Exploring Convolutional Neural Networks for Image Classification

Paridhi Awadheshpratap Singh
Student ID: a1865487
Email: a1865487@adelaide.edu.au

ABSTRACT

This paper dives into an evolving and complex field of image processing and classification. Categorisation of images into a set of predefined groups is called as image classification. Despite the ongoing research in this dynamic field, earlier studies have produced rudimentary and limited image analyses, resulting in the considerable challenge of effective image classification. This paper addresses the need for more sophisticated solutions by focusing on Convolutional Neural Networks commonly known as CNNs, a type of deep neural network known for its effectiveness in visual image analysis. CNNs use a multilayer perceptron with a hierarchical structure, culminating in a fully connected layer for comprehensive output processing. We have used the CIFAR-10 dataset for our analysis. For validation purposes the research code has been made available, on GitHub with a timestamp. In summary, this research aims to explore the possibilities for advancement in image classification by implementing CNN architecture.

I. Introduction

In recent years, advancements in image classification technology have been significant, particularly with the adoption of Convolutional Neural Networks, or CNNs. Though they have been in existance since the 1990s, gaining recognition with Yann LeCun's LeNet, it is the last few years that have unleashed their true potential which was possible due to advances in training these complicated networks on datasets extensively.

Image classification which is the process of categorizing images into different unique classes, is a fundamental task with widespread applications. It plays a pivotal role in various domains, including autonomous driving and enhancing user experiences on social media platforms like Facebook and Google Photos. Image classification presents unique challenges due to variations in scale, lighting, perspective, and object arrangement. The effectiveness of image classification techniques often extends to other critical computer vision tasks, such as object detection, localization, and segmentation.

A compelling example of the interplay between image classification and other AI domains is image captioning. This task involves generating descriptive sentences for images and is at the intersection of computer vision and natural language processing [1].

Recognizing the importance of CNNs in image classification, this research paper aims to provide a comprehensive understanding of their implementation and performance in practical applications. By dissecting their architectural intricacies, functionality, and nuances of training, our objective is to reveal the latest potential of CNNs.

For robust evaluation, we have utilized the well-established CIFAR-10 dataset, recognized as a benchmark for image classification which is publicly available and generously provided by Alex Krizhevsky [2]. Its diverse array of images offers a challenging testbed, mirroring the complexities encountered in real-world image classification scenarios.

The research code has been made available on GitHub with timestamps. This research endeavor extends beyond theoretical exploration, seeking to bridge the gap between theoretical advancements and practical applications, with the overarching goal of further enhancing image classification methodologies using CNNs.

II. Method Description

Convolutional Neural Networks(CNNs) are a class of Deep Neural Networks which are extensively used for identifying the features of an image and hence analysing a visual image. Image classification, image and video recognition, medical image analysis, natural language processing and computer vision are few of the sectors in which CNNs are deeply used.

CNN is very useful in image recognition due to its high accuracy and this image recognition technique is used in various sectors like security, phone, recommendation systems, etc.

CNNs use a special kind of linear function wherein two functions are multiplied to produce a third function that expresses how the shape of one function is modified by the other. This function is called "Convolution".

A basic CNN architecture constitutes of two main components

- Feature Extraction : A convolution tool segregates and recognises the various features of an image for analyses. A number of pairs of convolution or pooling layers constitute the network of feature extraction.
- Fully Connected Layer : The output from the convolution process is utilised in this layer which then predicts the class of the images per the extracted features in previous stages.

This CNN model of feature extraction aims to reduce the number of features present in a dataset by summarising the

existing features contained in an original set of features [3]. A diagramatic representation of a CNN model is:

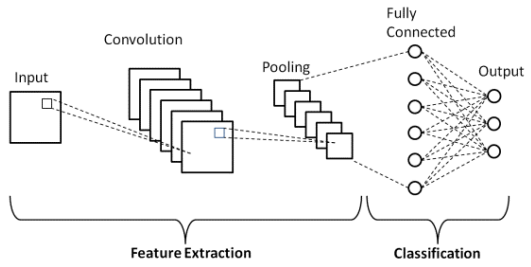

Fig. 1. CNN Architecture [3]

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

A. Convolution Layer

The first layer of the network that is used for feature extraction from input images is the convolution layer. It involves a basic process: it commences with a kernel, essentially a compact matrix of weights. This kernel moves across the 2D input data, executing element wise multiplication with the section of the input it's currently positioned on, and subsequently adding up the outcomes to generate a solitary output pixel.

The kernel repeats this operation as it moves across every position, thereby transforming a 2D feature matrix into another 2D feature matrix. In essence, the output features are computed as weighted sums, where the weights correspond to the values in the kernel. These weights are applied to input features that are approximately situated in the same spot on the input layer as the output pixel.

The inclusion of an input feature in this "approximate location" is directly determined by whether it falls within the kernel's area responsible for generating the output. Consequently, the kernel's size directly influences the number of input features involved in creating a new output feature – determining the extent of combination [4].

