniques, especially Convolutional Neural Networks (CNNs), which have demonstrated extraordinary performance in categorization tasks.

Moreover, indoor scene recognition beccame a major difficulty as the area develops. CNNs perform poorly when faced with the complexities of inside situations, while they excel at identifying outdoor scenes. The core of the problem is the variety of indoor scenes: some, like hallways, may be adequately described by global spatial features, but others, like bookshops, require a more detailed approach.

This becomes more challenging when indoor scene identification calls for a model that can handle both local and global discriminative information with ease. Considering interior environments are inherently unpredictable, a more sophisticated knowledge is needed than what can be achieved through traditional scene recognition algorithms. With an emphasis on interior situations, this project undertakes a comprehensive examination of the cutting-edge models in scene recognition. We explore the difficulties presented by the distinctive qualities of interior environments and the necessity for models that can effectively combine both local and global data for accurate recognition. We investigate how to tackle the complexities of indoor scene recognition by employing ensemble methods that make use of contextual data at various scales and semantic insights from identified objects.

II. BACKGROUND WORK

Recent advances in scene recognition have largely been driven by the widespread adoption of deep learning methods. It is true that outdoor scene recognition has achieved notable success, but the transition to indoor environments presents unique challenges, making it an intricate problem in high-level vision. This study investigates the use of Residual Neural Networks (ResNet) to improve indoor scene recognition in response to these challenges. A ResNet is a powerful tool for navigating the complexity and variety inherent in indoor scenes due to its unique architecture featuring residual blocks that facilitate the learning of residual functions.

To address the complex nature of indoor scenes, ResNet offers the ability to capture both local and global features. Some environments require an emphasis on spatial structures, while others require an object-centric analytical approach. This adaptability is particularly advantageous in such situations. The proposed approach recognizes that successful indoor scene recognition requires models that can accommodate indoor environments with a wide range of characteristics. This study contributes valuable insight and solutions to the ongoing challenge of indoor scene recognition in the field of high-level vision by applying ResNet [2] to the MIT indoor scene recognition dataset – a benchmark for comprehensive representation of indoor settings. The background work sets the stage for addressing the intricacies of indoor scene recognition through nuanced and effective applications of deep learning.

III. THEORETICAL AND CONCEPTUAL STUDY

A. ResNet

Residual Network (ResNet) is a type of deep convolution neural network (CNN) architecture which serves as a deep learning model applied in computer vision tasks. It is specifically crafted to accommodate an large array of convolutional layers. The goal of ResNet's design was to overcome the difficulty of training extremely deep neural networks, in which increasing network depth frequently results in performance plateaus or even declines due to vanishing gradients.

The training of neural networks involves a backpropagation process dependent on gradient descent, wherein the objective is to minimize the loss function by adjusting weights. The use of residual blocks, which include skip connections or shortcuts that let the network skip one or more layers during training, is the main innovation in ResNet. By allowing the gradient to flow through the network more directly, these skip connections help to mitigate the vanishing gradient issue and make training very deep networks easier.

The identity path, which sends the input straight to the following layer, and the residual path, which modifies the input, are the two primary paths that make up a residual block. The total of the identity and residual paths is the block's final output.

ResNet presents a novel solution to the vanishing gradient problem known as "skip connections." Multiple identity mappings, initially made up of impact-free convolutional layers, are stacked together in ResNet. The activations from the layer before are then reused and these layers are skipped. By reducing the number of layers in the network, this skipping process speeds up the initial training stage.

After retraining the network, every layer grows, and the remaining segments called residual parts are given the chance to investigate a larger portion of the feature space present in the input image. The majority of ResNet models use batch normalization and nonlinearity in addition to skipping two or

three layers at a time. This architecture facilitates the more effective maintenance and dissemination of valuable information throughout the network.
There are several depths of ResNet architectures, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. The numbers represent the total number of network layers. Deep neural networks' performance on several computer vision tasks, such as segmentation, object detection, and image classification, has significantly improved thanks to the widespread adoption of ResNet. We chose to implement the ResNet-18 model for this project.

