Scene Recognition With Convolutional Neural Networks

Sowmya Sivaramakrishnan
Department of Computer Science
The University of Texas at Dallas
Dallas, USA
sxs230043@utdallas.edu

Sisira
Department of Computer Science
The University of Texas at Dallas
Dallas, USA
sisira.sisira@utdallas.edu

Scott Vu
Department of Computer Science
The University of Texas at Dallas
Dallas, USA
smv210000@utdallas.edu

***Abstract*—Scene recognition using Convolutional Neural networks is an efficient method for classifying scenes in images. CNN's ability to identify complex patterns and spatial correlations in images with their hierarchical feature learning makes it a powerful approach for scene recognition. In this project, we introduce an innovative method for scene recognition involving a custom-built CNN trained on a dataset containing images belonging to three distinct classes: buildings, forests, and mountains. Our CNN comprises convolutional, max-pooling, and two fully connected layers. Our method employs intricate convolutions for RGB image classification, and we utilize the stochastic gradient descent (SGD) optimization technique with mini batches.**

Keywords—CNN, Scene Recognition, Image Classification, Neural Networks

I. Introduction

The task of recognizing scenes from images is a crucial one with widespread applications in today's world. One of its key applications are in the field of healthcare imaging where scene recognition is used to recognize abnormalities such as tumors from complex medical images such as those produced from MRIs.

Surveillance systems and security applications also use scene recognition but in a more real-time video analysis manner instead of image analysis. In surveillance systems, they employ this method to analyze the feeds and detect unusual activities and potential threats to security. Apart from this even retailers use scene recognition on real-time footage to eventually generate analytics concerning the footfall and to improve their store layouts and thereby the customer experience. Scene recognition is also widely used in VR, AR, robotics, autonomous vehicles etc.

In the past few decades, convolutional neural networks have increasingly gained popularity as one of the most efficient architectures that can be used for image classification and other object detection related tasks. Its success mainly stems from its ability to automatically teach itself to successfully identify patterns and hierarchical features that hold relevance in the context of the specific use cases. The field of computer vision, especially, has undergone a major revolution due to the self-learning capabilities of CNNs, the impact of which is evident in tasks such as scene recognition.

While other methods such as support vector machines, bag-of-words models, spatial pyramid matching, principal component analysis etc. were really good methods used in the past for the task of scene recognition, none of them performed remotely as well as CNNs do. For most traditional methods, computing features, identifying the relevant ones and utilizing them were all separate tasks which eventually impacted the efficiencies.

CNN on the other hand excels in the task of identifying relevant and hierarchical features, which is extremely important when dealing with complex images in huge volumes. The recognition capabilities of these networks also have zero reliance on the spatial arrangement of objects which is critical when performing scene recognition since multiple images of the same class need not necessarily have comparable spatial location of objects. These and a few other unique abilities of convolutional neural nets are the key motivation behind our proposal of utilizing CNNs for scene recognition in this paper.

II. Background Work

The history of machine learning methods used for scene recognition is intertwined with the evolution of the field of computer vision over decades since scene recognition is essentially a sub-category in computer vision. The early approaches had their foundation in traditional machine learning methods.

In the late 20th century, scientists would manually design features. They extracted certain types of descriptors such as those based on texture, color and shape in order to describe scenes. These methodologies had minimal ability to deal with variations in perspective, shift in lighting etc.

Following this the bag-of-words (BoW) models became popular for scene identification in the early 2000s. BoW models, which were influenced by text retrieval techniques, represented pictures as visual word histograms. "Bag of Visual Words and Fusion Methods for Action Recognition: Comprehensive Study and Good Practice" by H. Jégou et al. was a comprehensive study which explored BoW method in comparison with other fusion methods for the task of feature identification [4].

In most of these models a visual vocabulary was created using local characteristics such as SIFT (Scale-Invariant Feature Transform), and pictures were then classified according to how frequently these visual words appeared. In one such paper, "Grammatical Inference and SIFT for Scene Recognition" the authors Lucas C. Ribas et al. propose the use of SIFT to change images to strings followed by the use of induction to recognize the scene. BoW models, while somewhat successful, were devoid of spatial information.

