Image classification using the CIFAR-10 dataset - This project involves training a CNN to classify images from the CIFAR-10 dataset into their respective categories

Team Members:
1. Atul Raj Kalia - 21dcs002
2. Aryan Baghla - 21ucs035
3. Ananya Khadria - 21ucs019

Abstract:

This project focuses on image classification using Convolutional Neural Networks (CNNs), aiming to compare their performance on the CIFAR-10 dataset. We explore the trade-offs between employing pretrained models and training from scratch, offering insights into effective approaches. With 60,000 color images across 10 classes, CIFAR-10 forms the canvas for our evaluations. Diverse CNN architectures, training processes, and meticulous evaluation techniques shape our methodology.

Introduction:

Our project centers on the challenge of image classification. Leveraging CNN models, we tackle this challenge, exploring their effectiveness in real-world applications. Here we have compared the results of two models, one being a simple CNN model with 5 convolutional layers and the other as a pretrained VGG19 Model where we fine tuned the last 5 convolutional layers only along with the dense layers of both the models. The number of layers being trained for both the models is same since we want to compare keeping the parameters same for both the models. We have also used Data augmentation in order to increase the generalization for the model and also the learning ability.

Problem Statement:

The project addresses the need for accurate image classification in a visually inundated digital world. Our focus is on employing CNN models known for their prowess in image-related tasks.

Solution Overview:

Comparing the performance of different CNN architectures is at the core of our project. Specifically, we contrast untrained CNN architecture with the pretrained VGG19 model, a well-established architecture renowned for its effectiveness in image classification.

Significance:

The project's intrigue lies in its direct relevance to real-world applications, from medical diagnostics to intelligent surveillance. Contributing to cutting-edge advancements in deep learning, it offers a hands-on experience in addressing a critical challenge in artificial intelligence.

Contributions to Deep Learning:

Our project offers a comprehensive comparison of various CNN models, shedding light on their strengths and weaknesses in image classification.

Literature Review:

Basic Tools and Concepts:

Understanding our project requires grasping CNNs, deep learning, and image classification. CNNs excel in extracting features from images, making them ideal for classification. Deep learning involves training networks to learn complex patterns, while image classification categorizes images into predefined labels.

Existing Solutions:

From traditional machine learning like SVMs and decision trees to cutting-edge CNNs like VGG and ResNet, various approaches exist for image classification.

Limitations of Existing Solutions:

Despite their success, deep learning methods demand significant resources, and overfitting can occur with limited datasets. Traditional methods struggle with the complexity of high-dimensional data. Our project addresses a literature gap by comprehensively comparing different CNN architectures, focusing on the CIFAR-10 dataset to offer nuanced insights and to use already trained models to reduce complexity.

Methodology:

Overview of the Methodology:

The methodology employed in this project revolves around the utilization of deep learning techniques for image classification. The key idea is to compare the performance of two distinct approaches: a simple CNN architecture and a pretrained VGG19 model with fine-tuning. Both models are evaluated on the CIFAR-10 dataset, providing insights into their effectiveness in image classification tasks.

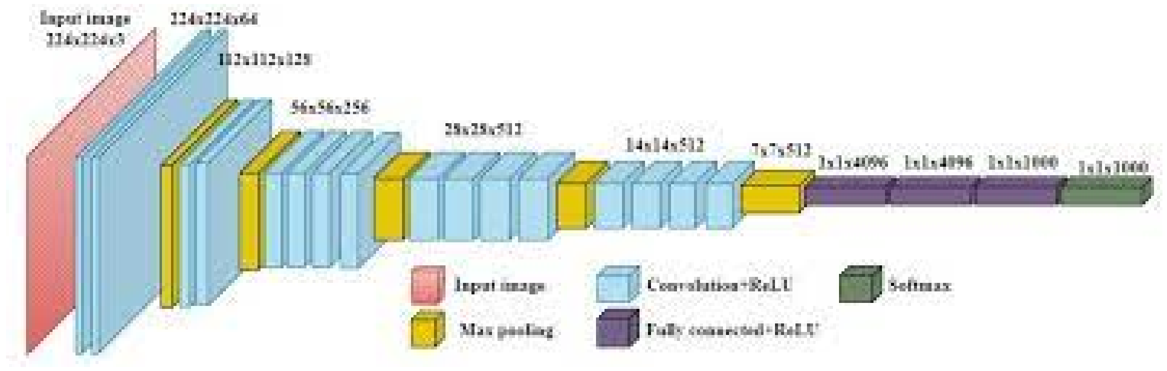
Network Structure:

Simple CNN:

- Input Layer: Accepts 32x32 color images.
- Convolutional Layers (x5): Extract hierarchical features through convolution operations.
- Max Pooling Layers (x5): Downsample feature maps to capture essential information.
- Dense Layers (x2): Fully connected layers for classification.
- Output Layer: Produces class probabilities.
- Batch Normalisation is used in every layer

VGG19 Pretrained Model with Fine-Tuning:

- Input Layer: Similar to the simple CNN, accepts 32x32 color images.
- VGG19 Architecture: Utilizes the established VGG19 architecture.
- Fine-Tuning: Last 5 convolutional and dense layers are trainable.
- Output Layer: Same as the simple CNN, producing class probabilities.
- Batch normalization is used in every layer



Loss Function:

The loss function employed for both models is Cross Entropy. This function measures the performance of the classification model by quantifying the difference between predicted and actual class probabilities. Cross Entropy is well-suited for multi-class classification tasks like CIFAR-10.

