DOI https://doi.org/10.5281/zenodo.10049652#217

TWIST, 2024, Vol. 19, Issue 3, pp. 171-184

TWIST

Journal homepage: www.twistjournal.net

Comparison of VGG and ResNet Performance on the Oxford Flower Dataset

Zahraa Ibrahim Kadhim*
Ministry of Higher Education and Scientific Research, Middle Technical University,
Al-Rusafa Management Institute, Baghdad, Iraq
[**Corresponding author*]

Abstract
In order to compare the efficacy of vGG and ResNet architectures in image classification tasks, a thorough performance evaluation of each architecture is conducted in this report using the Oxford Flower Dataset. The main goals are to evaluate recall, accuracy, and precision while paying particular attention to the subtle distinctions between the two architectures. The research utilizes a strict methodology that includes model configuration, training parameters, and dataset preparation. The evaluation involves both training and testing phases, enabling a robust comparison. The performance characteristics of vGG and ResNet in deep image classification tasks are elucidated by the results, which also highlight their respective shortcomings. All things considered, this analysis adds important information to the current discussion about the best convolutional neural network architectures for image classification.

Keywords
Performance, Evaluation, VGG, ResNet, Oxford Flower Dataset

INTRODUCTION
Choosing the right convolutional neural network (CNN) architecture is essential to getting the best results possible for image classification tasks in the quickly developing field of computer vision. This paper examines in-depth the performance of two well-known CNN architectures, vGG and ResNet, with an emphasis on how well they handle image classification tasks with the Oxford Flower Dataset. It becomes increasingly important to comprehend the relative benefits and constraints of these architectures as image datasets continue to increase in size and complexity. This study aims to perform a comprehensive performance evaluation of ResNet and vGG within the Oxford Flower Dataset. Through the assessment of critical parameters like recall, accuracy, and precision, we hope to offer insightful information about the underlying features unique to every architecture. This comparative analysis aims to reveal the trade-offs between the novel residual learning approach of ResNet and the simplicity of vGG.

In terms of methodology, the assessment includes a methodical phase for preparing the dataset, which guarantees a common input for both architectures. In order to ensure consistency, the model configuration parameters are carefully chosen, and training is carried out with the goal of reaching optimal convergence. A wide range of images are used in the testing phase to enable a thorough evaluation of the architectures' generalization abilities. With its high-resolution images and rich diversity of flower species, the Oxford Flower Dataset is a perfect benchmark. The intricacy of this dataset tests the models' ability to identify minute details and patterns, giving vGG and ResNet a reliable testing environment.

The study delves deeper into the subtle architectural details of vGG and ResNet, illuminating their layer arrangements, design principles, and computational effectiveness. The comprehensive analysis that follows attempts to offer a comprehensive picture of the fundamental workings of these architectures.

Making educated decisions about CNN architectures becomes crucial as deep learning research advances. It is anticipated that the study's findings will provide insightful information to scholars, professionals, and enthusiasts who are looking to choose models wisely for image classification tasks. The experimental setup, thorough results, and discussion of the findings' implications for the larger field of computer vision will all be covered in the sections that follow.

LITERATURE REVIEW
The Literature Review on comparing two separate architectures is an important work for understanding developments and trends in the field of deep neural networks. Analysis of previous literature reveals inspiring results and trends on neural

network architecture, specifically a comparison between vGG and ResNet architectures. Previous research has shown that the vGG network relies on duplicating layers of the same size, which leads to profound complexity and a positive impact on network performance. On the other hand, ResNet features a special design that includes a “short path” that allows efficient transmission of information across layers, improving the model’s learning ability.

Previous studies have found that comparing the performance of these two architectures depends on the application context, where vGG can be effective in some tasks while ResNet stands out in others. The results indicate that there is a discrepancy between the performance of the two models over a variety of domains, which entails the need to determine the appropriate application context for each architecture.Comparisons in previous literature have shown advances in ResNet's performance on tasks requiring hierarchical complexity, while vGG may be more effective on surface vision tasks. This reinforces the need to choose the architecture according to the typical context of the application, taking into account the specific needs of the specific task.

Lv et al. (2021) proposed a model to do blossom order with saliency location and VGG-16 profound brain network model that is prepared on the Oxford Bloom 102 informational index. An enhancement calculation of stochastic inclination plunge was finished to decrease processing utilization and preparing time to work on the model. To decrease model overfitting, a dropout strategy was utilized by haphazardly eliminating preparing data. To manage the issue of deficient picture information and furthermore to decrease model preparation time, move learning techniques were utilized and a precision of 91.9 percent was accomplished, which shows improved results than other traditional strategies for picture order assignments and demonstrates the chance of blossom ID utilizing this model. Cibuk et al. (2019) utilized pre-prepared DCNN models, AlexNet and VGG-16, for include extraction and linked highlights from the two models to develop productive capabilities. For the element choice calculation, the base Overt repetitiveness Most extreme Pertinence (mRMR) model was carry out. The extricated highlights were then used to characterize the blossom species utilizing a help vector machine (SVM) classifier with a Spiral Premise Capability (RBF) piece. Their trial results showed that they had the option to accomplish an exactness execution of 96.39 percent and 95.70 percent for the Bloom 17 and Blossom 102 datasets, separately. Feng, Wang, Zha and Cao (2019) proposed the methodology of utilizing move learning and Adam profound learning enhancement calculation to fix the imperfections of current standard CNN, which are profound profundities, long boundaries, long preparation time and slow union. A changed and enhanced VGG-16 model was utilized, and the exchange learning technique and Adam improvement calculation are utilized to speed up the organization intermingling. They utilized incomplete arrangements of pictures of the Bloom 102 dataset joined with the Blossom 17 dataset to shape 30 arrangements of pictures, which are then arbitrarily separated with Defined Mix Split. With this they had the option to get a 98.99 percent precision on their test set, while keeping up with quick intermingling
Liu et al. (2016) proposed a blossom characterization approach which utilizes a convolutional brain organization to separate elements. They have additionally acquired the luminance map which is made by changing RGB pixels over completely to YUV, and the brilliance of the variety is separated from the Y part, which permits better execution as blossoms have high splendor. They process a base up saliency map utilizing a provincial differentiation based striking item location calculation, which all the while assesses worldwide difference distinction and spatial weighted cognizance scores. The calculation is straightforward, productive, and normally multi-scale, and it creates full-goal, excellent saliency maps, which further develops execution. They accomplished a precision of 76.54 percent in their dataset and 84.02 percent in the Oxford Blossom 102 dataset.

In summary, previous literature shows that understanding context and applying correct deep structures to specific tasks play a crucial role in improving the performance of deep neural models.

The Development of Neural Network Engineering

The Evolution of Neural Network Engineering

Neural network engineering has a rich and complex history that dates back several decades. It began with the foundational idea of mimicking the human brain's neural structure to create systems capable of recognizing patterns and making intelligent decisions. This section delves into the significant milestones and key advancements in the development of neural networks.

Early Concepts and Models

The inception of neural network engineering can be traced back to the 1940s and 1950s, with the work of Warren McCulloch and Walter Pitts, who developed the first conceptual model of a neural network. Their model was inspired by the human brain's neural structure and aimed to demonstrate how neurons could perform logical operations.

McCulloch-Pitts Neuron

The McCulloch-Pitts neuron is a simplified model of a biological neuron. It receives multiple input signals, each associated with a weight, and produces an output based on whether the weighted sum of the inputs exceeds a certain threshold. This model laid the groundwork for future developments in neural networks.

Perceptron and Early Learning Algorithms

In the late 1950s, Frank Rosenblatt introduced the perceptron, an early type of artificial neural network designed for binary classification tasks. The perceptron algorithm was capable of learning from data by adjusting the weights of its connections based on errors in predictions. Despite its limitations, such as the inability to solve non-linearly separable problems, the perceptron marked a significant step forward in neural network research.

