**Improving CNN Results for Computer Vision Applications: A Comparison of VGGNet and Other Architectures**

Briston Domino

Department of Computer Science, The College of St. Scholastica

CIS 3115 Sec 500: Machine Learning Data Science

Professor: Tom Gibbons

May 12, 2023

**Abstract**

This research aims to enhance image classification performance in Convolutional Neural Networks (CNN) by comparing various architectures, including LeNet, AlexNet, VGGNet, and MobileNets, using the Caltech dataset. The primary focus is on the VGGNet architecture, renowned for its computational efficiency and broad application potential in computer vision. The research also emphasizes migrating the Project from the Learning Notebook 7 to a Python IDE (PyCharm) to leverage better coding and debugging tools. This study results in improved image classification, with the modified VGGNet model for the Caltech 20 dataset, highlighting enhanced performance over the Learning Notebook 7. The results offer valuable insights into the potential of deeper CNN architectures like VGGNet in computer vision applications. However, the research acknowledges the need for further improvements and extended training duration for more accurate results.

**Improving CNN Results for Computer Vision Applications: A Comparison of VGGNet and Other Architectures**

# Introduction

This paper investigates the performance of various convolutional neural network (CNN) architectures, including LeNet, AlexNet, VGGNet, and MobileNets, in computing vision tasks. I aim to improve upon the results from a previous project (Learning Notebook Practice 7) using VGGNet, a deeper CNN architecture. The study used the Caltech dataset to compare the VGGNet model's accuracy with the last Project's model. Results show that VGGNet's architecture offers better performance and has the potential for further application in computer vision. I will go through the process of the project code and the results. I will also talk about running this Project on my computer and the hardware I used.

# Literature Review

# Let us first talk about the different CNN architectures, including LeNet, AlexNet, VGGNet, and Mobile Nets. Also, I will be talking about the Caltech dataset. VGGNet is, developed by researchers at Oxford University, is a 16-layer convolutional neural network with up to ninety-five million apartments, trained on over one billion images (VGGNet: A Computer Guide, 2021). Despite its large filters and extensive data requirements, VGGNet remains computationally efficient and is a strong baseline for various computer vision applications. The VGGNet's deep feature representations are incorporated into other neural network architectures, such as SSD and YOLO (Different Types of CNN Architectures, 2021).

LeNet is the first CNN architecture designed for handwritten digit recognition and was developed in 1998 by Yann LeCun and colleagues. Although it initially struggled with the vanishing gradients problem, introducing max-pooling between convolutional layers reduced the spatial size of images, preventing overfitting and enabling more effective training. LeNet's architecture remains relevant today and is used in various applications, including traffic sign reconstruction and face detection (Different Types of CNN Architectures, 2021).

AlexNet was developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton, popularizing deep learning and CNN architectures. It won the ImageNet Large Scale Visual Recognition Challenge in 2012 and was designed for large-scale image datasets. With five convolutional layers, max-pooling layers, three fully connected layers, and two dropout layers, AlexNet has approximately sixty million parameters and utilizes the ReLU activation function (Different Types of CNN Architectures, 2021).

MobileNet, developed by Andrew G. Trillion and others, are compact CNNs designed for mobile devices with low-latency image classification and object detection. Their small size allows for real-time operation on embedded devices like smartphones and drones. The flexible architecture supports 100-300 layers of CNNs, often outperforming larger architectures like VGGNet. Google's mobile Vision API, integrated into Android phones to automatically identify popular objects in images, is a real-life application of MobileNets (Different Types of CNN Architectures, 2021).

I will use the Caltech dataset for the training aspect of my machine-learning Project. The Caltech datasets, also known as Caltech 101 and Caltech 256, are popular benchmark datasets for computer vision research. It was created by researchers at the California Institute of Technology (Caltech), consisting of labeled images collected from the web. Caltech 101 was introduced in 2003 and contained 9,144 images divided into 101 categories, including objects, animals, and scenes. The number of images per category varies, with about 40 to 800 images in each. The Caltech 101 dataset was designed to evaluate object recognition algorithms and is suitable for tasks such as image classification, object detection, and segmentation (Li, Andreeto, Ranzato, & Perona, 2022). Caltech 256 is the extension to Caltech 101, and the Caltech 256 datasets were introduced in 2007. It contains 30,607 images divided into 256 object categories, with a minimum of eighty images per category. The dataset is more challenging than Caltech 101 due to the increased number of categories and the more significant variability in object appearance (Griffin, G., Holub, A., & Perona, P. (2022). The primary goal of this dataset is to provide a standardized set of images for testing and evaluating various computer vision algorithms, particularly conventional neural networks (CNNs). While the Caltech datasets have contributed to the development of many computer vision models, they have some limitations, such as the relatively small number of images per category and the inherent biases in the pictures.