There are two techniques used commonly in a convolution layer:

1. Padding : There are instances where the corners of the matrix gets "trimmed-off" during the sliding process of the filter over the input layer. This may create an aggregation bias from the convolution layer. To make sure all pixels are considered evenly we add some extra fake pixels to the original matrix (usually of value 0, hence the oft-used term "zero padding") so that the edge pixels can be at the centre during the sliding process and hence help in producing an output of the same size as the input [4].

2. Striding : When the size of the desired output is less than the input size, we achieve it by either using a pooling layer which is explained in detail in the next section , or by using a stride, wherein we reduce the spatial dimensions by increasing the number of channels.

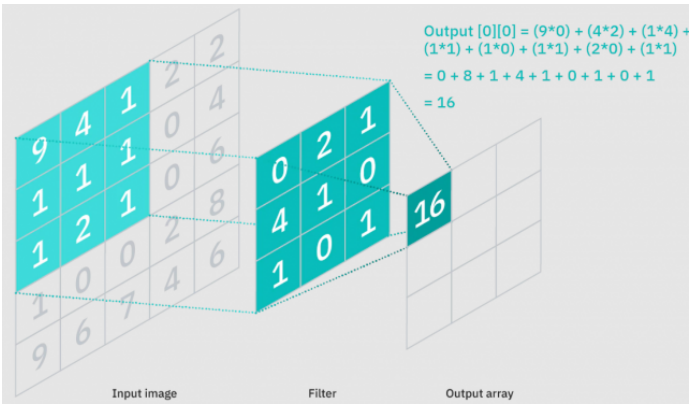

Fig. 2. Convolution Operation [5]

The concept of the stride is to intentionally *skip* certain positions of the kernel as it moves. A stride of 1 corresponds to selecting positions that are one pixel apart, effectively covering every position and functioning as a regular convolution. A stride of 2 involves selecting positions that are two pixels apart, skipping every alternate position, which results in a reduction of size by approximately a factor of 2. Similarly, a stride of 3 entails skipping every two positions, leading to a size reduction of roughly a factor of 3, and so forth [4].

B. Pooling Layer

A lot of times we introduce another layer following the convolution layer which primarily focuses on reducing the computational costs by decreasing the size of the convolution feature map call the Pooling layer. This is achieved by decreasing the number of connections between layers and independent operation on each layer. We use various types of pooling layer depending on the method we implement in our model. The features generated by the convolution layer are summarised by this layer [3].

The most commonly used pooling layers are:

- Max Pooling: Max pooling is a pooling operation in which the largest element within the filter's region on the feature map is chosen. Consequently, the output generated by a max-pooling layer consists of a feature map highlighting the most significant characteristics from the preceding feature map [6].
- Average Pooling: Average pooling calculates the mean of the elements contained within the filter's region on the feature map. Therefore, whereas max pooling identifies the most dominant feature in a specific section of the feature map, average pooling provides the average of the features found within that section [6].
- Global Pooling: Global pooling operations condense each channel within the feature map into a single value. Consequently, a feature map of dimensions $n_h \times n_w \times n_c$ is transformed into a $1 \times 1 \times n_c$ feature map. This process is

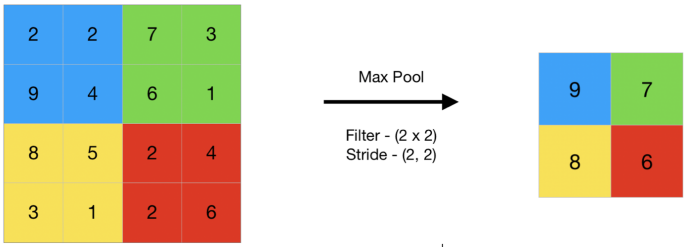

Fig. 3. Max Poling [6]

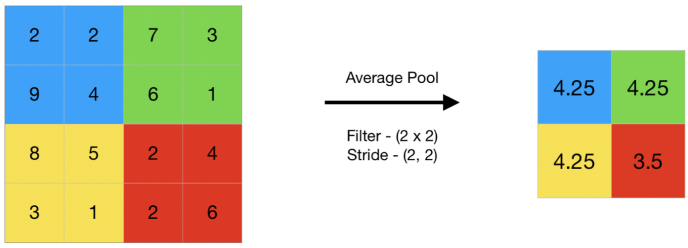

Fig. 4. Average [6]

analogous to employing a filter with dimensions $n_h \times n_w$, matching the feature map's dimensions. Moreover, this global pooling can take the form of either global max pooling or global average pooling [6].

C. Fully Connected Layer

The weights ans biases along with the neurons constitute the Fully Connected layer (FC). This is used as a connecting layer between two distinct layers. Typically, these layers are positioned preceding the output layer and constitute the final layers within a CNN Architecture.

In this context, the input image, having traversed preceding layers, is flattened and directed into the FC layer. The flattened vector then proceeds through several additional FC layers, where mathematical operations commonly occur. This is the stage at which the classification process commences. The rationale behind connecting two layers is that a pair of fully connected layers tends to yield superior performance when compared to a single connected layer. In the context of CNNs, these layers contribute to a reduction in the need for direct human supervision [3].

