Nature Scenery Image Classification with Deep Neural Networks

Pankhuri Bhatnagar
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
pxb220012@utdallas.edu

Mohit Gullani
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
mxg210090@utdallas.edu

Abhilash Veluvali
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
hxv220011@utdallas.edu

Garima Rahangdale
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
gxr220005@utdallas.edu

Abstract

Image classification is a crucial task in computer vision, where machines learn to identify patterns within images. This research aims to develop a deep learning model for classifying outdoor scenes into two predefined categories. The proposed implementation utilizes feedforward neural networks, backpropagation, and gradient descent algorithms to efficiently process nature images. Extensive experimentation with hyperparameter tuning iterations resulted in an algorithm achieving an accuracy of approximately 70% in classifying various outdoor scenes. The findings demonstrate the effectiveness of the approach in image recognition tasks and contribute to the advancement of deep learning techniques for computer vision applications.

Keywords: Convolutional Neural Network (CNN), Image Classification, Spark, Deep Learning, Sigmoid, Computer Vision.

INTRODUCTION

Recent years have seen notable breakthroughs in the field of computer vision, particularly due to developments in deep learning methods. Deep Neural Networks (DNNs) have revolutionized a variety of applications including object recognition, face detection, medical imaging, and autonomous vehicles. DNNs have become a potent tool for handling complicated picture categorization problems. In this project, we demonstrate a Deep Neural Network implementation for image classification using PySpark and TensorFlow in combination.

A robust data processing framework based on Apache Spark called PySpark has distributed computing capabilities, making it ideal for effectively managing huge datasets. A solid basis for quickly creating and training deep neural networks is offered by TensorFlow, a Google-developed open-source machine learning toolkit.

Our objective is to create an image classifier that can correctly categorize pictures taken from the well-known Intel Image Classification dataset, which includes pictures of various landscapes like mountains, forests, buildings, and oceans. To enhance the performance of the model, we will use a specifically created DNN architecture with a variety of hyperparameters that can be changed, such as the quantity of hidden layers and learning rate.

The implementation is carried out in a sequential manner, beginning with feature extraction from images and data preprocessing. The construction and training of the DNN using PySpark and TensorFlow will then be shown, along with a variety of activation functions, cost functions, and optimization strategies. To fine-tune the network's weights and biases, we'll also provide unique functions for forward and backward propagation in addition to gradient descent.

We will utilize a subset of the Intel Image Classification dataset for training and a different subset for testing in order to assess the performance of our DNN model. We will track the training and testing loss over several iterations in order to understand the convergence behavior. We will also compute confusion matrices and accuracy metrics to evaluate the model's classification performance thoroughly.

The experimental findings from various hyperparameter combinations will be presented in the study, demonstrating how hidden layers and learning rates affect model convergence and accuracy. We will next determine the configuration that performs the best and talk about the conclusions drawn from the experimental research.

BACKGROUND WORK

Numerous neural network types have enabled a wide range of applications, including image categorization, which has fueled the deep learning revolution. Artificial Neural Networks (ANN) and Deep Neural Networks (DNN) stand out among these networks because of their substantial distinctions in architecture, learning, data processing, per-

formance, and hardware. ANN is best suited for processing small to medium-sized datasets because it typically only has one layer or a few layers of neurons. DNN, on the other hand, has numerous layers of neurons and is better able to handle complex datasets. DNN also uses cutting-edge learning techniques, improving accuracy and performance, particularly for complex tasks like speech and picture recognition.

From the machine learning perspective, transfer learning refers to leveraging a pre-existing model trained on a similar task as a starting point for a new related task. Compared to DNN, transfer learning has several advantages in big data applications.

Two significant advantages are faster training and improved accuracy. It reduces the time and computational resources and addresses the problem of limited data availability by allowing the new model to utilize the vast dataset used to train the pre-trained model. This can improve performance and accuracy, mainly when the new task has limited data.

CNN (Convolutional Neural Network) is a deep learning model specialized for image classification tasks. It automatically employs convolutional and pooling layers to learn relevant patterns and features from raw pixel data. These layers help detect edges, textures, and shapes while reducing spatial dimensions to retain essential information. CNNs use backpropagation during training to adjust internal parameters and optimize the network for accurate image classification, making them highly effective in various computer vision applications.