In order to overcome the spatial constraints of BoW models, spatial pyramid matching was developed. SPM applied BoW models to each of the progressively finer spatial grids that were created from the pictures. This permits the spatial connections between visual words to be captured. SPM showed a notable enhancement in its ability to recognize scenes, particularly those with varied spatial arrangements.

Another approach that became popular in the 2000s was the histogram of oriented gradients or HOG method. While this was mainly used for object detection, it did happen to find some applications in scene recognition too. This algorithm offered a potent representation for obtaining edge and shape information by describing the distribution of gradient orientations in an image. It worked effectively for most part at identifying scenes with clear patterns only.

As a local feature descriptor that was unaffected by changes in illumination, rotation, or scale, SIFT was first presented. SIFT was first created for object identification, but by offering strong and unique local properties, it also helped with scene recognition. Because of its invariance characteristics, it could handle changes in scene pictures. Although this approach can accommodate the differences between similar scenes, it is a computationally expensive method. This led to further discovery of newer and more efficient techniques.

For scene recognition, there was a change in the 2010s towards merging several elements and using machine learning methods. To provide more thorough representations, elements such as texture features, colour histograms, and local descriptors were fused together. For classification, Support Vector Machines (SVMs) and Random Forests were frequently employed.

One such paper which proposed the use of SVM for learning features was “Alex-Net model based Scene image classification technique”. In this paper Jing Sun et al. propose a classical Alex-Net which learns features from the dataset with the utilization of SVM. The features are first learnt with the help of this model and subsequently Lib-SVM was used to classify the scenes[3].

Principal Component Analysis was another method that was used over the years for scene recognition problems. In the paper “Scene Character Recognition Using PCANet” the authors attempt to learn features of characters in the scenes for scene character recognition using a PCA network that consists of cascading PCA, which is a modified form of the classical PCA, that is similar to deep neural networks but the 64% accuracy it achieved was lower than deep learning models.

A ground-breaking era dawned upon scene recognition when deep learning, particularly Convolutional Neural Networks (CNNs), emerged. CNNs showcased unprecedented capabilities in extracting hierarchical features from raw pixel data by itself. This completely eliminated the conventional methods of manually engineering the features which wasn’t particularly effective when the complexity of images increased multifold. CNNs have proven to be excellent algorithms for end-to-end learning to map all the way from input images to the output class labels. These key features of CNNs make them adaptable for different complexities of scenes.

Another advantage of using these networks is the reduced parameters to be learnt. The parameters are shared in the convolutional layers thereby reducing the number of those that need to be learnt. This contributes significantly to the increase in efficiency when the problem scales up to higher levels of complexities.

The architecture of convolutional networks imitates the functioning of neurons in our brain while processing visuals. The pooling layers and local receptive fields help in focusing on local patterns and the hierarchies on a spatial level which in turn allows it to efficiently recognize the scenes and the objects in them.

Transfer learning is one more important feature of CNNs which allows it to utilize the knowledge it learnt in one domain to another domain, making it extremely easy and probably more effective to use pre-trained models for different problem statements from varying domains.

As a result of this shift towards CNNs and the utilization of large-scale datasets, scene recognition algorithms have become highly applicable to numerous real-world applications due to their remarkable advancements in accuracy and robustness.

III. Theoretical And Conceptual Study

One among the many key characteristics of Convolutional Neural Networks is the connectivity in its many layers. Not all layers of these networks are fully connected to the adjacent layers as one might expect [5]. Considering that these are mostly used while working with image and other spatial data, this design choice improves the efficiency of the processing. The learning phase of these networks involves a series of forward and backward passes that enable the learning.

A. Forward Pass

The start of the forward pass phase is marked by the feeding of images, in the form of pixel values that represent them, to the input layer. This data of pixels then undergoes something called convolution.