B. ResNet-18

ResNet-18 structure consists of a convolutional layer and eight residual building blocks (batch normalization and rectified linear unit (ReLU) activation functions). It has a residual building block as the basic structure. The structure of the residual building block is:

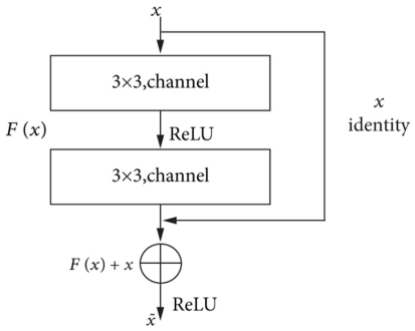

Fig.1. ResNet-18 Building Block [2]

It has 17 convolutional layers, a max-pooling layer, and a fully connected layer. A classical ResNet-18 model involves 33.16 million parameters, in which ReLU activation function and batch normalization (BN) are applied to the back of entire convolutional layers in “basic block.” The structure of ResNet-18 is:

Layer name	Output size	18-layer
Conv1	112 × 112	7 × 7, 64, stride 2
		3 × 3 max pool, stride 2
Conv2_x	56 × 56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
Conv3_x	28 × 28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
Conv4_x	14 × 14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
Conv5_x	7 × 7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$

Fig.2. CNN layer Size [2]

ResNet-18 Architecture:

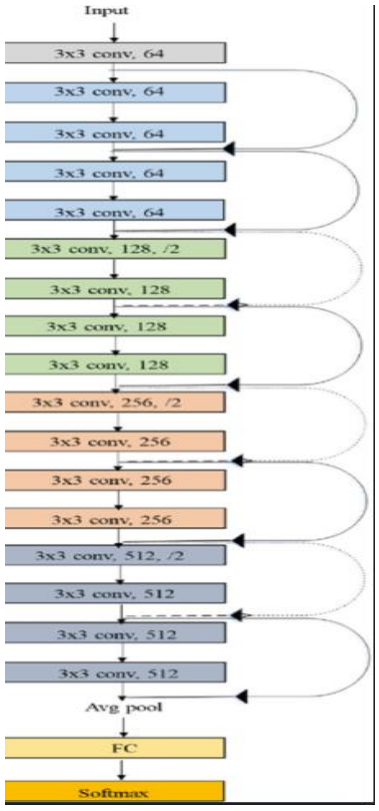

Fig.3. ResNet-18 Architecture [2]

A portion of the ImageNet database was used to train the pretrained model, ResNet-18. The model can classify images into 1000 object categories after being trained on over a million images.
Components of ResNet:

- An RGB Image with default dimensions 224x224 pixels is inputted to ResNet-18.
- Low Level features are extracted by a convolutional layer that processes the input image.
- Residual blocks, each comprise two convolutional layers. The key innovation lies in the integration of a "shortcut" or "skip connection" that circumvents these convolutional layers, directly adding the input to the block to its output. This innovative architecture facilitates smoother gradient flow during backpropagation, effectively mitigating the vanishing gradient problem in deep neural networks. Following each convolutional layer, batch normalization and rectified linear unit (ReLU) activation functions are typically applied. The skip connection plays a crucial role in preserving valuable information from earlier layers, enhancing the network's capacity to learn and retain essential features throughout the training process.
- The application of a global average pooling layer comes after the sequence of residual blocks. This layer generates a single value for each feature by averaging each feature map. By doing this, the

feature maps' spatial dimensions are reduced to a set size.

- The global average pooling layer's flattened output is fed into the final, fully connected layer, which generates the final classification scores. Usually, the number of neurons in this layer equals the number of classes in the classification task.

Deeper networks can be trained because residual connections facilitate the parameter gradients' easier propagation from the output layer to the network's earlier layers. On more challenging tasks, this deeper network can lead to higher accuracy.