DATASET

The image collection contains 1478 photos of mountains and 1591 photos of buildings, all of which have a resolution of 150x150 pixels and are in the JPEG format. These images are split into two sets for training and validation. The training set consists of 1478 mountain photos and 1591 building photos, while the validation set contains 525 mountain photos and 437 building photos.

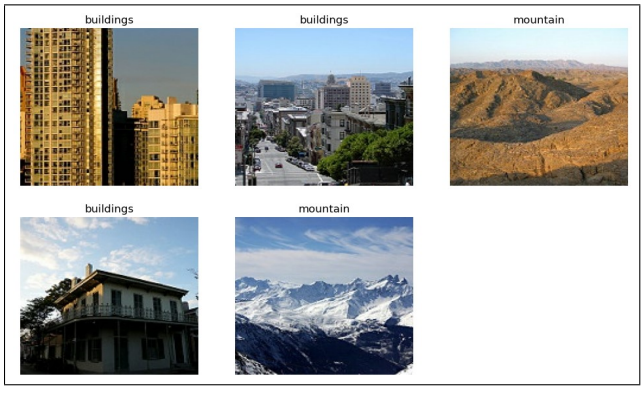

Fig. 1 Sample Images

IMPLEMENTED ALGORITHM

- ***Data Preprocessing:-***
  An essential and fundamental step in getting data ready for use in mathematical models is preprocessing it. It includes several processing stages designed to change the input data into a format that complies with the specifications of the chosen model. These fundamental procedures are critical in ensuring the data is appropriately prepared and optimized for practical analysis and inference. As a result, they improve the overall performance and accuracy of the mathematical models used in various applications. Some operations used are described below:
  - **RGB to Grayscale:** An image's height, width, and three separate color channels are represented as a tensor with the dimensions (height, width, 3), which serves as the first input. The aforementioned color channels are methodically subjected to a specialized weighted average in order to produce the grayscale intensity values. The result is a grayscale image, which has intrinsic advantages over RGB images due to its lower memory and processing power requirements. The computational load that models must bear is significantly reduced as a result of this positive transition. As a result, the procedure guarantees effective resource usage while protecting crucial information required for a variety of applications.
  - **Rescaling:** The [0, 255] range, where 0 represents black and 255 represents white pixels, is where pixel values in an image are generally found. It is common practice to modify these pixel values and put them into a more controllable range of [0, 1] to assure consistency and promote smoother model training with improved stability and convergence. Due to this standardization process, the model grows more adept at processing the data, resulting in enhanced performance and desirable results.
  - **Resizing:** During resizing the images, attempt is to reduce the height and width 150 pixels while keeping the original aspect ratio. This scaling method is frequently used when preparing data for computer vision applications, especially when convolutional neural networks (CNNs) are involved. The model's computational complexity is greatly decreased through image shrinking, resulting in quicker training and evaluation. As a result, the model may concentrate on the image's most crucial components, improving the processing speed and resource usage.
  - **Flattening:** A crucial step in image processing called "flattening" turns a 2D or 3D pixel array into a controllable 1D array. This phase comes before feeding the photos into a machine learning model after decreasing them to a fixed size. The model can efficiently learn from visual patterns by treating each pixel as a unique feature, leading to exact predictions in image-based ML applications.

- ***Data Augmentation:-***
  By changing copies of already-existing data or creating

new instances artificially from the existing dataset, image augmentation techniques aim to increase dataset size. It is used as a regularization technique during the training of machine learning models to lessen overfitting. In this study, we use picture augmentation methods from computer vision and deep learning to increase the training dataset and improve the performance of the machine learning model. The model's resistance to changes in illumination, orientation, and other aspects that affect image quality is increased by giving it more examples, which enhances its accuracy and generalization skills.
Several image augmentation techniques were employed, both individually and in combination:

- **Flipping:** Mirroring images horizontally or vertically aids the model in distinguishing objects with different orientations.
- **Rotation:** By rotating images a specific number of degrees (e.g., 90 or 180 degrees), the model learns to recognize objects in different directions, simulating the effect of tilted or inclined cameras.
- **Cropping:** Reducing the image size enables the model to recognize partially obscured objects or objects located in different image regions.
- **Scaling:** Scaling images up or down supports the model's ability to identify objects of varying sizes.

Our experimental results highlight the efficacy of image augmentation in enhancing machine learning models, contributing to improved performance and robustness in computer vision tasks.