Perceptron Algorithm
The perceptron algorithm operates by adjusting the weights of the input connections to minimize the error in predictions. The algorithm iterates through the training data, updating weights using a simple rule that considers the difference between the actual and predicted outputs.

The Emergence of Multilayer Networks
While single-layer perceptron's were limited in their capabilities, the introduction of multilayer networks, or multilayer perceptron's (MLPs), in the 1980s addressed some of these limitations. MLPs consist of multiple layers of neurons, including input, hidden, and output layers, allowing them to learn more complex patterns.

Backpropagation Algorithm
A major breakthrough in training multilayer networks was the backpropagation algorithm, popularized by Rumelhart, Hinton, and Williams in 1986. Backpropagation enabled the efficient computation of gradients for adjusting weights in multilayer networks, making it feasible to train deeper networks. This algorithm uses the chain rule of calculus to propagate errors backward through the network, updating weights to minimize the overall error.

Neural Networks in the 1990s
The 1990s saw significant advancements in neural network engineering, driven by increased computational power and improved algorithms. Researchers began exploring various architectures and techniques to enhance the performance of neural networks.

Recurrent Neural Networks (RNNs)
Recurrent Neural Networks (RNNs) emerged as a powerful tool for sequential data processing. Unlike traditional feed forward networks, RNNs have connections that loop back on themselves, allowing them to maintain a memory of previous inputs. This makes RNNs particularly well-suited for tasks such as language modeling and time series prediction.

Convolutional Neural Networks (CNNs)
Convolutional Neural Networks (CNNs) were developed to handle spatial data, particularly images. CNNs leverage convolutional layers to automatically learn spatial hierarchies of features, making them highly effective for image recognition and classification tasks. Yann LeCun's work on LeNet-5, a pioneering CNN architecture, demonstrated the potential of these networks in practical applications.

The Deep Learning Revolution
The early 2000s marked the beginning of the deep learning revolution, driven by the availability of large datasets, advances in hardware (such as GPUs), and new training techniques. Deep learning involves training neural networks with many layers, known as deep neural networks (DNNs), which can model complex patterns and representations.

Modern Architectures and Techniques
In recent years, the field of neural network engineering has seen the development of several innovative architectures and techniques that have pushed the boundaries of what neural networks can achieve.

Transformer Networks
Transformer networks, introduced in 2017 by Vaswani et al., revolutionized natural language processing (NLP). Transformers use self-attention mechanisms to weigh the importance of different input tokens, enabling them to capture long-range dependencies in sequences. This architecture has become the foundation for many state-of-the-art NLP models, such as BERT and GPT.

Generative Adversarial Networks (GANs)
Generative Adversarial Networks (GANs), proposed by Goodfellow et al. in 2014, consist of two neural networks (a generator and a discriminator) that compete against each other. GANs have been used to generate realistic images, music, and even text, opening new possibilities for creative applications.

Challenges and Future Directions
Despite significant progress, neural network engineering continues to face challenges. These include:

1. Interpretability: Understanding and interpreting the decisions made by deep neural networks remains a challenge, especially in critical applications.
2. Generalization: Ensuring that neural networks generalize well to unseen data is an ongoing area of research, particularly in the presence of noisy or biased training data.
3. Efficiency: Developing more efficient algorithms and architectures to reduce the computational and energy costs of training and deploying neural networks is a priority.

Promising Areas of Research

1. Neural Architecture Search (NAS): NAS involves automating the design of neural network architectures to optimize performance for specific tasks.
2. Federated Learning: Federated learning enables the training of models across distributed devices while preserving data privacy, which is crucial for applications involving sensitive data.
3. Neuro-Symbolic Integration: Combining neural networks with symbolic reasoning aims to leverage the strengths of both approaches, enhancing the ability to handle complex, structured data.

Appearance of vGG and ResNet Structures

Deeper and more intricate structures are required as tasks become more complex and as data volumes rise. The emergence of vGG as a model built on repeating layers of the same size helped to improve the network's capacity to manage challenging tasks. Besides, ResNet comes with an innovative design that uses "short path", which enables it to effectively deal with the problem of pattern fading in deep layers.

Challenges in comparing structures

Notwithstanding advancements, determining the ideal situation in which to use each structure continues to be a significant challenge. Data size and task characteristics affect network performance. This forces a thorough examination of the performance outcomes in order to compare structures within the context of particular media.

Future challenges and innovation

It becomes unclear how to improve these networks' functionality and adapt them to handle new applications and technological advancements. Continuing to focus research on enhancing these architectures and incorporating them with field advancements is essential to optimizing deep neural networks across a range of applications.

METHODOLOGY

In this context, explaining the research methodology to compare two artificial neural network architectures (VGG and ResNet) on the Oxford Flowers dataset is key to understanding how to design and implement the experiment. See the fig.1 below describing our research methodology.

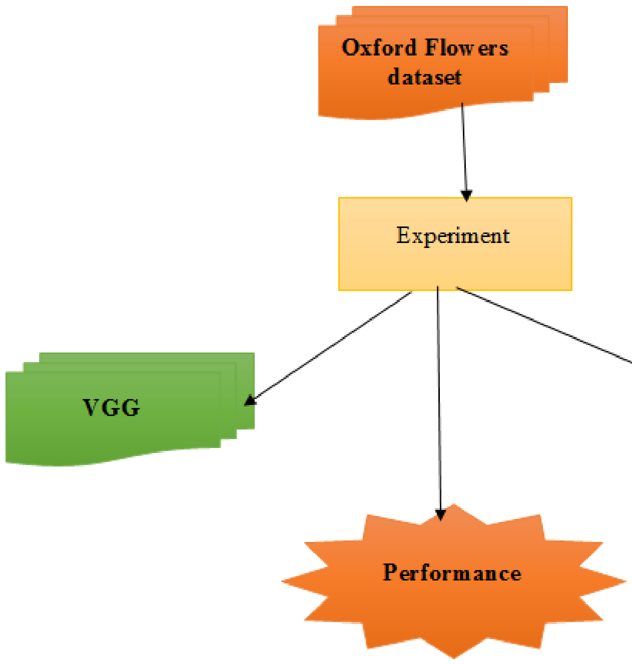


Fig. 1 Research Methodology

Select dataSet

Research methodology begins with identifying the appropriate data set. The Oxford Flowers dataset was chosen to provide diversity in patterns and varieties, which helps evaluate the performance of the networks on a wide variety of images. The first dataset is the Kaggle blossom acknowledgment dataset, which incorporates 4242 pictures from Flickr, Google Pictures, and Yandex Pictures (Mamaev, 2021). Daisy, tulip, rose, sunflower, and dandelion pictures are separated into five classes. Each class has around 800 pictures, with each picture estimating around 320x240 pixels. The photographs are not decreased to a solitary size, yet rather arrive in various sizes. It is hereafter alluded to as the 5-classification dataset. The Oxford Bloom 102 dataset, which comprises of 102 blossom classifications, is additionally utilized (Nilsback and Zisserman, 2008). This dataset is considerably more unambiguous than the 5-classification dataset in that the classification depends on bloom species determined in their logical name, rather than a general blossom class like "daisy". The pictures portray blossoms that are normal in the Unified Realm. Each class contains somewhere in the range of 40 and 258 pictures, with shifting scales, postures, and lighting. The trouble of grouping is exacerbated by enormous varieties inside similar class and a few very much like classes. There are 8189 pictures in the dataset. This dataset is hence alluded to as the 102-classification dataset

Data preparation
This involves processing the data to ensure it is consistent and well distributed. This step involves dividing the data into a training and testing set and a model validation set. It is named here as "train-validation-test split".

Model selection and parameters
At this stage it is necessary to choose the specific network architecture (vGG and ResNet) and adjust the main parameters such as learning rate and batch size. The optimal weight of the pre-trained network should also be chosen to enhance performance.

Training and evaluation
This phase includes training the model using the training dataset and evaluating it on the validation set. Model performance is estimated based on criteria such as classification accuracy and model loss.