IV. MODEL IMPLEMENTATION

A. Data Set

For the implementation we used MIT indoor scene recognition dataset [1] comprises 15,620 images across 67 indoor categories, each with at least 100 images. Recognizing indoor scenes is challenging due to the varied characteristics of indoor environments. Some scenes are well-defined by global spatial properties, while others are better characterized by the objects they contain.

Fig.4 . Sample data [1]

B. Implementation

To build a ResNet model from scratch using only Python, without relying on built-in functions, involves implementing the fundamental components of deep learning models. As per project guideline, manually coding the convolutional layer by performing element-wise operations for each filter and input patch, implementing batch normalization to normalize activations, and incorporating activation functions like ReLU to introduce non-linearity. The core of ResNet lies in the residual blocks, where identity blocks are particularly crucial. These blocks consist of multiple convolutional layers, batch normalization, and ReLU activation, with the input added to the output to create a residual connection. The ResNet architecture is constructed by stacking these blocks, with variations depending on the specific ResNet model (e.g., ResNet-18, ResNet-50). The architecture also includes pooling layers, such as max pooling, for down sampling feature maps. Further, the fully connected layer is used with global average pooling, reducing spatial dimensions of the images to 1x1. Training the model involves implementing a suitable loss function (categorical cross-entropy), backpropagation for gradient computation, and an optimization algorithm (stochastic gradient descent). Additional considerations include weight initialization, learning rate scheduling, and data loading procedures for efficient training.

C. Evaluation

Categorical Cross-Entropy (CCE) [4] is a commonly used loss function in classification tasks, especially for multi-class classification problems. It measures the dissimilarity between the predicted probability distribution and the true probability distribution of the classes. The formula for categorical cross-entropy is as follows:

$$CE = -\sum_{i=1}^{i=N} y\_true_i \cdot log(y\_pred_i)$$

$$CE = -\sum_{i=1}^{i=N} y_i \cdot log(\widehat{y_i})$$

$$CE = -[y_1 \cdot log(\widehat{y_1}) + y_2 \cdot log(\widehat{y_2}) + y_3 \cdot log(\widehat{y_3})]$$

The categorical cross-entropy loss encourages the predicted probability distribution to align closely with the true distribution, penalizing deviations from the correct class. It is a crucial component in the training of classification models, and minimizing this loss leads to improved model performance in assigning accurate class probabilities.

V. RESULTS AND ANALYSIS

In our experiment, we divided the dataset into training and testing sets using an 80-20 split, the training data contains 12,496 images. To handle the large dataset, we further divided the training set into batches of a specified size (71). Each batch, containing a subset of the training data, was processed sequentially during training. For a single epoch, the entire training dataset was iterated through, with each batch contributing to the model's parameter updates.

Over the course of 20 epochs, we repeatedly trained the entire dataset, so that the model will learn from the complete dataset multiple times. Throughout this training process, we recorded and monitored the training loss, testing loss, and accuracy precision, recall and f1 score. The training loss reflects the model's performance on the training set, indicating how well it fits the data. Simultaneously, the testing loss gauges the model's generalization to unseen data, ensuring that it doesn't overfit the training set. Accuracy, measured as the percentage of correctly classified instances, provides insights into the overall model performance.

We trained the model with different train to test splits and compared the metrics for each of them. We found that the model generalized well for 80 to 20 ratio. For other splits the model tends to overfit to training set while not performed well with the testing data. Also, the model took very high amount of time for training. We used google colab to train the model. So initially we use 40-60 split so that we will have less number of batches in each epoch and training will take less time. As the training data size increased, training time

increased significantly and model generalized well. Our model was able to perform well. It met our expectations as we have faced many issues while training due to the large size of the dataset and restrictions of google colab.

