Image Classification using Deep Learning

Harsh Sanjay Rathi
The University of Adelaide
North Terrace, Adelaide, SA, 5000
A1882311@adelaide.edu.au

Abstract

Image recognition is a challenging task for computer vision these days which involves categorizing images in highly detailed level. Often multilayer classification with images is more complex when it comes to sub categorical objects. In this project I have explored multilayer classification of images by two architectures of CNN and compared their performance. The projects have several key points, I collected a data set having high resolution images with over 100 subclasses in it. I took VGG19 and ResNet-152 as my CNN models and fine-tune them for the specific task of recognition and classification of images. To improve my model training I have implemented data augmentation techniques, transfer learning techniques and optimize hyperparameters. While my training process I kept an eye on my model's performance by tracing training loss and validation accuracy, on other hand I explored variety of batch size and epochs numbers to find best accuracy of the model and prevent model from over fitting. My project aims to contribute to the field of image classification by CNN by comparing VGG19 and ResNet-152 architectures with each other. This project focuses on the multiclassification CNN.

1. Introduction

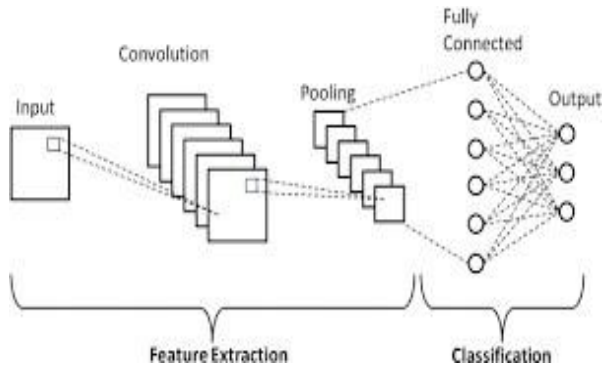


Figure 1: Convolutional Nural Network

A basic CNN model looks like 'figure1', which has two parts in it feature extraction and classification. Feature extraction contains an input parameter where inputs in the form of images are given to the algorithm. These images are nothing but pixels in the form of numbers to the computer and each number represents its own color or density.

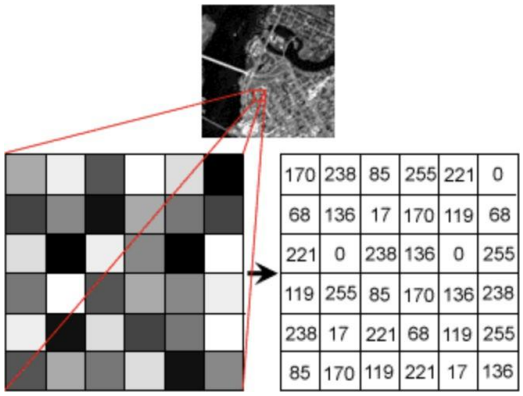


Figure 2: example of numerical information of the image

The convolutional and pooling layers extract meaningful data in the form of numbers from the given input image as shown in the 'figure 2'. Now convolutional layer which extracts specific information from the image in the form of matrix which is mostly in the shape 3 by 3 matrix. Then this convolutional layer is attached to the pooling layer which again extracts the data from convolutional layer and has the size of 2 by 2 matrix. Now these features are then connected to fully connected layers, this is a way to process information and make decisions by letting every part of the network share its knowledge and collaborate to understand the big picture. And then we check if we get our desired output. So fully connected layers and output layer is the classification part of the CNN.

This was the basic construction of the CNN model. Note that we can have multiple convolutional and pooling layers in our model and multiple hidden layers depending on the nature of the dataset and what we expect from our model as shown in the 'figure3'.