Convolution, an operation carried out in the convolutional layer of the CNN, is one of the central concepts in the functioning of a CNN. In this process a collection of learnable filters of small sizes called kernels are utilized, which perform feature detection by identifying various features and observable patterns appearing in the incoming data. A single value produced by the dot product between the kernel and the corresponding values in the input data is computed by sliding the kernels over the said data. The output feature map is constituted of these dot product values. These kernels are useful in capturing the finer details in each local area they are on. The stride value, i.e. the number of steps by which the kernel slides each time can be specified by us and so can the weight values of the kernel matrix [1]. The formula for convolution is given as follows:

$$\mathrm{O}(i, j) = \sum_{\mathrm{m}}\sum_{\mathrm{n}} \mathrm{I}(i - m, j - n) \cdot \mathrm{K}(m, n) \qquad (1)$$

After the convolution operation is completed, the output feature map that we obtained is subjected to an activation function. This step allows the model to learn involuted relationships in the data. ReLU expanded as Rectified Linear

Unit is one of the most preferred activation functions that's used at this step. It improves the expressiveness and introduces non-linearity in the model while addressing the vanishing gradient problem too. The CNN's capacity to identify patterns and traits within data might be restricted if it lacked non-linearity. Sparsity in the feature maps is a result of ReLU's trait of putting negative values to zero. Due to this sparsity, only neurons that are active (have positive values) contribute to the layers that come after; neurons that are inactive (have negative values) have no effect on the network's calculations. Because fewer computations are needed, sparse activation facilitates faster and more effective training and inference. The formula for this activation is as follows:

$$f(x) = \max(0,x) \tag{2}$$

After activation, the activated feature maps make their way through the pooling layer, a crucial component in addressing the challenges posed by the convolutional layer's large number of features. Specifically, the pooling layer is designed to address two primary challenges: the elevated computational complexity and the potential for overfitting. To overcome these obstacles, the pooling layer utilizes down-pooling, which effectively reduces the feature maps' dimensionality while retaining vital information. Instead of dealing with each pixel individually, the pooling layer consolidates statistics from the feature maps, providing a condensed representation of each feature's local region [5]. This method is typically accomplished using either max-pooling or mean-pooling, both of which entail partitioning the output feature maps from the preceding layer. [5].

Pooling produces feature maps of multiple channels, which are flattened into a 1D vector in the Flattening operation. The spatial representation is transformed into a linear format. For the fully connected layers to handle the features, this conversion is vital. The Dense Layer, commonly referred to as the fully connected layer, forms extensive links connecting each neuron with every neuron in the preceding layer. It contains the weight and bias matrices represented in vector form, as seen in the linear transform formula:

$$Z = X \cdot W + B \tag{3}$$

where Z represents the linear transform vector from the dense layer, X is the one-dimensional vector coming from the flatten layer, W is a matrix containing the weights, and B is the bias vector.

In our proposed system we have added two dense or fully connected layers in the end for improving model efficiency when the complexity of input increases. With each additional dense layer, the ability of the system to identify more elaborate and detailed features. In classification tasks, the conclusive output layer incorporates a dense layer, where the number of neurons corresponds to the classes defined in the specific problem. The proposed system deals with a three-class problem; hence we have an output layer comprising of three neurons.

In the end another activation function is employed in the output layer [2]. Among the many activation functions, in this layer we chose to use Softmax activation function which is best suited when the problem involves multiple classes. By using this activation function, the class specific logits or raw scores can be transformed into probabilities which are indicative of the confidence of the model with respect to each class.

This activation function takes the input in vector format and outputs a vector where every element lies in the range of 0 to 1 while the summation of the elements of that vector equals one. The mathematical formula for the activation function is given in the following figure:

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Fig. 1. Softmax Activation Function Formula [5]

Where $S(x_i)$ is the probability for the class I and x_i is the raw score that's given as the input to the activation from the fully connected layer. With the application of this activation function to the output from the output layer, the forward pass comes to an end.