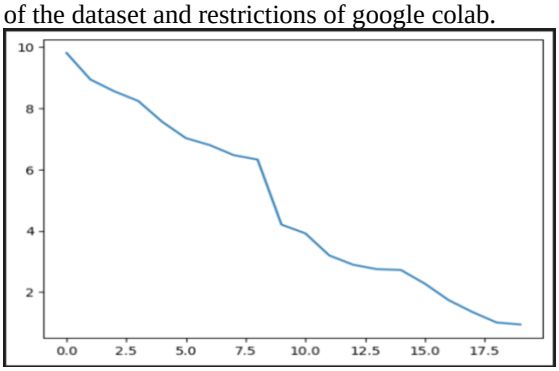

Fig.5 . Training Loss vs epochs

In the above figure shows the training loss from the data split for the 20 epochs it shows that 9.807 value for the first epoch and it reaches 0.939 for the 20th epoch

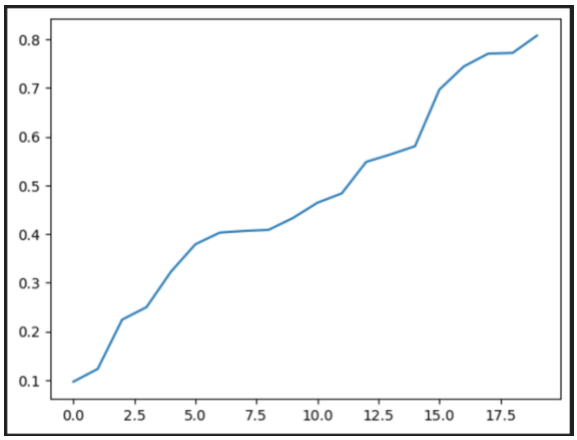

Fig.6 . Training Accuracy vs epochs

In the above figure shows the training accuracy from the data split for the 20 epochs it shows that 0.0966 value for the first epoch and it reaches 0.807 for the 20th epoch

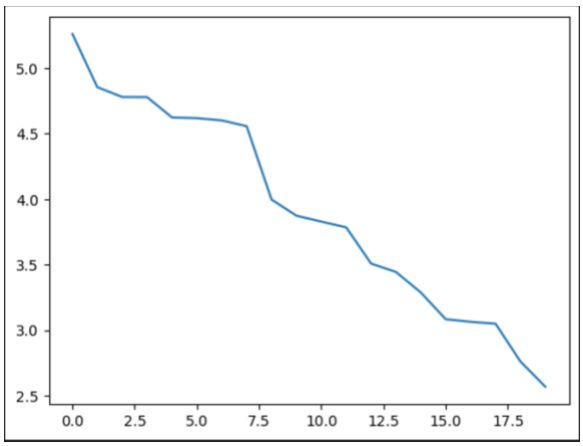

Fig.7 . Testing Loss vs epochs

In the above figure shows the testing loss loss from the data split for the 20 epochs it shows that 5.260 value for the first epoch and it reaches 2.568 for the 20th epoch

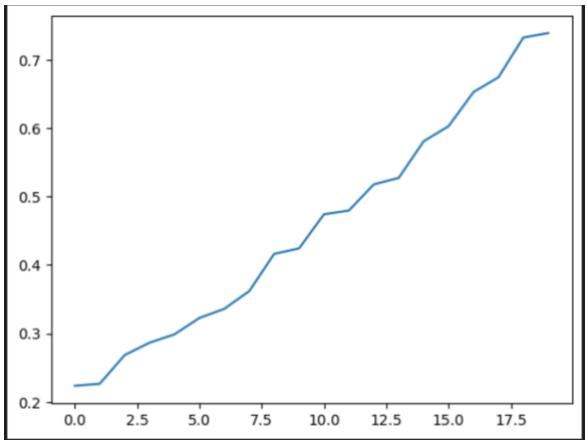

Fig.8 . Testing Accuracy vs epochs

In the above figure shows the testing accuracy 0.739 for given testing split.

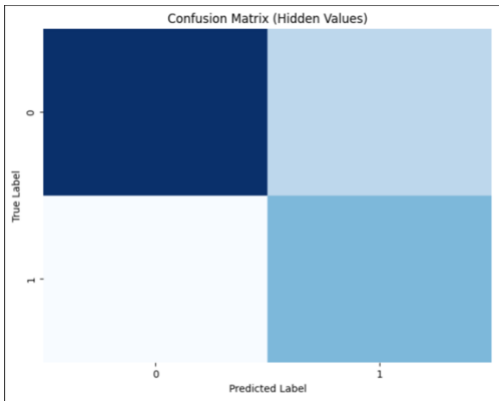

Fig.9 . Confusion Matrix

The above confusion matrix is produced from the given True Positive(TP) = 2011 , True Negative(TN) = 313, False Positive(FP) = 45, False Negative(FN) = 483 parameters based on the true label and predicted label.