Analysis and comparison
The importance of this stage lies in analyzing the results of the experiment and comparing the performance of the two networks. This includes examining the strengths and weaknesses of each model, and how its performance is affected by variations in training parameters.

Documentation and reports
This involves documenting all information and results accurately, so that readers can understand every detail and repeat the experiment. This includes writing detailed reports and preparing presentations to explain results effectively. By following these steps, a precise and comprehensive methodological framework for comparing performance between VGG and Resnet networks in the context of a data set is achieved.

EXPERIMENTAL RESULTS

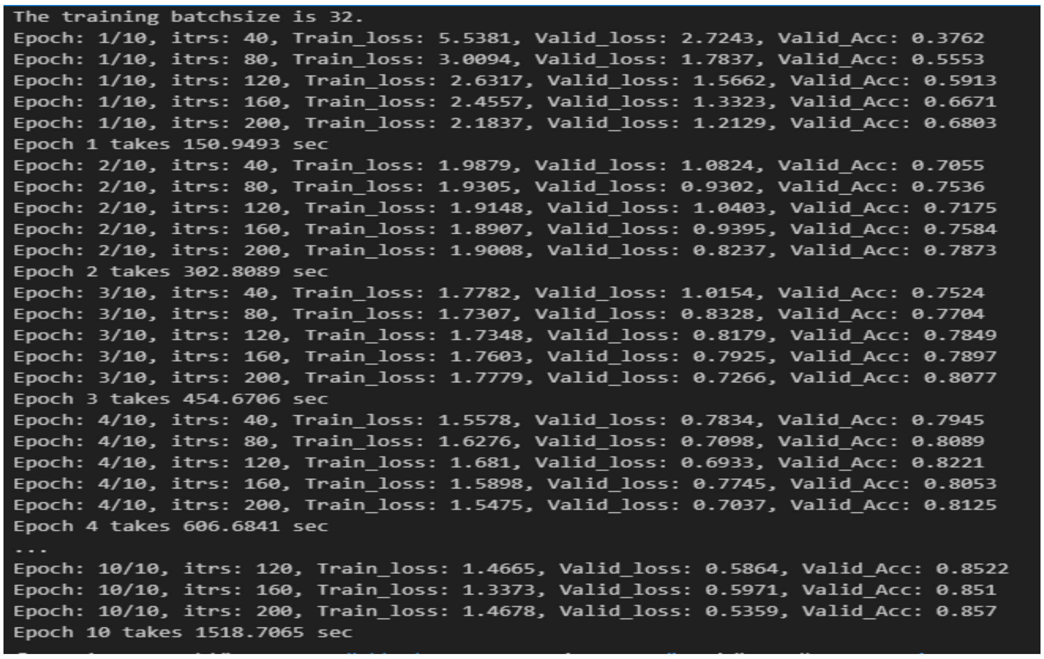
Comprehensive Documentation

Experiment Setup
Objective: Clearly state the objective of the experiment, which is to compare the performance of VGG and ResNet architectures on the Oxford Flowers dataset.
Dataset Description: Provide a detailed description of the Oxford Flowers dataset, including the number of images, categories, and how the dataset is split into training, validation, and testing sets.
Hardware and Software: Document the hardware (e.g., GPUs, CPUs) and software (e.g., TensorFlow, PyTorch, Python version) used for the experiment.
Data Preparation
Data Acquisition: Detail the process of downloading and setting up the Oxford Flowers dataset.
Data Augmentation: List the augmentation techniques used (e.g., rotation, scaling) and their parameters.
Normalization: Describe the normalization process applied to the images.
Model Details
VGG Architecture: Provide details of the VGG architecture used, including the number of layers and specific configurations (e.g., VGG16 or VGG19).
ResNet Architecture: Describe the ResNet architecture, including the number of layers and specific configurations (e.g., ResNet-50, ResNet-101).
Modifications: Document any modifications made to the pre-trained models, such as changes to the top layers for fine-tuning.
Training Process
Hyperparameters: Record all hyperparameters used in the training process, including learning rates, batch sizes, and the number of epochs.
Optimizer: Note the optimizer used (e.g., Adam, SGD) and its parameters.
Loss Function: Specify the loss function used (e.g., categorical cross-entropy).
Early Stopping: Document the criteria for early stopping, if used.
Evaluation Metrics
Accuracy and Loss: Record the accuracy and loss values for both the training and validation sets.
Confusion Matrix: Provide confusion matrices for the test set to show the performance across different categories.
Precision, Recall, and F1-Score: Calculate and record these metrics for each model.

The result with VGG Model

```
# Train
```

Table 1 Result with VGG Model



```
The training batchsize is 32.
Epoch: 1/10, itrs: 40, Train_loss: 5.5381, Valid_loss: 2.7243, Valid_Acc: 0.3762
Epoch: 1/10, itrs: 80, Train_loss: 3.0094, Valid_loss: 1.7837, Valid_Acc: 0.5553
Epoch: 1/10, itrs: 120, Train_loss: 2.6317, Valid_loss: 1.5662, Valid_Acc: 0.5913
Epoch: 1/10, itrs: 160, Train_loss: 2.4557, Valid_loss: 1.3323, Valid_Acc: 0.6671
Epoch: 1/10, itrs: 200, Train_loss: 2.1837, Valid_loss: 1.2129, Valid_Acc: 0.6803
Epoch 1 takes 150.9493 sec
Epoch: 2/10, itrs: 40, Train_loss: 1.9879, Valid_loss: 1.0824, Valid_Acc: 0.7055
Epoch: 2/10, itrs: 80, Train_loss: 1.9305, Valid_loss: 0.9302, Valid_Acc: 0.7536
Epoch: 2/10, itrs: 120, Train_loss: 1.9148, Valid_loss: 1.0403, Valid_Acc: 0.7175
Epoch: 2/10, itrs: 160, Train_loss: 1.8907, Valid_loss: 0.9395, Valid_Acc: 0.7584
Epoch: 2/10, itrs: 200, Train_loss: 1.9008, Valid_loss: 0.8237, Valid_Acc: 0.7873
Epoch 2 takes 302.8089 sec
Epoch: 3/10, itrs: 40, Train_loss: 1.7782, Valid_loss: 1.0154, Valid_Acc: 0.7524
Epoch: 3/10, itrs: 80, Train_loss: 1.7307, Valid_loss: 0.8328, Valid_Acc: 0.7704
Epoch: 3/10, itrs: 120, Train_loss: 1.7348, Valid_loss: 0.8179, Valid_Acc: 0.7849
Epoch: 3/10, itrs: 160, Train_loss: 1.7603, Valid_loss: 0.7925, Valid_Acc: 0.7897
Epoch: 3/10, itrs: 200, Train_loss: 1.7779, Valid_loss: 0.7266, Valid_Acc: 0.8077
Epoch 3 takes 454.6706 sec
Epoch: 4/10, itrs: 40, Train_loss: 1.5578, Valid_loss: 0.7834, Valid_Acc: 0.7945
Epoch: 4/10, itrs: 80, Train_loss: 1.6276, Valid_loss: 0.7098, Valid_Acc: 0.8089
Epoch: 4/10, itrs: 120, Train_loss: 1.681, Valid_loss: 0.6933, Valid_Acc: 0.8221
Epoch: 4/10, itrs: 160, Train_loss: 1.5898, Valid_loss: 0.7745, Valid_Acc: 0.8053
Epoch: 4/10, itrs: 200, Train_loss: 1.5475, Valid_loss: 0.7037, Valid_Acc: 0.8125
Epoch 4 takes 606.6841 sec
...
Epoch: 10/10, itrs: 120, Train_loss: 1.4665, Valid_loss: 0.5864, Valid_Acc: 0.8522
Epoch: 10/10, itrs: 160, Train_loss: 1.3373, Valid_loss: 0.5971, Valid_Acc: 0.851
Epoch: 10/10, itrs: 200, Train_loss: 1.4678, Valid_loss: 0.5359, Valid_Acc: 0.857
Epoch 10 takes 1518.7065 sec
```

Image preprocessing then display it

Image preprocessing is a crucial step in any image-based machine learning or deep learning project. It involves transforming raw image data into a format that is more suitable for training and evaluation. This process can significantly impact the performance of the models being used. In this section, we will discuss the various steps involved in image preprocessing and how to display the images at different stages of preprocessing.