D. Dropout

Sometimes the model we implement my encounter a situation wherein the model works too well on the training dataset that it creates a negative impact on the performance of the model when used on an "unknown" or new dataset. This situation is called as overfitting and to avoid this problem we introduce a dropout layer in our neural network where we drop a few neurons from the network to reduce the size of the model. This is achieved during the training process. 30 percent of the neurons are dropped out from the network randomly when a dropout of 0.3 is passed.

Dropout enhances the performance of a machine learning model by mitigating the risk of overfitting, achieved through the simplification of the network. It accomplishes this by selectively deactivating neurons within the neural network during the training process [3].

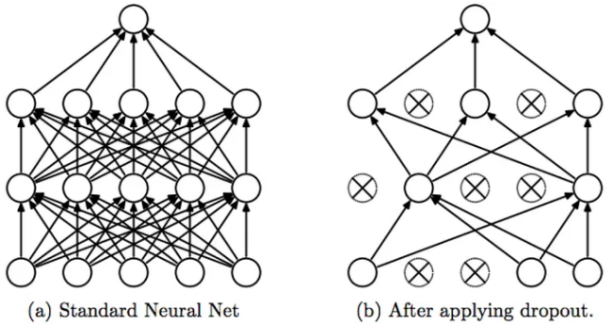

Fig. 5. Dropout [7]

E. Activation Function

The activation function stands as one of the CNN model's pivotal parameters. It plays a crucial role in comprehending and approximating intricate relationships among the network's variables, creating a bridge to convey which model information should be triggered during the forward pass and which should remain dormant as the network concludes its processing.

Incorporating non-linearity into the network, the activation function introduces a range of widely employed functions such as ReLU, Softmax, tanH, and Sigmoid. Each of these functions has distinct applications: for binary classification CNN models, the preference leans towards sigmoid and softmax functions, while for multi-class classification, softmax is generally employed. In straightforward terms, the activation functions within a CNN model serve as the gatekeepers, making determinations regarding the significance of input through mathematical operations for prediction purposes [3].

Most commonly used activation functions along with their graphs are:

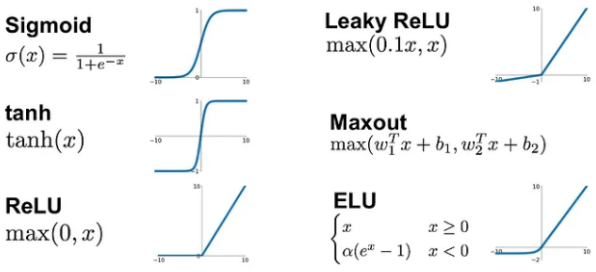

Fig. 6. Commonly used Activation Functions [8]

These are the vital components of a CNN Architecture employed in our analysis. There are a number of components we add for an efficient working of our model. Some of the techniques that we used in our experimentation are:

- Batch Normalisation :Batch normalization is crucial in deep neural networks as it addresses the internal covariate

shift issue by normalizing intermediate outputs within each batch during training. This normalization stabilizes the optimization process, allowing for higher learning rates, faster convergence, and improved generalization. The process involves calculating batch mean and variance for each feature, normalizing the activations, and applying learnable scaling and shifting. During inference, population statistics from training are used for consistency. This method enhances gradient flow and training efficiency. [9].

- Data Augmentation : Image augmentation is a method for creating additional training data for deep learning models by manipulating existing images. It serves to expand the dataset and enhance the model's resilience. Various common techniques, including rotation, shifting, flipping, introducing noise, and applying blur, are employed selectively based on the specific problem and model. The application of image augmentation has the potential to enhance both the accuracy and generalization capabilities of the model.
- Residual connections : A residual block is a type of neural network layer that incorporates a skip connection from the input to the output. This enables the network to learn the residual or difference between them. It addresses the problem of diminishing accuracy as more layers are added to a neural network, caused by issues like vanishing gradients and the curse of dimensionality. By bypassing the training of some layers and allowing the input to directly pass to the output, it facilitates the learning of identity functions and the propagation of larger gradients to initial layers. Residual blocks can be thought of as training an ensemble of diverse models, dynamically adjusting the number of layers during training, and enabling memory flow from initial to final layers. Different arrangements of pre-activations and batch normalizations can be employed for optimal gradient flow. A residual block is illustrated below [10].