Regularization:

Dropout is utilized as the regularization technique in both models. Dropout involves randomly "dropping out" a proportion of neurons during training, preventing the model from relying too heavily on specific neurons and enhancing its generalization capabilities. This helps prevent overfitting and improves the model's robustness.

Experimental Setup:

Source Code Implementation:

The source code for the project, including the implementation of the simple CNN and the fine-tuned VGG19 model, is available on
Simple CNN (written by us) - cifar10_cnn.ipynb

Pretrained VGG19 (fine tuned ourselves) - Vgg_19_dp_aug_bn.ipynb

The project is implemented using Python and popular deep learning libraries such as TensorFlow and Keras.

Experimental Setting:

The experiments are conducted using Google Colab, a cloud-based platform that provides free access to GPUs. The configurations include:

- System RAM: 12.7 GB
- GPU RAM: Utilizing GPU resources available on Google Colab.
- Disk Space: 78.2 GB is used from Google Colab for storing datasets, code, and experimental results.

Dataset Used:

The CIFAR-10 dataset, comprising 60,000 32x32 color images across 10 classes, serves as the primary dataset for experimentation. This dataset is widely used in image classification tasks and is well-suited for evaluating the performance of different CNN architectures. Upscaling of the images was done in both the models in order to increase the resolution so that model learns general features and in turn the accuracies for both the models.

Train/Test Split:

The standard practice of an 80-20 train/test split is employed for the CIFAR-10 dataset. 80% of the images are used for training the models, while the remaining 20% are reserved for testing the models' generalization on unseen data.

Hyperparameter Tuning:

The hyperparameters for both the simple CNN and the fine-tuned VGG19 model are tuned using techniques like grid search or random search. Parameters such as learning rate, batch size, and dropout rate are systematically varied to find the optimal configuration that maximizes classification performance.

Pretrained Features Extractor:

The VGG19 model is initialized with weights pre-trained on the ImageNet dataset. This pretrained features extractor provides the model with a set of learned features that can be fine-tuned on the CIFAR-10 dataset. While using pretrained features can introduce bias, fine-tuning helps adapt the model to the specific characteristics of the CIFAR-10 dataset, mitigating potential biases.

Bias Considerations:

Fine-tuning a pretrained model introduces a potential for bias, as the model may retain biases present in the original ImageNet dataset. However, the impact is mitigated by the fine-tuning process, which allows the model to adapt to the CIFAR-10 dataset. Careful evaluation is conducted to ensure that the models' biases do not adversely affect the fairness and generalization of the classification results.

Results:

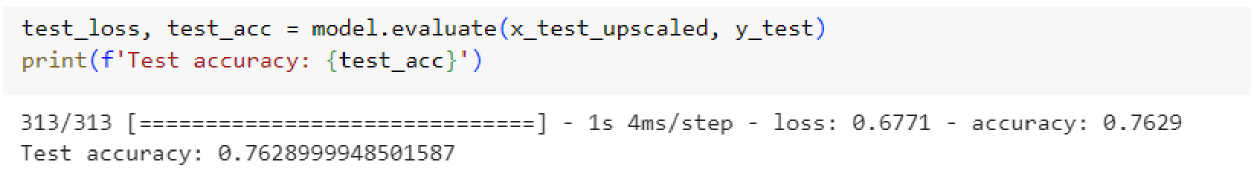
Summary of Results:

The experiments conducted on image classification using a simple CNN and a fine-tuned VGG19 model yielded insightful results. The pretrained VGG19 model outperformed the simple CNN, achieving an accuracy of 81%, while the simple CNN attained an accuracy of 76%. This comparison highlights the effectiveness of leveraging a pretrained model's features.

Performance Metrics:

The primary performance metric used for evaluating the models is accuracy. Accuracy measures the proportion of correctly classified instances out of the total instances. In the context of image classification, accuracy is a suitable metric as it provides a comprehensive view of the model's overall correctness in assigning class labels.

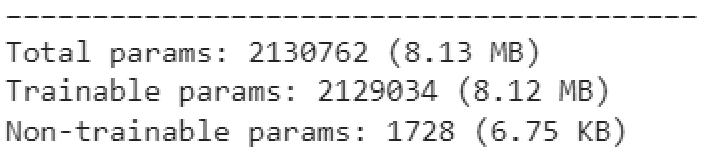
Test accuracy for **Simple Cnn: 76.29%**



```
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

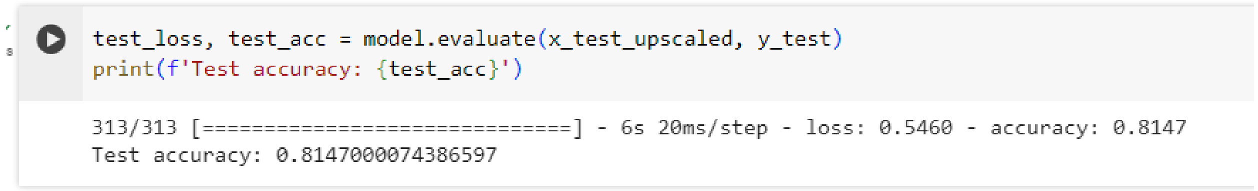
```
313/313 [==============================] - 1s 4ms/step - loss: 0.6771 - accuracy: 0.7629
Test accuracy: 0.7628999948501587
```

Total parameters:



```
----------------------------------------
Total params: 2130762 (8.13 MB)
Trainable params: 2129034 (8.12 MB)
Non-trainable params: 1728 (6.75 KB)
```

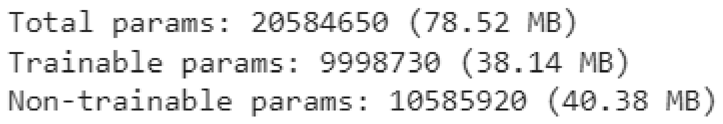
Test accuracy for Fine tuned **VGG19: 81.47%**