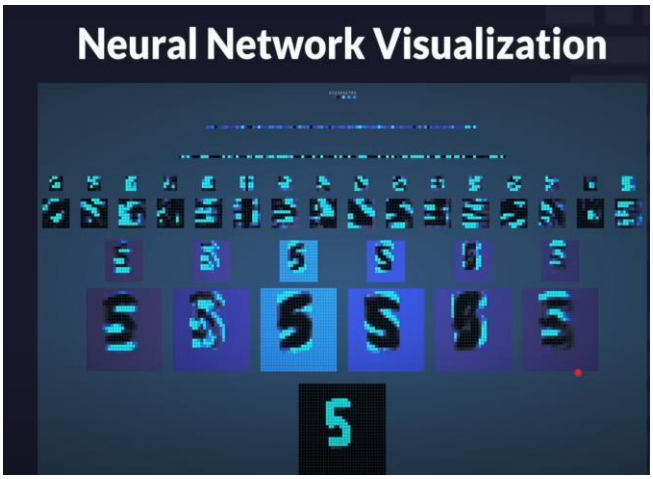


Figure 3: multilayer CNN

1.1. Dataset: 100-bird-species

The data set I used are the images of birds with 100 different species, designed for image classification tasks. It contains thousands of bird images in various dimensions. First of all, I gave the paths of my directories (training, testing, validation) to my code then created a dictionary where key is class names and values are the list of file paths which are corresponding to each class after that I created a variable which stores total number of unique classes in it, Then I mapped classes names to its numeric variable as computer requires the numeric label not class label. Then I selected 20 classes with highest frequency for a visual representation of my data.

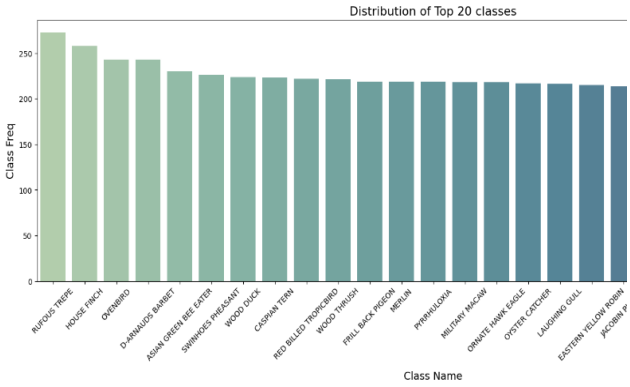


Figure 4: top 20 classes

My approach was just to make sure that my data is well-structured and explore this dataset before proceeding with model training and evaluation. It ensures that data is organized, labeled appropriately, and ready for use in machine learning models. Additionally, visualizing the distribution of classes can help me to identify any class imbalances that may need to be addressed during model training.

1.2. Convolution

In introduction we might get a rough idea about convolutional layer. Here, we will discuss how it extracts the data from the input layers. We now know that the images are nothing but a 2dmatrix. The below figure shows the basic math behind the convolution layer. It has a filter generally of 3 by 3 identity matrix; this filter is multiplied by input to give us a new matrix of 3 by 3. This new matrix is then summed to get a single number. Then the filter again multiplies with the next numbers of input. Thus, this process is repeated till each of the input feature is extracted.

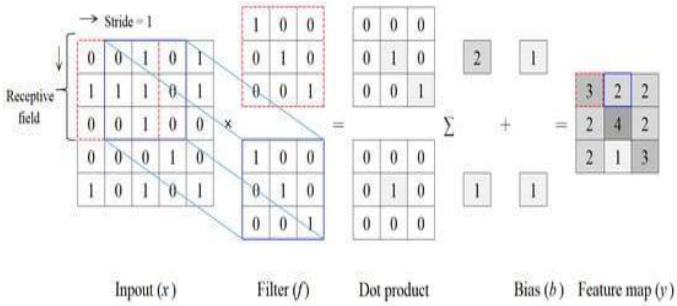


Figure 5: convolutional layer mathematics

1.3. Pooling

After convolutional layer pooling layer is introduced, pooling layers first divides input in 2 by 2 matrices and then from each matrix it pools the important features of the input data. This can be done in two types:

1.3.1 Max Pooling:

Max Pooling will focus on larger parameters of input matrix and will only extract large parameters from the input matrix.

