A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

**Project Title:**

Comparative Analysis of Deep Learning Architecture from Brain Tumour classification using MRI

Images; Evaluate the Impact Data Augmentation Techniques.

**Student Name and SRN:**

AKSA SIBI ANNAM,22070385

Supervisor: Dr. Alyssa Darke

Date Submitted: 29 August 2024

Word Count: 10,748

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct-v17.0.pdf) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the reportor code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Aksa Sibi Annam

Student Name signature: Aksa Sibi Annam

Student SRN number: 22070385

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ACKNOWLEGGEMENT

I would want to highlight that my post-graduate studies have been an amazing learning experience and to thank everyone who has helped me along the way as I near the completion of my studies.

To begin with, I would want to thank Almighty God for His unending inspiration and blessings, as well as for giving me the bravery and confidence to go forward with confidence and self-belief.

To my supervisor, Alyssa Darke, for all her help and guidance throughout this project, I would like to express my gratitude. I am grateful for her unwavering support and her tolerance for my curiosity.

In addition, I want to thank all of my lecturers at the University of Hertfordshire for their assistance and guidance throughout my course, which enabled me to better understand the material.

This would not have been possible without the unfailing support and encouragement of my parents, siblings, and friends, for which I am also grateful.

ABSTRACT

This project work, perform a comparative study of deep learning models (ResNet101, VGG16 and, Inception v3) for the classification of brain tumors utilizing MRI images towards finding out which model emerges as the best one regarding other techniques related to data augmentation that they may be combined with. This study uses the Kaggle Brain Tumor MRI dataset, which contains 5,712 training images and 1,311 testing images as divided into glioma, meningioma pituitary tumors and non-tumors groups to develop VGG16 model along with pretrained models of (ResNet101and InceptionV3). This study investigates these models and compares the difference in terms of performance metrics including sensitivity to find out their behavior. The study will focus to improve the performance and robustness of all these models by applying data augmentation techniques. The aim is to test which models work best for clinical utility, increasing the accuracy and turn around time of brain tumor diagnoses. It offers insights towards identifying appropriate deep learning architectures and augmentation schemes for brain tumor classification; thus, improving on the research domain of medical image analysis.

“Artificial intelligence has the potential to democratize access to education, healthcare and economic opportunities. Let's strive to make AI technology accessible and beneficial for all.”

##### **-Rana el Kaliouby**

CEO of Affectiva

Contents

[1. Introduction 9](#_Toc175836227)

[2. Review of Literature 11](#_Toc175836228)

[3. Methodology 14](#_Toc175836229)

[3.1 Brief overview 14](#_Toc175836230)

[3.2 Dataset 14](#_Toc175836231)

[3.2.1 Training Dataset 14](#_Toc175836232)

[3.2.2 Testing Dataset 15](#_Toc175836233)

[3.3 Pre-Processing of Data 16](#_Toc175836234)

[3.4 Data Augmentation 16](#_Toc175836235)

[3.4.1 First Data Augmentation: 16](#_Toc175836236)

[3.4.2 Second Data Augmentation: 17](#_Toc175836237)

[3.5 Creating a Training Set and a Validation Set 18](#_Toc175836238)

[3.6 Classification of Models. 19](#_Toc175836239)

[3.6.1 Inception V3 22](#_Toc175836240)

[3.6.2 ResNet101 24](#_Toc175836241)

[3.6.3 VGG 16 25](#_Toc175836242)

[4. Results 29](#_Toc175836243)

[4.1 First data Augmentation result 29](#_Toc175836244)

[4.1.1 Inception V3 29](#_Toc175836245)

[4.1.2 ResNet101 31](#_Toc175836246)

[4.1.3 VGG16 33](#_Toc175836247)

[4.2 Data Augmentation 2: 36](#_Toc175836248)

[4.2.1 Inception V3 36](#_Toc175836249)

[4.2.2 ResNet101 38](#_Toc175836250)

[4.2.3.VGG16 40](#_Toc175836251)

[5. Discussion of Result 43](#_Toc175836252)

[5.1 Classification of models 43](#_Toc175836253)

[5.2 Impact of Data Augmentation in model classification 43](#_Toc175836254)

[5.3 Comparison with other paper 43](#_Toc175836255)

[5.4 Some applications in real life 44](#_Toc175836256)

[5.5 Improvement of models 45](#_Toc175836257)

[5.6 Limitations 45](#_Toc175836258)

[6. Conclusion 47](#_Toc175836259)

[7. Reference 48](#_Toc175836260)

[8. Appendix 50](#_Toc175836261)

List of Figures

[Figure 1.Distribution of Training Data 15](#_Toc175836265)

[Figure 2. Distribution of Testing Data 15](#_Toc175836266)

[Figure 3. First data augmentation 17](#_Toc175836267)

[Figure 4. Second Data Augmentation 18](#_Toc175836268)

[Figure 5. ReLU function 20](#_Toc175836269)

[Figure 6.softmax function 21](#_Toc175836270)

[Figure 7. softmax function 22](#_Toc175836271)

[Figure 8. Training Graph of InceptionV3 for first data augmentation 30](#_Toc175836272)

[Figure 9. Confusion Matrix of InceptionV3 for first data augmentation 30](#_Toc175836273)

[Figure 10. Classification report of InceptionV3 for the first data augmentation 31](#_Toc175836274)

[Figure 11. Graph of ResNet101 for the first data augmentation 32](#_Toc175836275)

[Figure 12. Confusion Matrix of Resnet101 for the first data augmentation 32](#_Toc175836276)

[Figure 13. Classification report of ResNet101 33](#_Toc175836277)

[Figure 14. Graph of VGG16 for the first data augmentation. 34](#_Toc175836278)

[Figure 15. Confusion Matrix of VGG16 for the first data augmentation 34](#_Toc175836279)

[Figure 16. Classification report of VGG16 for the first data augmentation 35](#_Toc175836280)

[Figure 17. Graph of inceptionV3 for the second data augmentation. 36](#_Toc175836281)

[Figure 18. Confusion matrix of inceptionV3 for the second data augmentation 37](#_Toc175836282)

[Figure 19. Classification report of InceptionV3 for second data augmentation 37](#_Toc175836283)

[Figure 20. Graph of ResNet101 for second data augmentation 39](#_Toc175836284)

[Figure 21. Confusion matrix of ResNet101 for the second data augmentation 39](#_Toc175836285)

[Figure 22. Classification report of ResNet101of second data augmentation 40](#_Toc175836286)

[Figure 23. Graph of VGG16 of the second data augmentation 41](#_Toc175836287)

[Figure 24. Confusion Matrix of VGG16 the second data augmentation 41](#_Toc175836288)

[Figure 25. Classification report VGG16 of second data augmentation 42](#_Toc175836289)

List of tables

[Table 1.Architecture of Inception V3 24](#_Toc175836300)

[Table 2. Architecture of ResNet 101 25](#_Toc175836301)

[Table 3. Architecture of VGG16 28](#_Toc175836302)

[Table 4. Result of models for the First data augmentation 29](#_Toc175836303)

[Table 5. Result of models for the Second Data Augmentation 36](#_Toc175836304)

# 1. Introduction

The brain tumor identification is a matter of utmost importance in the fields of medical science and healthcare. The likelihood of a successful course of therapy and recovery is greatly increased by early and accurate identification of brain tumors. Headaches, seizures, cognitive deficits, and physical disabilities are just a few of the neurological symptoms and problems that can result from brain tumors, whether they are benign or malignant. Timely intervention—such as radiation therapy, chemotherapy, or surgical resection—allows for timely diagnosis and stops the tumor from spreading and causing more harm. Çinar and Yildirim (2020) highlighted that precise identification aids in the appropriate categorization and assessment of the tumor, which is crucial for formulating a successful treatment strategy catered to the individual patient's needs. In summary, early diagnosis of brain tumors is critical for both increasing the chances of survival and improving the quality of life for those impacted by the disease by lowering the risk of severe neurological deficits and improving overall health outcomes.

Over the years, advances in medical imaging and diagnostic technology have led to a substantial evolution in the diagnosis of brain tumors. In order to provide an appropriate diagnosis, early approaches frequently relied too heavily on clinical symptoms and simple imaging modalities like X-rays. A major advancement in the viewing of brain regions was made possible by the development of computed tomography (CT) in the 1970s. The 1980s saw the emergence of magnetic resonance imaging (MRI), which further altered this by improving the contrast and resolution of images of soft tissues. A number of researchers conducted groundbreaking work in the use of MRI to detect brain tumors. For example, Hounsfield's invention of the CT scan in 1973—for which he was awarded the Nobel Prize in Physiology or Medicine—created the foundation for more sophisticated imaging methods. Research on the clinical uses of MRI in brain tumor detection, such as that conducted by Holland et al. (1986), by the late 1980s and early 1990s, showed that MRI was superior to CT scans in providing detailed images of brain tumors, allowing for more precise diagnosis and treatment planning. Numerous studies have verified further breakthroughs in imaging technologies. A notable study conducted in 1988 by Stark and Bradley, for instance, described the use of MRI in the diagnosis and treatment of brain tumors and highlighted how well it works to determine the kind, size, and location of the tumor. As highlighted in more recent research by Zoph et al. (2020) and Çinar and Yildirim (2020), the use of artificial intelligence (AI) and machine learning into imaging techniques has improved the accuracy and speed of brain tumor identification in recent years.

Brain tumor detection has a major impact on early diagnosis, treatment planning, and monitoring, and has various important uses in both patient care and medical research. To improve the accuracy of surgical planning and post-operative monitoring, Holland et al. (1986) for example showed how to employ MRI with gadolinium contrast to precisely identify and classify brain tumors. By enabling prompt interventions and early identification using modern imaging techniques such as MRI and CT scans, patient outcomes and survival rates are improved. Furthermore, by combining imaging data with genetic and molecular analysis, customized therapy regimens can be created. Moreover, studies employing AI-driven models for non-invasive tumor classification (Çinar & Yildirim, 2020) demonstrate how routine imaging helps track the advancement of tumors and the effectiveness of treatment. The use of deep convolutional neural networks in multi-classifying MRI images of brain tumors was also investigated by Khan et al. (2021), demonstrating how AI can greatly improve treatment planning and diagnostic accuracy. These developments highlight how important it is to identify brain tumors accurately in order to improve patient outcomes and quality of life.

The superior ability to precisely detect and categorize cancers from MRI images, deep learning models like VGG16, Inception V3, and ResNet101 have become indispensable instruments in the investigation of brain tumors. VGG16, with its deep but straightforward architecture, excels in feature extraction, making it highly effective for identifying different types of brain tumours, including gliomas, meningiomas, and pituitary tumours. This model has a particular significance, because it lowers human error and raises diagnostic precision, this model is especially useful in clinical settings where it can improve early diagnosis and treatment planning. Inception V3 is noted for its complex architecture that allows for the capture of diverse image features through multiple convolution filters. This complexity enables it to handle variations in image quality and size, making it useful in both clinical and research applications. The ability of Inception V3 to differentiate between benign and malignant tumors is very useful for enhancing diagnostic precision and advancing cutting-edge research endeavors. ResNet101, with its deep residual learning framework, mitigates the vanishing gradient problem, enabling the training of very deep networks. This capability allows ResNet101 to detect subtle anomalies and patterns in MRI scans that might be missed by less sophisticated models]. It is perfect for clinical diagnostics and the creation of automated diagnostic tools because to its depth and resilience, which improves the precision and effectiveness of brain tumor identification.

These models are applied across various sectors: in healthcare, they assist radiologists in early and precise diagnosis; in medical research, they support the study of tumour progression and the evaluation of new treatments; and in telemedicine, they enable remote diagnosis and monitoring, making specialized care more accessible[].In general, diagnostic accuracy and patient outcomes are much enhanced when VGG16, Inception V3, and ResNet101 are incorporated into medical imaging procedures. Furthermore, in order to assess how different data augmentation methods affect the improvement of each model's robustness and generalizability, the research will apply these strategies to each model. The study aims to ascertain the most efficacious model for possible clinical implementation by pinpointing the augmentation tactics that enhance model performance. Assuring the models' accuracy and dependability in actual medical situations is part of this. Additionally, by improving diagnostic tools and methods for brain tumor identification and classification, the study hopes to provide important insights into the application of deep learning in medical image analysis.

# 2. Review of Literature

Recent years have witnessed notable breakthroughs in the use of deep learning to medical image analysis, particularly in the categorization of brain tumors using MRI scans. Several deep learning architectures are examined in this literature review, such as VGG-16, ResNet-101, and Inception-v3, which have shown a great deal of promise in improving diagnostic efficiency and accuracy. Studies evaluating the effectiveness of these models in identifying and categorizing brain tumors have emphasized both their advantages and disadvantages. The review also looks at how transfer learning and data augmentation strategies might enhance the performance and durability of models. The review highlights possible directions for more study and development by contrasting different models and approaches. It also offers insights into the state of deep learning applications in brain tumor diagnostics now.

With architectures like VGG-16, ResNet-50, and Inception-v3 demonstrating great potential, recent advances in deep learning have significantly improved the categorization of brain tumors using MRI scans. Because of its depth and efficient feature extraction, VGG-16 had the highest accuracy; ResNet-50 solved the vanishing gradient issue but required more training time (He et al., 2016; Simonyan & Zisserman, 2014). According to Szegedy et al. (2016), Inception-v3 was effective but had stability problems with validation accuracy. Through the diversification and consistency of training samples, data augmentation approaches such as rotation and intensity normalization have improved model robustness (Chollet, 2018). Transfer learning has further enhanced performance and shortened training time by utilizing pretrained models, such as ImageNet, to adapt pretrained weights to new tasks (Talo et al., 2019). With the potential to be further improved utilizing more sophisticated models like EfficientNet, VGG-16 performed exceptionally well overall at balancing accuracy and efficiency (Srinivas et al., 2022).

Çınar et al.'s paper "Comparison of Deep Learning Models for Brain Tumor Classification Using MRI Images" uses a standardized MRI dataset to test five CNN architectures: VGG-19, DenseNet-169, AlexNet, InceptionV3, and ResNet-101. The most accurate model was ResNet-101 (98.6%), followed by VGG-19 (97.2%). ResNet-101 outperforms other models because of its skip connections and deep design, which both solve the vanishing gradient issue. The efficiency of deeper, more complicated models, such as ResNet-101, in brain tumor detection from MRI scans is demonstrated by this study (Çınar et al., 2022).

Pre-trained deep learning models' efficacy in classifying brain MRI images is examined in Krishnapriya and Karuna's paper from 2023. Using transfer learning and data augmentation techniques, the models analyzed are VGG-19, VGG-16, ResNet50, and Inception V3. The study attempts to address the urgent need for rapid and reliable brain tumor identification, as these lesions can have serious consequences if not found in a timely manner. MRI is emphasized for its superior resolution and capacity to identify abnormalities in the brain. By automating feature extraction and decreasing dependence on human feature development, deep learning—particularly CNNs—has completely changed the classification of medical images. With an accuracy of 99.48%, recall of 98.76%, precision of 100%, and an F1 score of 99.17%, the VGG-19 model with transfer learning outperformed the other models, according to the study. Comparing pre-trained models to traditional methods, they show great accuracy with less training data, indicating the significant potential of applying them for brain tumor classification (Krishnapriya & Karuna, 2023).

In their paper from 2024, Mukesh Kumar and Sree Sankar compare the CNN architectures VGG16, ResNet, Inception, and DenseNet for the purpose of detecting brain tumors from MRI scans. With 89% recall, 87% F1 score, 92% accuracy, 92% precision, and 89% recall, VGG16 fared better than other models. Its better performance than ResNet (80%), Inception (84%), and DenseNet (85%) was attributed to its more straightforward architecture with sequential convolutional layers. This study highlights the practical relevance and effectiveness of VGG16 in medical imaging, particularly in situations where computational resources are few (Kumar & Sankar, 2024).

Ullah et al. (2023) have refined pre-trained deep learning models, such as CNN, ResNet101, InceptionV3, VGG16, and VGG19, to achieve better categorization of brain tumors. The models achieved high precision, recall, and F1 scores up to 97% on three public datasets by implementing data augmentation techniques as rotation, scaling, and flipping. The study shows how well these methods work to increase model robustness and performance; VGG16 and InceptionV3 demonstrate noteworthy outcomes. With early tumor discovery, there may be a decrease in the severe impairments brought on by a delayed diagnosis.

To classify brain cancers from MRI scans, Mahmoud et al. (2023) used the Aquila Optimizer (AQO) in conjunction with advanced deep learning models such as VGG-16, VGG-19, and Inception-V3. The VGG-19 model demonstrated the efficacy of integrating CNN models with optimization approaches, achieving the greatest accuracy of 98.95%. This method improves the accuracy of tumor classification, which is important for prompt and precise medical imaging diagnosis.

A generic CNN and six pre-trained models (ResNet50, InceptionV3, InceptionResNetV2, Xception, MobileNetV2, and EfficientNetB0) are among the seven deep convolutional neural network (CNN) models for brain tumor classification that are evaluated in the study by Gómez-Guzmán et al. (2023) utilizing MRI images. Seventy-two MRI pictures total, split into four categories (pituitary, meningioma, glioma, and healthy brains), made up the dataset. The model that performed the best, InceptionV3, had an accuracy of 97.12%, demonstrating how well it could classify brain tumors. This study highlights the potential to improve medical imaging diagnostic accuracy by utilizing sophisticated CNN architectures in conjunction with thorough preparation and data augmentation methods.