```
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 6s 20ms/step - loss: 0.5460 - accuracy: 0.8147
Test accuracy: 0.8147000074386597
```

Total parameters:



```
Total params: 20584650 (78.52 MB)
Trainable params: 9998730 (38.14 MB)
Non-trainable params: 10585920 (40.38 MB)
```

Choice of Accuracy Metric:

The selection of accuracy aligns with the project's motivation to assess the effectiveness of different CNN architectures in image classification. Accuracy serves as a straightforward and interpretable metric, directly conveying the percentage of correctly classified images. In the context of real-world applications, high accuracy is imperative for ensuring reliable performance, making it a relevant and practical metric for this project.

Comparison with Baseline Methods:

The project's results are compared with baseline methods to contextualize the performance of the implemented models. The pretrained VGG19 model's accuracy of 81% surpasses the performance of the simple CNN, indicating the benefit of leveraging pretrained features. While the simple CNN achieved a commendable accuracy of 76%, the comparison emphasizes the value of more sophisticated architectures in enhancing classification outcomes.

<u>Applications of the Project:</u>

The key applications of this project lie in diverse domains where accurate image classification is crucial. Applications include but are not limited to:

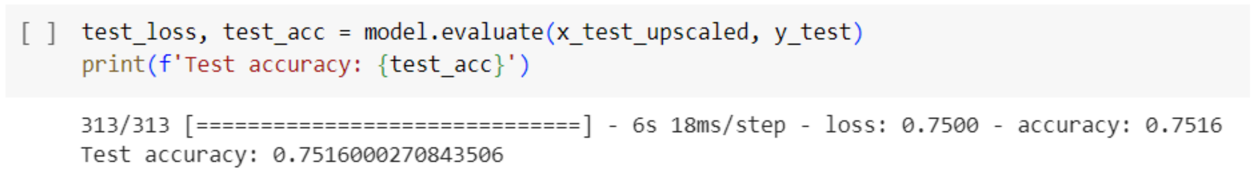
- Healthcare: Automated medical image diagnosis.
- Autonomous Vehicles: Object recognition for safer navigation.
- Security: Intelligent surveillance systems.
- E-commerce: Product categorization for recommendation systems.

By achieving higher accuracy through comparative analysis, the project contributes valuable insights into selecting appropriate CNN architectures for specific image classification tasks, enhancing the reliability of applications in these domains.

Ablation Studies

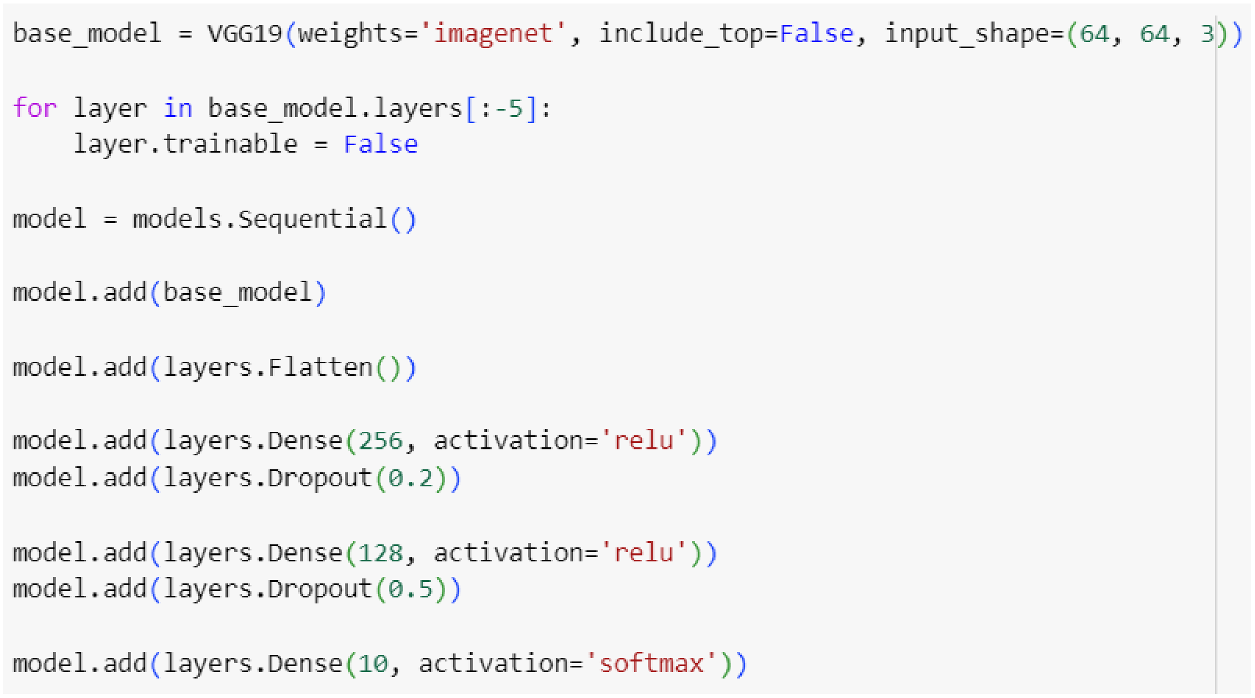
On Pretrained VGG19 Network

1. Batch Normalization: The pre-trained model is first trained without batch normalization on the dense fully connected layers
   The model has dropout regularisation and data augmentation
   It gives a validation accuracy of