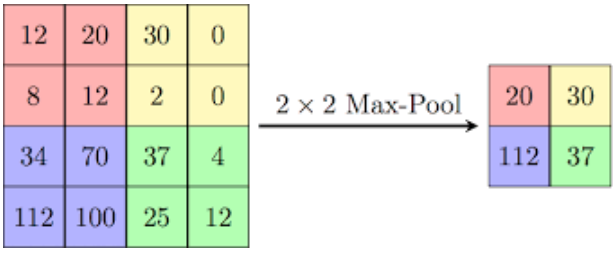
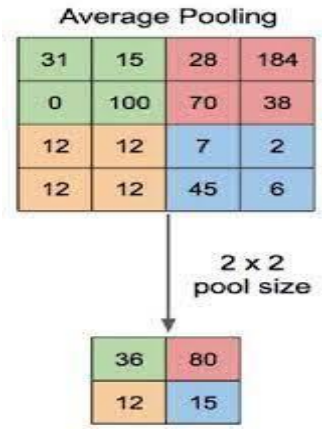


Figure 6: max pooling

1.3.2 Average pooling:

Average pooling will calculate the average of input matrix and extract the average of all features.



1.4. Loss Function:

Loss functions are used to optimize the model during training. I used Cross-Entropy Loss It measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-Entropy Loss quantifies the dissimilarity between the predicted probability distribution (predicted by the model) and the true probability distribution (the actual labels). The Cross-Entropy Loss looks at how well the model's guess (70% for cat) matches the real answer ("cat"). If the model's guess and the real answer match perfectly, the loss is very low. If they are completely wrong, the loss is high. It's like measuring the difference between what the model thinks and what's true.

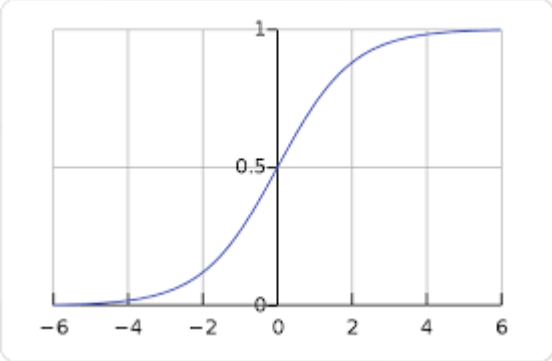


Figure 7: cross entropy loss graph

1.5. Activation function(ReLu):

The Rectified Linear Unit (ReLU) activation function is a simple, yet highly effective activation function used in neural networks, including our model Convolutional Neural Networks (CNNs) such as ResNet and VGG16. I have used ReLu activation function in both of my architectures. ReLU works as, for any input value x, if x is greater than or equal to zero, it returns x unchanged. If x is less than zero, it transforms it to zero. Which means it makes all the x negative values zero.

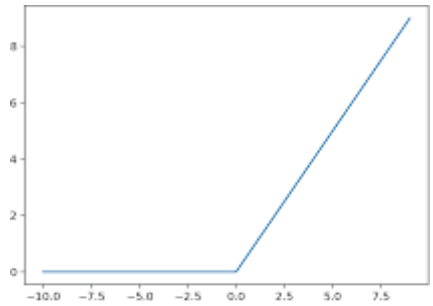


Figure 8: ReLu

1.6. Optimizer:

Optimizers are important when it comes to error handling. Basically, optimizer's goal is to reduce error landscape occurred when there is difference between expected value and output value in our model. I used Stochastic Gradient Descent (SGD) which reduces the error slope by taking small steps towards the value zero. This step is determined by the nature of our model and data set.

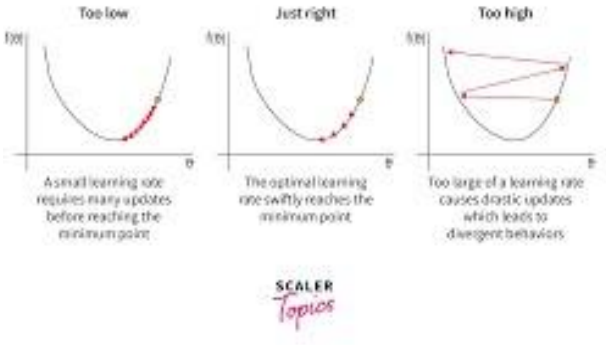


Figure 9: Stochastic Gradient Descent (SGD)