Brain tumor identification from MRI pictures has significantly improved thanks to deep learning, especially when employing CNNs like VGG16 and Inception V3. These models perform well because of their strong feature extraction capabilities and excellent accuracy. By making use of pre-trained models, transfer learning improves performance even further by cutting training time and increasing accuracy with little data. Techniques for augmenting data, including rotating and scaling images, increase the resilience of the model. According to studies, models such as VGG16 and Inception V3 can attain accuracy rates nearly equal to 98%, proving its usefulness in facilitating precise and effective identification of brain tumors.

The study of the literature demonstrates the noteworthy progress made in the use of deep learning models to classify brain tumors using magnetic resonance imaging. Certain architectures, such VGG-16, ResNet-50, and Inception-v3, have demonstrated significant potential and have individually improved diagnostic efficiency and accuracy. Though ResNet-50 tackles important problems like the vanishing gradient problem, VGG-16 is lauded for its deep architecture and efficient feature extraction skills. Though it can be unstable at times, Inception-v3 is still very effective at extracting a wide range of features. The use of data augmentation strategies has been essential for enhancing the consistency, diversity, and resilience of the model. With pretrained models in particular, transfer learning has further optimized performance and shortened training timeframes, showing notable gains in accuracy with less data. The comparative study models show that deeper models, such as ResNet-101, frequently perform better, highlighting the significance of model depth and complexity. All things considered, deep learning models—in particular, VGG-16, Inception-v3, and ResNet101—have enormous potential to improve brain tumor diagnosis and open the door to more precise, effective, and early detection techniques in clinical settings.

# 3. Methodology

## 3.1 Brief overview

This project was built in google Collaboratory in Python language using the libraries for the models used. The numpy, pandas, matplotlib, seaborn, os, cv2 (OpenCV), tqdm. the libraries tenserflow, keras and sklearn were used for model development. The next crucial step was data preparation, which involved preparing the data for model training by performing operations like scaling and normalization. To improve the model's resilience and capacity for generalization, data augmentation methods like rotation, width and height shifts, zooming, shearing, and horizontal flipping were used on the training images. Subsequent augmentation was implemented to enhance the model's functionality. The creation of a VGG16 model from scratch marked the beginning of model construction. In order to take use of their sophisticated architectures and previously acquired features, pre-trained models like ResNet101 and Inception V3 were also used. These models were trained using the prepared data, and their parameters were adjusted to maximize efficiency. In order to avoid overfitting and guarantee optimal model performance, the training procedure comprised several epochs and batch sizes, and model checkpointing. Plotting graphs revealed information about the models' learning progress and efficacy by tracking and visualizing the models' accuracy and loss. Overall, by applying sophisticated deep learning techniques and data augmentation tactics, this effort shows how to classify brain tumors using MRI scans with great accuracy. Deep learning's potential to improve medical imaging diagnostic tools is demonstrated by the mix of pre-trained and custom-built models.

## 3.2 Dataset

The "Brain Tumor MRI Dataset" created by Masoud Nickparvar on the Kaggle platform is where the datasets used in this study were found. The training dataset and the testing dataset are the two primary components of this dataset, which consists of MRI pictures of brain tumors. Gliomas, meningiomas, pituitary tumors, and no tumor are the four classes into which the photos are divided([MRI dataset](mailto:https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset?select=Training)).

### 3.2.1 Training Dataset

The images in the training dataset are utilized to educate the deep learning models. The data required for the models to learn and distinguish between different kinds of brain tumors and healthy brain scans is provided by the labels applied to each image, which correspond to their respective categories. Glioma, meningioma, pituitary tumor, and no tumor are the four categories into which the photos are arranged within Google Drive. The data consists of 5700 images. The count representing graph is show below:

A graph of classes in training data

Description automatically generated

Figure 1.Distribution of Training Data

### 3.2.2 Testing Dataset

The testing dataset is only used to assess the model's performance; it contains images that are comparable to those in the training set. This dataset makes it possible to evaluate the trained models' ability to generalize to new, untested data objectively. The testing dataset, like the training dataset, is made up of labeled photos that are kept in matching directories. The data consists of 1311 images. The graph indicating the number of images in this set is given below:

A graph of a number of classes

Description automatically generated

Figure 2. Distribution of Testing Data

## 3.3 Pre-Processing of Data

Initially, the JPG were obtained by extracting the zipped file thar was acquired from the Kaggle website containing the photos. After that, these files were uploaded to Google Drive, so that it can be easily accessed. Resizing the photos, standardizing the pixel values, and translating the image labels into a appropriate format for model training were among the preprocessing activities involved in data preparation. The Pre-processing the data is an essential step in our study to guarantee that the deep learning models work as well they can on the brain tumor classification objective. To improve the model's capacity for generalization and to broaden the dataset's diversity, a variety of techniques, including rotation, shifting, zooming, and flipping, are applied to the training images at this phase. In order to normalize pixel values and place them into a consistent range of [0, 1], both training and testing images are rescaled. The model's capacity to learn from the features in the photos is enhanced by this normalization, which makes sure the data is processed effectively.

## 3.4 Data Augmentation

In image processing applications like brain tumor classification, where it is critical to train models on a dataset that captures as much variability as possible, data augmentation is very significant. The model's ability to identify cancers under various circumstances and scenarios is improved by employing data augmentation, which eventually results in more precise and trustworthy predictions. Two methods of data augmentation were used in the brain tumor classification project to improve the performance and generalization capacities of the model. By creating fresh, different samples from the current dataset, these strategies sought to increase the diversity of the training data, which would aid in the models' learning of more robust features and lower the possibility of overfitting.

### 3.4.1 First Data Augmentation:

Using the *‘ImageDataGenerator’* class from the Keras library, the first data augmentation method in the project was put into practice. The training photos were subjected to a few changes using this technique, which increased the dataset's diversity and improved the model's resilience and generalization. The subsequent adjustments were made:

A screen shot of a computer program

Description automatically generated

Figure 3. First data augmentation

1. Rescaling: Pixel values were normalized to fall between 0 and 1 by rescaling each one by a factor of 1/255. This typical preprocessing stage aids in the model's quicker training and improved performance.
2. Rotation: A 40-degree rotation was applied at random to the photos. As a result, the model was able to identify brain tumors of any orientation.
3. Width and Height shift: Up to 5% of the overall width and height of the photos could be moved both vertically and horizontally. By simulating minute movements or displacements in the image, these adjustments improved the model's resistance to such alterations.
4. Zoom Range: Random zooms of up to 10% were applied to the photographs. Because of this change, the model was able to identify tumors at various scales, regardless of how big or little they appeared on the MRI scan.
5. Shear Range: The images were tilted slightly and sheared by up to 10% to replicate variances in how the pictures might be taken in the actual world.
6. Horizontal Flip: The pictures had been horizontally flipped. As a result, mirror versions of the original data were produced, adding to the dataset's diversity.
7. Vertical flipping: Vertical flipping which turns the photos upside down along the vertical axis is not used. By producing a mirror image along the vertical axis, this transformation offers an extra data alternative.
8. Fill Mode: To maintain the augmented images' visual coherence, *“fill\_mode='nearest’” was* applied to fill in any missing pixel values with the closest pixel when executing transformations that moved pixels.

### 3.4.2 Second Data Augmentation:

In order to perform the second data augmentation on photos during the training of deep learning models, the code snippet you gave is a setting for ‘Keras' ImageDataGenerator class’. The following provides an explanation for each parameter that the *‘ImageDataGenerator’* uses:

A screen shot of a computer code

Description automatically generated

Figure 4. Second Data Augmentation

1. Rescaling: In order to standardize the data and increase the stability of the model training process, this is frequently done. The images' pixel values are scaled from a range of 0-255 to a range of 0-1.
2. Roration range: During training, this randomly rotates the photos by up to 20 degrees, which aids in the model's invariance to the object orientation in the images
3. Width Shift Range: This causes an arbitrary horizontal shift of the images of up to 20% of their width. This augmentation aids in the model's learning to identify items that are not precisely centred.
4. Height Shift Range: As with ‘width\_shift\_range’, this vertically moves the images by a maximum of 20% of their height.
5. Zoom Range: This applies a 20% random zoom in or out to the photos. This aids the model's ability to identify things at various scales.
6. Shear Range: This applies a shear transformation to the images, distorting them along an axis. This can make the model more robust to geometric transformations.
7. Horizontal Flip: The photos are then randomly flipped horizontally. This augmentation is particularly helpful for photographs of symmetrical objects or some medical images when the left-right orientation is irrelevant.
8. Vertical Flip: This causes the images to flip vertically at random. This aids in the model's orientation invariance, much like inverting it horizontally.
9. Fill Mode: This describes how new pixels that result from transformations like shifts, rotations, or zooms should be filled in. The *'reflect*' mode fills these pixels with the border of the picture reflected in them.

Using this configuration, it is able produce a broad range of augmented images, which will assist your deep learning models become more broadly applicable. Because medical imaging datasets are frequently tiny, the application of these techniques is especially crucial, as augmentation serves to broaden the training data's diversity.

## 3.5 Creating a Training Set and a Validation Set

The next stage is to generate the training and validation datasets after data augmentation and rescaling. To do this, batches of picture data with real-time data augmentation are generated using the ‘ImageDataGenerator’ class. To preserve the integrity of the testing procedure, only rescaled, augmentation-free photos are used in the validation set, while the augmented images are used in the training set.

* The directories holding the training and test pictures are where’ training\_set’ and ‘test\_set’ are generated. Every image is downsized to 224 by 224 pixels, which is the same size as the input that the deep learning models (such as VGG16, ResNet101, and Inception V3) expect.
* The batch\_size is set to 16, which means that during training and validation, 16 photos are handled concurrently. A compromise is struck between memory consumption and computational efficiency with this batch size.
* 'Class\_mode' is set to 'categorical', meaning that the output labels are one-hot encoded, suitable for jobs involving several classes of categorization.

The deep learning models (ResNet101, VGG16, and Inception V3) are then trained on the enriched training data and their performance is assessed on the validation set, after which the training and validation sets are prepared. The models are trained via the fit method after being constructed with an appropriate optimizer and loss function. The efficacy of each architecture will be assessed during this process by tracking important performance indicators like accuracy, sensitivity, and specificity.

In summary, improving the model's capacity to generalize to new data requires completing the data pre-processing stages of augmentation and rescaling. Robust training requires that the models be exposed to a variety of normalized data, which is ensured by creating the training and validation datasets using the’ ImageDataGenerator’ class. The models will be trained, their parameters adjusted, and their performance assessed to determine the best architecture for classifying brain tumors.

## 3.6 Classification of Models.

The research focuses on classifying brain tumor MRI scans into four separate categories: glioma, meningioma, pituitary tumor, and no tumor. It does this by applying three sophisticated deep learning algorithms: ResNet101, InceptionV3, and VGG16-based Convolutional Neural Network (CNN). Using a deep convolutional architecture with numerous layers to extract complex characteristics from the images, the VGG16 model was constructed from the ground up. Detailed feature learning is made easier by this model's simple yet sophisticated architecture. ResNet101 and InceptionV3, on the other hand, are pre-trained models that have been adjusted for this particular classification job. Using Inception modules, which allow the model to capture characteristics at different scales and increase efficiency and effectiveness in identifying complicated patterns, makes InceptionV3 stand out. The vanishing gradient issue is addressed by ResNet101 with its residual connections, which enables the training of much deeper networks and enhances feature extraction and classification accuracy. In order to improve automated medical diagnosis and treatment planning, the project seeks to identify the most efficient algorithm for the robust and accurate categorization of brain tumors by utilizing the advantages of these various architectures.

**Activation Functions** used in these models they are theNeural networks r to add non-linearity to the model, which allows it to learn and represent complex patterns that go beyond simple linear correlations. Activation functions enable neural networks to stack many layers and collect hierarchical representations of data at each layer. This is crucial for resolving complicated tasks like natural language processing and picture recognition. Furthermore, because activation functions regulate neuronal output, they can be used for a variety of tasks, such as regression, where the outputs are continuous values, and classification, where the outputs are interpreted as probabilities. They are also essential for enhancing learning dynamics, helping to avoid problems like the vanishing gradient problem and enabling quicker and more reliable training. Neural networks would essentially be unable to handle non-linear data found in the actual world if they were only able to conduct linear transformations in the absence of activation functions. The activation functions used in these models are the ReLU(Rectified Linear Unit) and the Softmax activation functions.

**i. ReLU:**

Rectified Linear Unit (ReLU) is the one of the activation functions that is most frequently employed in deep learning, especially in convolutional neural networks (CNNs). A straightforward function that outputs zero otherwise and the input directly if it is positive defines it. ReLU has a low computational overhead due to this simple procedure, which is important for deep network training. The ReLU mathematical function defines as:

ReLU(x) = max(0,x)

The plot of ReLU function is given in Fig.5

A graph with a blue line

Description automatically generated

Figure 5. ReLU function

[source:https://images.app.goo.gl/ytd2rBda6zVxWPW86](mailto:https://images.app.goo.gl/ytd2rBda6zVxWPW86)

In addition, ReLU maintains a gradient of 1 for positive inputs, which permits efficient backpropagation and solves the vanishing gradient issue shared by sigmoid and tanh activation functions. ReLU also promotes sparse activation, which involves activating only a portion of neurons, which increases model efficacy and lowers the risk of overfitting.

**ii. Softmax:**

When solving multi-class classification issues, neural networks' output layer frequently uses the softmax activation function. It enables the model to assign a likelihood to each class by converting a vector of raw scores (logits) into probabilities. Each input score is first exponentiated by the softmax function, which then normalizes the results by dividing by the total of all exponentiated scores. Mathematically the function is defined in the fig.6:

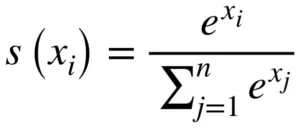


Figure 6.softmax function

[source:https://miro.medium.com/v2/resize:fit:600/format:webp/0\*fbg5QEc2Lv8IIKcq.png](mailto:https://miro.medium.com/v2/resize:fit:600/format:webp/0*fbg5QEc2Lv8IIKcq.png)

Here, x is the input vector and xi represent each element in the input and S(xi) represent the output vector.

The graph of Softmax function is shown in Fig.7

A graph of a function

Description automatically generated

Figure 7. softmax function

[source:https://cdn.botpenguin.com/assets/website/Softmax\_Function\_07fe934386.png](mailto:https://cdn.botpenguin.com/assets/website/Softmax_Function_07fe934386.png)

Softmax have the ability to assign probabilities to various classes makes it especially helpful in situations when the model needs to produce predictions that are comprehensible and straightforward. Softmax is an excellent choice for gradient-based optimization in training since it is a smooth and differentiable function. The purpose of categorizing inputs into one of several possible classes is a common use for it, including text and image classification.

### 3.6.1 Inception V3

The research makes use of the robust and effective InceptionV3 deep learning architecture, which has been optimized for the particular purpose of brain tumor classification after being pre-trained on the ImageNet dataset. The foundation of InceptionV3's design is the idea of "Inception modules," which apply convolutions of various sizes (1x1, 3x3, and 5x5) in parallel to enable the model to learn from several scales at once. The model is better able to identify and categorize the many patterns seen in brain tumor MRI images thanks to its multi-scale approach, which aids in the acquisition of a broad variety of spatial variables.

The input layer of the model accepts images with dimensions of 224x224x3. It then has multiple convolutional layers and max-pooling layers that are intended to minimize spatial dimensions while maintaining important features. Several Inception modules make up the model's core, and to guarantee stability and non-linearity, batch normalization and activation layers come after each module. Concatenating these modules results in mixed layers that incorporate the multi-scale features that were recovered by concurrent convolutional procedures.

The model employs a Global Average Pooling layer to minimize the spatial dimensions after the final Inception module, and a dropout layer to stop overfitting. The learning capability of the model is increased by adding a fully connected dense layer with 1,024 units and ReLU activation. Glioma, meningioma, pituitary tumor, and no tumor are the four categories into which the images are classified by the final output layer using a softmax activation function.

The InceptionV3 model retains the features it learnt from the ImageNet dataset by freezing the lower layers during training, leaving only the top layers of the model refined. While keeping high accuracy, this transfer learning method dramatically reduces training time. Assuring strong performance for multi-class classification problems, the model is constructed using the Adam optimizer and the categorical cross-entropy loss function. Combining efficiency and complexity, the InceptionV3 model has about 23.9 million parameters, of which about 2.1 million may be trained. This makes it an excellent choice for the difficult task of classifying brain tumors.