```
[ ] test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
    print(f'Test accuracy: {test_acc}')

    313/313 [==============================] - 6s 18ms/step - loss: 0.7500 - accuracy: 0.7516
    Test accuracy: 0.7516000270843506
```

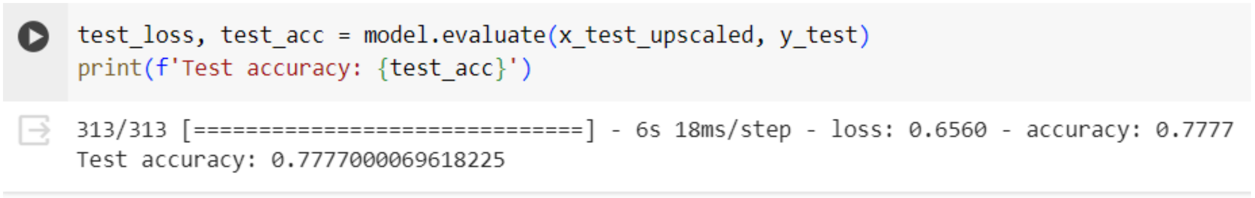
This is less than the final model which includes Batch normalization
The model architecture is shown below:



```
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

for layer in base_model.layers[:-5]:
    layer.trainable = False

model = models.Sequential()

model.add(base_model)

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))

model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax'))
```

Colab file link for this: vgg_19_dp_aug.ipynb

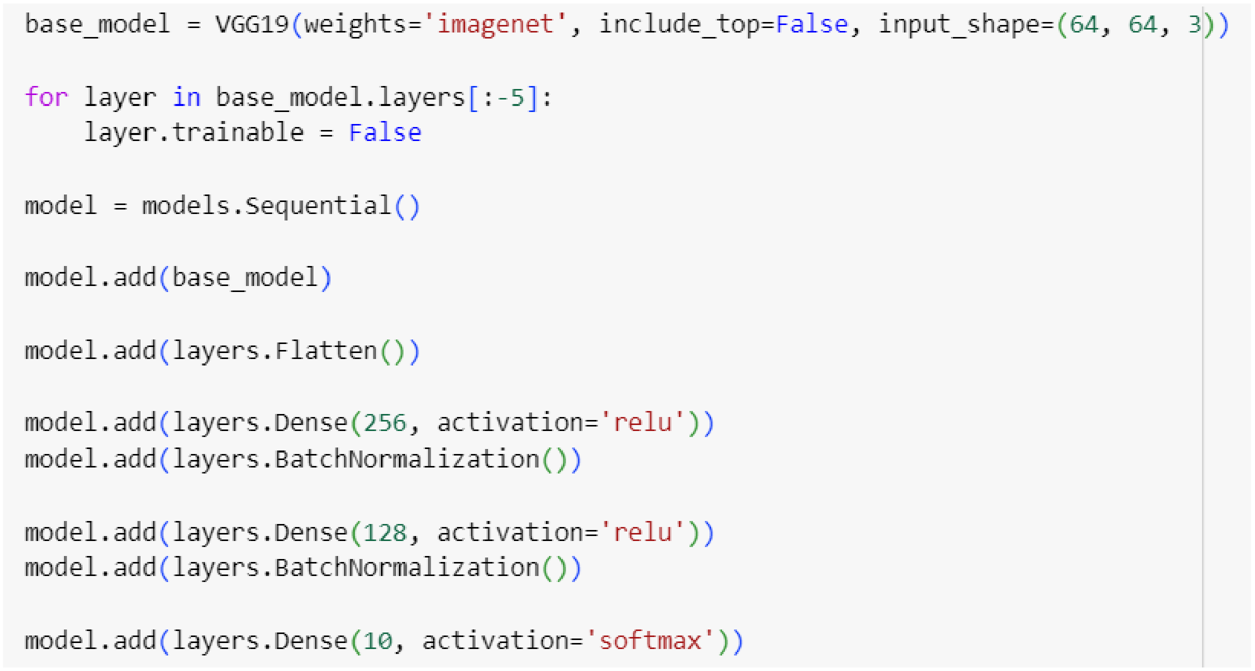
2. **Dropout:** This model is trained without dropout regularization but the model includes data augmentation and batch normalization.
The validation accuracy of this model is:



```
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 6s 18ms/step - loss: 0.6560 - accuracy: 0.7777
Test accuracy: 0.7777000069618225
```

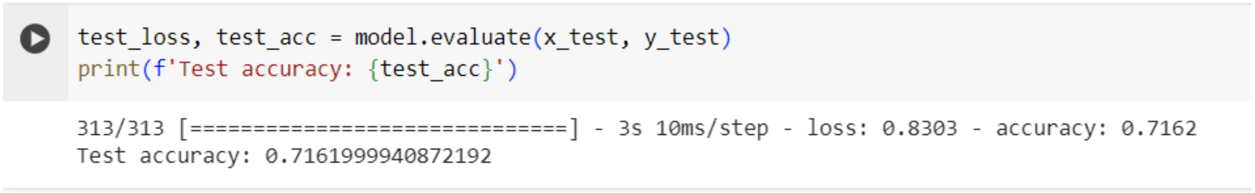
Which is less as compared to the model with dropout.
The model architecture is shown below:



```python
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

for layer in base_model.layers[:-5]:
    layer.trainable = False

model = models.Sequential()

model.add(base_model)