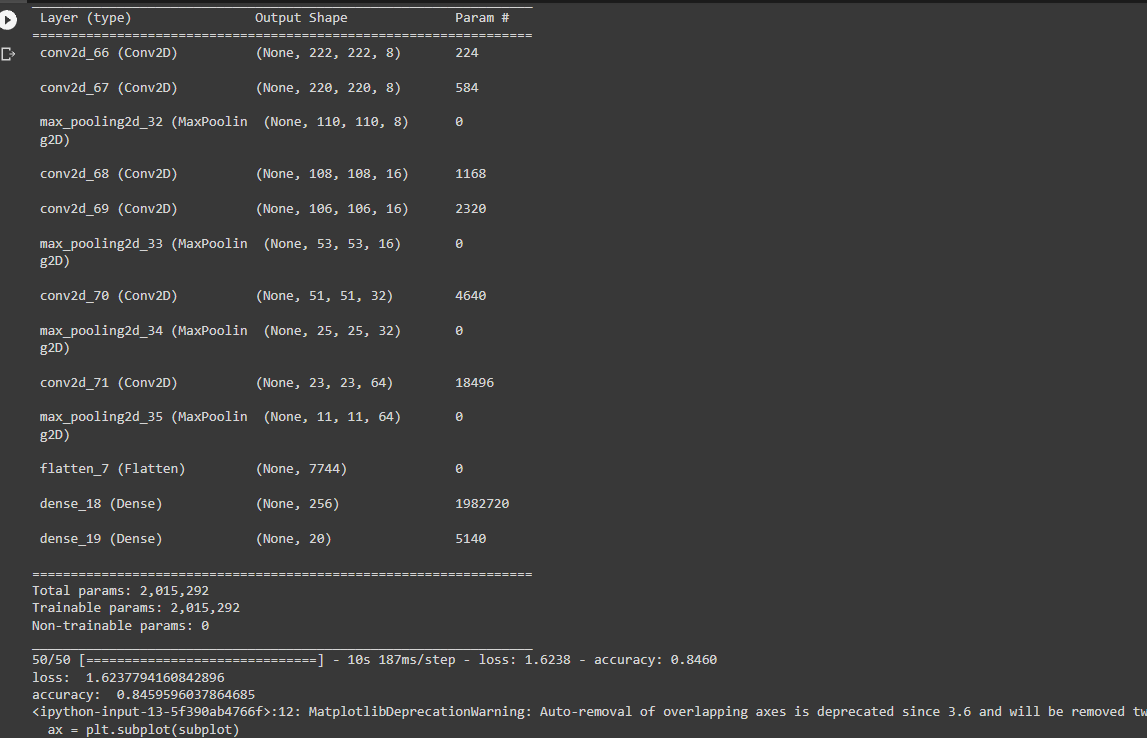
Keras Convolutional Neural Networks (CNN) are neural networks that utilize convolution, a mathematical operation performed on matrices, to process and classify images. These networks consist of various layers, including Conv1D, Conv2D, Conv3D, and other specialized layers. In Keras CNN, input images are passed through convolutional layers containing filters (matrices containing numbers) to produce the output images. The convolution process involves overlaying the fitter at a specific location on the input images, performing element-wise multiplication of the filter matrix and corresponding image values, and summing the products to create the output image. Keras CNNs are often used for image classification tasks, such as classifying images in the CIFAR-10 dataset. To use Kears CNN, import the necessary libraries and classes, prepare the raw input data, verify the dataset, create a convolutional base using Conv2D and MaxPooling2D layers, add a dense layer on top for classification, compile and train the model, and finally evaluate the model's performance, Kears CNN's are particularly useful for image classification tasks due to their ability to effectively learn and extract features from images (*Keras CNN: Learn how to use Keras CNN with examples?* 2023).

# Results

I will be using these architectures and the Caltech database for this Project. I will start by telling you the process it took for this Project. The first thing I had to do was to move the learning Notebook 7 over to a Python IDE called PyCharm. I did this because the IDE was easier to code and had many debugging tools built into the IDE to give visuals if there were errors in the code right away. It was also straightforward to change something quickly, then rerun the code, and there are all sorts of other features and add-ons packed into one application to aid in smooth, fast coding development for this Project. The first thing to do to move it to PyCharm was to copy the code from Learning Notebook 7. I spelled the code into seven different Scripts in PyCharm, each doing something different. Then the next step I took was to get the Caltech dataset from Caltech's website. After that, I split the dataset for training and testing. Then I specified the new test and training data locations by saving them into weights in the project folder.