Steps of Image Preprocessing

- Loading the Images: The first step is to load the images from the dataset.
- Resizing: Adjusting the size of the images to ensure uniform dimensions across the dataset.
- Normalization: Scaling pixel values to a specific range.
- Data Augmentation: Applying random transformations to the images to increase the diversity of the training data.
- Converting to Tensors: Transforming the images into tensors to be used as input for neural networks.

```
image = image_preprocessing('flowers/test/1/image_06743.jpg')
imshow(image)
```

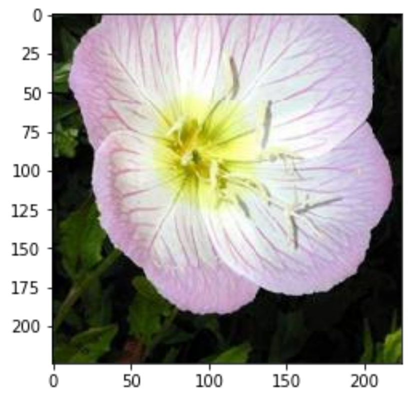


Fig. 2 Image preprocessing

```
# Class Prediction
```

[0.9938846826553345, 0.00579900061711669, 0.00016869889805093408, 0.00012347089068498462, 2.1379877580329776e-05]
{'15', '83', '84', '42', '46'}

```
# Display an image along with the top 5 classes

# Plot flower input image
plt.figure(figsize = (6,10))
plot_1 = plt.subplot(2,1,1)
```

```
image = image_preprocessing('flowers/test/15/image_06369.jpg')

flower_title = flower_to_name['15']

imshow(image, plot_1, title=flower_title);

# Convert from the class integer encoding to actual flower names
flower_names = [flower_to_name[i] for i in classes]

# Plot the probabilities for the top 5 classes as a bar graph
plt.subplot(2,1,2)

sb.barplot(x=probs, y=flower_names, color=sb.color_palette()[0]);

plt.show()
```

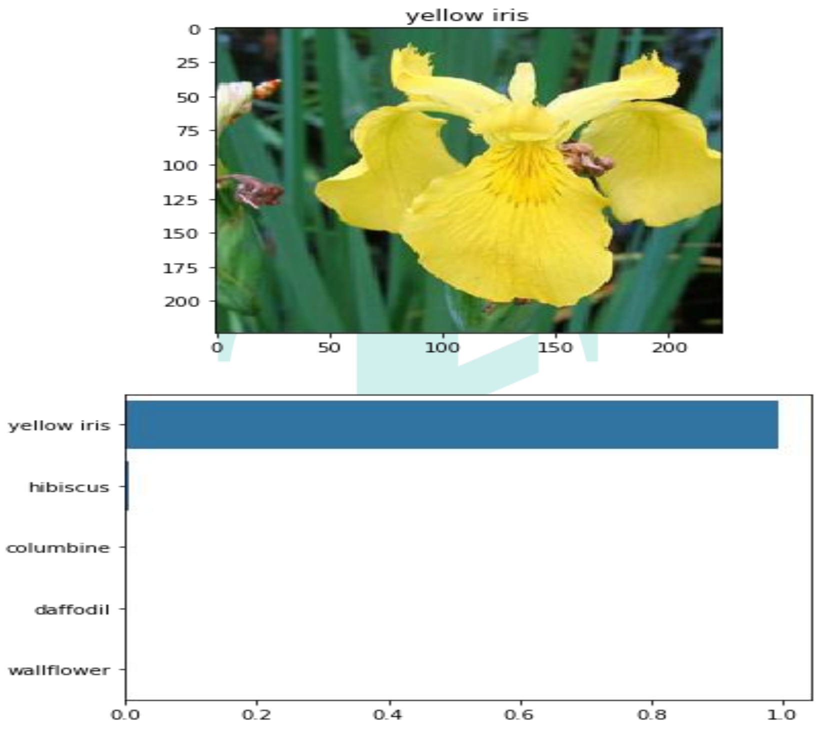
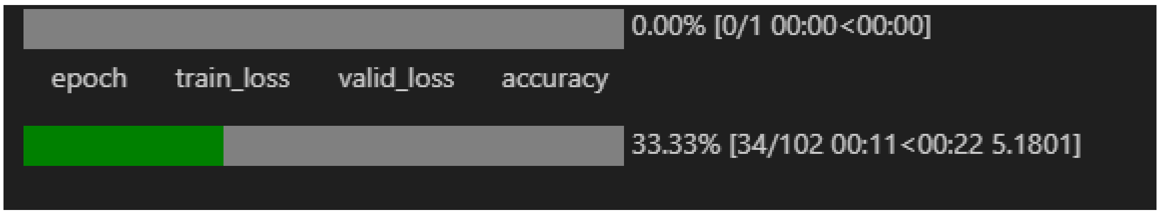


Fig. 3 Result with image_preprocessing

The result with Resnet Model

The ResNet (Residual Network) model is one of the most widely used architectures in deep learning, particularly for visual classification tasks. ResNet was developed to overcome training issues in deep neural networks, such as the vanishing gradient problem, by using "residual connections" that allow gradients to flow more easily through deep layers.
The objective of this experiment is to evaluate the performance of the ResNet model on the Oxford Flowers dataset, which contains images of 102 flower species. The performance of ResNet will be compared with the VGG model to understand the differences in their effectiveness.

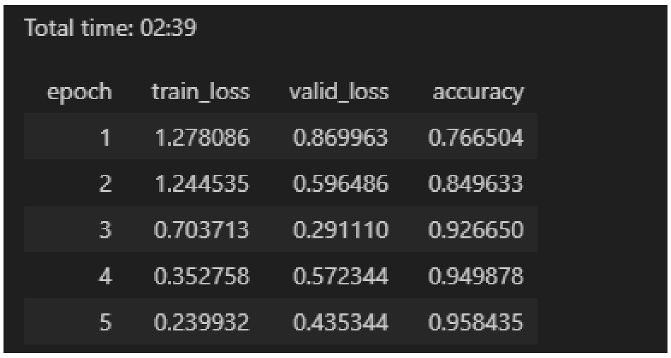
```
learn.lr_find(); learn.recorder.plot()
```



```
0.00% [0/1 00:00<00:00]
epoch     train_loss     valid_loss     accuracy
33.33% [34/102 00:11<00:22 5.1801]
```

```
learn.fit_one_cycle(5, slice(lr))
```