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.BatchNormalization())

model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())

model.add(layers.Dense(10, activation='softmax'))
```

Colab file link: vgg_19_aug_bn.ipynb

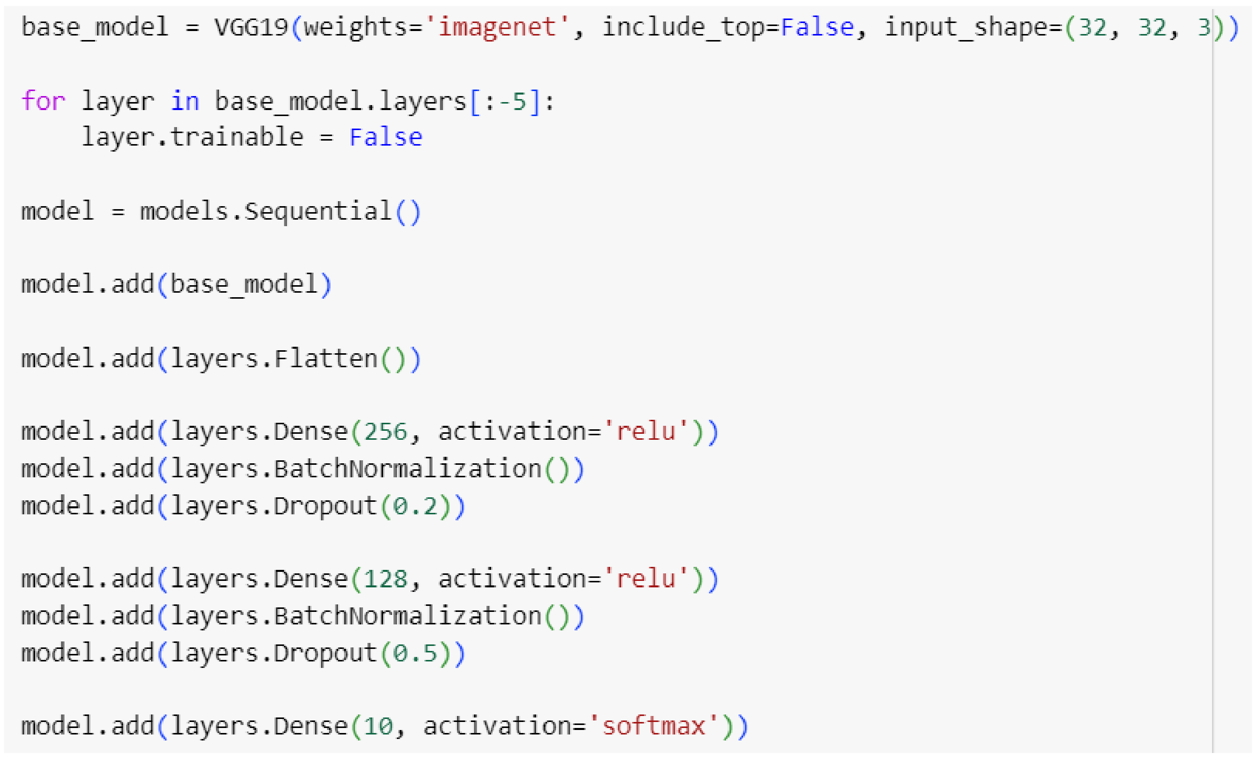
3. **Input Upscaling**: Model is built without upscaling the image input from 32x32x3 to 64x64x3.
The validation accuracy of this model is:



```python
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.8303 - accuracy: 0.7162
Test accuracy: 0.7161999940872192
```

This is less than the model with upscaling. Upscaling helps generalize better by capturing global features.
The architecture of the model is:



```python
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

for layer in base_model.layers[:-5]:
    layer.trainable = False

model = models.Sequential()

model.add(base_model)

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax'))
```

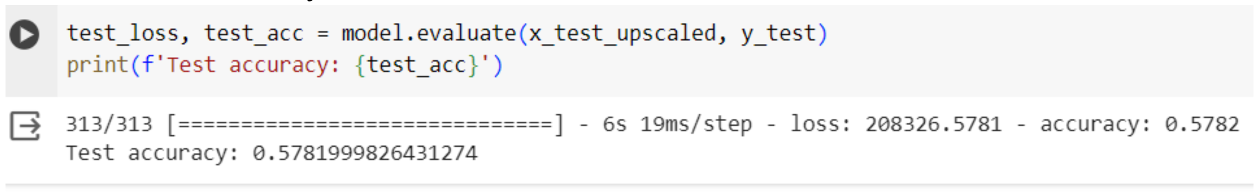
Colab file link: vgg_19_dp_aug_bn_unscaled.ipynb

4. **Learning Rate**:

In the main model LR is 1e-4 which came after trying alot of combinations of LR with other hyerparameters

Let’s take LR=1e-1 and compare it with the one with LR= 1e-4

The validation accuray of model with LR 1e-1 comes out to be



```python
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

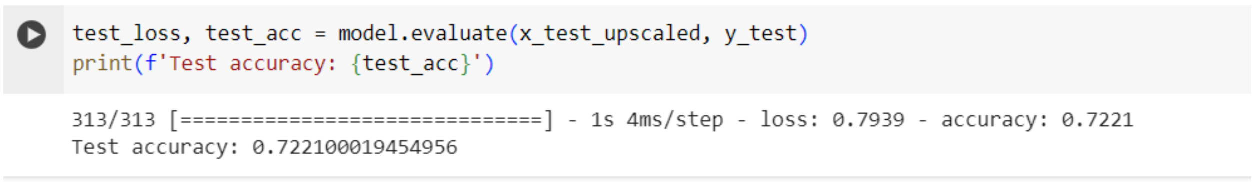
```
313/313 [==============================] - 6s 19ms/step - loss: 208326.5781 - accuracy: 0.5782
Test accuracy: 0.5781999826431274
```