- ***Neural Networks:-***
  Neural networks, inspired by the human brain, consists of neurons sending signal to one another.Neural networks depends on the trained data and are adaptive enough to continuously learn from their take and improve their accuracy.It is a powerful and vital part of deep learning algorithms.

  Neural networks consists of layers with interconnected nodes.A node is a mathematical function used in collecting and classifying information. Generally, neural networks contain 3 layers where input layer is first layer responsible for flow of information from outer world. This layer analyzes and processes the data and pass the output data to Hidden layer. Hidden layer accepts the data from previous hidden layers and passes onto the final layer which is output layer. Output layer produces the final result. The neural networks can have multiple hidden layers with interconnected nodes. Each connection is represented by parameters like weight and biases which helps in achieving highest accuracy. Neural networks have various applications in domians like forecasting and marketing, business analytics, risk assessment, image recoginition and natural language processing.Neural Networks has widely reduced the challenges of handling millions of data making it popular in the current era.

- ***Feed Forward Neural Network:-***
  A type of artificial neural network architecture known as a single hidden layer feedforward neural network, commonly referred to as a single hidden layer neural network, represents one of the fundamental and straightforward structures employed in the fields of machine learning and deep learning.

  The architecture of a single hidden layer feedforward neural network typically consists of three layers (input, hidden and output layers).

  The process of training a single hidden layer feedforward neural network involves feeding the input data forward through the network, calculating the output, comparing it with the true labels, and adjusting the model's parameters (weights and biases) using an optimization algorithm (e.g., gradient descent) to minimize the prediction error.

  While single hidden layer neural networks can be effective for certain simple tasks, they might struggle with more complex problems that require learning intricate representations. As a result, deep neural networks with multiple hidden layers, also known as deep learning models, have gained popularity due to their ability to learn hierarchical representations from data.

  One kind of artificial neural network architecture is a single hidden layer feedforward neural network, often known as a single-hidden-layer neural network. It is one of the most straightforward and fundamental neural network topologies utilized in deep learning and machine learning.

  Three layers—the input, hidden layer, and output layer—typically make up the design of a single hidden layer feedforward neural network.

  Feeding input data into a single hidden layer feedforward neural network, computing the output, comparing it to the true labels, and modifying the model's parameters (weights and biases) using an optimization algorithm (for example, gradient descent) to minimize the prediction error are all steps in the training process.

  Single hidden layer neural networks may perform well for some straightforward tasks, but they may struggle with more challenging issues that demand learning complex representations. Due to their capacity to learn hierarchical representations from data, deep neural networks with several hidden layers, commonly referred to as deep learning models, have become increasingly popular.

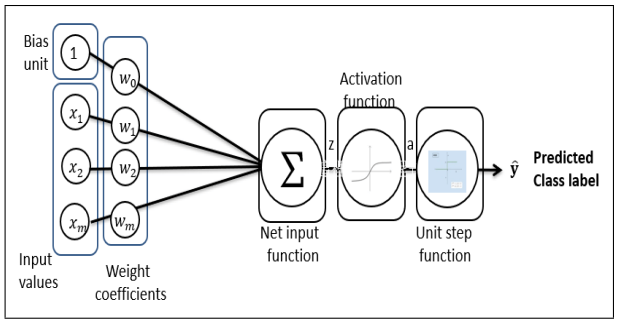

Fig. 2 Single Layer Perceptron Architecture

- ***Layers:-***
  - **Input Layer:** The number of inputs that the Input Layer gets is directly correlated with the number of neurons in that layer. With regard to our project, the image is 150 × 150 pixels in size, and its input layer thus has 22,500 neurons.
  - **Hidden Layer:** The Hidden Layer's hyper-parameters include the variable number of neurons that make up this layer. A task's complexity and kind will determine how many neurons are present in the hidden layer. Depending on how it was implemented, the buried layer's neurons ranged in number from 32 to 128.
  - **Output Layer:** The required number of outputs affects how many neurons are needed in the output layer. Since the goal is to categorize the image into two groups, in our case the output layer has two neurons.