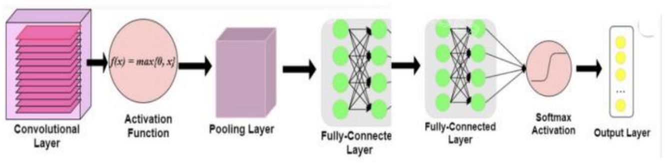

Fig. 2. Forward Pass in CNN

B. *Error Computation*

One of the most suited error computation method for image classification problems is the Cross Entropy Error. This method is also called the log loss function and is a focal error computation approach especially for convolutional neural networks. It aligns perfectly with the aim of CNNs to minimise the deviation between the predicted and actual probabilities of the classification. Cross entropy evaluates the variation between the distribution of output probabilities we get from Softmax and the distribution of the actual values. Since we chose Softmax as the activation function, this error computation method aligns the best with the probabilities that we receive as an output from the activation.

This loss function basically assigns a penalty which is levied every time the model expresses confidence in a wrong prediction. It assesses the confidence levels that the model displays through a prediction and not just whether or not the prediction is right. This property makes it intrinsically sensitive to the probabilities.

In case of a problem with a disproportionate class distribution, the penalty that cross entropy loss function imposes on errors made by the model is higher for the classes which have fewer cases than those with a majority. From a mathematical purview this error function involves taking negative log of the predicted probability. This logarithmic approach ensures a much more refined and insightful error computation. The figure below shows the formula for Cross Entropy Error calculation:

$$\mathrm{E} = -\sum_{i=0}^{9} y_i \log(p_i)$$

Fig. 3. Cross Entropy Error Formula [5]

Within the complex world of CNNs, where image classification tasks require accuracy and consistency, Cross wrong predictions, sensitivity to probabilities, and smooth transition to multiclass settings.

C. Backward Propagation

Backpropagation also called Backward pass constitutes of an iterative optimisation algorithm that minimises the error which we measured in the previous section using cross entropy loss function. This minimisation, in the context of CNNs utilised for image recognition tasks like the one at hand, allows for the network to learn and adapt to complex patterns in the data. In this phase we sequentially propagate the errors in the reverse direction starting from output layer through all the previous layers of the network to adjust the weights to reduce the error computed during cross entropy evaluation and enhance the overall efficiency of predictions done by the network[2].

To begin the backward pass, we first differentiate the loss with respect to the network's output. The outcome of this differentiation specifies the required modification in the output to reduce the loss. Since we used cross entropy to calculate the error in the last step of the forward pass, the cross entropy loss function is what we differentiate in this step to begin the backward pass phase and the differentiation is done with respect to the output we receive in the output layer after applying Softmax activation.

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial O} \times \frac{\partial O}{\partial Z} \tag{4}$$

The computation of gradients concerning weights of a particular layer involves multiplying the input activations (from the previous layer) with the partial derivative of the loss with respect to the current layer's outputs. Similarly, the bias gradients of a layer are computed as the partial derivative of the loss function with respect to that layer's bias. The fully connected layer of the model has a simpler backward pass for computing gradients than the other layers. When we backpropagate through the last fully connected layer, individual neuron's contribution to the overall error is evaluated by using the chain rule that we have learned in calculus which finds the gradients of the weights and the biases. Since the proposed system consists of two fully connected layers, the process gets repeated when the output from the last dense layer is passed back to the one before that.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \times \frac{\partial Z}{\partial W} \tag{5}$$

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Z} \times \frac{\partial Z}{\partial B} \tag{6}$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z} \times \frac{\partial Z}{\partial X} \tag{7}$$

There is a requirement to restore the gradients to their original two dimensional format before sending them to the max pooling layer, which is done and the feature maps are Entropy Error is a reliable option for error computation. It's a perfect fit because of its subtle way of penalising confidently passed to the pooling layer. In this layer during the forward pass, there were specific locations where the max values were selected. In the backward pass, we are going to send back the gradients only for those specific locations and let the rest have 0 gradients. This is achieved using the locality property and it makes sure that the pooling operation doesn't get reversed at the time of backpropagation.