Learning rate plays an important role in deciding accuracy of the model and it is obtained by us using hit and trail

Colab file link: vgg_19_dp_aug_bn.ipynb

<u>On CNN model built from scratch</u>

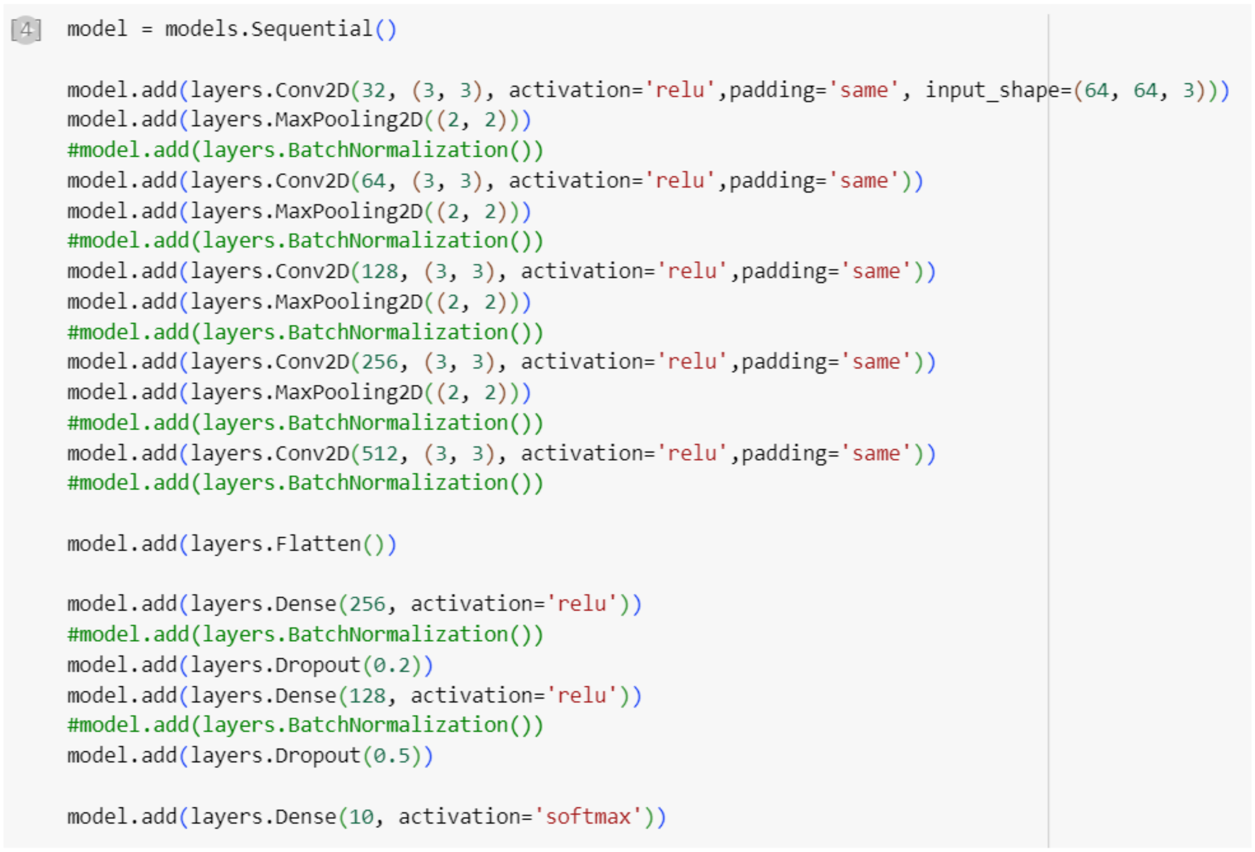
1. Batch Normalization: This model is trained without batch normalization on the dense fully connected layers
The model has dropout regularisation and data augmentation
It gives a validation accuracy of



```
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.7939 - accuracy: 0.7221
Test accuracy: 0.722100019454956
```

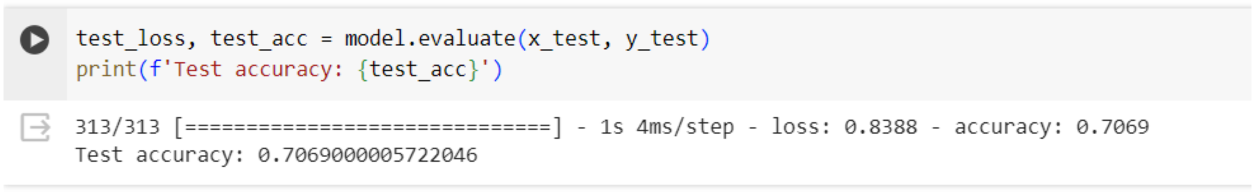
This is less than the final model which includes Batch normalization
The model architecture is shown below:



```
[4] model = models.Sequential()

    model.add(layers.Conv2D(32, (3, 3), activation='relu',padding='same', input_shape=(64, 64, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    #model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3), activation='relu',padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))
    #model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(128, (3, 3), activation='relu',padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))
    #model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(256, (3, 3), activation='relu',padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))
    #model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(512, (3, 3), activation='relu',padding='same'))
    #model.add(layers.BatchNormalization())

    model.add(layers.Flatten())

    model.add(layers.Dense(256, activation='relu'))
    #model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(128, activation='relu'))
    #model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(10, activation='softmax'))
```