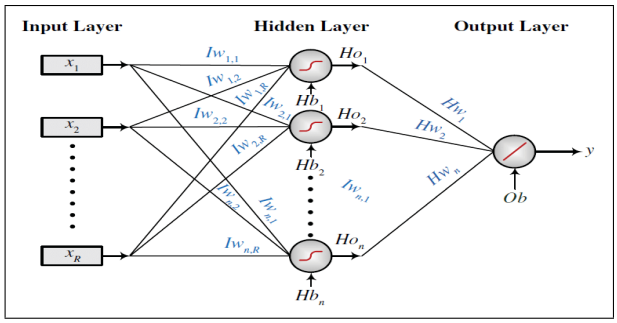

Fig. 3 Layers

- ***Activation Functions:-***
  Activation functions are mathematical functions applied to the input signal of a neuron, generating an output signal. Subsequently, the output signal serves as input for the next layer of neurons. Forward propagation involves passing a neuron's output signal to the subsequent higher layer in the neural network.

  Activation functions can be viewed as transfer functions because they carry the output of one neuron to the input of the next neuron in the neural network. In this context, activation functions are essential for the functioning of neural networks, as they facilitate the flow of information between neurons.
  - **Sigmoid:** The Sigmoid Activation Function maps input values to a range between 0 and 1, making it suitable for binary classification tasks. Nonetheless, it is less ideal for deeper networks due to the vanishing gradient problem it encounters.
  - **ReLU:** The Rectified Linear Unit (ReLU) activation function adds non-linearity to the model by setting all negative inputs to zero and leaving positive inputs unchanged. Its widespread adoption can be attributed to its simplicity and effectiveness in addressing the vanishing gradient problem.
  - **Leaky ReLU:** The Leaky ReLU Activation Function is a modified version of ReLU that permits a small negative slope for negative inputs. Its purpose is to address the "dying ReLU" problem, which can cause specific neurons to constantly output zero by allowing a partial flow of information for negative inputs.
  - **SoftMax:** The Softmax Activation Function is predominantly employed in the output layer for multi-class classification tasks. It transforms a vector of real numbers into a probability distribution, enabling the neural network to make accurate class predictions based on the highest probability.

In conclusion, activation functions in neural networks prepare a neuron's output for the next layer while introducing non-linearity to the model. Among the commonly used activation functions—sigmoid, tanh, and ReLU—each offers distinct advantages and disadvantages.

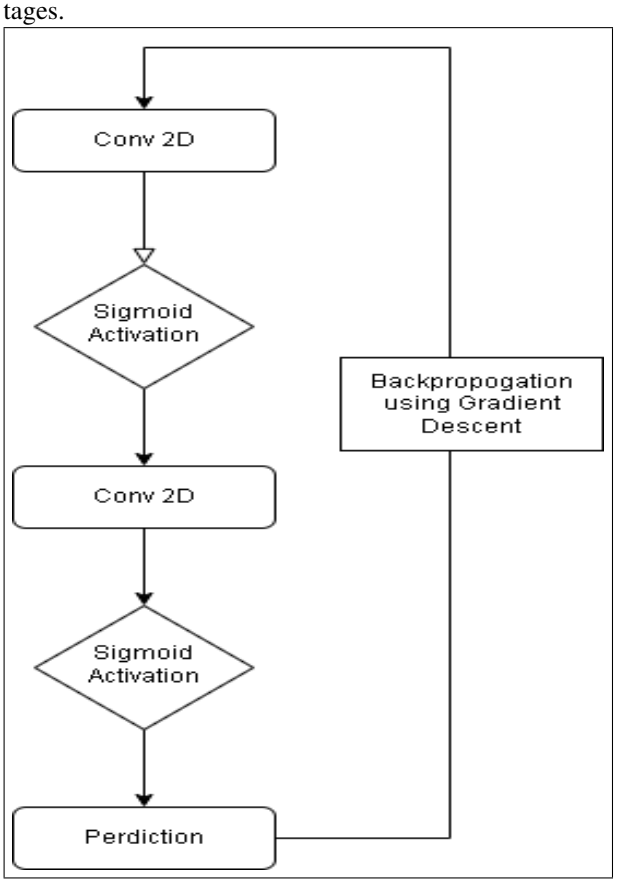

Fig. 4 Implemented Neural Architecture

- ***Gradient Descent and Cost Function:-***
  The primary goal of a machine learning model is to make accurate predictions for unseen data. To achieve this, the model minimizes a cost function that measures the difference between the predicted outputs and the actual outputs for a given set of inputs. The model's parameters, denoted as W and B, are defined in terms of the cost function. Gradient descent is a common optimization approach used to find the optimal values for these parameters. It involves iteratively updating the values of W and B by computing their derivatives with respect to the cost function. This process helps the model gradually improve its predictions over multiple iterations.