The InceptionV3 model's architecture is summarized in the following table, which was utilized for the project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer Type** | **Output Shape** | **Number of Parameters** | **Activation Function** | **Notes** |
| Input | (224,224,3) | 0 | nil | Input images 224x224 RGB images |
| Convolution 2D + Batch Normalization + ReLU | (111, 111, 32) | 960 | ReLU | 3x3 convolution with 32 filters |
| Convolution 2D + Batch Normalization + ReLU | (109, 109, 32) | 9,312 | ReLU | 3x3 convolution with 32 filters |
| Convolution 2D + Batch Normalization + ReLU | (109, 109, 64) | 18,624 | ReLU | 3x3 convolution with 64 filters |
| 2D Maxpooling | (54, 54, 64) | 0 |  | 3x3 maxpooling |
| Convolution 2D + Batch Normalization + ReLU | (52, 52, 192) | 143,616 | ReLU | 3x3 convolution with 192 filters |
| 2D Maxpooling | (25, 25, 192) | 0 |  | 3x3 maxpooling |
| Inception Modules(mixed 0-7) | Various | ~5.4 million | ReLU | A sequence of Inception modules including parallel pooling and convolutions |
| 2D Maxpooling | (5, 5, 768) | 0 |  | 3x3 maxpooling |
| Inception Modules (Mixed8-10) | Various | ~12.5 million | ReLU | More filter-equipped Inception modules |
| Global Average Pooling2D | (2048) | 0 |  | Reduce feature map to one vector |
| Dropout (0.5) | (2048) | 0 |  | Overfitting prevention using a dropout layer |
| Dense + ReLU | (1024) | 2,098,176 | ReLU | Layer having 1024 units that is fully connected |
| Dense + Softmax | (4) | 4,100 | Softmax | Four-class classification output layer using softmax activation |
| |  | | --- | | **Total Parameters** |  |  | | --- | |  | |  | 23,905,060 |  | **Trainable**: 2,102,276, **Non-trainable**: 21,802,784 |

Table 1.Architecture of Inception V3

### 3.6.2 ResNet101

ResNet101, which stands for "Residual Network with 101 layers," is a very advanced deep learning model created to address the difficulties involved in training extremely deep neural networks. ResNet101, created by Microsoft researchers as a member of the ResNet family, tackles a major difficulty in deep learning: the vanishing gradient problem, which impairs networks' capacity to learn as they go deeper. This issue arises when the gradients employed in backpropagation get too modest. This causes the learning process to stall because the weights are not updated much. ResNet101 overcomes this by using a groundbreaking technique, residual learning. The key innovation in ResNet101 is the introduction of residual blocks, where each block contains shortcut connections (also known as skip connections) that bypass one or more layers. Instead of directly learning a mapping from input to output, these blocks learn the residual, or the difference between the input and the desired output. This approach simplifies the learning process for the network, allowing it to focus on refining the residuals rather than relearning the entire input-output mapping at each layer.

ResNet101 is used as a pre-trained model in this project; its weights were first discovered using the ImageNet dataset. Once the network is optimized for the particular purpose of brain tumor classification, it is trained to classify MRI images into four groups: no tumor, pituitary tumor, glioma, and meningioma. The model can use its deep feature extraction capabilities while adjusting to the new classification goal by freezing the pre-trained layers and adding a global average pooling layer followed by a dense softmax layer. In particular, it is particularly well-suited for complicated diagnostic applications like tumor classification. The outcome is a highly accurate and efficient model that excels at recognizing minute patterns in medical images. The ResNet101 model demonstrates the strength and adaptability of residual learning, solidifying its place as a mainstay in contemporary deep learning architectures by maintaining excellent performance without experiencing the degradation problems typical of deep networks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer Type** | **Output Shape** | **Number of Parameters** | **Trainable** | **Summary** |
| Input | (224,224,3) | 0 | No | Input layer for 224x224 RGB images |
| ResNet 101(Functional) | (7, 7, 2048) | 42,658,176 | Non-Trainable | Pre-trained model, layers are frozen |
| Global Average 2D Pooling | (2048) | 0 | No | Reduces feature map to single vector |
| Dense | (4) | 8,196 | Yes | Fully connected layers for classification of four class of brain tumors using softmax |
| **Total Parameters** |  | 42,666,372 |  | **Trainable**: 8,196 (32.02 KB**), Non-trainable**: 42,658,176 (162.73 MB) |

Table 2. Architecture of ResNet 101

### 3.6.3 VGG 16

The computer vision area has greatly benefited from the deep convolutional neural network architecture known as VGG16. Known for its simplicity and efficiency, VGG16 was created by Oxford's Visual Geometry Group (VGG) and uses modest 3x3 convolutional filters constantly across the network. Thirteen convolutional layers, five max-pooling layers, three fully connected layers, and a softmax output layer make up the architecture's sixteen layers. In order to enable the model to capture progressively more complicated features as the spatial dimensions decrease, VGG16 is organized into successive blocks where the number of filters doubles after each max-pooling layer. VGG16's consistent architecture, which facilitates implementation and modification, is one of its main advantages. Using VGG16 as the foundational model, four kinds of brain tumors are identified in this project: glioma, meningioma, pituitary tumor, and no tumor. The model's deep design makes it a potent tool for medical image analysis since it enables it to attain high accuracy in image classification tasks when supplemented with regularization approaches like batch normalization and dropout. VGG16 is still a well-liked option because of its simple design and demonstrated performance, even with its comparatively high amount of parameters.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer Type** | **Output Shape** | **Number of Parameters** | **Trainable** | **Activation Function** | **Notes** |
| Input | (224,224,3) | 0 | Yes | - | Input layer for 224x224 RGB images |
| Conv2D | (224,224,64) | 1792 | Yes | ReLU | 3x3 convolution,64 filters, padding= ‘same’ |
| Conv2D | (224,224,64) | 36,928 | Yes | ReLU | 3x3 convolution,64 filters, padding= ‘same’ |
| MaxPooling2D | (112,112,64) | 0 | No | - | 2x2 MaxPooling, 2 stride |
| Batch Normalization | (112,112,64) | 256 | Yes | - | Normalizes activations to expedite training |
| Conv2D | (112,112,128) | 73,856 | Yes | ReLU | 3x3 convolution, 128 filters, padding = ‘same’ |
| Conv2D | (112,112,128) | 147,584 | Yes | ReLU | 3x3 convolution,128 filters, padding = ‘same’ |
| MaxPooling2D | (56,56,128) | 0 | No | \_ | 2x2 Maxpooling, 2 strides |
| BatcNormalization | (56,56,128) | 512 | Yes |  | Normalizes activations to expedite training |
| Conv2D | (56,56,256) | 295,168 | Yes | ReLU | 3x3 convolution, 256 filters, Padding = ‘same’ |
| Conv2D | (56,56,256) | 590,080 | Yes | ReLU | 3x3 convolution,256 filters, Pading = ‘same’ |
| Conv2D | (56,56,256) | 590,080 | Yes | ReLU | 3x3 convolution, 256 filters, Padding = ‘same’ |
| Maxpooling2D | (28,28,256) | 0 | No | - | 2x2 MaxPooling, 2 strides |
| BatchNormalization | (28,28,256) | 1,024 | Yes | - | Normalizes activations to expedite training |
| Conv2D | (28,28,512) | 1,180,160 | Yes | ReLU | 3x3 convloution, 512 filters, Padding = ‘same’ |
| Conv2D | (28,28,512) | 2,359,808 | Yes | ReLU | 3x3 convolution, 512 filters, Padding = ‘Same’ |
| Conv2D | (28,28,512) | 2,359,808 | Yes | ReLU | 3x3 convolution, 512 filters, padding = ‘same’ |
| MaxPoooling 2D | (14,14,512) | 0 | No | - | 2x2 MaxPooling, 2 strides |
| BatchNormalization | (14,14,512) | 2,048 | Yes | - | Normalizes activations to expedite training |
| Flatten | (100352) | 0 | No | - | Flatten the input |
| Dense | (512) | 51,380,736 | Yes | ReLU | Fully connected layers with 512 units |
| Dropout(0.2) | (512) | 0 | No |  | The dropout layer prevent overfitting |
| Dense | (512) | 262,656 | Yes | ReLU | Fully connected layers with 512 units |
| Dropout(0.2) | (512) | 0 | No |  | The dropout layer prevent overfitting |
| Dense | (4) | 2,052 | Yes | ReLU | Four output class and final classification layer |
| **Total Parameters** |  | 59284548 |  |  | **Trainable**: 59282628, **Non-trainable**:1920 |

Table 3. Architecture of VGG16

# 4. Results

In this project, three models are used: Inception V3, ResNet101, and VGG16. Each model is trained twice using different methods of data augmentation. The best-performing iterations of each model trained for the project are displayed in Tables 4 and 5, together with information on their accuracy, length of training, and predictive quality.

## 4.1 First data Augmentation result

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy** | **Training time**  **(s)** | **Predictive quality** |
| Inception V3 | 92 | 4261.105  (~1.1836 hrs) | Excellent |
| ResNet 101 | 72 | 27,997  (~7.7769hrs) | Average |
| VGG 16 | 63 | 17812.865  (~4.94 hrs) | Below Average |

Table 4. Result of models for the First data augmentation

### 4.1.1 Inception V3

The InceptionV3 model has demonstrated remarkable efficacy in categorizing pictures of brain tumors, as seen by its training metrics, validation metrics, confusion matrix, and classification report. The model's ability to learn and distinguish between distinct tumor types was demonstrated by the training accuracy, which increased gradually over 25 epochs from 74.72% to 90.77%. The validation accuracy rose quickly from 76.96% to 90.24% by the last epoch, much like the training accuracy did. The congruence of training and validation accuracy suggests that the model has well-generalized to new data, with minimal signs of overfitting. Additionally, as training went on, the validation and training losses decreased gradually, indicating that the model's accuracy and confidence in its predictions grew.A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Figure 8. Training Graph of InceptionV3 for first data augmentation

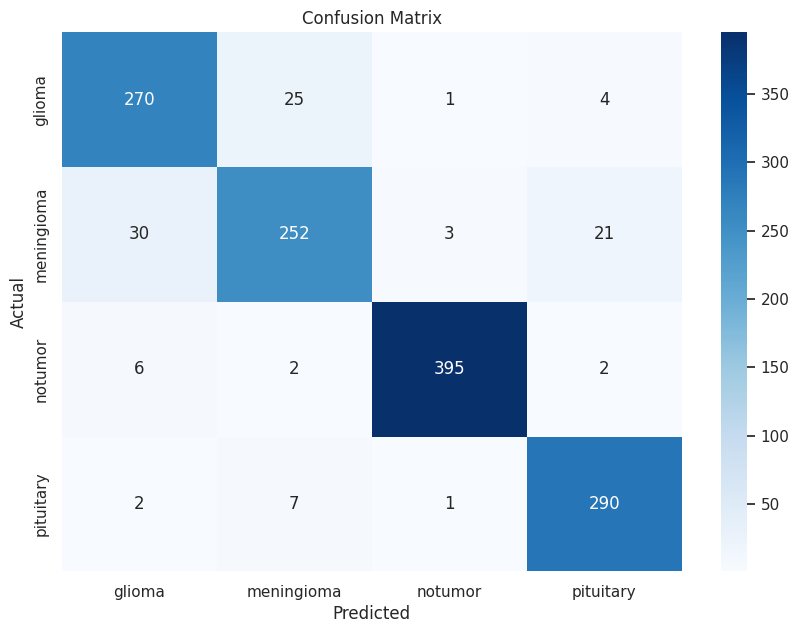


Figure 9. Confusion Matrix of InceptionV3 for first data augmentation

The confusion matrix further demonstrates the effectiveness of the concept across different classes. Out of 300 images of gliomas, the InceptionV3 model correctly identified 270 of them, with only a few misclassifications into meningiomas and pituitary tumors. Meningioma had similar results, with 252 out of 306 scan images correctly recognized; the other images were primarily mislabeled as gliomas and pituitary tumors. 395 out of 405 instances in which there was no tumor were correctly identified by the model, demonstrating its great ability to discriminate between photos with and without tumors. Pituitary tumor classification also shown a high degree of accuracy, with only 290 out of 300 patients receiving the proper classification and very few caeses of misclassification.

A screenshot of a computer screen

Description automatically generated

Figure 10. Classification report of InceptionV3 for the first data augmentation

The classification report provides an extensive breakdown of the model's performance metrics. The model produced very high precision values for all classes; the tumor class had the greatest precision value, 0.99, indicating that nearly all of the predictions were correct. Glioma, pituitary, and meningioma cancers all showed remarkable precision, proving the model's high level of accuracy across a range of tumor types. With a recall value of 0.98 for notumor leading, the model successfully identified almost all cases of notumor. Meningioma memory was comparatively lower at 0.82, indicating that this particular tumor type still needs to be detected with greater accuracy. Glioma and pituitary tumors both showed good recall. Every class consistently has strong F1 scores, demonstrating the model's

Overall, with an amazing 92% accuracy rate, the InceptionV3 model did a good job at classifying brain tumor images. The close alignment of precision, recall, and F1 scores across all classes demonstrates the model's dependability and resilience. The weighted and macro averages of 0.92 for precision, recall, and F1 score suggest that the model performs consistently well across all classes with no obvious shortcomings. Although there is much room for improvement in the classification of meningioma patients, overall, the model performs remarkably well, making it a reliable diagnostic tool.

### 4.1.2 ResNet101

Training and validation measures show that the ResNet101 model has performed more consistently and steadily over the course of 50 epochs. The validation accuracy followed a similar pattern, peaking at 72.85% by the conclusion of the last period after beginning at 56.98%. Starting at 52.63%, the training accuracy increased steadily to 74.19%. Overall, this trend suggests that the model learned the training data effectively. However, there are occasional fluctuations in the validation accuracy, which may indicate that the model had issues with samples. While the validation loss shows slight variances between datasets, the training and validation losses both decreased with time, indicating that the model became more confident in its predictions.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Figure 11. Graph of ResNet101 for the first data augmentation

A blue squares with white text

Description automatically generated

Figure 12. Confusion Matrix of Resnet101 for the first data augmentation

The confusion matrix reveals the ability of the model to classify every type of tumor. With 191 out of 300 glioma images correctly classified, the model did rather well in this regard; however, misclassifications occurred, particularly with meningiomas, where 68 scans were incorrectly classified. Similarly, only 134 of the 306 pictures of meningiomas could be accurately diagnosed by the model; many of them were mistakenly classified as tumors or gliomas. On the other hand, the model demonstrated exceptional performance in successfully classifying images of tumors, correctly classifying 341 out of 405 images, and correctly classifying 277 out of 300 images of pituitary tumors.

A screenshot of a computer screen

Description automatically generated

Figure 13. Classification report of ResNet101

The merits and drawbacks of the model are further emphasized in the categorization report. The model's notumor class (0.82) and pituitary class (0.70) achieved the best precision, demonstrating that these classes were frequently correctly predicted. The difficulty in differentiating between these comparable tumor types is reflected in the lower precision for meningiomas and gliomas. Meningiomas had a recall of only 0.44, demonstrating the challenge of correctly identifying this class, but pituitary tumors had a recall of 0.92, indicating that the model successfully identified majority of these cases. The model's difficulties with these more challenging classifications were further highlighted by the F1 scores, which were highest for tumor (0.83) and pituitary (0.79), glioma at 0.67, and meningioma at 0.50.

The ResNet101 model achieves an accuracy rate of 72% in the categorization of brain tumors, which is good overall performance. The model performs remarkably well in cases with notumor and pituitary, however misclassifications are more common in gliomas, notably meningiomas. The steady accuracy improvement over epochs demonstrates effective learning, even while the performance variability suggests room for improvement. To further enhance the model's performance, particularly in identifying more similar tumor types, extra phases such as hyperparameter tuning, dataset balancing, or advanced data augmentation techniques should be considered. These adjustments could improve the model's accuracy and generalizability across all tumor types, strengthening its diagnostic utility.

### 4.1.3 VGG16

The model's performance across a range of criteria is presented in detail in the VGG16 model training results, illustrating both its benefits and drawbacks. Training accuracy increases steadily over the course of ten epochs, reaching a peak of 59.49% in the last epoch after starting at 48.33% in the first. This steady increase indicates that new data is being effectively absorbed by the model from the training set. Furthermore, there are notable fluctuations in the validation accuracy as well; it begins in the first epoch at a low of 27.84%, peaks at 62.93% in the ninth epoch, and then gradually drops to 58.58% by the tenth epoch. These changes imply that the model may not be generalizing well to new data even during the learning phase because of overfitting during some training sessions.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Figure 14. Graph of VGG16 for the first data augmentation.

Although there is an overall downward trend, there are noticeable spikes in certain epochs of the validation loss, which also reflects this instability. This unpredictable behavior highlights how challenging it is to get consistent performance across multiple data sets and suggests that the model is either overfitting to the training set or not capturing the complexity of the data distribution. The model is improving its internal representations, as seen by the training loss, which dropped over the course of epochs from 2.0266 to 0.9560. But there were variations in the validation loss, suggesting that these gains would not necessarily translate into better results on new data.

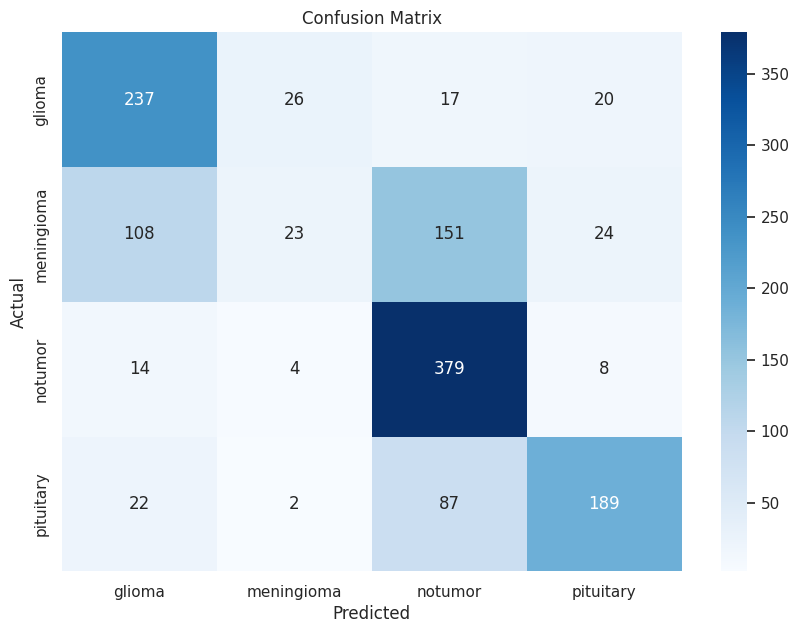


Figure 15. Confusion Matrix of VGG16 for the first data augmentation

The confusion matrix provides a deeper knowledge of how various classes are handled by the model. It is found that there is a noticeable imbalance in the model's ability to classify different types of brain tumors appropriately. For instance, the model correctly identifies most non-tumor pictures, as seen by its high recall of 94% in "notumor" detection scenarios. However, in "meningioma" cases, it performs abjectly, recalling only 8 percent of the cases. This disparity raises the possibility that the model is biased and that some classes are being favored over others due to an imbalance in the training data or inherent difficulties in distinguishing between similar tumor kinds.