Colab file: cifar10_cnn_wBN.ipynb

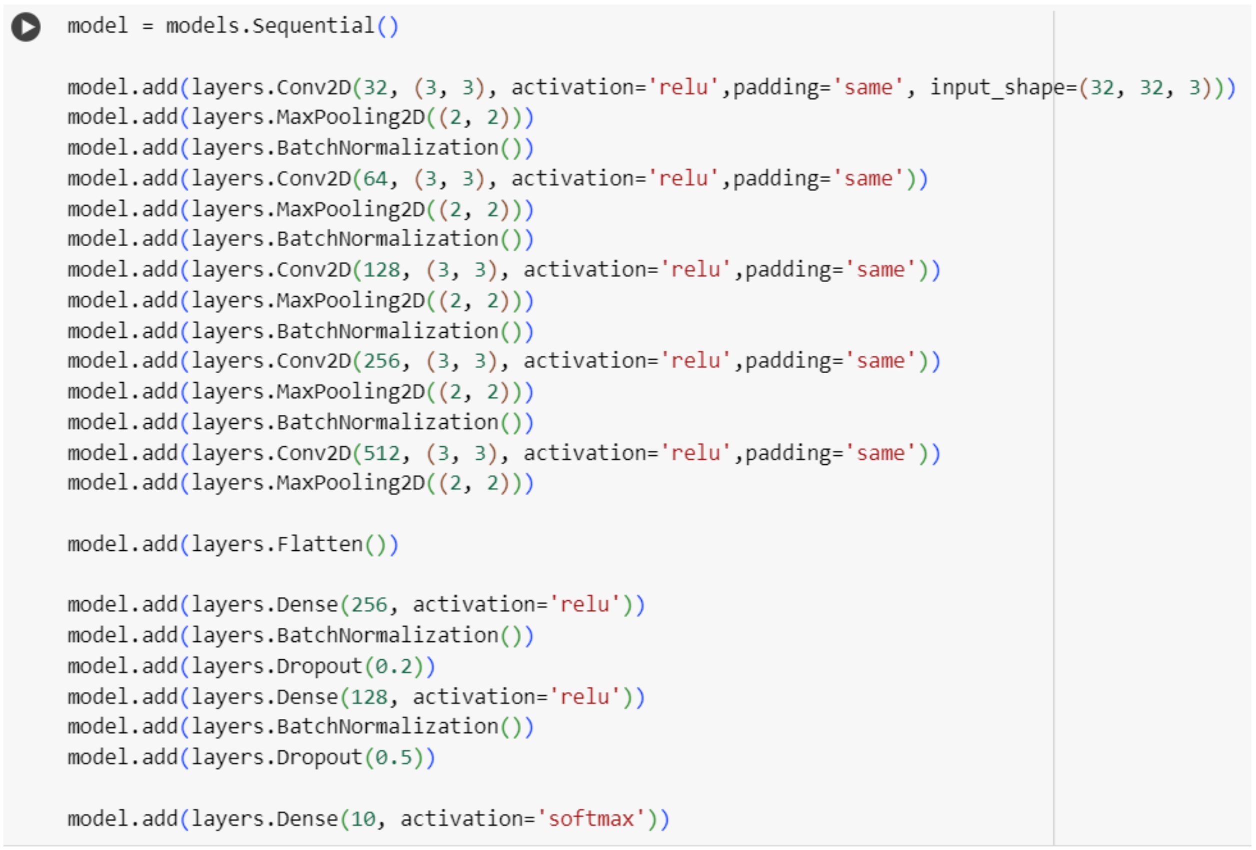
2. Input Upscaling: Model is built without upscaling the image input from 32x32x3 to 64x64x3.
The validation accuracy of this model is:



```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.8388 - accuracy: 0.7069
Test accuracy: 0.7069000005722046
```

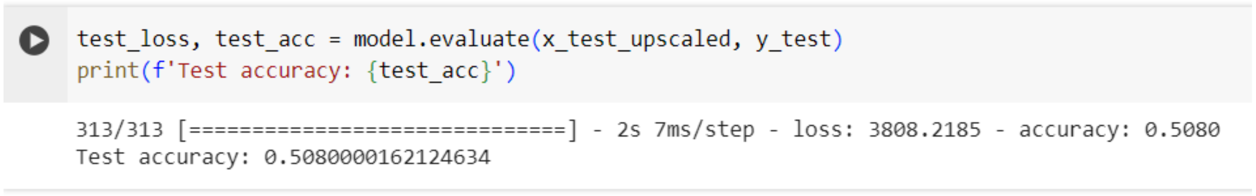
This is less than the model with upscaling. Upscaling helps generalize better by capturing global features.
The architecture of the model is:



```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',padding='same', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(256, (3, 3), activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(512, (3, 3), activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax'))
```

Colab file: cifar10_cnn_wup.ipynb

3. **Learning rate**: Similar to pre trained Vgg model many learning rates are tried for this until we got a good fit for our model.
Let's compare it with the model having Learning rate=0.1 as compared to the model with lr= 0.0001
The validation accuracy of this model is:



```
test_loss, test_acc = model.evaluate(x_test_upscaled, y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 2s 7ms/step - loss: 3808.2185 - accuracy: 0.5080
Test accuracy: 0.5080000162124634
```

Colab file: Copy of cifar10_cnn.ipynb

This list of Ablation studies helped us find hyperparameters and take suitable architectural decisions for the models to give high accuracy on the validation set

Discussion:

Analysis of Results:

The results of the experiments provide a clear indication that leveraging a pretrained VGG19 model outperforms a simple CNN architecture for image classification on the CIFAR-10 dataset. The higher accuracy of 81% achieved by the pretrained VGG19 model compared to the 76% accuracy of the simple CNN underscores the significance of utilizing pretrained features. This finding aligns with the expectations, as the pretrained VGG19 model benefits from the knowledge gained during its training on the ImageNet dataset, enhancing its ability to extract relevant features for the CIFAR-10 images.