Since we're working backwards, the next layer to work with is the ReLU activation. When the output is greater than zero for a ReLU neuron, the gradient is sent to the previous layer without modifications and when it's a negative value the gradient that is sent is zero. Using ReLU ensures that convergence happens faster during training owing to the simplicity of its derivative.

$$\text{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Fig. 4. ReLU Gradient Formula

The next layer is the convolutional layer where we are required to compute gradients of the kernels by performing the convolution operation between differentiation of the loss w.r.t the outputs and the inputs from our previous layer. As mentioned earlier backward propagation is iterative and eventually it keeps updating the weights to give us much more efficient models.

IV. Proposed System

In this project, we aim to provide an efficient implementation of scene recognition from image data using a convolutional neural network of high accuracy. The model constructed as part of this research performs much better than the conventional methods used for this use case. CNNs are known to perform better when complex images are involved, with greater ability to self-learn intricate hierarchical features in the images, which goes undetected by traditional models. The following sections give a detailed description of the dataset, the output expected from our proposed model and the architecture and algorithm involved in creating this model.

A. Dataset

The dataset used for this model is a widely used and publicly available dataset found in Kaggle; follow this link https://www.kaggle.com/datasets/nitishabharathi/scene-classification/data. It is a dataset specifically designed to be used for building scene classification models. It contains about ~5k images of natural scenes from around the world, categorized into 6 classes [7]. From the dataset, we picked a subset of classes: buildings, forests, and mountains, which are the three classes our model aims to train on and classify accurately. Instead of setting a standard split for the training and testing dataset, we opted for a constant fit of 85:15, to ensure uniform class composition and avoid overfitting. Each image is resized into a (3,28,28) tensor with 3 color channels. To ensure generality, we shuffle the dataset before proceeding to model building. Also, the class labels for the images are one-hot encoded to compare against the SoftMax produced class probabilities.

B. *Output*

The proposed Convolutional Neural Network (CNN) model aims to accurately categorize input images depicting specific scenes, distinguishing among buildings, forests, or mountains. In the training phase we evaluate the model accuracy and then run the test data through it to obtain the output.

C. *Architecture*

The architecture of the system proposed herein begins with the training data, which is essentially pixel values for the images being fed to the input layer. This input is then passed onto the convolution layer which, as the name suggests, performs a convolution operation using kernels of fixed size with weights that we set. This kernel then slides continually till we obtain an output feature map which consists of the dot product values between the kernel and the pixel values at the locations when the kernel slid. This output is then passed through the ReLU activation function which returns the input value as it is when the value is positive and zero for negative inputs coming from the convolution layer.