1.7. VGG 16 Model Implementation:

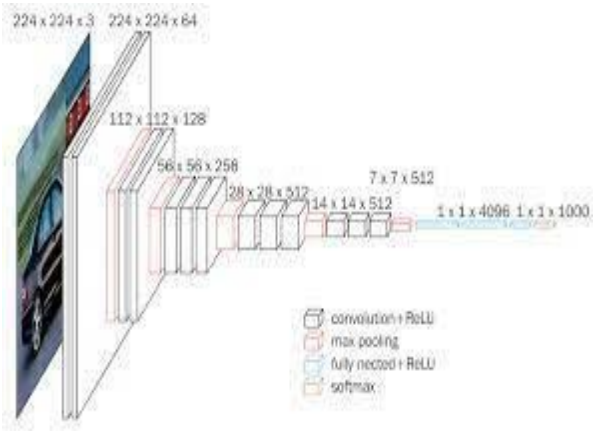


Figure 10:VGG16 model

VGG16 is a very simple but effective architecture used in image classification. VGG stands for Visual Geometry Group and 16 stands for 16 convolutional layers. To implement VGG16 in my programming I used pytorch deep learning framework.

1.8. ResNet-152:

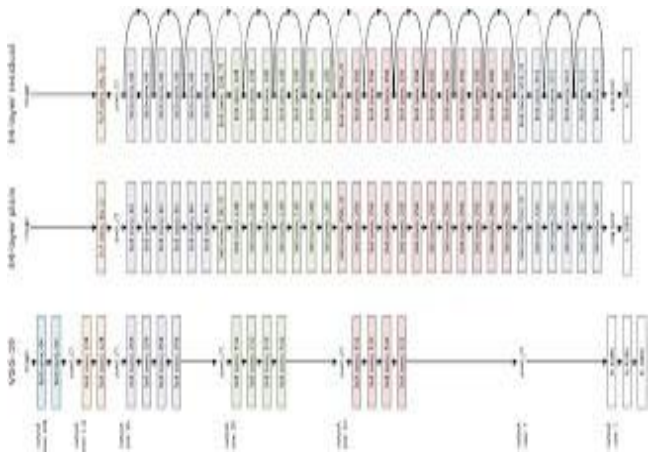


Figure 11:ResNet-152

ResNet is a type of deep convolutional neural network (CNN) architecture used in image classification and other computer vision tasks. What makes ResNet unique is its use of residual blocks, which contain skip connections or shortcuts that allow the network to skip over certain layers during training. These skip connections help combat the vanishing gradient problem, making it possible to train very deep neural networks effectively. ResNet enables the training of extremely deep neural networks, which can capture intricate features in images and achieve high accuracy in tasks like image recognition. It has been a breakthrough in the field of deep learning and is widely used in various computer vision applications. ResNet models come in different depths, such as ResNet-50, ResNet-101, and ResNet-152, with a varying number of layers, allowing you to choose the right model for your specific task.

1.9. Experimental Analysis

Firstly, for VGG16 I set epochs to 10, after each epoch, I performed a validation loop to evaluate the model's performance on the validation dataset. This helps prevent overfitting and allows us to monitor how well the model generalizes. After training and validating, I likely tested the model on my test dataset to assess its performance on unseen data.

After that I included code for visualizing and analyzing the training and validation performance, which may include plotting loss and accuracy which I will show in my experimental analysis. fine-tuning was done for specific dataset by adjusting the classifier layers and training it on bird species dataset.

First, I imported some libraries like pytorch, torchvision, and other common Python libraries. After this defining configuration was really important because my data set was about 2gb and my laptop is not that fast to train this huge data in couple of ours, so I used GPU’s which helped me in training my data faster. Later I loaded bird species dataset, which includes training, validation, and test sets. Applied data transformations such as resizing, data augmentation, normalization, and batching using PyTorch's DataLoader and transforms.

I created instances of the VGG16 model using PyTorch's torchvision.models module. This model is pre-defined in PyTorch and is available for use. By setting the pretrained argument to True, I loaded pre-trained weights for VGG16.

In VGG16, the classifier consists of fully connected layers. I modified the classifier by changing the output size of the final fully connected layer to match the number

of classes in bird species dataset. This step is crucial for adapting the model to our specific classification task.

Then I implemented a training loop that includes iterations over my training dataset, forward and backward passes, and parameter updates. Then calculated the loss and tracked accuracy during training.

Secondly, for ResNet-152. I imported the required libraries, including PyTorch, torchvision, and other utility libraries.

Then I loaded and preprocessed your bird dataset. This included setting up data transformations, splitting the dataset into training, validation, and test sets, and creating data loaders for each of these sets.