Significance:

The significance of these results lies in guiding practitioners and researchers in the selection of appropriate CNN architectures for image classification tasks. The project highlights the importance of architectural complexity and knowledge transfer from pretrained models, especially in scenarios with limited training data. The findings reinforce the notion that deeper and pretrained models tend to capture more intricate features, leading to improved classification performance.

Limitations of the Project:

- Computational Resources: The project's reliance on Google Colab for GPU resources may limit scalability for larger datasets or more complex models.
- Dataset Size: CIFAR-10, while widely used, is a relatively small dataset. The findings may not directly translate to more extensive datasets with diverse characteristics.

Failures:

The models gave accuracy below and equal to 10% multiple times in the start which was overcome by doing changes in the

- Dropout percent
- Learning rate needed to be changed in order to converge better
- Multiple combinations of both were made and executed in order to get the model accuracies of 76% and 81% respectively for the simple cnn model and the pretrained and fine tuned VGG19 model.

Biggest Risk and Mitigation:

The most significant risk leading to project failure could be an overreliance on the specific characteristics of the CIFAR-10 dataset. To address this, the project could benefit from additional experiments on other datasets to validate the generalizability of the findings. Incorporating a wider range of datasets would mitigate the risk of the models being too tailored to the peculiarities of CIFAR-10.

Future Scope:

The project opens avenues for future exploration and expansion:

- Transfer Learning Variants: Investigate other transfer learning variants and architectures beyond VGG19 to discern their effectiveness in diverse contexts.
- Data Augmentation Techniques: Explore advanced data augmentation techniques to enhance model generalization.
- Ensemble Methods: Implement ensemble methods combining multiple architectures for improved robustness.
- Real-world Applications: Extend the project to real-world applications in specific domains such as healthcare, security, and autonomous systems.

New Applications:

The project's findings lay the groundwork for various applications:

- Medical Imaging: Enhanced image classification for disease identification.
- Remote Sensing: Improved object recognition in satellite imagery.
- Retail: Fine-grained product categorization for e-commerce.

By expanding the project's scope to these applications, the deep learning models can contribute to more accurate and reliable automated decision-making in diverse fields.

Conclusion:

Key Takeaways:

This project, focused on the comparative analysis of image classification models using a simple CNN and a fine-tuned VGG19 with upscaling, yields significant insights into the nuances of architectural choices. The key takeaway is the demonstrable superiority of the VGG19 model, particularly when upscaling is incorporated, showcasing the importance of leveraging pretrained features and advanced architectural components for improved image classification performance.

Key Concepts:

Architectural Impact: The project emphasizes the critical role of model architecture in image classification tasks, with the VGG19 model's intricacies and upscaling contributing to its superior performance.
Transfer Learning Significance: Leveraging pretrained models, such as VGG19, is showcased as a valuable strategy, especially when fine-tuned on specific datasets like CIFAR-10. The transferability of knowledge from pretrained models proves beneficial for tasks with limited data.
Fine-tuning Considerations: The project underscores the significance of fine-tuning, particularly when upscaling is involved. Fine-tuning enables adapting pretrained models to dataset-specific characteristics, enhancing their ability to capture detailed features.

Contribution to Deep Learning:

This project contributes to the field of deep learning by offering a comprehensive comparison of CNN architectures for image classification. The findings provide practical guidance for selecting models based on the intricacies of the task and dataset. The exploration of upscaling in VGG19 adds a nuanced understanding of architectural components, shedding light on factors influencing model performance.

Future Scope:

The project's future scope lies in further exploring and refining architectural choices for image classification:

- Advanced Architectures: Investigate and compare the performance of newer CNN architectures.
- Hyperparameter Optimization: Explore optimal hyperparameter configurations for models with upscaling, considering trade-offs between accuracy and computational efficiency.
- Transfer Learning Variations: Assess the impact of different transfer learning strategies beyond the conventional fine-tuning, exploring avenues like progressive unfreezing.

The insights gained from this project can help advancements in designing more effective and efficient image classification models, fostering ongoing progress in the dynamic field of deep learning.

References:

1. I. Oztel, G. Yolcu and C. Oz, "Performance Comparison of Transfer Learning and Training from Scratch Approaches for Deep Facial Expression Recognition,"

2019 4th International Conference on Computer Science and Engineering (UBMK), Samsun, Turkey, 2019, pp. 1-6, doi: 10.1109/UBMK.2019.8907203.
2. S. Tammina, "Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images," *Int. J. Sci. Res. Pub.*, vol. 9, no. 10, ISSN: 2250-3153, 2019, doi: 10.29322/IJSRP.9.10.2019.p9420.
3. For the codes, we referred Kaggle website: https://www.kaggle.com/code/kmkarakaya/transfer-learning-for-image-classification for the transfer learning :https://www.kaggle.com/code/souravkgoyal/fine-tuning-pretrained-model-keras for the fine tuning part.

Contribution of team members:

Sr No.	**Name**	**Roll Number**	**%age Contribution**
1.	Aryan Baghla	21ucs035	33.34
2,	Atul Raj Kalia	21dcs002	33.33
3.	Ananya Khadria	21ucs019	33.33
		Total	**100%**