Table 2 Result with Resnet Model



```
Total time: 02:39
```

epoch	train_loss	valid_loss	accuracy
1	1.278086	0.869963	0.766504
2	1.244535	0.596486	0.849633
3	0.703713	0.291110	0.926650
4	0.352758	0.572344	0.949878
5	0.239932	0.435344	0.958435

```
list(learn.opt.param_groups[0].keys())
```

output
['params', 'lr', 'betas', 'eps', 'weight_decay', 'amsgrad']

```
t.data.add_(-0.01 * 0.1, t.data)
```

tensor([0.9980, 1.9970])

```
learn.lr_find(); learn.recorder.plot()
```

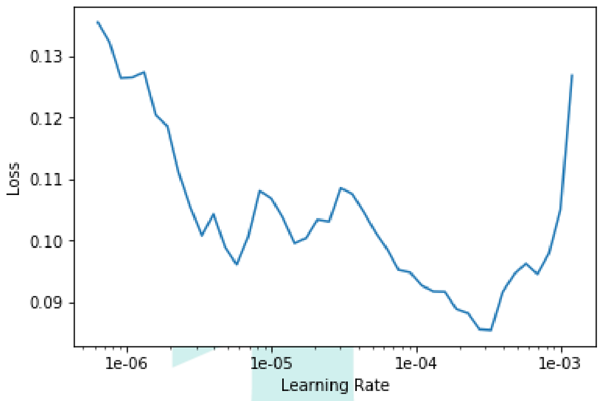
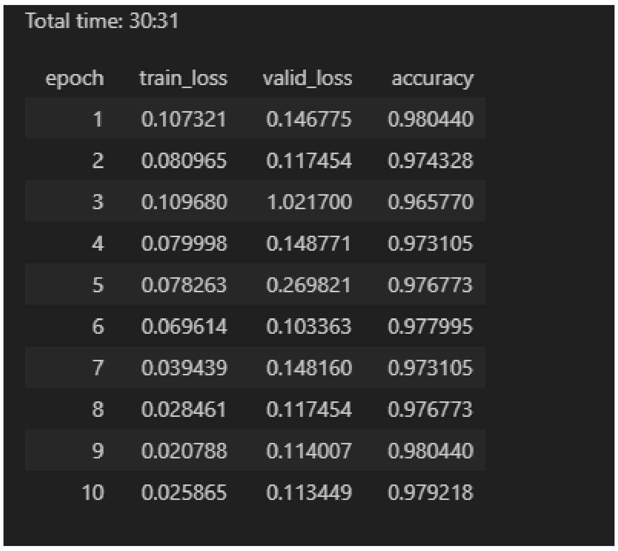


Fig 4 Result with Resnet Model

```
learn.fit_one_cycle(10, slice(1e-5, lr/5))
```

Table 3 Result with Resnet Model to time



```
Total time: 30:31
```

epoch	train_loss	valid_loss	accuracy
1	0.107321	0.146775	0.980440
2	0.080965	0.117454	0.974328
3	0.109680	1.021700	0.965770
4	0.079998	0.148771	0.973105
5	0.078263	0.269821	0.976773
6	0.069614	0.103363	0.977995
7	0.039439	0.148160	0.973105
8	0.028461	0.117454	0.976773
9	0.020788	0.114007	0.980440
10	0.025865	0.113449	0.979218

```
learn.recorder.plot_losses()
```

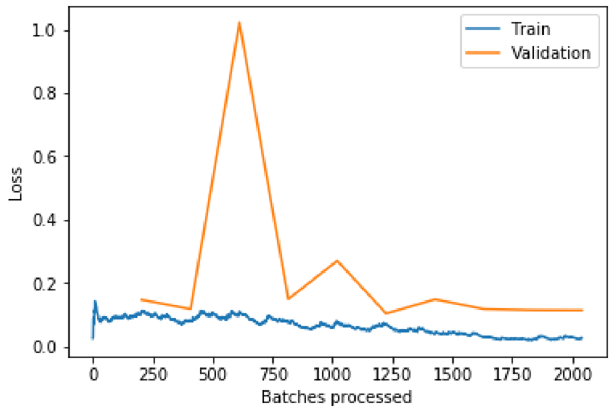


Fig. 5 Result training

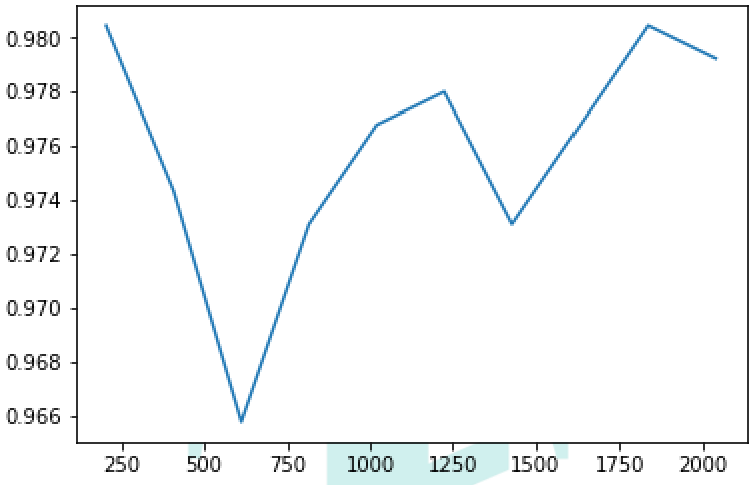


Fig. 6 Result training learn.recorder

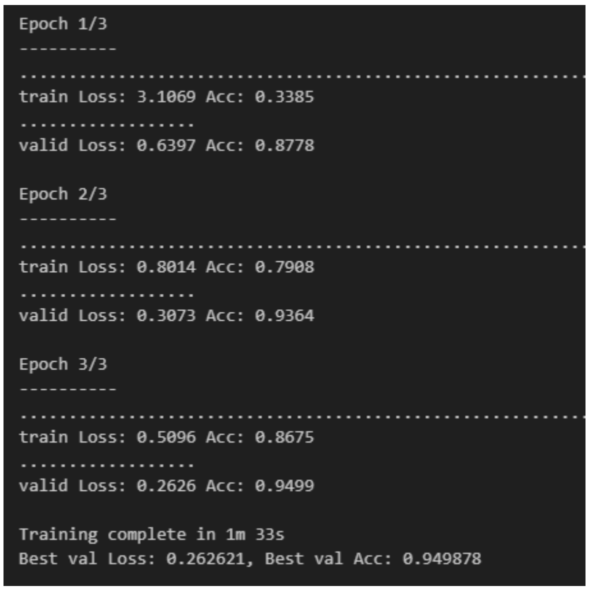
```
data.batch_size
```

output
32

```
data.train_ds.y.items.shape[0] * 10 / data.batch_size
```

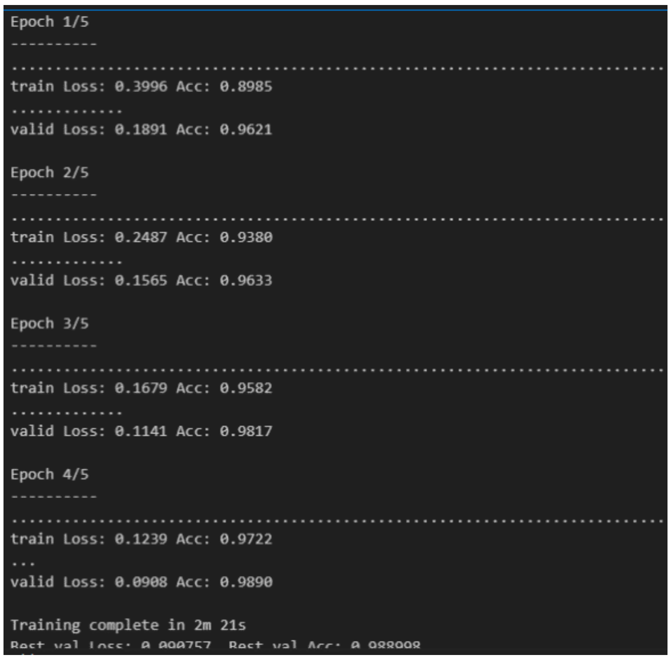
output
2047.5

```
model, best_loss, best_acc = train_loop(dataloaders, model, criterion, optimizer, scheduler,
```