Loaded the pretrained ResNet-152 model from torchvision.models. This model has already been trained on a large dataset (e.g., ImageNet) and can be fine-tuned for your specific task.

Then I modified the model to suit our classification task. This involved changing the number of output units in the final fully connected layer to match the number of bird species classes in your dataset.

Then implementation of training loop where I iterated over the training dataset, performed forward and backward passes, updated the model's weights, and calculated training loss and accuracy. I repeated this process for multiple epochs.

Similarly, I implemented a validation loop where you used the validation dataset to evaluate the model's performance and calculate validation loss and accuracy. Throughout the training and validation loops, you printed and collected training and validation loss values and accuracies for ResNet-152.

1.10. Visual analysis and comparison of VGG16 and ResNet-152:

```
Epoch [0/10] Batch [0/2645]
Train Loss (ResNet-152): 11.708264350891113
Train Loss (VGG16): 20.122209548950195
Epoch [0/10] Batch [500/2645]
Train Loss (ResNet-152): 5.738012506576355
Train Loss (VGG16): 6.681730448366877
Epoch [0/10] Batch [1000/2645]
Train Loss (ResNet-152): 4.230944102698868
Train Loss (VGG16): 6.55587874592601
Epoch [0/10] Batch [1500/2645]
Train Loss (ResNet-152): 3.348139261500507
Train Loss (VGG16): 6.42818740111522
Epoch [0/10] Batch [2000/2645]
Train Loss (ResNet-152): 2.797504099546701
Train Loss (VGG16): 6.24538603453324
Epoch [0/10] Batch [2500/2645]
Train Loss (ResNet-152): 2.4198725321372954
Train Loss (VGG16): 6.018053512104222
Epoch [1/10] Batch [0/2645]
Train Loss (ResNet-152): 0.39102932810783386
Train Loss (VGG16): 4.537906646728516
Epoch [1/10] Batch [500/2645]
Train Loss (ResNet-152): 0.5524505460095739
Train Loss (VGG16): 4.177401505068628
Epoch [1/10] Batch [1000/2645]
Train Loss (ResNet-152): 0.5386601380118123
Train Loss (VGG16): 3.841858174060132
Epoch [1/10] Batch [1500/2645]
Train Loss (ResNet-152): 0.526451242220235
Train Loss (VGG16): 3.546113299656359
Epoch [1/10] Batch [2000/2645]
Train Loss (ResNet-152): 0.5194390743769806
Train Loss (VGG16): 3.281945514297676
Epoch [1/10] Batch [2500/2645]
```

Figure 12: training analyses

The above figure gives us a detailed information about train loss of both of the architectures VGG 16 and ResNet-152. We can see that loss function of ResNet-152 started from11.708 and till the end it became 0.013.

On the other hand, loss function of VGG 16 started from 20.122 and till the end it became 0.236.

Observing both architectures, both were good in gradually decreasing the loss. However, this was a pretrained models which I used in my coding and ResNet-152 was pretrained on a large deep learning model and so its performance is good when compared with VGG 16.