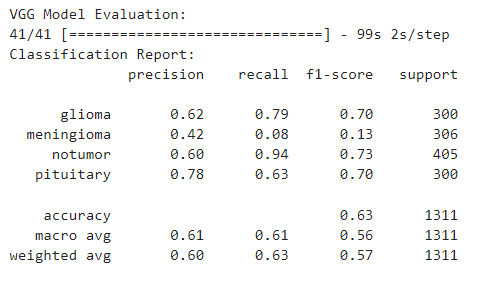


Figure 16. Classification report of VGG16 for the first data augmentation

The classification report describes this discrepancy in more detail. The model achieves a high precision for the "glioma" and "pituitary tumor" classes, with precision scores of 0.62 and 0.78, respectively. This implies that when the model makes a forecast, it usually correctly identifies these classifications. However, "meningioma" has a very low recall rate (0.08), which negatively impacts the class's overall score. The F1 scores, which strike a balance between recall and precision, represent this distinction. Meningioma scores a pituitary tumor and glioma score far lower, at 0.13 and 0.70, respectively. This disparity draws attention to the model's shortcomings for certain tumor types, which are brought about either by the visual similarities between them or by the dearth of representative samples in the training set.

The model reveals its variable performance across classes with an overall modest accuracy rating of 63%. Given that the model does not perform consistently well across all classes, training and data balancing still need to be improved, as indicated by the global average F1 score of 0.56 for the model. Moreover, a large amount of processing power is needed for each training period, suggesting that the model's architecture is complex and could benefit from optimization techniques.

## 4.2 Data Augmentation 2:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy**  **(%)** | **Training time**  **(s)** | **Predictive quality** |
| Inception V3 | 86 | 6,849s  (1.9025hrs) | Very good |
| ResNet 101 | 64 | 24,196.206s  (6.7211hrs) | Below Average |
| VGG 16 | 71 | 23,813s  (6.6147hrs) | Average |

Table 5. Result of models for the Second Data Augmentation

### 4.2.1 Inception V3

The InceptionV3 model, which was trained using the second data augmentation strategy, excels in classifying images of brain tumors. The training and validation accuracy of the model increased steadily over the course of 25 epochs, from initial values of 69.65% and 79.56%, respectively, to 84.88% and 85.35% at the final epoch. The model learns to distinguish between the distinct classes effectively, as evidenced by the steady increase, even though the congruence between training and validation accuracy shows decent generalization to unknown data. Throughout the training, there was a noticeable decrease in both the training and validation losses; the former went from 0.8394 to 0.4021, while the latter went from 0.5384 to 0.3490. This trend indicates that as training progressed, the model's accuracy and confidence rose, indicating that it improved its ability to make precise predictions.

A graph of a training and training accuracy

Description automatically generated with medium confidence

Figure 17. Graph of inceptionV3 for the second data augmentation.

A screenshot of a graph

Description automatically generated

Figure 18. Confusion matrix of inceptionV3 for the second data augmentation

The confusion matrix provides detailed information about the model's performance with respect to different types of tumors. Of the 300 glioma images, the model accurately identified 249; the remaining cases were misclassified as meningiomas and pituitary tumors. The model successfully detected 208 images of meningioma out of 306 images; however, some images were mislabeled as pituitary tumors or gliomas. The model displayed reliability in distinguishing between images including and lacking tumors by accurately identifying 388 out of 405 images, demonstrating excellent performance in classifying cases that were not cancers. Likewise, with relatively few errors, the model identified 288 out of 300 images of pituitary tumors with remarkable accuracy. These findings demonstrate the model's robustness across most tumor forms, even though there is still room for improvement in terms of discriminating between closely similar classifications such as glioma and meningioma.

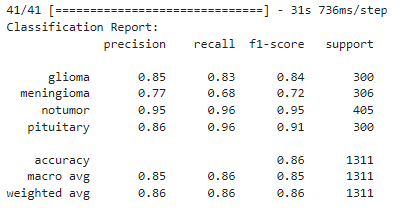


Figure 19. Classification report of InceptionV3 for second data augmentation

The classification report, which shows strong recall and precision scores for every class, offers more proof of the model's effectiveness. Meningioma and glioma precision values were 0.85 and 0.77, respectively, while pituitary and tumorous neoplasm precision values were much higher at 0.95 and 0.86. This shows that, overall, the model has a great ability to predict the true class for each sample with accuracy. A similar trend may be observed in the memory values: tumors and the pituitary gland yield the highest recall (0.96), whereas gliomas and meningiomas yield remember values of 0.68 and 0.83, respectively. The F1 values, which balance memory and precision, were consistently high, with the highest scores for pituitary tumors (0.91) and tumors (0.95).

The InceptionV3 model's overall accuracy of 86% shows that it is quite successful at classifying brain tumors. As a dependable tool for this activity, the model's performance is typified by good precision, recall, and F1 scores across most tumor types. Although the model is robust, more optimization may improve its capacity to differentiate between these related tumor types, as seen by the slightly lower scores for the glioma and meningioma classes. However, the model continues to perform admirably overall, offering precise and trustworthy classifications for the great majority of situations.

### 4.2.2 ResNet101

The ResNet101 model does rather well in brain tumor classification, with many clear advantages and much potential for further development. The model demonstrated a steady increase in training accuracy during 50 training epochs, going from 46.26% to 64.89%. Simultaneously, there was an increase in the validation accuracy as well, going from 47.29% to 64.53%. This continuous development indicates that the model was able to learn from and adapt to the training data; nevertheless, the comparatively slow rate of accuracy gain indicates that the model had challenges fully capturing the complexity of the data. Moreover, there are concerns about potential overfitting because to the validation accuracy's frequent fluctuations, particularly in the middle epochs, which show difficulties extrapolating to previously unidentified data. The training loss decreased from 1.1819 to 0.8741 and the validation loss from 1.2119 to 0.9049, indicating a general downward trend in both the training and validation loss curves. The model was progressively minimizing error, but the relatively high loss values and the heterogeneity in validation loss suggest that the model struggled to generate consistently accurate predictions across the different classes.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Figure 20. Graph of ResNet101 for second data augmentation

A blue squares with black numbers

Description automatically generated

Figure 21. Confusion matrix of ResNet101 for the second data augmentation

The confusion matrix provides a detailed study of the model's performance for different types of tumors. Pituitary tumor classification was the model's strongest suit, successfully identifying 241 out of 300 cases; non-tumor detection was its second best suit, correctly classifying 305 out of 405 cases. However, the model found it particularly difficult to classify tumors from meningiomas and lymphomas. Specifically, it correctly identified only 163 out of 300 cases of gliomas; a large number of these cases were misclassified as other forms of tumors. Like the meningioma cases, only 125 out of 306 cases were correctly detected by the model, and there was a substantial percentage of misclassification into other categories.

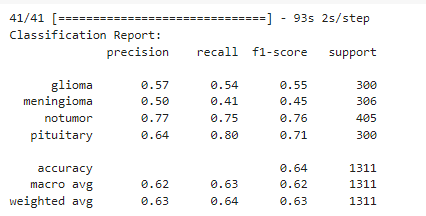


Figure 22. Classification report of ResNet101of second data augmentation

The classification report illustrates these challenges even more by demonstrating a notable difference in the model's performance among classes. While comparably stronger precision, recall, and F1 scores indicated the model's performance in reliably recognizing and classifying notumor and pituitary tumor classifications, the metrics for glioma and meningioma were noticeably lower. Gliomas and meningiomas were predicted by the model to have corresponding values of 0.57 and 0.50, suggesting that these conditions were commonly misdiagnosed. Glioma (0.54) and meningioma (0.41) had lower recall ratings, suggesting that a large number of actual occurrences of these tumors were missed. Recall measures how well each form of tumor can be identified in real-world cases by the model.The model's inability to accurately distinguish between meningiomas and gliomas was further highlighted by the low F1 scores for both classes, which balance accuracy against recall.

The ResNet101 model's total accuracy of 64% indicates that, despite its basic competence in brain tumor classification, there is a great deal of room for improvement. Further refinement is necessary, as the model has shown promise in notumor and pituitary instances, but it is unable to correctly diagnose gliomas and meningiomas. These issues could be fixed with stronger model architecture, more efficient hyperparameter adjustments, and improved data preprocessing techniques, leading to more reliable and accurate predictions for all tumor types. These improvements would be crucial to turning the model from a passable performer into a more dependable and potent instrument for categorizing brain cancers.

### 4.2.3.VGG16

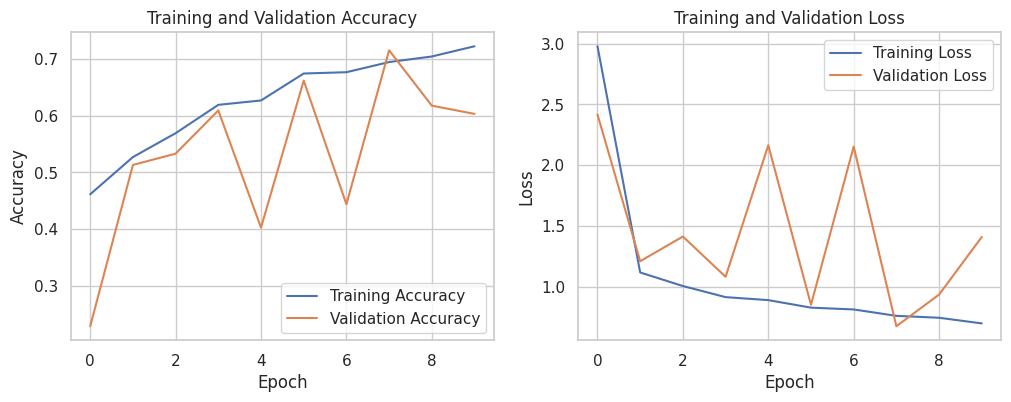
The VGG16 model, which was built from scratch, performs unevenly in brain tumor picture categorization, showing both distinct benefits and drawbacks. Throughout the training procedure, accuracy grew gradually, starting at 46.16% and ending at 72.26% at the end of the tenth phase. As the model gained experience, its accuracy and confidence grew, as shown by the sharp decline in training loss from 2.9799 to 0.6958. On the other hand, the validation measurements paint a more complex picture. Validation accuracy starts at a low of 22.88% and fluctuates a lot during training, reaching a peak of 60.34% in the last epoch. Furthermore, the validation loss showed significant spikes in a few epochs (epochs 5 and 7, for instance, 2.1643 and 2.1530, respectively), which indicated that the model had problems with overfitting and generalization.

Figure 23. Graph of VGG16 of the second data augmentation

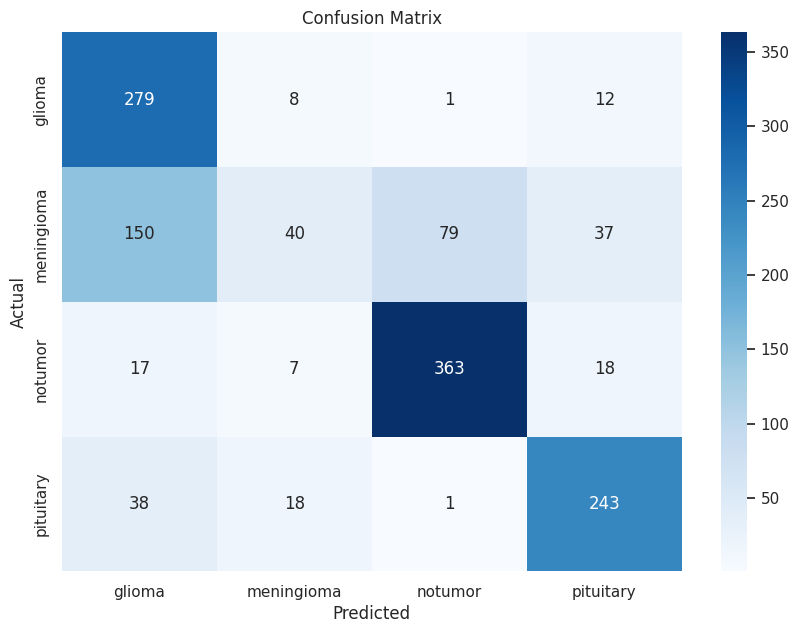


Figure 24. Confusion Matrix of VGG16 the second data augmentation

The VGG16 model's confusion matrix demonstrates that while the model does well in recognizing certain tumor types, it struggles with others. It performs well in the tumor and glioma categories, correctly recognizing 279 out of 300 cases (93% recall) and 364 out of 405 cases (90% remember) respectively. Regretfully, the model's performance with meningioma is subpar; just 40 out of 306 cases were correctly classified, resulting in a 13% recall rate. This suggests a significant mistake in classification, most likely confusing meningioma for other types of tumors. The model performs more balanced when it comes to pituitary tumors, correctly identifying 243 out of 300 instances (81% recall).

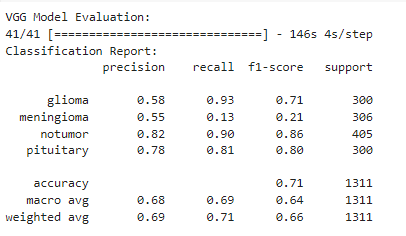


Figure 25. Classification report VGG16 of second data augmentation

The classification report emphasizes these differences in performance even more. In terms of gliomas, the model achieves an F1 score of 0.71 and a precision of 0.58, indicating a balanced capacity to detect true positives while minimizing false positives. Here, the notumor category's good performance—which has an F1 score of 0.86 and a precision of 0.82—demonstrates the trustworthiness of the model. However, the meningioma category has significant deficiencies, as seen by its 0.55 classification accuracy, 0.13 recall, and 0.21 F1 score. The pituitary tumor category performs better, with a precision of 0.78, recall of 0.81, and an F1 score of 0.80, indicating moderate efficacy. Even if the model excels in a number of areas, it still needs to be improved overall, particularly in correctly classifying meningiomas.

The model's overall accuracy of 71% is impressive for a model built from scratch, there is still need for improvement given its uneven performance across different tumor types. The model's inability to accurately categorize meningiomas tempers its initial great performance in glioma and tumor detection, suggesting that further refinement is required. This may include adding complexity to the model, enhancing data preparation, or applying more sophisticated methods in order to address overfitting and improve generalization. More work is required to make the VGG16 model a more reliable and practical tool for classifying brain tumors in all categories, even if it shows promise, especially in some classes.

# 5. Discussion of Result

## 5.1 Classification of models

When the three models of the two distinct data augmentations—two iterations of InceptionV3, ResNet101, and VGG16—are compared, it becomes evident that the first iteration of InceptionV3 performs the best, with an accuracy of 92% and robust, consistent outcomes across all tumor types, particularly in the classification of pituitary and notumors. Even though the second InceptionV3 model had slightly more difficulty distinguishing between gliomas and meningiomas, it was still reliable with an accuracy rate of 86%. The initial ResNet101 model was the most accurate in classifying pituitaries and tumors, with an accuracy rate of 72%. Gliomas and meningiomas, however, were difficult for it to handle, particularly in the second model, which had the lowest overall accuracy of 64%.The two VGG16 models were created from scratch and performed similarly, with an accuracy of about 71%; they were very good at classifying tumors and gliomas, but they had the lowest recall of any model when it came to meningiomas. In general, the InceptionV3 models—especially the first one—are the most dependable, whereas ResNet101 and VGG16 models still need to be improved upon in order to improve their classification accuracy and generalization ability.

## 5.2 Impact of Data Augmentation in model classification

Data augmentation significantly affects the performance and generalization of classification models in complex tasks such as brain tumor classification. In the six models that were studied, different augmentation techniques yielded different outcomes (InceptionV3, ResNet101, and VGG16). The original InceptionV3 model, which achieved the highest accuracy of 92%, was able to more effectively generalize across a range of tumor types, including challenging cases such as gliomas and meningiomas, thanks to effective data augmentation. This shows that using the right augmentation techniques can reduce overfitting and increase the model's robustness. However, there were also inconsistent results with the ResNet101 and VGG16 models; while the first ResNet101 model showed some progress, the second ResNet101 model and the VGG16 models had difficulties with consistency and accuracy, particularly when it came to categorizing meningiomas. This implies that although data augmentation can yield substantial advantages, its efficacy is intimately linked to the particular model architecture and the match between augmentation methods and the model's learning process.

## 5.3 Comparison with other paper

The findings support previous research while also pointing to possible areas for future improvement when comparing the accuracy and data used in study with those from the publications in the literature review. To enhance performance, your study used different data augmentation techniques including rotations and flips on brain MRI images. Deep learning models like InceptionV3, ResNet101, and VGG16 were used. The first InceptionV3 model achieved 92% accuracy, the second model 86%, the first ResNet101 model 72%, the second 64%, and the two VGG16 models around 71%. These are noteworthy results from these methodologies. These findings align with a number of studies that highlight the value of optimizing models and augmenting data to enhance classification performance.

Çinar and Yildirim (2020) employed a hybrid convolutional neural network design to detect brain cancers with an accuracy of 97%, demonstrating the efficacy of combining many network layers to increase performance. Zoph et al. (2020) have provided support for the exceptional performance of your InceptionV3 models by highlighting the crucial impact that suitable data augmentation approaches may play in improving model accuracy. The InceptionV3 results obtained from your research are comparable and in line with how the literature describes InceptionV3's strong handling of complex visual elements—particularly the 92% correctness of the original model.