  The derivative indicates the rate at which the cost function changes concerning a specific parameter, and its sign indicates the direction in which the parameter should be adjusted to minimize the cost function.

  When the derivative of the cost function is positive, we move in the opposite direction to reduce the cost. Conversely, if the derivative is negative, we progress in a positive direction. The learning rate, denoted as alpha, governs the size of each parameter update. Careful selection of the learning rate is crucial to prevent divergence and ensure stable convergence during the optimization process. It determines the magnitude of the step taken in the direction of the gradient update.

  Using an excessively high learning rate can lead to overshooting the optimal result, while a very low learning rate may cause slow convergence, requiring numerous iterations for the algorithm to reach the best outcome. Typically, a learning rate of 0.01 is a reasonable starting point, but it should be adjusted based on the cost function's behavior during training. Various gradient descent methods, such as batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, can be employed to compute the gradient. These methods differ in the amount of data used to calculate the gradient for each iteration. Equation 4 provides the formula for updating the weights and bias based on the gradient. In summary, gradient descent is a fundamental optimization technique in machine learning that helps determine the best parameter values by iteratively updating them in the direction of the cost function's gradient. The choice of learning rate plays a crucial role in the algorithm's performance as an essential hyperparameter.

$$SSE(y_n) = E = 1/2 \sum_{n=1}^{N} (y_n - y*_n)^2$$

$$MSE(y_n) = E = 1/n \sum_{n=1}^{N} (y_n - y*_n)^2$$

N is the dataset size
y is the predicted output
y is the real output*

Equation 1 Cost Functions

$$W \neg\ W - a \frac{\delta E}{\delta W}$$

Equation 2 Gradient calculation for weights

$$B \neg\ B - a \frac{\delta E}{\delta B}$$

Equation 3 Gradient calculation for Bias

Backward Propagation: The purpose of the backward propagation is to update fresh weights and bias factors that, when combined with gradient descent, lower the error.

$$\begin{cases} w_{ij}^{(1)} = w_{ij}^{(1)} - \alpha \dfrac{\partial E}{\partial w_{ij}^{(1)}} \\ b_j^{(1)} = b_j^{(1)} - \alpha \dfrac{\partial E}{\partial b_j^{(1)}} \end{cases}$$

$$\begin{cases} w_{jk}^{(2)} = w_{jk}^{(2)} - \alpha \dfrac{\partial E}{\partial w_{jk}^{(2)}} \\ b_k^{(2)} = b_k^{(2)} - \alpha \dfrac{\partial E}{\partial b_k^{(2)}} \end{cases}$$

Equation 4 Gradient update calculation for weights and Bias

$$\frac{\partial E}{\partial b_k^{(2)}} = (y_k - y*_k) \times f'(\hat{y}_k)$$

$$\frac{\partial E}{\partial w_{jk}^{(2)}} = \frac{\partial E}{\partial b_k^{(2)}} \times h_j$$

$$\frac{\partial E}{\partial b_j^{(1)}} = f'(\hat{h}_j) \sum_{k=1}^{k} \frac{\partial E}{\partial b_k^{(2)}} \times w_{jk}^{(2)}$$

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial b_j^{(1)}} \times x_i$$

Equation 5 Derivative calculation for error with respect to weights and Bias

EXPERIMENTS AND RESULTS

Hyperparameter tuning is performed to determine the best accuracy of a model. In the model described, model is trained mainly on Hidden layer , Learning rate and Iterations parameters.