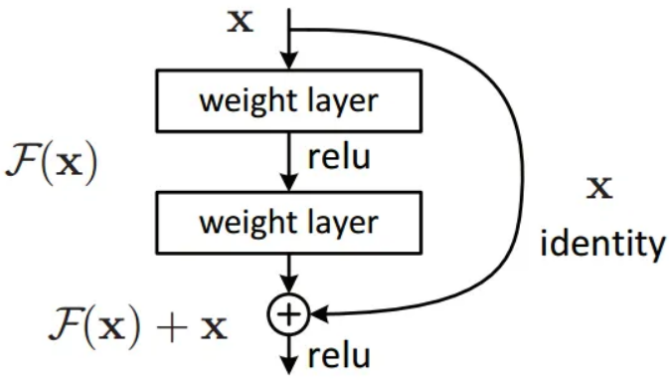

Fig. 7. Single Residual Block [10]

- Batch Normalisation : Batch Normalization is believed to accelerate training and simplify the learning process for several reasons. One key factor is that it normalizes inputs to ensure they fall within a comparable value range, which expedites learning. An intuitive perspective is that Batch Normalisation extends this normalization not only to the input but also across the network's layer values. In convolutional layers, where shared filters span the feature maps of the input (typically representing height and width in images), it makes sense to normalize the output in a similar shared manner across these feature maps.
- Learning rate scheduling : We experiment with different learning to know which one fits the model better. A popular technique used for scheduling the learning rate in training Convolutional Neural Networks (CNNs) for image classification is the **One Cycle Learning Rate Policy**. It consists of two phases: a cycle phase and a decay phase. The cycle phase oscillates the learning rate between a minimum and a maximum value, while the decay phase gradually reduces the learning rate to zero. The 1-cycle policy aims to achieve fast and robust convergence of neural networks by using high learning rates to escape saddle points and low learning rates to avoid divergence. The 1-cycle policy is based on the idea of cyclical learning rates, which vary the learning rate periodically within a range. The 1-cycle policy is also related to the concept of super-convergence, which refers to the phenomenon of training neural networks very quickly using large learning rates. The 1-cycle policy has been shown to improve the accuracy and generalization of various neural network architectures on different tasks [11].
- Weight Decay : Weight decay is a regularization technique that penalizes the complexity of a neural network by adding the sum of squares of its weights to the loss function. It prevents overfitting by shrinking the weights towards zero, which reduces the variance of the model. Weight decay is also known as L2 regularization or ridge regression. We have used weight decay to improve the accuracy and generalization of an image classification model trained on our dataset [12].
- Gradient Clipping : Gradient clipping is a technique to deal with the problem of exploding gradients in recurrent neural networks (RNNs). It limits the magnitude of the gradients to a predefined threshold, preventing them from becoming too large and causing numerical instability. Gradient clipping can be useful for convolutional neural networks (CNNs) as well, especially when they are very deep or have large learning rates. Gradient clipping can help CNNs avoid divergence and achieve faster convergence [13].

III. Method Implementation

The entire experiment was done using Python as the primary programming language for implementing and training Convolutional Neural Networks (CNNs) due to its extensive libraries, such as TensorFlow and PyTorch, which facilitate efficient deep learning model development and experimentation.

A. Cifar10Model

To facilitate our experiment, we divided the dataset into Training, Testing and Validation datasets by allocating 45,000 images for training, 10,000 images for testing while 5,000 images for validation. We then divided the images into batches for further processing. A grid of one such batch is attached below.

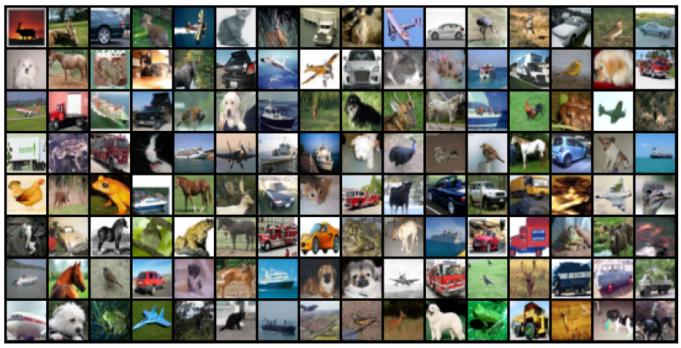

Fig. 8. Grid of images

We implemented a basic image classification model based on a Convolutional Neural Network (CNN) for the task of classifying images from the CIFAR-10 dataset from scratch. The model architecture is defined by the 'Cifar10Model' class, which extends a base class, 'ImageClassificationBase'. The model comprises a series of convolutional layers, rectified linear unit (ReLU) activation functions, and max-pooling layers. It starts with an initial convolutional layer configured to accept 3-channel RGB images of size (3, 32, 32) and the subsequent layers progressively reduce the spatial dimensions while increasing the number of feature maps. The final fully connected layer outputs predictions for the ten distinct classes present in the CIFAR-10 dataset. Training and evaluation procedures are facilitated by the 'fit' and 'evaluate' functions and the model is optimized using gradient descent with a specified learning rate and optimizer. - The 'ImageClassificationBase' class provides essential methods for training and evaluation, including loss calculation, accuracy measurement, and epoch-level results reporting. Training involves iterating through the training dataset, computing gradients, and updating model parameters. After each training epoch, the model's performance is evaluated on the validation set to ensure generalization. The 'evaluate' function computes validation losses and accuracies. The 'fit' function orchestrates the training process across multiple epochs and collects a history of training and validation metrics [14].