My coding link is here - github code

1.10.1 Accuracy and loss graph of validation data of both architectures:

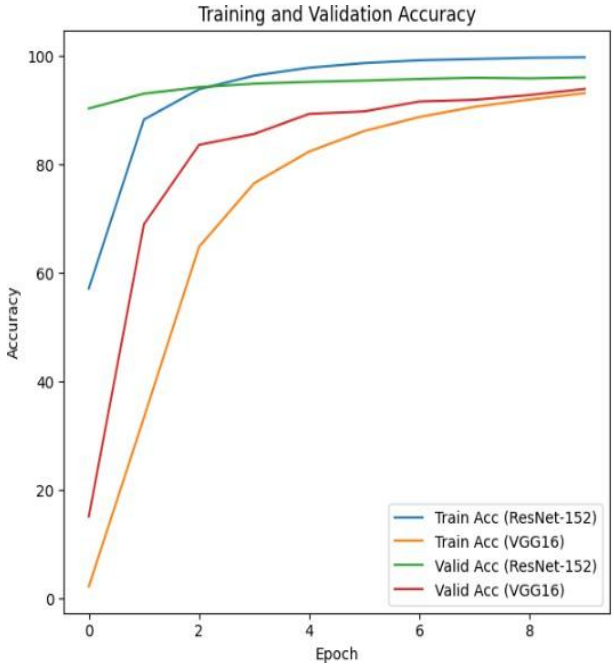


Figure 13: training and validation accuracy

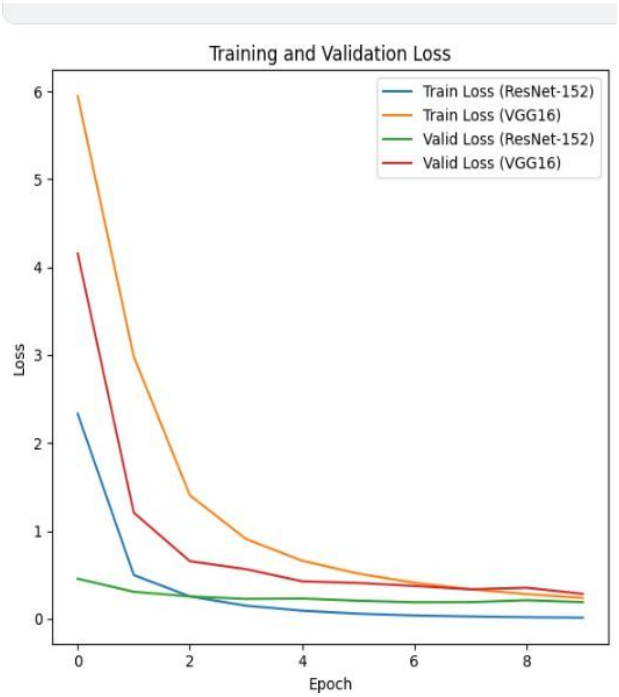


Figure 14: Training and validation loss

These are my final graphs obtained from the coding I have done. The first graph shows the accuracy of both the models and the second graph shows the validation loss of both the models.

Final Validation Accuracy (ResNet-152): 96.07619047619048 %
Final Validation Accuracy (VGG16): 93.94285714285714 %

There is not a big difference in these two architectures but the reason why ResNet-152 has more accuracy is because ResNet-152 is a much deeper neural network than VGG-16. It has 152 layers compared to VGG-16's 16 layers. Deeper networks can capture more complex patterns and features in the data. ResNet introduces residual connections, which allow the network to learn residual functions. These connections help mitigate the vanishing gradient problem during training, making it easier for the network to learn and optimize its parameters effectively. Depending on how the models are fine-tuned and pretrained, ResNet-152 might have been pretrained on larger or more diverse datasets, which could improve its generalization to your specific task.

1.11. Conclusion

This paper was just to share a comparison between two architectures of CNN which helps in image classification and to share the basics of CNN. Now I think that both of the architectures are good at their places because VGG 16 is simple but effective architecture as it has 93 % accuracy. On other hand, ResNet-152 is complex and trained on large deep learning data sets compared to VGG 16. So, it was expected that ResNet-152 should overperform VGG 16. Mostly, the selection of the models depends on the nature of the data set we have and the expectations we keep towards our algorithm. There was a chance of development for me too when I was coding, as I could have increased the epochs or changed activation function through which I could have understood the best suiting parameters. But the problem was my data set was huge and despite using GPU it took like 3 to 4 hours to train the model. Lastly, I am satisfied with both of the models and it was worth to compare them in this project as I understood many different things about architectures and when and which models to be selected.

1.12. References

- Arzmi, MH, P. P. Abdul Majeed, A, Muazu Musa, R, Mohd Razman, MA, Gan, H-S, Mohd Khairuddin, I & Ab. Nasir, AF 2023, Deep Learning in Cancer Diagnostics A Feature-based Transfer Learning Evaluation, 1st ed. 2023., Springer Nature Singapore, Singapore.
- huggingface.co. (2023). microsoft/resnet-152 · Hugging Face. [online] Available at: https://huggingface.co/microsoft/resnet-152 [Accessed 1 Nov. 2023].
- CNN special report. Iconic Singapore 2019, Cable News Network CNN, Atlanta, GA.
- www.mathworks.com. (n.d.). VGG-16 convolutional neural network - MATLAB vgg16. [online] Available at: https://www.mathworks.com/help/deeplearning/ref/vgg16.html.