Result of the hyperparameter tuning is as follows:

max_iterations	rate	hidden_layers	train_loss	train_accuracy	test_loss	test_accuracy
20	0.2	32	0.237422	0.649837	0.482328	0.275362
30	0.2	32	0.237875	0.625000	0.494802	0.268286
20	0.3	32	0.244604	0.612378	0.527027	0.263201
30	0.3	32	0.235981	0.641287	0.512474	0.263843
20	0.2	64	0.247650	0.603420	0.511435	0.274667
30	0.2	64	0.227156	0.650244	0.483368	0.302072
20	0.3	64	0.224887	0.655130	0.484407	0.297065
30	0.3	64	0.226300	0.646580	0.521830	0.279730
20	0.2	128	0.252797	0.597313	0.462578	0.299584
30	0.2	128	0.227806	0.662459	0.465696	0.321494
20	0.3	128	0.221137	0.686889	0.476091	0.299862
30	0.3	128	0.216289	0.697883	0.476091	0.310645

Table 1 : Hyperparameter Tuning

From above , best accuracy is achieved at Hidden Layer : 128, Iterations: 30 and Learning Rate: 0.3.

For different hyperparameters, we plot the accuracy learning graphs.

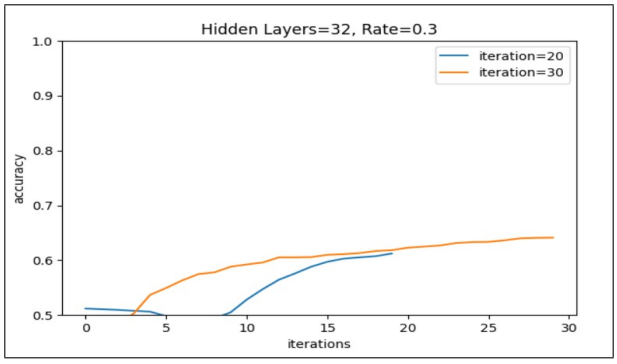

Fig. 5 Accuracy vs Iterations
Hidden Layers = 32, Learning Rate = 0.3

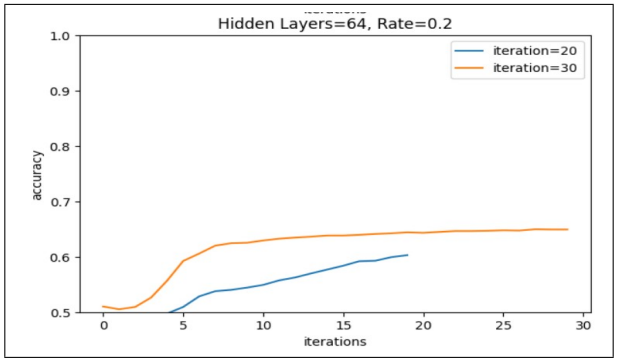

Fig. 6 Accuracy vs Iterations
Hidden Layers = 64, Learning Rate = 0.2

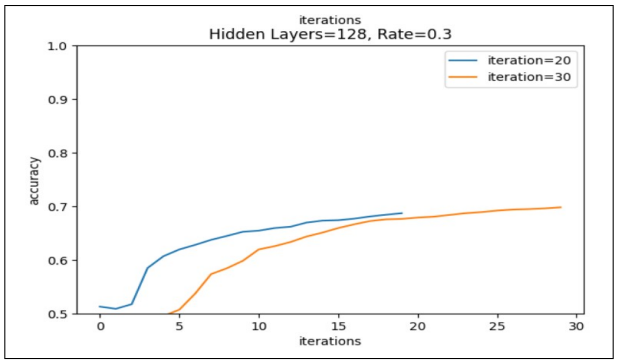

Fig. 7 Accuracy vs Iterations
Hidden Layers = 128, Learning Rate = 0.3

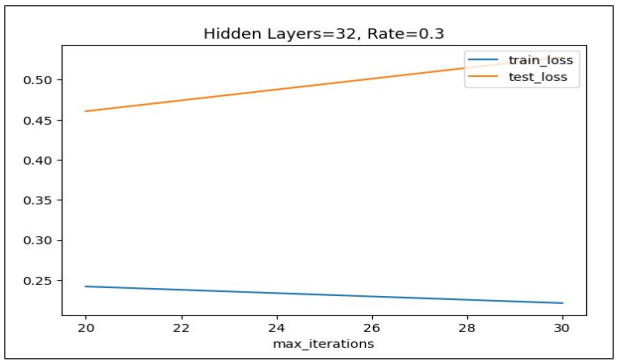

Fig. 8 Loss vs Iterations
Hidden Layers = 32, Learning Rate = 0.3

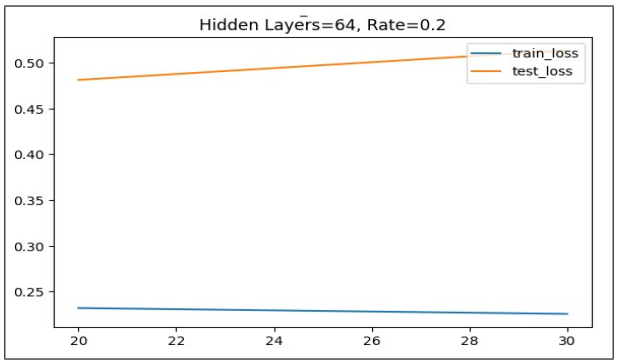

Fig. 9 Loss vs Iterations
Hidden Layers = 64, Learning Rate = 0.2

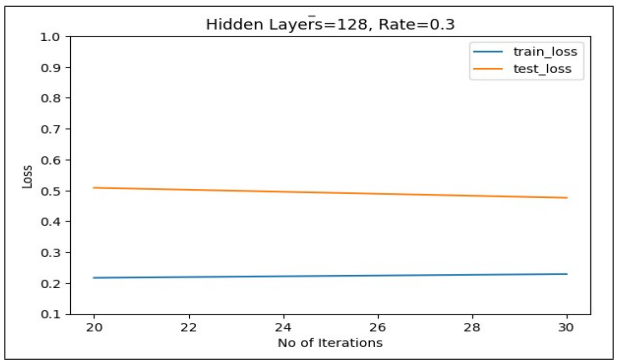

Fig. 10 Loss vs Iterations
Hidden Layers = 128, Learning Rate = 0.3

CONCLUSION

It can take a while to classify images, especially when working with large datasets. Environments with big data clusters can divide the burden among several nodes, enabling quicker processing.

We found one simple neural network method for classifying images in a big data environment: PySpark was used to create a two-layer neural network for image classification, and NumPy was utilized for mathematical computations involving cost functions and sigmoid functions.

Additionally, we noted that Image Augmentation has been demonstrated to boost accuracy by 3% because it enables the machine learning model to more effectively generalize and identify things in fresh, previously undiscovered photos by creating new images with various variants.