To run this Project on my computer, I first had to install Windows Subsystem for Linux (WSL). Then in the terminal in Windows, I installed all the required drivers to get It running. These drivers include the CUDA driver for the GPU for training, Nvidia Nsight Compute, Nvidia Visual Profiler, TensorFlow, Python 3, and ImageMagick. So, what do these drivers do? Let us first start with the NVIDIA Nsight Compute. The NVIDIA Nsight Compute is a powerful profiling tool that helps optimize the performance of GPU-accelerated machine learning applications by providing detailed insights into kernel execution and resource usage. Next is Nvidia Visual Profiler, a comprehensive performance analysis tool that allows developers to visualize and optimize the execution of GPU-accelerated machine learning applications by identifying performance bottlenecks and resource usage issues. Lastly, ImageMagick is a versatile image manipulation tool that can be used for preprocessing tasks, such as resizing, cropping, and transforming images, to prepare them for use as input data in machine-learning CNN models. After all of these drivers, I was able to run this on my computer using the terminal. I saw a terrible performance on the neural network from Learning Notebook 7. So, I tried to remove the clutter directory to see if it would improve performance, but the performance still remained poor.

I ran the machine learning project on my desktop computer for testing and training. I had to run this on my computer because it took way too long on the free Colab Google Server and kept running out of GPUs to use, making it even longer. My specs on my computer are for the CPU; I have an AMD Ryzen 7 7700x processor. For GPU, I have an Nvidia GeForce RTX 3060ti, a 1TB NVMe SSD making training and testing way faster, and I have 32GB of DDR5 RAM. Even with my specs being way better than the Google free Server, it still took about 5 1⁄2 hours to train. One big reason it took this long is that it has over 134,719,296 total parameters! I have predicted how long it will take to run on the Google free server, and it will take about 10 hours or more to train with the same parameters for all of the CNN architectures!

Before discussing my results, let us first discuss the results from Learning Notebook 7. The highest results are 0.1800 for Training accuracy and 0.2562 for the highest testing accuracy, which could be better. Ok, so that is Learning Notebook 7; Now let us talk about my results, and they are for the Baseline CNN for Caltech 20 accuracy is 0.84595 with a loss of 1.6237. Then, for Baseline CNN Caltech 256, the accuracy is 0.25806, and the loss is 4.5579. The score for the VGG16 for Caltech 20 for accuracy is 0.08585, and the loss is about three. Lastly, the best accuracy for VGG16 for Caltech 256 is 0.02688, and the loss is about 5.4. So, as you can see, the best result is the CNN for Caltech 20. This one is the modified version of the one in Learning Notebook 7. If I had more time, I would train it more to increase the training accuracy score. Below are all of its Parameters, Conv2D, MaxPooling2D, etc. From both Learning Notebook 7 and my modified version project.   


(My Modified Version)

# (From Learning Notebook 7)

# Conclusion

In conclusion, this Project offered valuable insights into the world of Convolutional Neural Networks (CNN) and their potential in computer vision applications. Despite the challenges encountered, such as transitioning from Learning Notebook 7 to PyCharm, coding in Python, and optimizing the model's parameters to achieve higher accuracy, the Project demonstrated the importance of exploring deeper CNN architectures. Although the current mode's accuracy did not fully meet my expectations, the modified version from Learning Notebook 7 Baseline CNN from Caltech 20 still outperformed the original. With more time and resources, further improvements can likely be made, allowing for the development of even more accurate and efficient models. Additionally, future work could explore other CNN architectures or data augmentation techniques to enhance the image classification performance further. Overall, this Project not only highlights the power of deep learning in computer vision but also underscores the importance of continuous learning and experimentation in the rapidly evolving field of machine learning.

# 

# References

Kumar, A. (2022, April 12). *Different types of CNN Architectures explained: Examples*. Data   
Analytics. <https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/>

Nepal, P. (2020, December 8). *VGGNet architecture explained*. Medium. <https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6>

VARSHNEY, P. (2020, July 28). *VGGNet-16 architecture: A complete guide*. Kaggle. <https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide>

Griffin, G., Holub, A., & Perona, P. (2022). Caltech 256 (1.0) [Data set]. CaltechDATA. <https://doi.org/10.22002/D1.20087>

Li, F.-F., Andreeto, M., Ranzato, M., & Perona, P. (2022). Caltech 101 (1.0) [Data set]. CaltechDATA. <https://doi.org/10.22002/D1.20086>

Educba. (2023, March 15). *Keras CNN: Learn how to use Keras CNN with examples.* EDUCBA. <https://www.educba.com/keras-cnn/>

Link to notebook: <https://colab.research.google.com/drive/1ninBPm9vtmbaK7XIVBUf0gC1rBhTTx_i?usp=sharing>