The '`to_device`' function transfers the model and data to the appropriate computing device (GPU or CPU) for efficient training. Once trained, the model is utilized for image classification tasks by passing image data through the 'forward' method of the 'Cifar10Model'.

B. ResNet9

We conducted additional experiments to enhance the performance of our model. We implemented a ResNet9 from scratch using the model 'Cifar10Model' as the base model. We made some alternations in this model which are:

- Standardization: We normalized the image tensors by subtracting the mean and dividing by the standard deviation of pixels across each channel. Normalizing the data prevents the pixel values from any one channel from disproportionately affecting the losses and gradients [10].
- Data augmentation: We applied random transformations while loading images from the training dataset. Specifically, we will pad each image by 4 pixels, and then take a random crop of size 32 x 32 pixels, and then flip the image horizontally with a 50 percent probability.
- Residual connections: One of the key changes to our CNN model was the addition of the residual block, which adds the original input back to the output feature map obtained by passing the input through one or more convolutional layers. We used the ResNet9 architecture [10].
- Batch normalization : After each convolutional layer, we added a batch normalization layer, which normalizes the outputs of the previous layer. This is somewhat similar to data normalization, except it's applied to the outputs of a layer, and the mean and standard deviation are learned parameters [9].
- Learning rate scheduling : Instead of using a fixed learning rate, we will use a learning rate scheduler, which will change the learning rate after every batch of training. There are many strategies for varying the learning rate during training, and we used the "One Cycle Learning Rate Policy".
- Weight Decay : We added weight decay to the optimizer, yet another regularization technique which prevents the weights from becoming too large by adding an additional term to the loss function [12].
- Gradient clipping : We also added gradient clipping, which helps limit the values of gradients to a small range to prevent undesirable changes in model parameters due to large gradient values during training [13].
- Adam optimizer : Instead of SGD (stochastic gradient descent), we used the Adam optimizer which uses techniques like momentum and adaptive learning rates for faster training. There are many other optimizers to choose from and experiment with.

We used transfer learning to see how different CNN models perform with our dataset. This technique helps improve the efficiency and effectiveness of CNNs in image classification tasks, especially when dealing with limited data and computational resources. We have employed another model DenseNet121 for our analysis using Transfer Learning.

C. DenseNet121

DenseNet121, short for "Densely Connected Convolutional Networks," represents a family of CNNs that embrace a dense connectivity pattern between layers. This design fosters improved feature reuse and gradient flow across the entire network. DenseNet is a well-established and widely adopted

pre-trained CNN architecture, frequently employed for image classification tasks.

DenseNet121 is built on a series of dense blocks, each with multiple convolutional layers. What's unique is that each dense block takes input from the previous block and all previous blocks, creating strong connections between all the layers for better information flow.

The "121" in DenseNet121 refers to its 121 layers, and the special feature here is that the output of each layer is used as input for all the following layers. To manage the complexity, DenseNet121 includes transition layers between the dense blocks. These transition layers have batch normalization, a 1x1 convolutional layer, and pooling to reduce dimensionality and the number of feature maps. DenseNet121 has fewer parameters but higher accuracy in image classification tasks. It also avoids overfitting, which means it can generalize well to new data. DenseNet121 can learn rich features of images, which can improve its performance in image classification. However, DenseNet121 also has some drawbacks. It can be slow and costly to train and run compared to other architectures. It may need more memory to store intermediate feature maps, which can limit the size of the images it can handle and make it less suitable for some devices. DenseNet121 may also depend too much on earlier layers, which may affect its ability to learn more complex features in later layers [15].

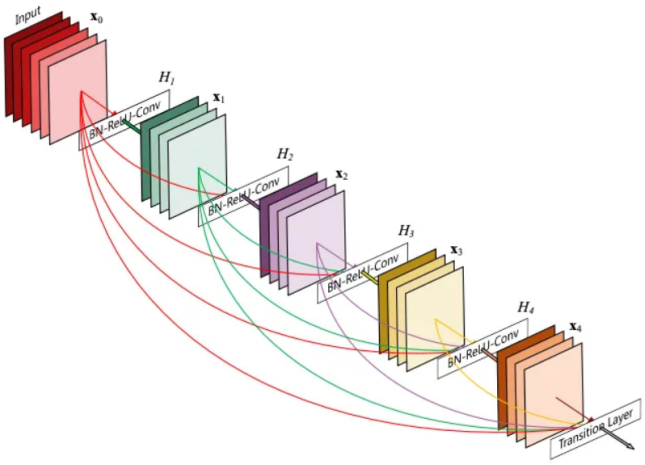

Fig. 9. DenseNet121 Architecture [15]

The Python code [14] for Image Classification using CNNs in this GitHub repository. The code contains three models: ResNet9 and Basic CNN, which are built from scratch, and DenseNet121, which uses Transfer Learning. The tutor jinan.zou@adelaide.edu.au has been granted access to the repository as per the assignment requirements and the timestamp has been recorded for verification.