We suggest that transfer learning-based models can provide improved performances based on our findings. Pre-trained models are frequently developed on substantial and varied datasets, which improves their ability to generalize to new datasets. Pre-trained models have already picked up on helpful visual cues and patterns that are pertinent to a wide range of jobs.

FUTURE WORK

In future research, the model can be enhanced to achieve high accuracy by applying Multilayer Perceptrons techniques. These techniques utilizes multiple hidden layers of neuron which provides fast and accurate learning.Moreover, to optimize the processing of large datasets and enable distributed computing PySpark and RDD techniques could be integrated.Additionally, research can be expanded towards transfer learning, wherein a model can be fed the knowledge gained by another pre-trained model.

This will accelerate the learning process, ensuring model's overall performance.

REFERENCES

[1] Wenting Qi, Charalampos Chelmis, "Improving Algorithmic Decision–Making in the Presence of Untrustworthy Training Data", 2021 IEEE International Conference on Big Data (Big Data), pp.1102-1108, 2021.

[2] Perez, L., & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. ArXiv. /abs/1712.04621

[3] Shorten, C., and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 60.

[4] B. Ding, H. Qian and J. Zhou, "Activation functions and their characteristics in deep neural networks," 2018 Chinese Control And Decision Conference (CCDC), Shenyang, China, 2018, pp. 1836-1841, doi: 10.1109/CCDC.2018.8407425.

[5] C. V. Gonzalez Zelaya, "Towards Explaining the Effects of Data Preprocessing on Machine Learning," 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 2086-2090, doi: 10.1109/ICDE.2019.00245.

[6] Durdu, Akif & Cetin, Halil & Komur, Hasan. (2014). Robot Imitation of Human Arm via Artificial Neural Network. Proceedings of the 16th International Conference on Mechatronics, Mechatronika 2014. 10.1109/MECHATRONIKA.2014.7018286.

[7] Simonyan, K., and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[9] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536.

[10] Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford University Press

[11] V. K. Verma, V. Kansal and P. Bhatnagar, "Patient Identification using Facial Recognition," 2020 International Conference on Futuristic Technologies in Control Systems & Renewable Energy (ICFCR), Malappuram, India, 2020, pp. 1-7, doi: 10.1109/ICFCR50903.2020.9250002.