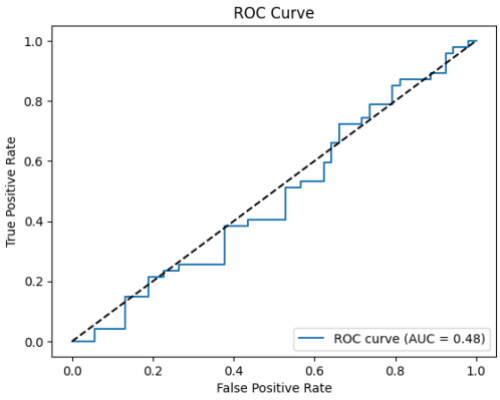

Fig.9 . ROC Curve

As displayed in the above ROC curve, our model was able to perform slightly better than a random guessing model

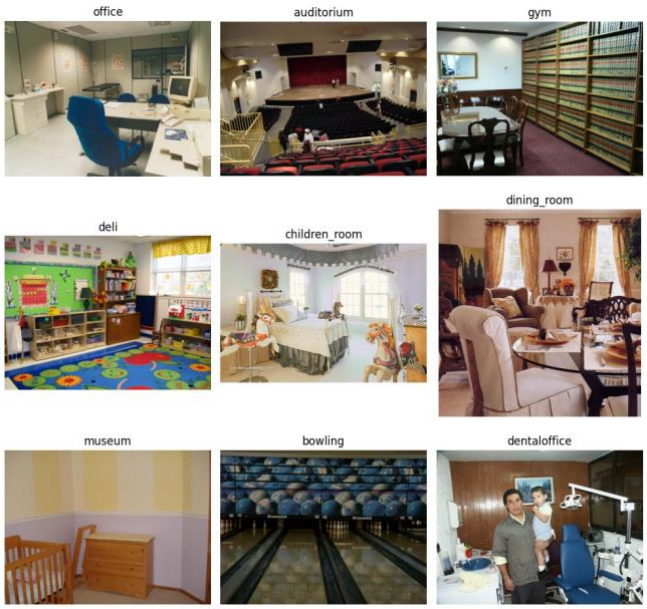

Fig.10. The above figure is a sample classification done by our trained model on test dataset. Here u can see that our model correctly classified 6 out of 9 images. It incorrectly classified library as gym, children_room as museum and kidergarden as deli.

VI. CONCLUSION AND FUTURE WORK

We are satisfied with the results we were able to achieve in the given amount of time and constraints. We learned how convolution layer works in detail and how batch normalization helps during training. While we are implementing the model, we are able learn more about how the back propagation works in convolution layer. If we could use libraries like pytorch to implement the model, we assume that we could have achieved a better performance, since these libraries are written to use the GPUs efficiently and quickly. Since we implemented all the layers of models from scratch using python, the model took significantly large amount of time to train and during training we faced multiple issues with implementation. So, in future we aim to compare our model with a model implemented using pytorch in different aspects like training accuracy, testing accuracy, training time for one epoch. Also we would like to compare how other models will work in scene recognition and how does the performance of will change if we increase the number of layers in the resnet model.

REFERENCES

[1] https://www.kaggle.com/datasets/itsahmad/indoor-scenes-cvpr-2019

[2] https://arxiv.org/abs/1512.03385

[3] https://arxiv.org/pdf/2007.01806.pdf

[4] https://neuralthreads.medium.com/categorical-cross-entropy-loss-the-most-important-loss-function-d3792151d05b

[5] https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

[6] https://wcvanvan.medium.com/backpropagation-for-batch-normalization-87ad825f3a50