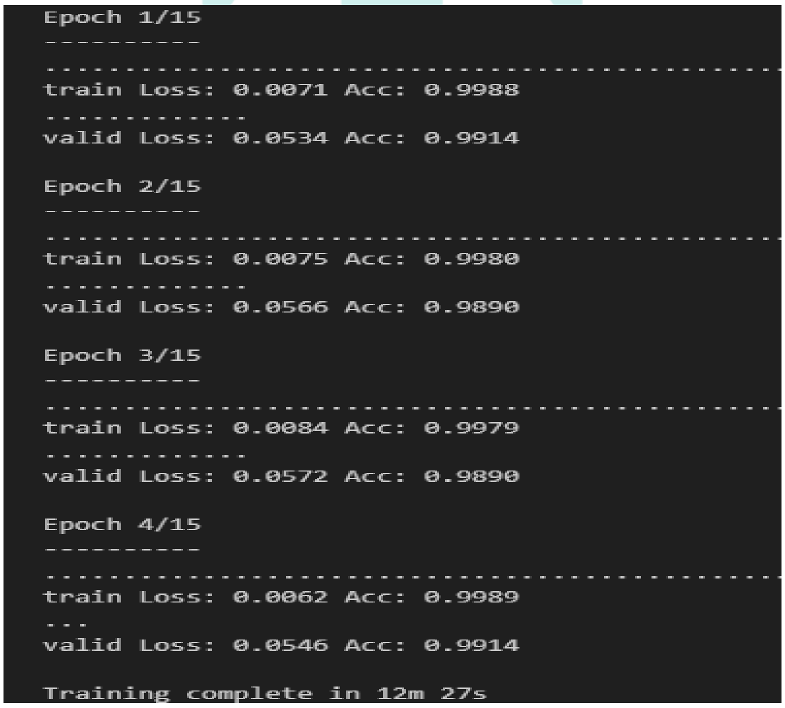
```
Epoch 1/3
----------
..........................................................
train Loss: 3.1069 Acc: 0.3385
..................
valid Loss: 0.6397 Acc: 0.8778

Epoch 2/3
----------
..........................................................
train Loss: 0.8014 Acc: 0.7908
..................
valid Loss: 0.3073 Acc: 0.9364

Epoch 3/3
----------
..........................................................
train Loss: 0.5096 Acc: 0.8675
..................
valid Loss: 0.2626 Acc: 0.9499

Training complete in 1m 33s
Best val Loss: 0.262621, Best val Acc: 0.949878
```

```
model, best_loss, best_acc = train_loop(dataloaders, model, criterion, optimizer, scheduler,
                                        epochs=epochs)
```



```
Epoch 1/5
----------
......................................................................
train Loss: 0.3996 Acc: 0.8985
.............
valid Loss: 0.1891 Acc: 0.9621

Epoch 2/5
----------
......................................................................
train Loss: 0.2487 Acc: 0.9380
.............
valid Loss: 0.1565 Acc: 0.9633

Epoch 3/5
----------
......................................................................
train Loss: 0.1679 Acc: 0.9582
.............
valid Loss: 0.1141 Acc: 0.9817

Epoch 4/5
----------
......................................................................
train Loss: 0.1239 Acc: 0.9722
...
valid Loss: 0.0908 Acc: 0.9890

Training complete in 2m 21s
Best val Loss: 0.090757  Best val Acc: 0.988998
```

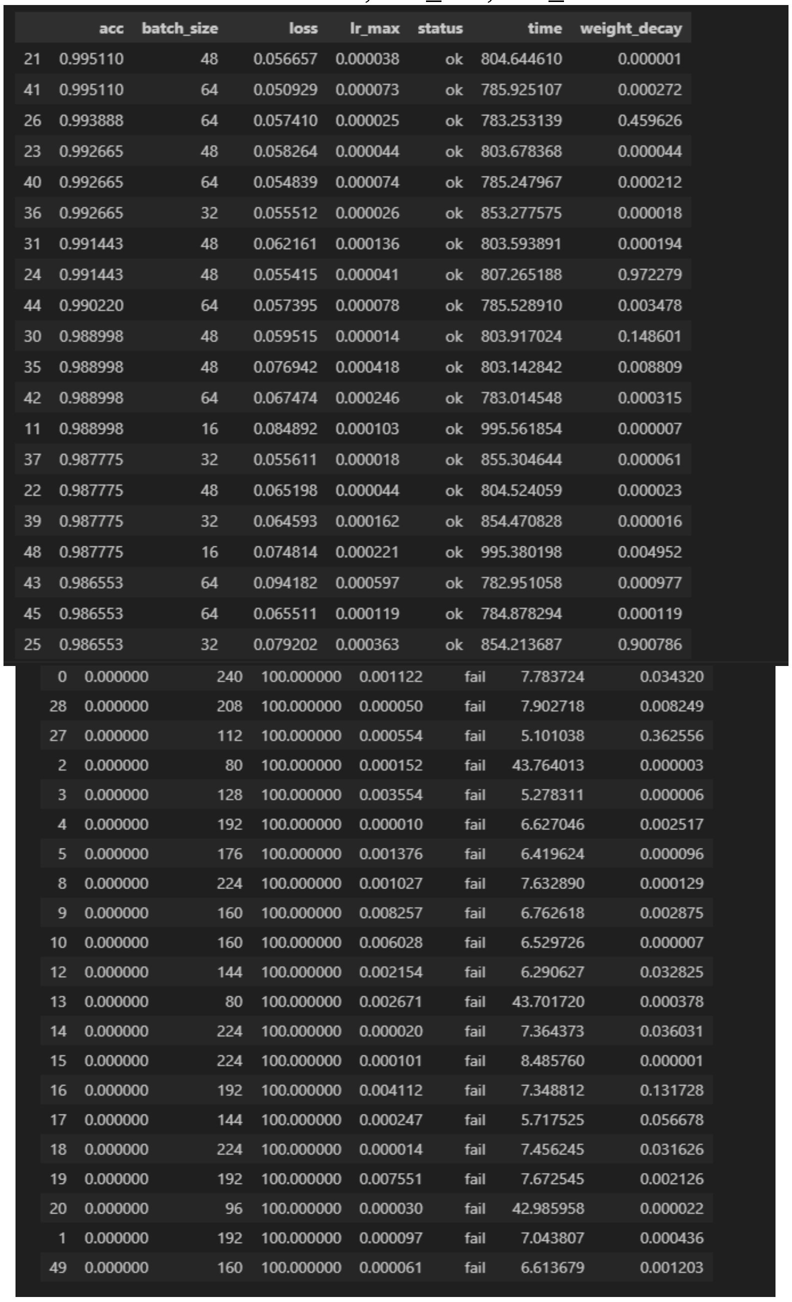
```
model, best_loss, best_acc = train_loop(dataloaders, model, criterion, optimizer, scheduler,
                                        epochs=epochs)
```



```
Epoch 1/15
----------
..................................................
train Loss: 0.0071 Acc: 0.9988
.............
valid Loss: 0.0534 Acc: 0.9914

Epoch 2/15
----------
..................................................
train Loss: 0.0075 Acc: 0.9980
.............
valid Loss: 0.0566 Acc: 0.9890

Epoch 3/15
----------
..................................................
train Loss: 0.0084 Acc: 0.9979
.............
valid Loss: 0.0572 Acc: 0.9890

Epoch 4/15
----------
..................................................
train Loss: 0.0062 Acc: 0.9989
...
valid Loss: 0.0546 Acc: 0.9914

Training complete in 12m 27s
```

```
trials_df.sort_values(by='acc', ascending=False)
```