IV. Experiments and Analysis

We first implemented a traditional CNN model with 8 hidden layers in its architecture. To understand if the model fits the dataset well we produced plots of validation accuracy and training and validation losses against the number of epochs. The plots are illustrated below.

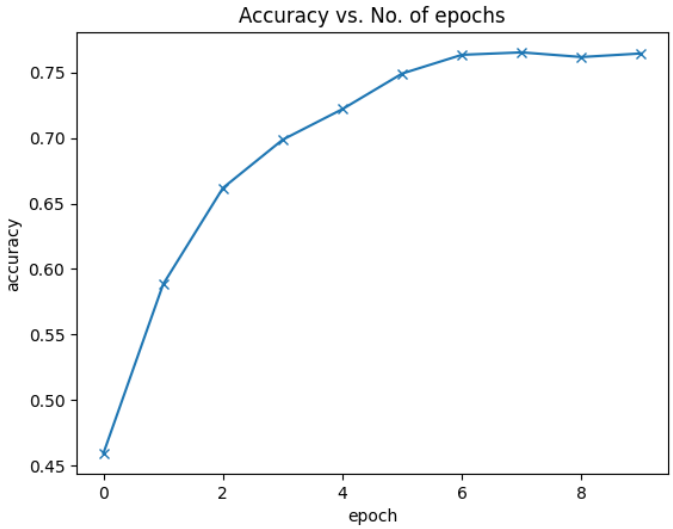

Fig. 10. Cifar10Model Accuracy

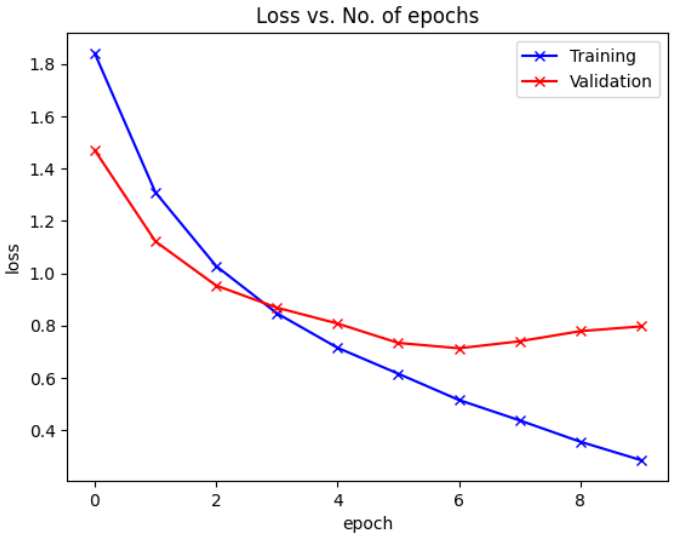

Fig. 11. Cifar10Model Training and Validation Losses

While we did attain a validation accuracy of 76.45 percent with this model, we observed that the model begins to exhibit overfitting after just 4 epochs, as evidenced by the divergence seen in the loss graph displayed above. Additionally, we faced issues with vanishing or exploding gradients in this model.

In an effort to address these challenges and identify a more suitable model for our dataset, we transitioned to another model, DenseNet121. This model incorporates dense blocks in its architecture, each consisting of multiple convolution layers. Notably, DenseNet121 is renowned for its capability to mitigate overfitting and effectively generalize new data. It's

also recognized for delivering superior results while employing fewer parameters. Here are the outcomes we obtained using this model:

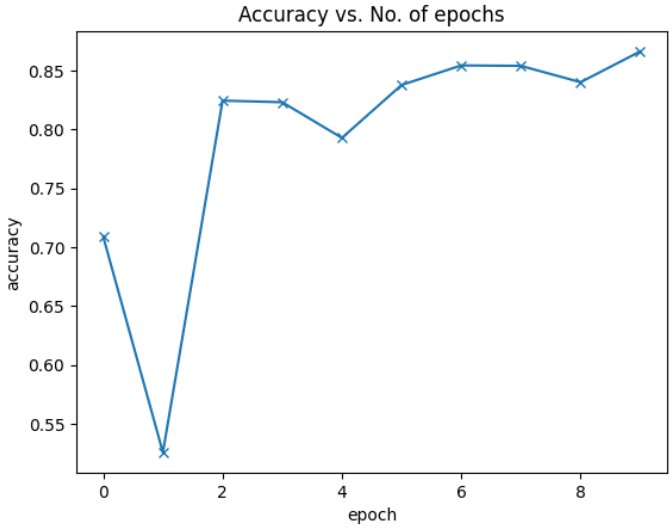

Fig. 12. DenseNet121 Accuracy

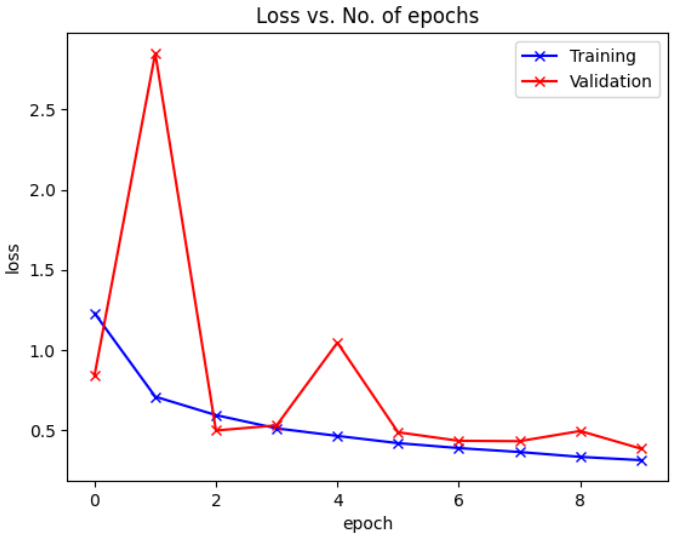

Fig. 13. DenseNet121 Training and Validation Losses

We successfully attained an accuracy of 86.62 percent on the validation dataset with the DenseNet121 model. Nevertheless, this achievement came at the cost of significantly increased computation time, leading to slower training. Furthermore, the model consumed a substantial amount of memory to store its feature maps, presenting challenges in terms of scalability and running on multiple devices.

Subsequently, we explored a less dense model, the ResNet9, which features a 9-layer architecture. We adapted the 'Cifar10Model' that we initially developed from the ground up, incorporating various enhancements such as residual blocks, batch normalization, data augmentation, learning rate scheduling, weight decay, gradient clipping, and dropout regularization. Here are the results we obtained with this model:

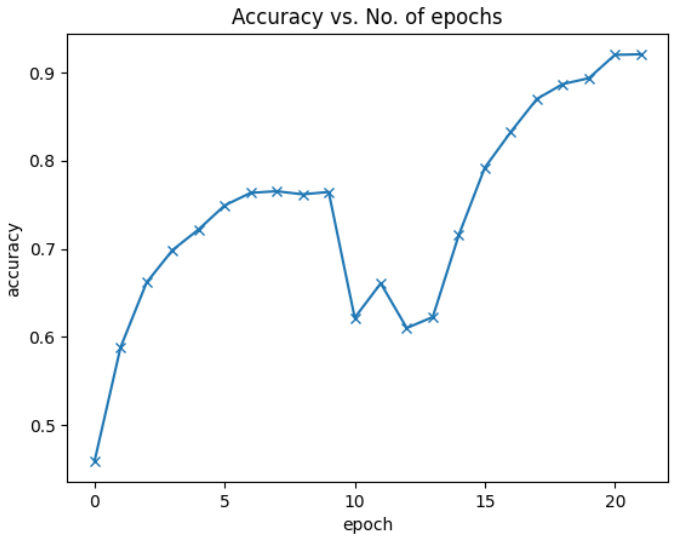

Fig. 14. ResNet9 Accuracy

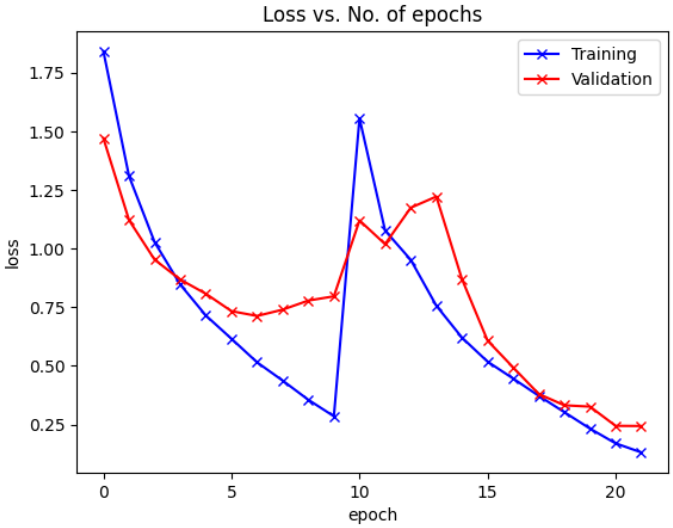

Fig. 15. ResNet9 Training and Validation Losses

We obtained an accuracy of 92.05 percent on the validation dataset with the ResNet9 model. Here's a table summarizing the findings from all three models (Cifar10Model, DenseNet121, and ResNet9):

Model	Epochs	Learning Rate (max)	Optimizer	Test Set Accuracy
Cifar10Model	10	0.001	Adam	76.16
DenseNet121	10	0.01	SGD	86.46
ResNet9	12	0.01 (max)	Adam	91.58

TABLE I
MODEL PERFORMANCE

Based on the plots and the table presented above, it's evident that the ResNet9 model is the most well-suited choice for our dataset. Despite having a similar number of parameters as the Cifar10Model, ResNet9 outperforms the latter by a substantial margin. It also effectively mitigates issues related to overfitting and vanishing gradients. Furthermore, ResNet9 surpasses DenseNet121 in terms of results while also offering the advantages of reduced computation time and memory usage.