The ResNet101 models in this study performed in a variety of ways, despite the fact that their 72% and 64% accuracy rates are marginally lower than those found in other studies. ResNet architectures such as ResNet101, for instance, can achieve up to 98.6% accuracy when used properly with hybrid network designs or optimal settings, according to Çinar and Yildirim (2020). This discrepancy suggests that, despite the strength of your ResNet101 models, they might be significantly improved by applying advanced optimization techniques, such as hybrid architectures or improved data augmentation methods, to boost their classification accuracy for brain tumors.

## 5.4 Some applications in real life

The use of deep learning models like InceptionV3, ResNet101, and VGG16 for the classification of brain tumors has many important applications in the healthcare sector. These models can significantly increase diagnosis accuracy by assisting radiologists in reading magnetic resonance imaging (MRI) data. Better patient outcomes depend on the earlier and more accurate identification of brain tumors, which this can facilitate. More sophisticated models, such as InceptionV3, can be applied for in-depth analysis because they can handle a wide range of image features and variations, while VGG16, which is well-known for its feature extraction capabilities, can be used for initial tumor screening in clinical settings because of its comparatively low computational requirements.

These models are useful for studies on the development of tumors, the effectiveness of treatments, and new therapeutic strategies in medicine. They have the potential to interpret large amounts of imaging data quickly and provide insights that human analysis would miss. In underserved or rural locations, the use of such models in telemedicine increases patient access to specialized treatment by facilitating remote monitoring and diagnosis. Furthermore, the integration of models such as ResNet101, which has remarkable proficiency in identifying subtle patterns in MRI scans, into the creation of AI-powered diagnostic instruments may enhance automated systems meant to identify precocious signs of brain disorders. In summary, the application of these models in diverse settings can improve the precision of diagnosis, stimulate investigation, and facilitate the availability of superior medical care.

## 5.5 Improvement of models

**VGG16**

* To identify tumor-specific features more accurately, use a pre-trained VGG16 model and use fine tuning it using brain MRI data.
* To increase generalization and robustness to changes in the images, use sophisticated data augmentation techniques including rotations, flips, elastic deformations, and contrast alterations.
* To minimize overfitting, employ more comprehensive regularization techniques (L2 regularization, dropout) and data augmentation tactics.

**ResNet101**

* Reduce the regularization strength or modify the residual block layout and depth to address underfitting problems.
* To reduce overfitting, incorporate early halting and dropout layers into the model design and keep it simple.
* Utilize advanced optimization methods to increase convergence and decrease loss, such as learning rate schedulers or adaptive optimizers (like AdamW).

**Inception V3**

* To maximize the advantage of different strengths and increase overall accuracy, use ensemble learning by mixing InceptionV3 with multiple models.
* Use more complex data augmentation methods to improve generalization and decrease overfitting, such as rotations, elastic transformations, and contrast modifications.
* To address underfitting, adjust the learning rate or add extra layers or nodes to the model to make it more complicated.

## 5.6 Limitations

A major impediment to this research is the length of time needed for training, especially for models like VGG16, whose deep architecture necessitates a large amount of computational power and time for convergence. The models may not be sophisticated enough or well-trained enough to capture the key characteristics of brain tumors, as evidenced by the underfitting of learning curves. This could be the result of insufficient epochs or subpar learning rates. Reducing the batch size can also greatly extend the training period, which slows down and requires more resources in order to enhance model accuracy and lessen overfitting. All of these elements together show how important it is to strike a balance between model complexity, computational effectiveness, and training duration in order to produce reliable and accurate results.

# 6. Conclusion

In summary, there was variation in the classification of brain cancers by the Inception V3, ResNet101, and VGG16 models. The Inception V3 model was particularly noteworthy for its rapid training times and high accuracy (92% with the first augmentation, 86% with the second), which made it a great option for real-time clinical application. The fact that ResNet101 had trouble with considerably longer training cycles highlighted the need for optimization to achieve a trade-off between speed and performance. It was nevertheless able to achieve a moderate level of accuracy (72% and 64%). The least successful model, VGG16, suggests that considerable adjustments to its model architecture and training methods are needed. Its training periods were relatively long, and its accuracy was the lowest (63% and 71%). Other problems with the study were long training times (particularly for deeper models such as ResNet101 and VGG16) and the potential for under- or overfitting based on the data augmentation method used. These models have significant real-time applications that improve patient outcomes by aiding in early diagnosis, increasing diagnostic precision, and supporting medical professionals in clinical decision-making, despite these challenges.

**Future scope:**

* Focus on enhancing the deep learning models through the use of advanced techniques such as attention mechanisms, hyperparameter building, and transfer learning with updated architectures in order to reduce training times and increase accuracy.
* Combining MRI scans with multimodal data sources, such as genetic information, clinical records, and patient demographics, may lead to a more thorough and precise diagnosis.
* By using the models in real-time clinical settings, you may enhance early detection and treatment, enable remote diagnostics via cloud-based systems, and expand access to specialist care in underprivileged regions.
* Provide models with continuous learning from new data so they can adapt to new tumor types or subtypes and evolving medical imaging recommendations.
* To increase the models' acceptability, generalizability, and robustness in clinical practice, work with healthcare facilities and academic institutions to validate the models on bigger, more varied datasets.

# 7. Reference

Hounsfield, G. N. (1973). "Computerized transverse axial scanning (tomography): Part 1. Description of system." *British Journal of Radiology*, 46(552): 1016-1022.5[https://doi.org/10.1259/0007-1285-46-552-1016](mailto:https://doi.org/10.1259/0007-1285-46-552-1016)

Zoph, B., Cubuk, E. D., Ghiasi, G., Lin, T.-Y., Shlens, J., & Le, Q. V. (2020). "Learning Data Augmentation Strategies for Object Detection." In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer Vision – ECCV 2020. Lecture Notes in Computer Science* (vol. 12372, pp. 566–583). Springer.[https://link.springer.com/chapter/10.1007/978-3-030-58583-9\_34](mailto:https://link.springer.com/chapter/10.1007/978-3-030-58583-9_34)

Çinar, A., & Yildirim, M. (2020). "Detection of tumors on brain MRI images using the hybrid convolutional neural network architecture." *Medical Hypotheses*, 139, 109684.5. [https://www.sciencedirect.com/science/article/abs/pii/S0306987720301717?via%3Dihub](mailto:https://www.sciencedirect.com/science/article/abs/pii/S0306987720301717?via%3Dihub)

Khan, M. A., et al. "Multi-Classification of Brain Tumor MRI Images Using Deep Convolutional Neural Network with Fully Optimized Framework." Iranian Journal of Science and Technology, Transactions of Electrical Engineering, vol. 45, pp. 1015–1036, 2021. <mailto:https://link.springer.com/article/10.1007/s40998-021-00426-9>

Raghuvanshi, S., & Dhariwal, S. (2023). The VGG16 Method Is a Powerful Tool for Detecting Brain Tumors Using Deep Learning Techniques. Engineering Proceedings.[https://www.mdpi.com/2673-4591/59/1/46](mailto:https://www.mdpi.com/2673-4591/59/1/46)

Okanda Nyatega, C., Jajere Adamu, M., & Kawuwa, H. B. (2022). Brain Tumor Analysis Using Deep Learning and VGG-16 Ensembling Learning Approaches. Applied Sciences

[https://www.mdpi.com/2076-3417/12/14/7282](mailto:https://www.mdpi.com/2076-3417/12/14/7282)

Srinivas, C., Prasad K. S., N., Zakariah, M., Alothaibi, Y. A., Shaukat, K., Partibane, B., & Awal, H. (2022). *Deep Transfer Learning Approaches in Performance Analysis of Brain Tumor Classification Using MRI Images*. Computational and Mathematical Methods in Medicine, 2022.[https://onlinelibrary.wiley.com/doi/10.1155/2022/3264367](mailto:https://onlinelibrary.wiley.com/doi/10.1155/2022/3264367)

Çınar, N., Kaya, B., & Kaya, M. (2022). *Comparison of Deep Learning Models for Brain Tumor Classification Using MRI Images*. In 2022 International Conference on Decision Aid Sciences and Applications (DASA), 23-25 March 2022, Chiangrai, Thailand. IEEE.[https://ieeexplore.ieee.org/document/9765250](mailto:https://ieeexplore.ieee.org/document/9765250)

Krishnapriya, S., & Karuna, Y. (2023). *Pre-trained Deep Learning Models for Brain MRI Image Classification*. Frontiers in Human Neuroscience, 17[https://www.frontiersin.org/journals/human-neuroscience/articles/10.3389/fnhum.2023.1150120/full](mailto:https://www.frontiersin.org/journals/human-neuroscience/articles/10.3389/fnhum.2023.1150120/full)

Kumar, C. M., & Sankar, J. S. (2024). *Comparative Analysis of Convolutional Neural Networks for Brain Tumor Detection: A Study of VGG16, ResNet, Inception, and DenseNet Models*. In 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC).[https://ieeexplore.ieee.org/document/10575770](mailto:https://ieeexplore.ieee.org/document/10575770)

Ullah, Z., Odeh, A., Khattak, I., & Al Hasan, M. *Enhancement of Pre-Trained Deep Learning Models to Improve Brain Tumor Classification*.[https://digitallibrary.aau.ac.ae/handle/123456789/1157](mailto:https://digitallibrary.aau.ac.ae/handle/123456789/1157)

Mahmoud, A., et al. (2023). Advanced Deep Learning Approaches for Accurate Brain Tumor Classification in Medical Imaging. *Symmetry, 15*(3), 571.[https://www.mdpi.com/2073-8994/15/3/571](mailto:https://www.mdpi.com/2073-8994/15/3/571)

Gómez-Guzmán, M. A., Jiménez-Beristain, L., García-Guerrero, E. E., López-Bonilla, O. R., Tamayo-Pérez, U. J., Esqueda-Elizondo, J. J., Palomino-Vizcaino, K., & Inzunza-González, E. (2023). Classifying Brain Tumors on Magnetic Resonance Imaging by Using Convolutional Neural Networks. *Electronics, 12*(4), 955.[https://www.mdpi.com/2079-9292/12/4/955](mailto:https://www.mdpi.com/2079-9292/12/4/955)

Kukadiya, B. (2024). *Enhancement of Pre-Trained Deep Learning Models to Improve Brain Tumor Classification*. Master's Thesis, California State University, Northridge.[https://scholarworks.calstate.edu/concern/theses/j9602790r](mailto:https://scholarworks.calstate.edu/concern/theses/j9602790r)

1. Artificial intelligence has the potential to democratize access to education, healthcare and economic opportunities. Let's strive to make AI technology accessible and beneficial for all.

[https://peak.ai/hub/blog/16-inspiring-quotes-about-ai/](mailto:https://peak.ai/hub/blog/16-inspiring-quotes-about-ai/)

# 8. Appendix

The GitHub link attached to below, the first link is for the code of first data augmentation and second link is for the code of second data augmentation.

1. [Link of first data augmentation](mailto:https://github.com/annaaksa/Data_Science_Project/blob/main/First_dg_0_5.ipynb)
2. [Link of second data augmentation](mailto:https://github.com/annaaksa/Data_Science_Project/blob/main/Second_dg_0_2.ipynb)

Data Science Project

Comparative analysis of deep learning architecture from brain tumor classification using MRI images; evaluation the impact of data augmentation techniques.

Student ID:22070385

Project objectives are

* Develop and compare different deep learning models (ResNet, VGG16, Inception V3) for brain tumor classification (Develop VGG16 model and use pre-trained model for Inception V3 and ResNet).
* Evaluate and compare the performance of these models in terms of accuracy and sensitivity.
* Apply augmentation techniques for each model.
* Evaluate the impact.

The data set is downloaded from the Kaggle and uploaded in the personal drive

#install tensorflow 2.13.0

!pip install tensorflow==2.13.0

Requirement already satisfied: tensorflow==2.13.0 in /usr/local/lib/python3.10/dist-packages (2.13.0)

Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.4.0)

Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.6.3)

Requirement already satisfied: flatbuffers>=23.1.21 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (24.3.25)

Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.4.0)

Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.2.0)

Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.64.1)

Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.11.0)

Requirement already satisfied: keras<2.14,>=2.13.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.1)

Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (18.1.1)

Requirement already satisfied: numpy<=1.24.3,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.24.3)

Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.3.0)

Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (24.1)

Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.20.3)

Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (71.0.4)

Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.16.0)

Requirement already satisfied: tensorboard<2.14,>=2.13 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.0)

Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.0)

Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.4.0)

Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (4.5.0)

Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.16.0)

Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.37.1)

Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.13.0) (0.44.0)

Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.27.0)

Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (1.0.0)

Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.7)

Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.32.3)

Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.7.2)

Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.0.3)

Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (5.5.0)

Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.4.0)

Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (4.9)

Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (1.3.1)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.3.2)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.7)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.0.7)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2024.7.4)

Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.1.5)

Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.6.0)

Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.2.2)

#mount the google colab

from google.colab import drive

drive.mount('/content/drive')

Mounted at /content/drive

#importing the libraries

import os

import cv2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from sklearn.utils import shuffle

from sklearn.model\_selection import train\_test\_split as tts

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tqdm import tqdm

# Create a list of classes and label the total number of images.

count = [n\_train\_glioma, n\_train\_mening, n\_train\_pit, n\_train\_non, total\_train]

length = ['glioma', ' meningioma', 'pituitary', 'no tumor', 'total images']

# Create a loop to print the number of images belongs to each class

for i, sum in zip(length, count):

  print(f'Total counts of {i} is {sum}')

  print(100\*'-')

Total counts of glioma is 1317

----------------------------------------------------------------------------------------------------

Total counts of meningioma is 1331

----------------------------------------------------------------------------------------------------

Total counts of pituitary is 1457

----------------------------------------------------------------------------------------------------

Total counts of no tumor is 1595

----------------------------------------------------------------------------------------------------

Total counts of total images is 5700

----------------------------------------------------------------------------------------------------

# Define the name of all classes

class\_labels = ['Glioma', 'Meningioma', 'Pituitary', 'No Tumor']

# Creatng a list count for the number of images belong to each class

class\_counts = [n\_train\_glioma, n\_train\_mening, n\_train\_pit, n\_train\_non]

# Create a countplot

# Create a white grid background for the plot

sns.set(style="whitegrid")

# Give the figure size for the plot

plt.figure(figsize=(8, 6))

# Create a bar plot which correspond the class to the number of images

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Annotate the counts on top of the bars

for i, count in enumerate(class\_counts):

    ax.text(i, count + 2, str(count), ha='center', va='bottom', fontsize=12)

# Add titles and labels for the graph

plt.title("Distribution of Classes in Training Data")

plt.xlabel("Class Label")

plt.ylabel("Count")

plt.show()

<ipython-input-7-ea309d3e31b6>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Directory path of the test data

test\_dir="/content/drive/MyDrive/Colab Notebooks/Testing"

# Listing the files in each directory to count the number of testing images for each type of tumor.

n\_test\_glioma = len(os.listdir(f'{test\_dir}/glioma'))

n\_test\_mening = len(os.listdir(f'{test\_dir}/meningioma'))

n\_test\_pit = len(os.listdir(f'{test\_dir}/pituitary'))

n\_test\_non = len(os.listdir(f'{test\_dir}/notumor'))

# Total length of the whole classes combined

total\_test = n\_test\_glioma + n\_test\_mening + n\_test\_pit + n\_test\_non

Total counts of glioma is 300

----------------------------------------------------------------------------------------------------

Total counts of meningioma is 306

----------------------------------------------------------------------------------------------------

Total counts of pituitary is 300

----------------------------------------------------------------------------------------------------

Total counts of no tumor is 405

----------------------------------------------------------------------------------------------------

Total counts of total images is 1311

# Create a countplot

# Create a white grid background for the plot

sns.set(style="whitegrid")

# Give the figure size for the plot

plt.figure(figsize=(8, 6))

# Create a bar plot which correspond the class to the number of images

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Annotate the counts on top of the bars

for i, count in enumerate(class\_counts):

    ax.text(i, count + 2, str(count), ha='center', va='bottom', fontsize=12)

# Add titles and labels for the graph

plt.title("Distribution of Classes in Testing Data")

plt.xlabel("Class Label")

plt.ylabel("Count")

plt.show()

<ipython-input-10-bcf986257a76>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Create a Data generator on training and testing set with data augmentation

train\_datagen=ImageDataGenerator(rescale=1./255,

                                 rotation\_range=40,

                                 width\_shift\_range=0.05,

                                 height\_shift\_range=0.05,

                                 zoom\_range=0.1,

                                 shear\_range=0.1,

                                 horizontal\_flip=True,

                                 vertical\_flip=False,

                                 fill\_mode='nearest')

# Apply only rescaling for the test data

test\_datagen=ImageDataGenerator(rescale=1./255)

#Creating Training set and validation set

training\_set=train\_datagen.flow\_from\_directory(train\_dir,

                                               target\_size=(224,224),

                                               batch\_size=16,

                                               class\_mode='categorical')

test\_set=test\_datagen.flow\_from\_directory(test\_dir,

                                          target\_size=(224,224),

                                          batch\_size=16,

                                          class\_mode="categorical")

Found 5700 images belonging to 4 classes.

Found 1311 images belonging to 4 classes.