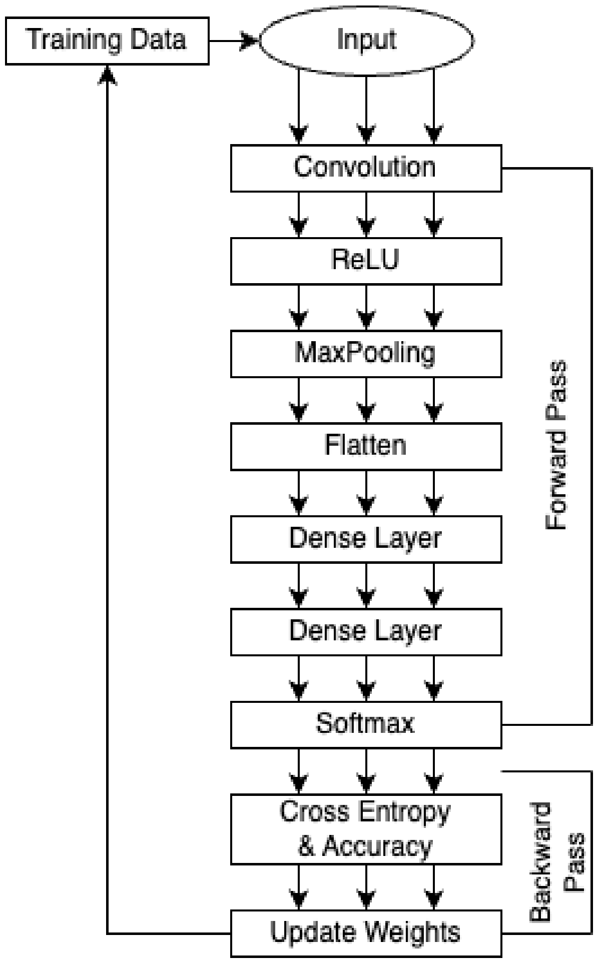

Fig. 5. Proposed System Architecture

The next layer in this architecture is the max pooling layer which works to reduce the dimensions of the feature maps. Aside from reducing the complexity by decreasing the dimensions, it also enables the model to represent spatial information and detect features at different levels of granularity. After the pooling layer the feature maps undergo flattening which is followed by two fully connected layers. The output from these layers is subjected to Softmax activation which gives us probabilistic outputs for the classifications.

Once we complete a forward pass, the backward pass occurs which evaluates the cross entropy error and in a bid to minimize this error it traverses backwards while continuously updating the weights. This process is done iteratively to improve the efficiency of the model when it comes to classifying the scenes from the test data.

D. *Algorithm*

Our algorithm features a complex Convolutional Neural Network (CNN) that is designed to handle RGB images. It utilizes Convolution, MaxPooling, and Dense or Fully Connected layers to perform precise and accurate classification.

The core of our network contains the Convolution Layer, which performs feature extraction. Using convolutional operations, input images are scanned using custom sized filters to detect patterns, edges, and textures. To enable the effective capture of complex data relationships, we implemented the Rectified Linear Unit (ReLU) activation function to introduce non-linearity into the network. Furthermore, resulting from the implementation of gradient loss through these filters, the layer can further fine-tune the complex features extracted to further boost classification accuracy.

To reduce data dimensionality, we incorporated a MaxPooling Layer. This layer preserves important features by condensing the representation of data while retaining essential information in each pooling window. It also reduces computational complexity alongside the retention of important image features, allowing more optimal data representation for classification.

The Dense or Fully Connected layers execute a step-by-step sequence of computations. Initially, the input image is flattened to a one-dimensional array, proceeding with transformations in the first dense layer. This layer performs a linear transformation denoted as ($W_1 \cdot X+B_1$). The data is then transformed again, but with the ReLU activation. Following this, the final dense layer performs its operation ($W_2 \cdot X+B_2$), resulting in the generation of class prediction probabilities through the application of softmax activation. These sequential steps are integral to the image classification process in our CNN.

The Forward Propagation involves the input data passing through various layers within the network architecture. The process begins with the convolutional layer, where the input undergoes cross-correlation with specific filters followed by the application of the Rectified Linear Unit (ReLU) activation. This operation fosters non-linearity, aiding in the extraction of essential features while preventing the propagation of extraneous information. Subsequently, the data flows through MaxPooling layers that condense the representations, preserving crucial information. Finally, the data transits through Dense or Fully Connected layers, culminating in the application of the SoftMax function, which generates probabilities across different classes, thereby facilitating the final classification output.

The Backpropagation is a pivotal step in our training methodology, functioning by computing prediction errors and then calculating gradients using the chain rule across the network layers. This process updates weights and biases iteratively, ensuring the refinement of the network for improved predictive accuracy and reduced error margins.

To further optimize, we incorporated Stochastic Gradient Descent (SGD) with mini-batches. Unlike traditional SGD or Batch Gradient Descent, we select random mini-batches of select sizes from the training data in each iteration. It provides a balance between the shortcomings of pure SGD, which can cause model instability due to fluctuations and susceptibility to noise, and the risks of overfitting inherent in iterating through the entire dataset. This improves our model's stability and generalization capabilities.

V. Results And Analysis

To implement the Convolutional Neural Network (CNN), we employed Python as the primary programming language and developed the model without relying on pre-existing frameworks like TensorFlow. The experiments were conducted in a CPU-based environment, leveraging Jupyter Notebooks for code development and execution.