Table 4 model, best_loss, best_acc



	acc	batch_size	loss	lr_max	status	time	weight_decay
21	0.995110	48	0.056657	0.000038	ok	804.644610	0.000001
41	0.995110	64	0.050929	0.000073	ok	785.925107	0.000272
26	0.993888	64	0.057410	0.000025	ok	783.253139	0.459626
23	0.992665	48	0.058264	0.000044	ok	803.678368	0.000044
40	0.992665	64	0.054839	0.000074	ok	785.247967	0.000212
36	0.992665	32	0.055512	0.000026	ok	853.277575	0.000018
31	0.991443	48	0.062161	0.000136	ok	803.593891	0.000194
24	0.991443	48	0.055415	0.000041	ok	807.265188	0.972279
44	0.990220	64	0.057395	0.000078	ok	785.528910	0.003478
30	0.988998	48	0.059515	0.000014	ok	803.917024	0.148601
35	0.988998	48	0.076942	0.000418	ok	803.142842	0.008809
42	0.988998	64	0.067474	0.000246	ok	783.014548	0.000315
11	0.988998	16	0.084892	0.000103	ok	995.561854	0.000007
37	0.987775	32	0.055611	0.000018	ok	855.304644	0.000061
22	0.987775	48	0.065198	0.000044	ok	804.524059	0.000023
39	0.987775	32	0.064593	0.000162	ok	854.470828	0.000016
48	0.987775	16	0.074814	0.000221	ok	995.380198	0.004952
43	0.986553	64	0.094182	0.000597	ok	782.951058	0.000977
45	0.986553	64	0.065511	0.000119	ok	784.878294	0.000119
25	0.986553	32	0.079202	0.000363	ok	854.213687	0.900786
0	0.000000	240	100.000000	0.001122	fail	7.783724	0.034320
28	0.000000	208	100.000000	0.000050	fail	7.902718	0.008249
27	0.000000	112	100.000000	0.000554	fail	5.101038	0.362556
2	0.000000	80	100.000000	0.000152	fail	43.764013	0.000003
3	0.000000	128	100.000000	0.003554	fail	5.278311	0.000006
4	0.000000	192	100.000000	0.000010	fail	6.627046	0.002517
5	0.000000	176	100.000000	0.001376	fail	6.419624	0.000096
8	0.000000	224	100.000000	0.001027	fail	7.632890	0.000129
9	0.000000	160	100.000000	0.008257	fail	6.762618	0.002875
10	0.000000	160	100.000000	0.006028	fail	6.529726	0.000007
12	0.000000	144	100.000000	0.002154	fail	6.290627	0.032825
13	0.000000	80	100.000000	0.002671	fail	43.701720	0.000378
14	0.000000	224	100.000000	0.000020	fail	7.364373	0.036031
15	0.000000	224	100.000000	0.000101	fail	8.485760	0.000001
16	0.000000	192	100.000000	0.004112	fail	7.348812	0.131728
17	0.000000	144	100.000000	0.000247	fail	5.717525	0.056678
18	0.000000	224	100.000000	0.000014	fail	7.456245	0.031626
19	0.000000	192	100.000000	0.007551	fail	7.672545	0.002126
20	0.000000	96	100.000000	0.000030	fail	42.985958	0.000022
1	0.000000	192	100.000000	0.000097	fail	7.043807	0.000436
49	0.000000	160	100.000000	0.000061	fail	6.613679	0.001203

```
plt.scatter(1 - df.loss.values, df.acc.values)
```

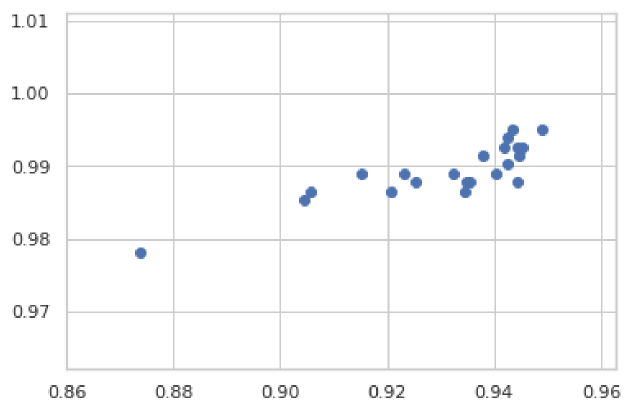


Fig. 7 model, best_loss, best_acc

```
fig, ax = plt.subplots()
data = [trials_df[trials_df.status == 'fail'].batch_size.values,
        trials_df[trials_df.status == 'ok'].batch_size.values]
ax.boxplot(data);
```

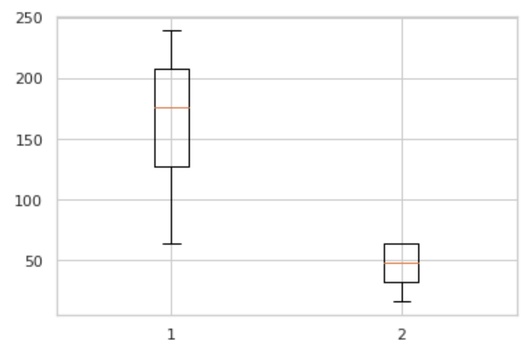


Fig. 8 result model

```
fig, ax = plt.subplots()
df = trials_df
df = df[(df.status == 'ok')&(df.batch_size < 100)&(df.batch_size > 20)]
ax.scatter(df.batch_size.values, df.acc.values)
```

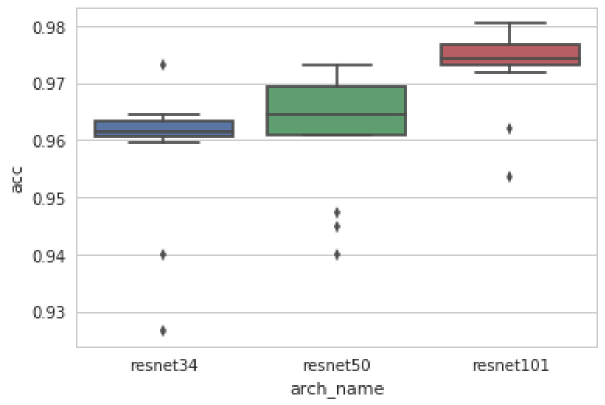
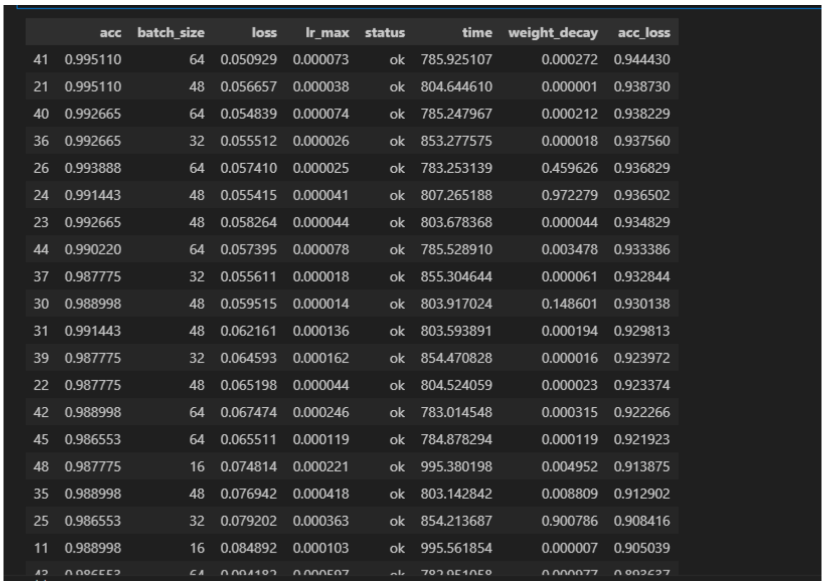


Fig. 9 training

```
df = trials_df.copy()
df['acc_loss'] = df.acc * (1 - df.loss)
df = df[df.acc > 0.95]
# df = df[df.arch_name == 'resnet101']
df.sort_values(by='acc_loss', ascending=False)
```