# Apply a batch size for the test and training of data

batch\_size=16

# Define a colour channel

color\_channel=3

# Define the image shape for the imout images

image\_shape=(224,224,3)

#import some more libraries for the model development

import tensorflow as tf

from tensorflow.keras.applications import InceptionV3, ResNet101

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input, LSTM, Conv2D, MaxPooling2D, MaxPool2D, Flatten, Dense, Dropout, Reshape, BatchNormalization,GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, LearningRateScheduler

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import regularizers

# Load the base model pre-trained model of Inception V3

base\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=image\_shape)

#  Freeze each layer in the base model to halt weight changes during training.

for layer in base\_model.layers:

    layer.trainable = False

# Customize the top fully connected layer

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dropout(0.5)(x)

x = Dense(1024, activation='relu')(x)

predictions = Dense(len(class\_labels), activation='softmax')(x)

# Create the complete model

model = Model(inputs=base\_model.input, outputs=predictions)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the summary

model.summary()

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5>

87910968/87910968 [==============================] - 3s 0us/step

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 224, 224, 3)] 0 []

conv2d\_10 (Conv2D) (None, 111, 111, 32) 864 ['input\_1[0][0]']

batch\_normalization\_4 (Bat (None, 111, 111, 32) 96 ['conv2d\_10[0][0]']

chNormalization)

activation (Activation) (None, 111, 111, 32) 0 ['batch\_normalization\_4[0][0]'

]

conv2d\_11 (Conv2D) (None, 109, 109, 32) 9216 ['activation[0][0]']

batch\_normalization\_5 (Bat (None, 109, 109, 32) 96 ['conv2d\_11[0][0]']

chNormalization)

activation\_1 (Activation) (None, 109, 109, 32) 0 ['batch\_normalization\_5[0][0]'

]

conv2d\_12 (Conv2D) (None, 109, 109, 64) 18432 ['activation\_1[0][0]']

batch\_normalization\_6 (Bat (None, 109, 109, 64) 192 ['conv2d\_12[0][0]']

chNormalization)

activation\_2 (Activation) (None, 109, 109, 64) 0 ['batch\_normalization\_6[0][0]'

]

max\_pooling2d\_4 (MaxPoolin (None, 54, 54, 64) 0 ['activation\_2[0][0]']

g2D)

conv2d\_13 (Conv2D) (None, 54, 54, 80) 5120 ['max\_pooling2d\_4[0][0]']

batch\_normalization\_7 (Bat (None, 54, 54, 80) 240 ['conv2d\_13[0][0]']

chNormalization)

activation\_3 (Activation) (None, 54, 54, 80) 0 ['batch\_normalization\_7[0][0]'

]

conv2d\_14 (Conv2D) (None, 52, 52, 192) 138240 ['activation\_3[0][0]']

batch\_normalization\_8 (Bat (None, 52, 52, 192) 576 ['conv2d\_14[0][0]']

chNormalization)

activation\_4 (Activation) (None, 52, 52, 192) 0 ['batch\_normalization\_8[0][0]'

]

max\_pooling2d\_5 (MaxPoolin (None, 25, 25, 192) 0 ['activation\_4[0][0]']

g2D)

conv2d\_18 (Conv2D) (None, 25, 25, 64) 12288 ['max\_pooling2d\_5[0][0]']

batch\_normalization\_12 (Ba (None, 25, 25, 64) 192 ['conv2d\_18[0][0]']

tchNormalization)

activation\_8 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_12[0][0]

']

conv2d\_16 (Conv2D) (None, 25, 25, 48) 9216 ['max\_pooling2d\_5[0][0]']

conv2d\_19 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_8[0][0]']

batch\_normalization\_10 (Ba (None, 25, 25, 48) 144 ['conv2d\_16[0][0]']

tchNormalization)

batch\_normalization\_13 (Ba (None, 25, 25, 96) 288 ['conv2d\_19[0][0]']

tchNormalization)

activation\_6 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_10[0][0]

']

activation\_9 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_13[0][0]

']

average\_pooling2d (Average (None, 25, 25, 192) 0 ['max\_pooling2d\_5[0][0]']

Pooling2D)

conv2d\_15 (Conv2D) (None, 25, 25, 64) 12288 ['max\_pooling2d\_5[0][0]']

conv2d\_17 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_6[0][0]']

conv2d\_20 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_9[0][0]']

conv2d\_21 (Conv2D) (None, 25, 25, 32) 6144 ['average\_pooling2d[0][0]']

batch\_normalization\_9 (Bat (None, 25, 25, 64) 192 ['conv2d\_15[0][0]']

chNormalization)

batch\_normalization\_11 (Ba (None, 25, 25, 64) 192 ['conv2d\_17[0][0]']

tchNormalization)

batch\_normalization\_14 (Ba (None, 25, 25, 96) 288 ['conv2d\_20[0][0]']

tchNormalization)

batch\_normalization\_15 (Ba (None, 25, 25, 32) 96 ['conv2d\_21[0][0]']

tchNormalization)

activation\_5 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_9[0][0]'

]

activation\_7 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_11[0][0]

']

activation\_10 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_14[0][0]

']

activation\_11 (Activation) (None, 25, 25, 32) 0 ['batch\_normalization\_15[0][0]

']

mixed0 (Concatenate) (None, 25, 25, 256) 0 ['activation\_5[0][0]',

'activation\_7[0][0]',

'activation\_10[0][0]',

'activation\_11[0][0]']

conv2d\_25 (Conv2D) (None, 25, 25, 64) 16384 ['mixed0[0][0]']

batch\_normalization\_19 (Ba (None, 25, 25, 64) 192 ['conv2d\_25[0][0]']

tchNormalization)

activation\_15 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_19[0][0]

']

conv2d\_23 (Conv2D) (None, 25, 25, 48) 12288 ['mixed0[0][0]']

conv2d\_26 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_15[0][0]']

batch\_normalization\_17 (Ba (None, 25, 25, 48) 144 ['conv2d\_23[0][0]']

tchNormalization)

batch\_normalization\_20 (Ba (None, 25, 25, 96) 288 ['conv2d\_26[0][0]']

tchNormalization)

activation\_13 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_17[0][0]

']

activation\_16 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_20[0][0]

']

average\_pooling2d\_1 (Avera (None, 25, 25, 256) 0 ['mixed0[0][0]']

gePooling2D)

conv2d\_22 (Conv2D) (None, 25, 25, 64) 16384 ['mixed0[0][0]']

conv2d\_24 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_13[0][0]']

conv2d\_27 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_16[0][0]']

conv2d\_28 (Conv2D) (None, 25, 25, 64) 16384 ['average\_pooling2d\_1[0][0]']

batch\_normalization\_16 (Ba (None, 25, 25, 64) 192 ['conv2d\_22[0][0]']

tchNormalization)

batch\_normalization\_18 (Ba (None, 25, 25, 64) 192 ['conv2d\_24[0][0]']

tchNormalization)

batch\_normalization\_21 (Ba (None, 25, 25, 96) 288 ['conv2d\_27[0][0]']

tchNormalization)

batch\_normalization\_22 (Ba (None, 25, 25, 64) 192 ['conv2d\_28[0][0]']

tchNormalization)

activation\_12 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_16[0][0]

']

activation\_14 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_18[0][0]

']

activation\_17 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_21[0][0]

']

activation\_18 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_22[0][0]

']

mixed1 (Concatenate) (None, 25, 25, 288) 0 ['activation\_12[0][0]',

'activation\_14[0][0]',

'activation\_17[0][0]',

'activation\_18[0][0]']

conv2d\_32 (Conv2D) (None, 25, 25, 64) 18432 ['mixed1[0][0]']

batch\_normalization\_26 (Ba (None, 25, 25, 64) 192 ['conv2d\_32[0][0]']

tchNormalization)

activation\_22 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_26[0][0]

']

conv2d\_30 (Conv2D) (None, 25, 25, 48) 13824 ['mixed1[0][0]']

conv2d\_33 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_22[0][0]']

batch\_normalization\_24 (Ba (None, 25, 25, 48) 144 ['conv2d\_30[0][0]']

tchNormalization)

batch\_normalization\_27 (Ba (None, 25, 25, 96) 288 ['conv2d\_33[0][0]']

tchNormalization)

activation\_20 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_24[0][0]

']

activation\_23 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_27[0][0]

']

average\_pooling2d\_2 (Avera (None, 25, 25, 288) 0 ['mixed1[0][0]']

gePooling2D)

conv2d\_29 (Conv2D) (None, 25, 25, 64) 18432 ['mixed1[0][0]']

conv2d\_31 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_20[0][0]']

conv2d\_34 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_23[0][0]']

conv2d\_35 (Conv2D) (None, 25, 25, 64) 18432 ['average\_pooling2d\_2[0][0]']

batch\_normalization\_23 (Ba (None, 25, 25, 64) 192 ['conv2d\_29[0][0]']

tchNormalization)

batch\_normalization\_25 (Ba (None, 25, 25, 64) 192 ['conv2d\_31[0][0]']

tchNormalization)

batch\_normalization\_28 (Ba (None, 25, 25, 96) 288 ['conv2d\_34[0][0]']

tchNormalization)

batch\_normalization\_29 (Ba (None, 25, 25, 64) 192 ['conv2d\_35[0][0]']

tchNormalization)

activation\_19 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_23[0][0]

']

activation\_21 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_25[0][0]

']

activation\_24 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_28[0][0]

']

activation\_25 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_29[0][0]

']

mixed2 (Concatenate) (None, 25, 25, 288) 0 ['activation\_19[0][0]',

'activation\_21[0][0]',

'activation\_24[0][0]',

'activation\_25[0][0]']

conv2d\_37 (Conv2D) (None, 25, 25, 64) 18432 ['mixed2[0][0]']

batch\_normalization\_31 (Ba (None, 25, 25, 64) 192 ['conv2d\_37[0][0]']

tchNormalization)

activation\_27 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_31[0][0]

']

conv2d\_38 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_27[0][0]']

batch\_normalization\_32 (Ba (None, 25, 25, 96) 288 ['conv2d\_38[0][0]']

tchNormalization)

activation\_28 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_32[0][0]

']

conv2d\_36 (Conv2D) (None, 12, 12, 384) 995328 ['mixed2[0][0]']

conv2d\_39 (Conv2D) (None, 12, 12, 96) 82944 ['activation\_28[0][0]']

batch\_normalization\_30 (Ba (None, 12, 12, 384) 1152 ['conv2d\_36[0][0]']

tchNormalization)

batch\_normalization\_33 (Ba (None, 12, 12, 96) 288 ['conv2d\_39[0][0]']

tchNormalization)

activation\_26 (Activation) (None, 12, 12, 384) 0 ['batch\_normalization\_30[0][0]

']

activation\_29 (Activation) (None, 12, 12, 96) 0 ['batch\_normalization\_33[0][0]

']

max\_pooling2d\_6 (MaxPoolin (None, 12, 12, 288) 0 ['mixed2[0][0]']

g2D)

mixed3 (Concatenate) (None, 12, 12, 768) 0 ['activation\_26[0][0]',

'activation\_29[0][0]',

'max\_pooling2d\_6[0][0]']

conv2d\_44 (Conv2D) (None, 12, 12, 128) 98304 ['mixed3[0][0]']

batch\_normalization\_38 (Ba (None, 12, 12, 128) 384 ['conv2d\_44[0][0]']

tchNormalization)

activation\_34 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_38[0][0]

']

conv2d\_45 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_34[0][0]']

batch\_normalization\_39 (Ba (None, 12, 12, 128) 384 ['conv2d\_45[0][0]']

tchNormalization)

activation\_35 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_39[0][0]

']

conv2d\_41 (Conv2D) (None, 12, 12, 128) 98304 ['mixed3[0][0]']

conv2d\_46 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_35[0][0]']

batch\_normalization\_35 (Ba (None, 12, 12, 128) 384 ['conv2d\_41[0][0]']

tchNormalization)

batch\_normalization\_40 (Ba (None, 12, 12, 128) 384 ['conv2d\_46[0][0]']

tchNormalization)

activation\_31 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_35[0][0]

']

activation\_36 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_40[0][0]

']

conv2d\_42 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_31[0][0]']

conv2d\_47 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_36[0][0]']

batch\_normalization\_36 (Ba (None, 12, 12, 128) 384 ['conv2d\_42[0][0]']

tchNormalization)

batch\_normalization\_41 (Ba (None, 12, 12, 128) 384 ['conv2d\_47[0][0]']

tchNormalization)

activation\_32 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_36[0][0]

']

activation\_37 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_41[0][0]

']

average\_pooling2d\_3 (Avera (None, 12, 12, 768) 0 ['mixed3[0][0]']

gePooling2D)

conv2d\_40 (Conv2D) (None, 12, 12, 192) 147456 ['mixed3[0][0]']

conv2d\_43 (Conv2D) (None, 12, 12, 192) 172032 ['activation\_32[0][0]']

conv2d\_48 (Conv2D) (None, 12, 12, 192) 172032 ['activation\_37[0][0]']

conv2d\_49 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_3[0][0]']

batch\_normalization\_34 (Ba (None, 12, 12, 192) 576 ['conv2d\_40[0][0]']

tchNormalization)

batch\_normalization\_37 (Ba (None, 12, 12, 192) 576 ['conv2d\_43[0][0]']

tchNormalization)

batch\_normalization\_42 (Ba (None, 12, 12, 192) 576 ['conv2d\_48[0][0]']

tchNormalization)

batch\_normalization\_43 (Ba (None, 12, 12, 192) 576 ['conv2d\_49[0][0]']

tchNormalization)

activation\_30 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_34[0][0]

']

activation\_33 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_37[0][0]

']

activation\_38 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_42[0][0]

']

activation\_39 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_43[0][0]

']

mixed4 (Concatenate) (None, 12, 12, 768) 0 ['activation\_30[0][0]',

'activation\_33[0][0]',

'activation\_38[0][0]',

'activation\_39[0][0]']

conv2d\_54 (Conv2D) (None, 12, 12, 160) 122880 ['mixed4[0][0]']

batch\_normalization\_48 (Ba (None, 12, 12, 160) 480 ['conv2d\_54[0][0]']

tchNormalization)

activation\_44 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_48[0][0]

']

conv2d\_55 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_44[0][0]']

batch\_normalization\_49 (Ba (None, 12, 12, 160) 480 ['conv2d\_55[0][0]']

tchNormalization)

activation\_45 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_49[0][0]

']

conv2d\_51 (Conv2D) (None, 12, 12, 160) 122880 ['mixed4[0][0]']

conv2d\_56 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_45[0][0]']

batch\_normalization\_45 (Ba (None, 12, 12, 160) 480 ['conv2d\_51[0][0]']

tchNormalization)

batch\_normalization\_50 (Ba (None, 12, 12, 160) 480 ['conv2d\_56[0][0]']

tchNormalization)

activation\_41 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_45[0][0]

']

activation\_46 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_50[0][0]

']

conv2d\_52 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_41[0][0]']

conv2d\_57 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_46[0][0]']

batch\_normalization\_46 (Ba (None, 12, 12, 160) 480 ['conv2d\_52[0][0]']

tchNormalization)

batch\_normalization\_51 (Ba (None, 12, 12, 160) 480 ['conv2d\_57[0][0]']

tchNormalization)

activation\_42 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_46[0][0]

']

activation\_47 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_51[0][0]

']

average\_pooling2d\_4 (Avera (None, 12, 12, 768) 0 ['mixed4[0][0]']

gePooling2D)

conv2d\_50 (Conv2D) (None, 12, 12, 192) 147456 ['mixed4[0][0]']

conv2d\_53 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_42[0][0]']

conv2d\_58 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_47[0][0]']

conv2d\_59 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_4[0][0]']

batch\_normalization\_44 (Ba (None, 12, 12, 192) 576 ['conv2d\_50[0][0]']

tchNormalization)

batch\_normalization\_47 (Ba (None, 12, 12, 192) 576 ['conv2d\_53[0][0]']

tchNormalization)

batch\_normalization\_52 (Ba (None, 12, 12, 192) 576 ['conv2d\_58[0][0]']

tchNormalization)

batch\_normalization\_53 (Ba (None, 12, 12, 192) 576 ['conv2d\_59[0][0]']

tchNormalization)

activation\_40 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_44[0][0]

']

activation\_43 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_47[0][0]

']

activation\_48 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_52[0][0]

']

activation\_49 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_53[0][0]

']

mixed5 (Concatenate) (None, 12, 12, 768) 0 ['activation\_40[0][0]',

'activation\_43[0][0]',

'activation\_48[0][0]',

'activation\_49[0][0]']

conv2d\_64 (Conv2D) (None, 12, 12, 160) 122880 ['mixed5[0][0]']

batch\_normalization\_58 (Ba (None, 12, 12, 160) 480 ['conv2d\_64[0][0]']

tchNormalization)

activation\_54 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_58[0][0]

']

conv2d\_65 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_54[0][0]']

batch\_normalization\_59 (Ba (None, 12, 12, 160) 480 ['conv2d\_65[0][0]']

tchNormalization)

activation\_55 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_59[0][0]

']

conv2d\_61 (Conv2D) (None, 12, 12, 160) 122880 ['mixed5[0][0]']

conv2d\_66 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_55[0][0]']

batch\_normalization\_55 (Ba (None, 12, 12, 160) 480 ['conv2d\_61[0][0]']

tchNormalization)

batch\_normalization\_60 (Ba (None, 12, 12, 160) 480 ['conv2d\_66[0][0]']

tchNormalization)

activation\_51 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_55[0][0]

']

activation\_56 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_60[0][0]

']

conv2d\_62 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_51[0][0]']

conv2d\_67 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_56[0][0]']

batch\_normalization\_56 (Ba (None, 12, 12, 160) 480 ['conv2d\_62[0][0]']

tchNormalization)

batch\_normalization\_61 (Ba (None, 12, 12, 160) 480 ['conv2d\_67[0][0]']

tchNormalization)

activation\_52 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_56[0][0]

']

activation\_57 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_61[0][0]