To explore the impact of various hyperparameters on model performance, we conducted an extensive exploration akin to a Grid Search. Employing multiple nested loops, we systematically tested numerous combinations of hyperparameters to gauge their effects on the model's accuracy and loss. This exhaustive analysis enabled us to identify the most promising configurations. From the comprehensive results obtained, we found the following noteworthy setups that showcase the influence of different parameters on the model's performance. It's important to note that our train-test split was maintained at 85% - 15%, ensuring consistent separation of data into distinct training and testing sets to ensure uniform class composition. Among these setups, we observed significant differences in accuracy and loss metrics, highlighting the sensitivity of the model to various hyperparameters.

TABLE I. Log Of Best Trials

Exp. No.	Parameters Chosen	Results
26	FC Layer Neurons: 95 Learning Rate: 0.00025 Epochs: 200 Batch Size: 750 Filter Size: (3 x 6 x 6)	Train Accuracy: 79.83% Train Loss: 0.171 Test Accuracy: 75.33% Test Loss: 0.236
36	FC Layer Neurons: 120 Learning Rate: 0.0003 Epochs: 225 Batch Size: 750 Filter Size: (3 x 6 x 6)	Train Accuracy: 81.23% Train Loss: 0.153 Test Accuracy: 75% Test Loss: 0.245

The exploration of configurations 26 and 36 offered valuable insights into the impact of varied parameters on the CNN's performance.

Comparing Configuration 26 and Configuration 36, batch size's influence on model performance is evident. Configuration 26, utilizing a batch size of 750, achieved a higher test accuracy of 75.3% compared to Configuration 36's accuracy of 75.0% with the same batch size. This trend implies that a larger batch size might contribute marginally to better generalization and convergence during training, leading to improved performance.

Both configurations shared similar settings for the number of fully connected layer neurons and filter size, showcasing their consistency. However, despite this similarity, Configuration 26 demonstrated better test accuracy and lower test loss, suggesting a subtle sensitivity of the model to these parameters and their combined effect on model performance.

Notably, both configurations employed different learning rates (0.00025 for Configuration 26 and 0.0003 for Configuration 36) and epochs (200 for Configuration 26 and 225 for Configuration 36). Configuration 26, with fewer epochs and a slightly lower learning rate, presented improved accuracy and lower loss, indicating a potential trend where a more conservative approach to these parameters could yield better performance. The fluctuations observed in the graphs could be attributed to the use of SGD. Multiple iterations during hyper-parameter tuning revealed that a lower learning rate is better suited for this model when using SGD, emphasizing the impact of optimization techniques on stability and convergence.

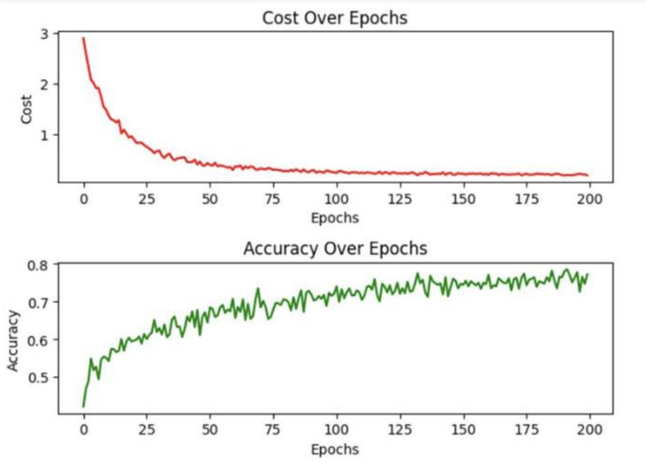

Fig. 6. Cost and Accuracy Plots Against Epochs

The graphical representation in Fig 6. illustrates the cost and accuracy trends over epochs and demonstrates the performance trajectory of our best-performing experiment. These graphs depict the evolution of the model's learning process, showcasing the decline in cost and the concurrent ascent in accuracy over epochs. These visual insights offer a comprehensive overview of the model's performance, encapsulating the progression and improvements observed in the best configuration. The graphical representation corresponds specifically to the hyperparameters and settings that yielded the most optimal results, providing a clear visualization of the learning dynamics throughout the training process.