Considering these factors, we confidently assert that ResNet9 stands as the most optimal model for our dataset.

V. Reflection on Project

Our experience in crafting and implementing a Convolutional Neural Network (CNN) for image classification has been a blend of challenges and rewards. This reflection offers insights into our key design decisions, the lessons we've gleaned, and potential directions for future endeavors.

The selection of an appropriate CNN architecture stood out as a pivotal choice. While we initially embraced widely-used architectures like the basic CNN, ResNet, and DenseNet, we are open to exploring other architectures such as MobileNet and EfficientNet in future projects. Each architecture brings its unique strengths, potentially yielding superior results tailored to specific image classification tasks.

Our project showcased the significant impact of data preprocessing, encompassing normalization and augmentation, on enhancing model performance and generalization. Augmentation techniques, including cropping and flipping, revealed substantial influence. To fortify our model, we anticipate delving deeper into data augmentation strategies and approaches.

Hyperparameter tuning, particularly concerning learning rates, batch sizes, and weight decay, proved to be an iterative process. Embracing learning rate schedules, such as the "One Cycle Learning Rate Policy," enriched our model training. In forthcoming projects, we can explore the adoption of automated hyperparameter optimization methods to streamline this fine-tuning process.

Our incorporation of regularization techniques like batch normalization, weight decay, and gradient clipping contributed positively to the model's overall generalization. As we continue to refine our models, we look forward to experimenting with novel regularization methods.

While assessing our model, we found the selection of performance metrics, such as accuracy and loss, to be instrumental in our comprehensive evaluation. Future endeavors might involve the integration of additional metrics tailored to specific applications, including precision, recall, or the F1 score.

Acknowledging the computational intensity of training deep CNNs, we prioritized the utilization of GPU resources for efficient training. In the future, we may explore further optimization through distributed computing or the exploration of cloud-based GPU solutions, enabling us to tackle larger datasets and more complex models effectively.

Our current project focused primarily on model performance. However, in certain applications, comprehending the decision-making process is of paramount importance. Subsequent work may encompass the integration of techniques that enhance model interpretability and explainability.

In summation, this project has provided us with invaluable hands-on experience in the design, implementation, and assessment of CNNs for image classification. We have witnessed the profound impact of various design choices and embraced the significance of continuous experimentation. As we move forward, we emphasize the importance of staying attuned to the latest developments in the realms of computer vision and deep learning to perpetually refine our models and expand our problem-solving capabilities.

References

[1] S. Padmanabhan, "Convolutional neural networks for image classification and captioning," Stanford, CA, 2016.

[2] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Toronto, ON, Canada*, 2009.

[3] M. Gurucharan, "Basic cnn architecture: Explaining 5 layers of convolutional neural network," 2020. [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture

[4] I. Shafkat, "Intuitively understanding convolutions for deep learning," 2018, (Accessed 25. 10. 2023). [Online]. Available: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[5] A. Kumar, "Real-world applications of convolutional neural networks," *Vitalflux*, vol. 18, p. 2023, 2021, (Accessed 25. 10. 2023). [Online]. Available: https://vitalflux.com/real-world-applications-of-convolutional-neural-networks

[6] S. Khosla, "Cnn — introduction to pooling layer," GeeksforGeeks (Accessed 25. 10. 2023), 2020.

[7] A. Budhiraja, "Dropout in (deep) machine learning," *Medium*, 2016, (Last accessed April 26, 2020). [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machi

[8] S. Jadon, "Introduction to different activation functions for deep learning," *Medium, Augmenting Humanity*, vol. 16, p. 140, 2018.

[9] N. Kumar, "Batch normalization and dropout in neural networks with pytorch," *URL: https://towardsdatascience. com/batch-normalization-and-dropoutin-neural-networks-explained-with-pytorch-47d7a8459bcd (cit. on p. 84)*, 2019 (Accessed 25. 10. 2023).

[10] S. Sahoo, "Residual blocks—building blocks of resnet," *Towards Data Science*, 2018 (Accessed 25. 10. 2023)).

[11] S. Gugger, "The 1cycle policy, apr 2018 (accessed 01. 11. 2023)," *URL https://sgugger. github. io/the-1cycle-policy. html*.

[12] D. Vasani, "This thing called weight decay," *Towards Data Science*, 2019 (Accessed 25. 10. 2023).

[13] W. Wong, "What is gradient clipping? a simple yet effective way to tackle exploding gradients," *Towards Data Science*, 2020 (Accessed 25. 10. 2023).

[14] A. N S, "Deep learning with pytorch: Zero to gans," *Jovian*, 2020 (Accessed 25. 10. 2023). [Online]. Available: https://jovian.com/learn/deep-learning-with-pytorch-zero-to-gans

[15] N. AI, "Densenet," *Medium*, 2023 (Accessed 25. 10. 2023). [Online]. Available: https://medium.com/nocoding-ai/densenet121-760df192f12d