']

average\_pooling2d\_5 (Avera (None, 12, 12, 768) 0 ['mixed5[0][0]']

gePooling2D)

conv2d\_60 (Conv2D) (None, 12, 12, 192) 147456 ['mixed5[0][0]']

conv2d\_63 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_52[0][0]']

conv2d\_68 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_57[0][0]']

conv2d\_69 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_5[0][0]']

batch\_normalization\_54 (Ba (None, 12, 12, 192) 576 ['conv2d\_60[0][0]']

tchNormalization)

batch\_normalization\_57 (Ba (None, 12, 12, 192) 576 ['conv2d\_63[0][0]']

tchNormalization)

batch\_normalization\_62 (Ba (None, 12, 12, 192) 576 ['conv2d\_68[0][0]']

tchNormalization)

batch\_normalization\_63 (Ba (None, 12, 12, 192) 576 ['conv2d\_69[0][0]']

tchNormalization)

activation\_50 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_54[0][0]

']

activation\_53 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_57[0][0]

']

activation\_58 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_62[0][0]

']

activation\_59 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_63[0][0]

']

mixed6 (Concatenate) (None, 12, 12, 768) 0 ['activation\_50[0][0]',

'activation\_53[0][0]',

'activation\_58[0][0]',

'activation\_59[0][0]']

conv2d\_74 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

batch\_normalization\_68 (Ba (None, 12, 12, 192) 576 ['conv2d\_74[0][0]']

tchNormalization)

activation\_64 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_68[0][0]

']

conv2d\_75 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_64[0][0]']

batch\_normalization\_69 (Ba (None, 12, 12, 192) 576 ['conv2d\_75[0][0]']

tchNormalization)

activation\_65 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_69[0][0]

']

conv2d\_71 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

conv2d\_76 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_65[0][0]']

batch\_normalization\_65 (Ba (None, 12, 12, 192) 576 ['conv2d\_71[0][0]']

tchNormalization)

batch\_normalization\_70 (Ba (None, 12, 12, 192) 576 ['conv2d\_76[0][0]']

tchNormalization)

activation\_61 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_65[0][0]

']

activation\_66 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_70[0][0]

']

conv2d\_72 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_61[0][0]']

conv2d\_77 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_66[0][0]']

batch\_normalization\_66 (Ba (None, 12, 12, 192) 576 ['conv2d\_72[0][0]']

tchNormalization)

batch\_normalization\_71 (Ba (None, 12, 12, 192) 576 ['conv2d\_77[0][0]']

tchNormalization)

activation\_62 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_66[0][0]

']

activation\_67 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_71[0][0]

']

average\_pooling2d\_6 (Avera (None, 12, 12, 768) 0 ['mixed6[0][0]']

gePooling2D)

conv2d\_70 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

conv2d\_73 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_62[0][0]']

conv2d\_78 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_67[0][0]']

conv2d\_79 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_6[0][0]']

batch\_normalization\_64 (Ba (None, 12, 12, 192) 576 ['conv2d\_70[0][0]']

tchNormalization)

batch\_normalization\_67 (Ba (None, 12, 12, 192) 576 ['conv2d\_73[0][0]']

tchNormalization)

batch\_normalization\_72 (Ba (None, 12, 12, 192) 576 ['conv2d\_78[0][0]']

tchNormalization)

batch\_normalization\_73 (Ba (None, 12, 12, 192) 576 ['conv2d\_79[0][0]']

tchNormalization)

activation\_60 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_64[0][0]

']

activation\_63 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_67[0][0]

']

activation\_68 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_72[0][0]

']

activation\_69 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_73[0][0]

']

mixed7 (Concatenate) (None, 12, 12, 768) 0 ['activation\_60[0][0]',

'activation\_63[0][0]',

'activation\_68[0][0]',

'activation\_69[0][0]']

conv2d\_82 (Conv2D) (None, 12, 12, 192) 147456 ['mixed7[0][0]']

batch\_normalization\_76 (Ba (None, 12, 12, 192) 576 ['conv2d\_82[0][0]']

tchNormalization)

activation\_72 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_76[0][0]

']

conv2d\_83 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_72[0][0]']

batch\_normalization\_77 (Ba (None, 12, 12, 192) 576 ['conv2d\_83[0][0]']

tchNormalization)

activation\_73 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_77[0][0]

']

conv2d\_80 (Conv2D) (None, 12, 12, 192) 147456 ['mixed7[0][0]']

conv2d\_84 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_73[0][0]']

batch\_normalization\_74 (Ba (None, 12, 12, 192) 576 ['conv2d\_80[0][0]']

tchNormalization)

batch\_normalization\_78 (Ba (None, 12, 12, 192) 576 ['conv2d\_84[0][0]']

tchNormalization)

activation\_70 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_74[0][0]

']

activation\_74 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_78[0][0]

']

conv2d\_81 (Conv2D) (None, 5, 5, 320) 552960 ['activation\_70[0][0]']

conv2d\_85 (Conv2D) (None, 5, 5, 192) 331776 ['activation\_74[0][0]']

batch\_normalization\_75 (Ba (None, 5, 5, 320) 960 ['conv2d\_81[0][0]']

tchNormalization)

batch\_normalization\_79 (Ba (None, 5, 5, 192) 576 ['conv2d\_85[0][0]']

tchNormalization)

activation\_71 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_75[0][0]

']

activation\_75 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_79[0][0]

']

max\_pooling2d\_7 (MaxPoolin (None, 5, 5, 768) 0 ['mixed7[0][0]']

g2D)

mixed8 (Concatenate) (None, 5, 5, 1280) 0 ['activation\_71[0][0]',

'activation\_75[0][0]',

'max\_pooling2d\_7[0][0]']

conv2d\_90 (Conv2D) (None, 5, 5, 448) 573440 ['mixed8[0][0]']

batch\_normalization\_84 (Ba (None, 5, 5, 448) 1344 ['conv2d\_90[0][0]']

tchNormalization)

activation\_80 (Activation) (None, 5, 5, 448) 0 ['batch\_normalization\_84[0][0]

']

conv2d\_87 (Conv2D) (None, 5, 5, 384) 491520 ['mixed8[0][0]']

conv2d\_91 (Conv2D) (None, 5, 5, 384) 1548288 ['activation\_80[0][0]']

batch\_normalization\_81 (Ba (None, 5, 5, 384) 1152 ['conv2d\_87[0][0]']

tchNormalization)

batch\_normalization\_85 (Ba (None, 5, 5, 384) 1152 ['conv2d\_91[0][0]']

tchNormalization)

activation\_77 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_81[0][0]

']

activation\_81 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_85[0][0]

']

conv2d\_88 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_77[0][0]']

conv2d\_89 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_77[0][0]']

conv2d\_92 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_81[0][0]']

conv2d\_93 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_81[0][0]']

average\_pooling2d\_7 (Avera (None, 5, 5, 1280) 0 ['mixed8[0][0]']

gePooling2D)

conv2d\_86 (Conv2D) (None, 5, 5, 320) 409600 ['mixed8[0][0]']

batch\_normalization\_82 (Ba (None, 5, 5, 384) 1152 ['conv2d\_88[0][0]']

tchNormalization)

batch\_normalization\_83 (Ba (None, 5, 5, 384) 1152 ['conv2d\_89[0][0]']

tchNormalization)

batch\_normalization\_86 (Ba (None, 5, 5, 384) 1152 ['conv2d\_92[0][0]']

tchNormalization)

batch\_normalization\_87 (Ba (None, 5, 5, 384) 1152 ['conv2d\_93[0][0]']

tchNormalization)

conv2d\_94 (Conv2D) (None, 5, 5, 192) 245760 ['average\_pooling2d\_7[0][0]']

batch\_normalization\_80 (Ba (None, 5, 5, 320) 960 ['conv2d\_86[0][0]']

tchNormalization)

activation\_78 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_82[0][0]

']

activation\_79 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_83[0][0]

']

activation\_82 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_86[0][0]

']

activation\_83 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_87[0][0]

']

batch\_normalization\_88 (Ba (None, 5, 5, 192) 576 ['conv2d\_94[0][0]']

tchNormalization)

activation\_76 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_80[0][0]

']

mixed9\_0 (Concatenate) (None, 5, 5, 768) 0 ['activation\_78[0][0]',

'activation\_79[0][0]']

concatenate (Concatenate) (None, 5, 5, 768) 0 ['activation\_82[0][0]',

'activation\_83[0][0]']

activation\_84 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_88[0][0]

']

mixed9 (Concatenate) (None, 5, 5, 2048) 0 ['activation\_76[0][0]',

'mixed9\_0[0][0]',

'concatenate[0][0]',

'activation\_84[0][0]']

conv2d\_99 (Conv2D) (None, 5, 5, 448) 917504 ['mixed9[0][0]']

batch\_normalization\_93 (Ba (None, 5, 5, 448) 1344 ['conv2d\_99[0][0]']

tchNormalization)

activation\_89 (Activation) (None, 5, 5, 448) 0 ['batch\_normalization\_93[0][0]

']

conv2d\_96 (Conv2D) (None, 5, 5, 384) 786432 ['mixed9[0][0]']

conv2d\_100 (Conv2D) (None, 5, 5, 384) 1548288 ['activation\_89[0][0]']

batch\_normalization\_90 (Ba (None, 5, 5, 384) 1152 ['conv2d\_96[0][0]']

tchNormalization)

batch\_normalization\_94 (Ba (None, 5, 5, 384) 1152 ['conv2d\_100[0][0]']

tchNormalization)

activation\_86 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_90[0][0]

']

activation\_90 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_94[0][0]

']

conv2d\_97 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_86[0][0]']

conv2d\_98 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_86[0][0]']

conv2d\_101 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_90[0][0]']

conv2d\_102 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_90[0][0]']

average\_pooling2d\_8 (Avera (None, 5, 5, 2048) 0 ['mixed9[0][0]']

gePooling2D)

conv2d\_95 (Conv2D) (None, 5, 5, 320) 655360 ['mixed9[0][0]']

batch\_normalization\_91 (Ba (None, 5, 5, 384) 1152 ['conv2d\_97[0][0]']

tchNormalization)

batch\_normalization\_92 (Ba (None, 5, 5, 384) 1152 ['conv2d\_98[0][0]']

tchNormalization)

batch\_normalization\_95 (Ba (None, 5, 5, 384) 1152 ['conv2d\_101[0][0]']

tchNormalization)

batch\_normalization\_96 (Ba (None, 5, 5, 384) 1152 ['conv2d\_102[0][0]']

tchNormalization)

conv2d\_103 (Conv2D) (None, 5, 5, 192) 393216 ['average\_pooling2d\_8[0][0]']

batch\_normalization\_89 (Ba (None, 5, 5, 320) 960 ['conv2d\_95[0][0]']

tchNormalization)

activation\_87 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_91[0][0]

']

activation\_88 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_92[0][0]

']

activation\_91 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_95[0][0]

']

activation\_92 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_96[0][0]

']

batch\_normalization\_97 (Ba (None, 5, 5, 192) 576 ['conv2d\_103[0][0]']

tchNormalization)

activation\_85 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_89[0][0]

']

mixed9\_1 (Concatenate) (None, 5, 5, 768) 0 ['activation\_87[0][0]',

'activation\_88[0][0]']

concatenate\_1 (Concatenate (None, 5, 5, 768) 0 ['activation\_91[0][0]',

) 'activation\_92[0][0]']

activation\_93 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_97[0][0]

']

mixed10 (Concatenate) (None, 5, 5, 2048) 0 ['activation\_85[0][0]',

'mixed9\_1[0][0]',

'concatenate\_1[0][0]',

'activation\_93[0][0]']

global\_average\_pooling2d ( (None, 2048) 0 ['mixed10[0][0]']

GlobalAveragePooling2D)

dropout\_2 (Dropout) (None, 2048) 0 ['global\_average\_pooling2d[0][

0]']

dense\_3 (Dense) (None, 1024) 2098176 ['dropout\_2[0][0]']

dense\_4 (Dense) (None, 4) 4100 ['dense\_3[0][0]']

==================================================================================================

Total params: 23905060 (91.19 MB)

Trainable params: 2102276 (8.02 MB)

Non-trainable params: 21802784 (83.17 MB)

# Define a path for saving the model

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen1/InceptionV3.h5', monitor='val\_accuracy', save\_best\_only=True)

# Model traing and the validation

history = model.fit(

    training\_set,

    epochs=25,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/25

357/357 [==============================] - ETA: 0s - loss: 0.6989 - accuracy: 0.7472

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

357/357 [==============================] - 189s 518ms/step - loss: 0.6989 - accuracy: 0.7472 - val\_loss: 0.5574 - val\_accuracy: 0.7696

Epoch 2/25

357/357 [==============================] - 173s 484ms/step - loss: 0.4982 - accuracy: 0.8132 - val\_loss: 0.4571 - val\_accuracy: 0.8284

Epoch 3/25

357/357 [==============================] - 173s 483ms/step - loss: 0.4559 - accuracy: 0.8258 - val\_loss: 0.4132 - val\_accuracy: 0.8482

Epoch 4/25

357/357 [==============================] - 169s 474ms/step - loss: 0.4143 - accuracy: 0.8412 - val\_loss: 0.3827 - val\_accuracy: 0.8535

Epoch 5/25

357/357 [==============================] - 168s 469ms/step - loss: 0.4005 - accuracy: 0.8467 - val\_loss: 0.3869 - val\_accuracy: 0.8513

Epoch 6/25

357/357 [==============================] - 168s 472ms/step - loss: 0.3834 - accuracy: 0.8530 - val\_loss: 0.3728 - val\_accuracy: 0.8619

Epoch 7/25

357/357 [==============================] - 167s 467ms/step - loss: 0.3777 - accuracy: 0.8504 - val\_loss: 0.3857 - val\_accuracy: 0.8383

Epoch 8/25

357/357 [==============================] - 166s 464ms/step - loss: 0.3512 - accuracy: 0.8686 - val\_loss: 0.3845 - val\_accuracy: 0.8413

Epoch 9/25

357/357 [==============================] - 168s 471ms/step - loss: 0.3519 - accuracy: 0.8618 - val\_loss: 0.3593 - val\_accuracy: 0.8642

Epoch 10/25

357/357 [==============================] - 170s 475ms/step - loss: 0.3413 - accuracy: 0.8758 - val\_loss: 0.3302 - val\_accuracy: 0.8810

Epoch 11/25

357/357 [==============================] - 167s 468ms/step - loss: 0.3306 - accuracy: 0.8721 - val\_loss: 0.3235 - val\_accuracy: 0.8658

Epoch 12/25

357/357 [==============================] - 167s 467ms/step - loss: 0.3201 - accuracy: 0.8812 - val\_loss: 0.3169 - val\_accuracy: 0.8688

Epoch 13/25

357/357 [==============================] - 165s 463ms/step - loss: 0.3149 - accuracy: 0.8823 - val\_loss: 0.3398 - val\_accuracy: 0.8581

Epoch 14/25

357/357 [==============================] - 169s 473ms/step - loss: 0.3060 - accuracy: 0.8856 - val\_loss: 0.2916 - val\_accuracy: 0.8963

Epoch 15/25

357/357 [==============================] - 167s 466ms/step - loss: 0.2982 - accuracy: 0.8865 - val\_loss: 0.2994 - val\_accuracy: 0.8833

Epoch 16/25

357/357 [==============================] - 170s 476ms/step - loss: 0.2934 - accuracy: 0.8872 - val\_loss: 0.2830 - val\_accuracy: 0.9054

Epoch 17/25

357/357 [==============================] - 168s 469ms/step - loss: 0.2737 - accuracy: 0.8996 - val\_loss: 0.2812 - val\_accuracy: 0.8917

Epoch 18/25

357/357 [==============================] - 164s 460ms/step - loss: 0.2837 - accuracy: 0.8918 - val\_loss: 0.2623 - val\_accuracy: 0.8970

Epoch 19/25

357/357 [==============================] - 168s 471ms/step - loss: 0.2848 - accuracy: 0.8928 - val\_loss: 0.2951 - val\_accuracy: 0.8886

Epoch 20/25

357/357 [==============================] - 168s 470ms/step - loss: 0.2805 - accuracy: 0.8919 - val\_loss: 0.2521 - val\_accuracy: 0.9085

Epoch 21/25

357/357 [==============================] - 169s 473ms/step - loss: 0.2656 - accuracy: 0.8965 - val\_loss: 0.2910 - val\_accuracy: 0.8810

Epoch 22/25

357/357 [==============================] - 168s 472ms/step - loss: 0.2870 - accuracy: 0.8882 - val\_loss: 0.2719 - val\_accuracy: 0.8993

Epoch 23/25

357/357 [==============================] - 169s 473ms/step - loss: 0.2556 - accuracy: 0.9040 - val\_loss: 0.2230 - val\_accuracy: 0.9230

Epoch 24/25

357/357 [==============================] - 168s 471ms/step - loss: 0.2617 - accuracy: 0.9040 - val\_loss: 0.2417 - val\_accuracy: 0.9138

Epoch 25/25

357/357 [==============================] - 167s 468ms/step - loss: 0.2494 - accuracy: 0.9077 - val\_loss: 0.2599 - val\_accuracy: 0.9024

# Create the plot for InceptionV3

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Load the base pre-trained model for the ResNet101, excluding the top layer

base\_model = ResNet101(weights='imagenet', include\_top=False, input\_shape=image\_shape)

# Freezing all the layers of the model for prevent from the weights during the training

for layer in base\_model.layers:

    layer.trainable = False

# Customize the top layer of the model

model = Sequential([

  base\_model,

  tf.keras.layers.GlobalAveragePooling2D(),

  tf.keras.layers.Dense(4, activation='softmax')

])

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet101_weights_tf_dim_ordering_tf_kernels_notop.h5>

171446536/171446536 [==============================] - 6s 0us/step

# Call the summary of the model

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

resnet101 (Functional) (None, 7, 7, 2048) 42658176

global\_average\_pooling2d ( (None, 2048) 0

GlobalAveragePooling2D)

dense (Dense) (None, 4) 8196

=================================================================

Total params: 42666372 (162.76 MB)