The graphical representations of cost over epochs reveal notable trends, with a visible decline in cost from an initial value of 3, reaching 0.17 by the 200th epoch. Concurrently, the accuracy over epochs graph depicts a steady upward trajectory, starting at 0.4 and resulting at 0.75 by the 200th epoch. These trends reflect the model's learning process, indicating an improvement in performance as training progresses. It's worth noting the fluctuations in both cost and accuracy, which could be attributed to the use of Stochastic Gradient Descent (SGD) as the optimization technique.

These trends align with the observed impact of varied hyperparameters, emphasizing the significance of batch size, learning rate, and epochs. The configurations highlight the subtle nuances associated with these parameters, showing how adjustments impact convergence and model stability. Configuration 26, with fewer epochs and a slightly lower learning rate, exhibited better performance in terms of accuracy and loss.

VI. Conclusion

These findings underline the importance of meticulous parameter exploration and optimization. Further investigations might involve systematic adjustments to individual parameters while keeping others constant to unveil their independent impact on the model's performance. This analytical approach will enable a more comprehensive understanding of each parameter's role in refining the CNN, aiming for heightened accuracy and minimized loss while considering optimization techniques for stability and performance enhancement.

The trends observed in the graphs corroborate the findings from the configurations, indicating a clear convergence and improvement in model performance over epochs. These graphical insights serve as visual validations of the model's learning process, demonstrating the gradual decrease in cost and simultaneous increase in accuracy, affirming the significance of a well-optimized set of hyperparameters in the model's efficacy. This holistic approach to analysis, incorporating both numerical data and visual representations, provides a comprehensive understanding of the CNN's behavior, aiding in the refinement of parameters for enhanced performance and stability.

VII. Future Work

While the CNN proposed in this paper is extremely robust and has a high accuracy, there is still potential for future enhancements that could improvise on the current proposal. One such improvisation could be the enhancement of the network to accommodate a wider range of classes of scenes instead of three, while ensuring that the accuracy is maintained.

Since we're working with CNNs, transfer-learning is one of their key properties which could be utilized. In the future, we could look at using a pre-trained CNN, which performs efficiently in another domain, for a scene recognition problem.

Alternatively, we could use ensemble methods and perform scene recognition using the outputs of multiple CNNs which might further improve the efficiency of this model.

Acknowledgment

We express our sincere appreciation to Prof. Anurag Nagar of the Department of Computer Science at The University of Texas at Dallas for his comprehensive and interactive teaching methods. His guidance and support throughout the project were indispensable, contributing significantly to our learning and the creation of this CNN model for scene recognition.

We are also thankful to the Department of Computer Science at The University of Texas at Dallas for creating an atmosphere that supports our intellectual and personal growth. This has enabled us to broaden our knowledge and succeed in both our professional and personal pursuits.

References

[1] Irhum Shafkat, "Intuitively Understanding Convolutions for Deep Learning" Medium, https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[2] C. C. AGGARWAL, Neural Networks and Deep Learning: A Textbook. S.l.: SPRINGER INTERNATIONAL PU, 2023.

[3] Sarfaraz Masood, et al. "Scene Recognition from Image Using Convolutional Neural Network" Procedia Computer Science Volume 167, 2020, Pages 1005-1012.

[4] Xiaojiang Peng, et al. "Bag of Visual Words and Fusion Methods for Action Recognition: Comprehensive Study and Good Practice" arXiv:1405.4506

[5] Sisira, Devarshaa Sikharam "MapReduce-based CNN for Handwritten Digit Recognition"

[6] N. Basit et al., "MapReduce-based deep learning with handwritten digit recognition case study," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2016, pp. 1690-1699, doi: 10.1109/BigData.2016.7840783.

[7] Steve Rogers, "Image Classification using Keras: Intel Scene Classification Challenge" Medium, https://medium.com/beovolytics/image-classification-using-keras-intel-scene-classification-challenge-b0b087374654.