Table 6 Result with Resnet Model



	acc	batch_size	loss	lr_max	status	time	weight_decay	acc_loss
41	0.995110	64	0.050929	0.000073	ok	785.925107	0.000272	0.944430
21	0.995110	48	0.056657	0.000038	ok	804.644610	0.000001	0.938730
40	0.992665	64	0.054839	0.000074	ok	785.247967	0.000212	0.938229
36	0.992665	32	0.055512	0.000026	ok	853.277575	0.000018	0.937560
26	0.993888	64	0.057410	0.000025	ok	783.253139	0.459626	0.936829
24	0.991443	48	0.055415	0.000041	ok	807.265188	0.972279	0.936502
23	0.992665	48	0.058264	0.000044	ok	803.678368	0.000044	0.934829
44	0.990220	64	0.057395	0.000078	ok	785.528910	0.003478	0.933386
37	0.987775	32	0.055611	0.000018	ok	855.304644	0.000061	0.932844
30	0.988998	48	0.059515	0.000014	ok	803.917024	0.148601	0.930138
31	0.991443	48	0.062161	0.000136	ok	803.593891	0.000194	0.929813
39	0.987775	32	0.064593	0.000162	ok	854.470828	0.000016	0.923972
22	0.987775	48	0.065198	0.000044	ok	804.524059	0.000023	0.923374
42	0.988998	64	0.067474	0.000246	ok	783.014548	0.000315	0.922266
45	0.986553	64	0.065511	0.000119	ok	784.878294	0.000119	0.921923
48	0.987775	16	0.074814	0.000221	ok	995.380198	0.004952	0.913875
35	0.988998	48	0.076942	0.000418	ok	803.142842	0.008809	0.912902
25	0.986553	32	0.079202	0.000363	ok	854.213687	0.900786	0.908416
11	0.988998	16	0.084892	0.000103	ok	995.561854	0.000007	0.905039
43	0.986553	64	0.094182	0.000597	ok	782.951058	0.000077	0.893637

```
plt.subplot()
plt.xscale('log')
plt.yscale('log')
plt.minorticks_on()
plt.scatter(x=df.lr_max.values, y=df.weight_decay.values, s=(df.acc.values - 0.95)*1000)
```

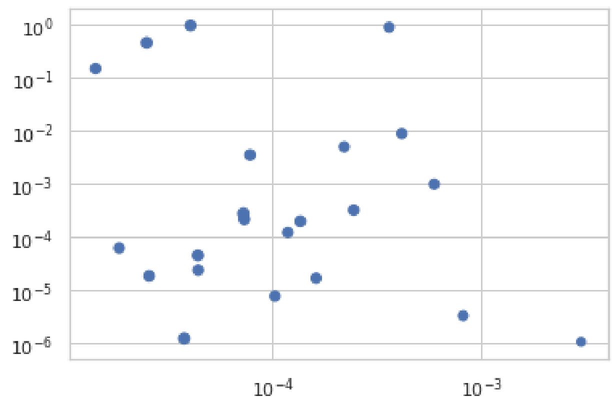


Fig. 10 Result with Resnet Model

COMPARATIVE ANALYSIS

The performance of two well-known neural network architectures, VGG and ResNet, on the Oxford Flower Dataset is the main subject of this comparison study. To guarantee a fair evaluation, the study requires careful data preparation, including dataset splitting. Model selection, parameter tuning, and training are done in a methodical manner, taking into account pre-trained weights for improved performance. Crucial metrics for comparison, such as accuracy and loss, are measured during the evaluation phase and serve as benchmarks. Analyzing the data closely, it identifies situations in which VGG or ResNet perform particularly well. The discussion of the findings' implications for practical applications provides insightful information on how to choose the best architectures for image classification tasks. The analysis adds to the deeper conversation in deep learning and computer vision.

Table 7 Comparative between VGG and Resnet Training

Train	VGG Model	Resnet Model
Train loss	5.53	1.27
Valid-loss	2.27	0.86
accuracy	0.37	0.76
Train-loss	1.98	1.24
Valid -loss	1.08	0.59
accuracy	0.70	0.84

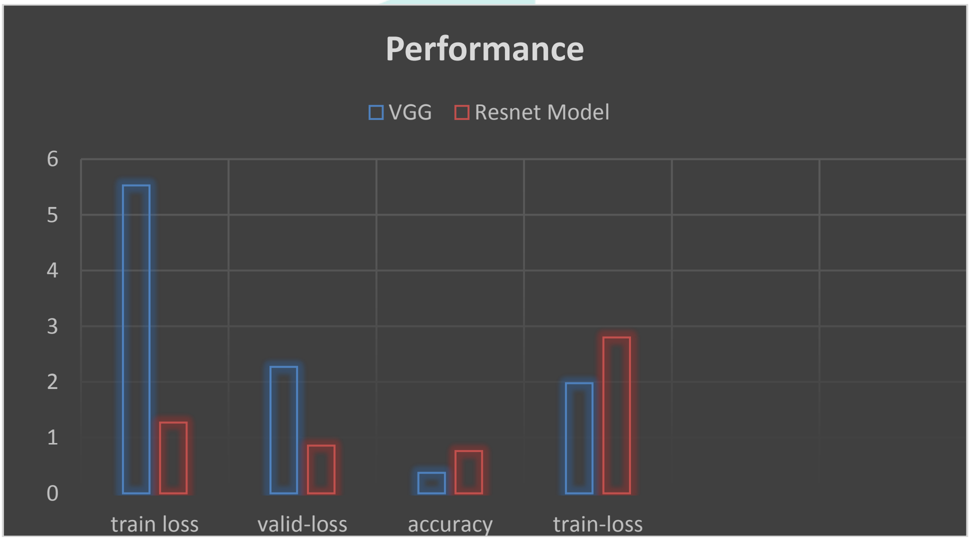


Fig. 11 Comparative between VGG and Resnet

CONCLUSION

This report undertakes a comprehensive evaluation of the vGG and ResNet architectures for image classification tasks using the Oxford Flower Dataset. The primary objectives encompass assessing recall, accuracy, and precision, with a keen focus on discerning the nuanced differences between the two architectures. A stringent methodology is employed, covering model configuration, training parameters, and dataset preparation. The evaluation spans both training and testing phases, facilitating a robust comparative analysis. Results shed light on the distinctive performance characteristics of vGG and ResNet in deep image classification, highlighting their respective strengths and limitations. In conclusion, this analysis contributes valuable insights to the ongoing discourse surrounding optimal convolutional neural network architectures for image classification.

REFERENCES

1. Cıbuk, M., Budak, U., Guo, Y., Cevdet Ince, M. and Sengur, A., (2019) ‘Efficient deep features selections and classification for flower species recognition’. Measurement, 137, pp.7-13.
2. Fchollet, (2016). Release VGG16, VGG19, and ResNet50 · fchollet/deep-learning-models., GitHub.
3. Feng, J., Wang, Z., Zha, M. and Cao, X. (2019) ‘Flower Recognition Based on Transfer Learning and Adam Deep Learning Optimization Algorithm’. Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence - RICAI 2019.
4. He, K., Zhang, X., Ren, S., Sun, J. (2015) ‘Deep Residual Learning for Image Recognition’ 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778
5. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters
6. Liu, Y., Tang, F., Zhou, D., Meng, Y. and Dong, W. (2016) ‘Flower classification via convolutional neural network’. 2016 IEEE International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA), Qingdao, China, pp. 110-116.
7. Lv, R., Li, Z., Zuo, J., Liu, J. (2021) ‘Flower Classification and Recognition Based on Significance Test and Transfer Learning’. 2021 IEEE International Conference on Consumer Electronics and Computer Engineering, ICCECE 2021. Institute of Electrical and Electronics Engineers Inc., pp. 649–652.
8. Mamaev, A. (2021). Flower Recognition, Kaggle [Online]. Available at: https://www.kaggle.com/alxmamaev/flowersrecognition (Accessed: 19 April 2021)
9. Nilsback, M., Zisserman, A. (2008). 102 Category Flower Dataset, Visual Geometry Group [Online]. Available at: https:// www.robots.ox.ac.uk/~vgg/data/flowers/102/ (Accessed: 19 April 2021)
10. Simonyan, K., Zisserman, A. (2015) ‘Very deep convolutional networks for large-scale image recognition’, 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, International Conference on Learning Representations, ICLR, San Diego, California, United States of America.
11. Ong, ZY., Chye, KK., Kang, HW., Tan, CW. (2021) ‘A Flower Recognition System using Deep Neural Network Coupled with Visual Geometry Group 19 Architecture’, International Conference On Digital Transformation And Applications 2021, ICDXA 2021, pp. 121-128 Kuala Lumpur, Malaysia.