Trainable params: 8196 (32.02 KB)

Non-trainable params: 42658176 (162.73 MB)

# Compile the moodel

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# For saving the best model a directory is given

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen1/Resnet101.h5', monitor='val\_accuracy', save\_best\_only=True)

# Model training and validation

history = model.fit(

    training\_set,

    epochs=50,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/50

357/357 [==============================] - ETA: 0s - loss: 1.1110 - accuracy: 0.5263

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

357/357 [==============================] - 2916s 8s/step - loss: 1.1110 - accuracy: 0.5263 - val\_loss: 1.0901 - val\_accuracy: 0.5698

Epoch 2/50

357/357 [==============================] - 522s 1s/step - loss: 1.0088 - accuracy: 0.6023 - val\_loss: 1.0470 - val\_accuracy: 0.5622

Epoch 3/50

357/357 [==============================] - 519s 1s/step - loss: 0.9525 - accuracy: 0.6309 - val\_loss: 0.9847 - val\_accuracy: 0.6384

Epoch 4/50

357/357 [==============================] - 528s 1s/step - loss: 0.9167 - accuracy: 0.6547 - val\_loss: 0.9685 - val\_accuracy: 0.6400

Epoch 5/50

357/357 [==============================] - 516s 1s/step - loss: 0.8888 - accuracy: 0.6535 - val\_loss: 0.9331 - val\_accuracy: 0.6369

Epoch 6/50

357/357 [==============================] - 523s 1s/step - loss: 0.8645 - accuracy: 0.6632 - val\_loss: 0.9045 - val\_accuracy: 0.6468

Epoch 7/50

357/357 [==============================] - 525s 1s/step - loss: 0.8511 - accuracy: 0.6695 - val\_loss: 0.9009 - val\_accuracy: 0.6712

Epoch 8/50

357/357 [==============================] - 525s 1s/step - loss: 0.8326 - accuracy: 0.6746 - val\_loss: 0.8706 - val\_accuracy: 0.6728

Epoch 9/50

357/357 [==============================] - 505s 1s/step - loss: 0.8189 - accuracy: 0.6807 - val\_loss: 0.8604 - val\_accuracy: 0.6712

Epoch 10/50

357/357 [==============================] - 500s 1s/step - loss: 0.8109 - accuracy: 0.6809 - val\_loss: 0.8541 - val\_accuracy: 0.6522

Epoch 11/50

357/357 [==============================] - 509s 1s/step - loss: 0.8029 - accuracy: 0.6793 - val\_loss: 0.8491 - val\_accuracy: 0.6789

Epoch 12/50

357/357 [==============================] - 507s 1s/step - loss: 0.7950 - accuracy: 0.6821 - val\_loss: 0.8777 - val\_accuracy: 0.6857

Epoch 13/50

357/357 [==============================] - 504s 1s/step - loss: 0.7763 - accuracy: 0.6893 - val\_loss: 0.8438 - val\_accuracy: 0.6766

Epoch 14/50

357/357 [==============================] - 507s 1s/step - loss: 0.7832 - accuracy: 0.6888 - val\_loss: 0.8433 - val\_accuracy: 0.6743

Epoch 15/50

357/357 [==============================] - 494s 1s/step - loss: 0.7675 - accuracy: 0.6998 - val\_loss: 0.8272 - val\_accuracy: 0.6834

Epoch 16/50

357/357 [==============================] - 504s 1s/step - loss: 0.7580 - accuracy: 0.7026 - val\_loss: 0.8267 - val\_accuracy: 0.6972

Epoch 17/50

357/357 [==============================] - 506s 1s/step - loss: 0.7544 - accuracy: 0.7021 - val\_loss: 0.8391 - val\_accuracy: 0.6728

Epoch 18/50

357/357 [==============================] - 508s 1s/step - loss: 0.7538 - accuracy: 0.7018 - val\_loss: 0.7976 - val\_accuracy: 0.7079

Epoch 19/50

357/357 [==============================] - 512s 1s/step - loss: 0.7509 - accuracy: 0.7011 - val\_loss: 0.8890 - val\_accuracy: 0.6400

Epoch 20/50

357/357 [==============================] - 502s 1s/step - loss: 0.7357 - accuracy: 0.7042 - val\_loss: 0.8453 - val\_accuracy: 0.6926

Epoch 21/50

357/357 [==============================] - 499s 1s/step - loss: 0.7343 - accuracy: 0.7109 - val\_loss: 0.8208 - val\_accuracy: 0.6903

Epoch 22/50

357/357 [==============================] - 501s 1s/step - loss: 0.7258 - accuracy: 0.7125 - val\_loss: 0.8146 - val\_accuracy: 0.6842

Epoch 23/50

357/357 [==============================] - 508s 1s/step - loss: 0.7200 - accuracy: 0.7195 - val\_loss: 0.7817 - val\_accuracy: 0.7124

Epoch 24/50

357/357 [==============================] - 505s 1s/step - loss: 0.7294 - accuracy: 0.7170 - val\_loss: 0.8032 - val\_accuracy: 0.6857

Epoch 25/50

357/357 [==============================] - 501s 1s/step - loss: 0.7254 - accuracy: 0.7147 - val\_loss: 0.7705 - val\_accuracy: 0.7086

Epoch 26/50

357/357 [==============================] - 509s 1s/step - loss: 0.7236 - accuracy: 0.7151 - val\_loss: 0.7824 - val\_accuracy: 0.6941

Epoch 27/50

357/357 [==============================] - 502s 1s/step - loss: 0.7205 - accuracy: 0.7170 - val\_loss: 0.7805 - val\_accuracy: 0.7010

Epoch 28/50

357/357 [==============================] - 498s 1s/step - loss: 0.7109 - accuracy: 0.7154 - val\_loss: 0.7816 - val\_accuracy: 0.6949

Epoch 29/50

357/357 [==============================] - 506s 1s/step - loss: 0.7132 - accuracy: 0.7219 - val\_loss: 0.7397 - val\_accuracy: 0.7155

Epoch 30/50

357/357 [==============================] - 509s 1s/step - loss: 0.7027 - accuracy: 0.7216 - val\_loss: 0.7822 - val\_accuracy: 0.6972

Epoch 31/50

357/357 [==============================] - 514s 1s/step - loss: 0.7078 - accuracy: 0.7202 - val\_loss: 0.7793 - val\_accuracy: 0.7140

Epoch 32/50

357/357 [==============================] - 511s 1s/step - loss: 0.7063 - accuracy: 0.7172 - val\_loss: 0.7421 - val\_accuracy: 0.7246

Epoch 33/50

357/357 [==============================] - 512s 1s/step - loss: 0.6867 - accuracy: 0.7298 - val\_loss: 0.7628 - val\_accuracy: 0.7147

Epoch 34/50

357/357 [==============================] - 510s 1s/step - loss: 0.6950 - accuracy: 0.7309 - val\_loss: 0.7467 - val\_accuracy: 0.7262

Epoch 35/50

357/357 [==============================] - 510s 1s/step - loss: 0.6974 - accuracy: 0.7230 - val\_loss: 0.7767 - val\_accuracy: 0.7056

Epoch 36/50

357/357 [==============================] - 512s 1s/step - loss: 0.6864 - accuracy: 0.7302 - val\_loss: 0.7381 - val\_accuracy: 0.7292

Epoch 37/50

357/357 [==============================] - 507s 1s/step - loss: 0.6857 - accuracy: 0.7256 - val\_loss: 0.7631 - val\_accuracy: 0.7140

Epoch 38/50

357/357 [==============================] - 511s 1s/step - loss: 0.7037 - accuracy: 0.7218 - val\_loss: 0.7933 - val\_accuracy: 0.7056

Epoch 39/50

357/357 [==============================] - 510s 1s/step - loss: 0.6789 - accuracy: 0.7402 - val\_loss: 0.7495 - val\_accuracy: 0.7101

Epoch 40/50

357/357 [==============================] - 509s 1s/step - loss: 0.6761 - accuracy: 0.7332 - val\_loss: 0.8056 - val\_accuracy: 0.7079

Epoch 41/50

357/357 [==============================] - 515s 1s/step - loss: 0.6834 - accuracy: 0.7325 - val\_loss: 0.7370 - val\_accuracy: 0.7208

Epoch 42/50

357/357 [==============================] - 511s 1s/step - loss: 0.6786 - accuracy: 0.7332 - val\_loss: 0.7670 - val\_accuracy: 0.7071

Epoch 43/50

357/357 [==============================] - 494s 1s/step - loss: 0.6772 - accuracy: 0.7325 - val\_loss: 0.7309 - val\_accuracy: 0.7254

Epoch 44/50

357/357 [==============================] - 545s 2s/step - loss: 0.6706 - accuracy: 0.7407 - val\_loss: 0.7343 - val\_accuracy: 0.7124

Epoch 45/50

357/357 [==============================] - 515s 1s/step - loss: 0.6771 - accuracy: 0.7309 - val\_loss: 0.7563 - val\_accuracy: 0.7223

Epoch 46/50

357/357 [==============================] - 526s 1s/step - loss: 0.6721 - accuracy: 0.7384 - val\_loss: 0.7274 - val\_accuracy: 0.7292

Epoch 47/50

357/357 [==============================] - 529s 1s/step - loss: 0.6785 - accuracy: 0.7384 - val\_loss: 0.7548 - val\_accuracy: 0.7170

Epoch 48/50

357/357 [==============================] - 531s 1s/step - loss: 0.6691 - accuracy: 0.7423 - val\_loss: 0.6967 - val\_accuracy: 0.7193

Epoch 49/50

357/357 [==============================] - 527s 1s/step - loss: 0.6611 - accuracy: 0.7442 - val\_loss: 0.7299 - val\_accuracy: 0.7285

Epoch 50/50

357/357 [==============================] - 528s 1s/step - loss: 0.6596 - accuracy: 0.7419 - val\_loss: 0.7238 - val\_accuracy: 0.7109

# Plot the training graph of the ResNet101 model

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Developmet of the VGG16 model from scratch

# Define the model

model = Sequential()

# Block 1

model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input\_shape=image\_shape))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 2

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 3

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 4

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(len(class\_labels), activation='softmax'))

model.summary()

Model: "sequential\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_20 (Conv2D) (None, 224, 224, 64) 1792

conv2d\_21 (Conv2D) (None, 224, 224, 64) 36928

max\_pooling2d\_8 (MaxPoolin (None, 112, 112, 64) 0

g2D)

batch\_normalization\_8 (Bat (None, 112, 112, 64) 256

chNormalization)

conv2d\_22 (Conv2D) (None, 112, 112, 128) 73856

conv2d\_23 (Conv2D) (None, 112, 112, 128) 147584

max\_pooling2d\_9 (MaxPoolin (None, 56, 56, 128) 0

g2D)

batch\_normalization\_9 (Bat (None, 56, 56, 128) 512

chNormalization)

conv2d\_24 (Conv2D) (None, 56, 56, 256) 295168

conv2d\_25 (Conv2D) (None, 56, 56, 256) 590080

conv2d\_26 (Conv2D) (None, 56, 56, 256) 590080

max\_pooling2d\_10 (MaxPooli (None, 28, 28, 256) 0

ng2D)

batch\_normalization\_10 (Ba (None, 28, 28, 256) 1024

tchNormalization)

conv2d\_27 (Conv2D) (None, 28, 28, 512) 1180160

conv2d\_28 (Conv2D) (None, 28, 28, 512) 2359808

conv2d\_29 (Conv2D) (None, 28, 28, 512) 2359808

max\_pooling2d\_11 (MaxPooli (None, 14, 14, 512) 0

ng2D)

batch\_normalization\_11 (Ba (None, 14, 14, 512) 2048

tchNormalization)

flatten\_2 (Flatten) (None, 100352) 0

dense\_6 (Dense) (None, 512) 51380736

dropout\_4 (Dropout) (None, 512) 0

dense\_7 (Dense) (None, 512) 262656

dropout\_5 (Dropout) (None, 512) 0

dense\_8 (Dense) (None, 4) 2052

=================================================================

Total params: 59284548 (226.15 MB)

Trainable params: 59282628 (226.15 MB)

Non-trainable params: 1920 (7.50 KB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen1/VGG16.h5', monitor='val\_accuracy', save\_best\_only=True)

# Train the model

history1 = model.fit(

    training\_set,

    epochs=10,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/10

357/357 [==============================] - 1771s 5s/step - loss: 2.0266 - accuracy: 0.4833 - val\_loss: 3.0815 - val\_accuracy: 0.2784

Epoch 2/10

357/357 [==============================] - 1777s 5s/step - loss: 1.2719 - accuracy: 0.4386 - val\_loss: 1.4387 - val\_accuracy: 0.4088

Epoch 3/10

357/357 [==============================] - 1782s 5s/step - loss: 1.2640 - accuracy: 0.3912 - val\_loss: 1.6579 - val\_accuracy: 0.4638

Epoch 4/10

357/357 [==============================] - 1776s 5s/step - loss: 1.1716 - accuracy: 0.4535 - val\_loss: 1.7875 - val\_accuracy: 0.4272

Epoch 5/10

357/357 [==============================] - 1768s 5s/step - loss: 1.1298 - accuracy: 0.4481 - val\_loss: 1.4294 - val\_accuracy: 0.3097

Epoch 6/10

357/357 [==============================] - 1767s 5s/step - loss: 1.1070 - accuracy: 0.4872 - val\_loss: 1.2353 - val\_accuracy: 0.4600

Epoch 7/10

357/357 [==============================] - 1782s 5s/step - loss: 1.1678 - accuracy: 0.4637 - val\_loss: 1.7510 - val\_accuracy: 0.4928

Epoch 8/10

357/357 [==============================] - 1781s 5s/step - loss: 1.0730 - accuracy: 0.4849 - val\_loss: 2.0613 - val\_accuracy: 0.3532

Epoch 9/10

357/357 [==============================] - 1786s 5s/step - loss: 1.0499 - accuracy: 0.5263 - val\_loss: 1.0536 - val\_accuracy: 0.6293

Epoch 10/10

357/357 [==============================] - 1764s 5s/step - loss: 0.9560 - accuracy: 0.5949 - val\_loss: 0.9587 - val\_accuracy: 0.5858

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history1.history['accuracy'], label='Training Accuracy')

plt.plot(history1.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history1.history['loss'], label='Training Loss')

plt.plot(history1.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Import libraries

import os

from tensorflow.keras.models import load\_model

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score,f1\_score

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

image\_size = 224

# Initialze empty cell to store the test images anf their corresponding labels

x\_test = []

y\_test = []

# To analyze the photos in the testing dataset, iterate over each class label.

for label in labels:

  # Build a path to the test images for the given label.

    testingpath = os.path.join('/content/drive/MyDrive/Colab Notebooks/Testing', label)

    for file in os.listdir(testingpath):

        image = cv2.imread(os.path.join(testingpath, file), cv2.IMREAD\_COLOR)

        image = cv2.resize(image, (image\_size, image\_size))

        x\_test.append(image)

        y\_test.append(labels.index(label))

x\_test = np.array(x\_test) / 255.0

y\_test = np.array(y\_test)

# Normalize the pixel values and convert the list of images to a numpy array.

y\_test\_categorical = tf.keras.utils.to\_categorical(y\_test, num\_classes=len(labels))

# Give specific locations for the saved model weights files for different architectures.

vgg\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen1/VGG16.h5'

inception\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen1/InceptionV3.h5'

resnet\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen1/Resnet101.h5'

# Load the pre-trained models from the specified file paths

vgg\_model = tf.keras.models.load\_model(vgg\_model\_path)

inception\_model = tf.keras.models.load\_model(inception\_model\_path)

resnet\_model = tf.keras.models.load\_model(resnet\_model\_path)

# Function to evaluate model

def evaluate\_model(model, x\_test, y\_test, y\_test\_categorical, labels):

    predictions = model.predict(x\_test)

    y\_pred = np.argmax(predictions, axis=1)

    y\_true = np.argmax(y\_test\_categorical, axis=1)

    # Classification report

    print("Classification Report:")

    print(classification\_report(y\_true, y\_pred, target\_names=labels))

    # Confusion matrix

    cm = confusion\_matrix(y\_true, y\_pred)

    plt.figure(figsize=(10, 7))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

    plt.xlabel('Predicted')

    plt.ylabel('Actual')

    plt.title('Confusion Matrix')

    plt.show()

# Evaluate model

print("VGG Model Evaluation:")

evaluate\_model(vgg\_model, x\_test, y\_test, y\_test\_categorical, labels)

VGG Model Evaluation:

41/41 [==============================] - 99s 2s/step

Classification Report:

precision recall f1-score support

glioma 0.62 0.79 0.70 300

meningioma 0.42 0.08 0.13 306

notumor 0.60 0.94 0.73 405

pituitary 0.78 0.63 0.70 300

accuracy 0.63 1311

macro avg 0.61 0.61 0.56 1311

weighted avg 0.60 0.63 0.57 1311

evaluate\_model(inception\_model, x\_test, y\_test, y\_test\_categorical, labels)

41/41 [==============================] - 31s 720ms/step

Classification Report:

precision recall f1-score support

glioma 0.88 0.90 0.89 300

meningioma 0.88 0.82 0.85 306

notumor 0.99 0.98 0.98 405

pituitary 0.91 0.97 0.94 300

accuracy 0.92 1311

macro avg 0.92 0.92 0.92 1311

weighted avg 0.92 0.92 0.92 1311

evaluate\_model(resnet\_model, x\_test, y\_test, y\_test\_categorical, labels)

41/41 [==============================] - 90s 2s/step

Classification Report:

precision recall f1-score support

glioma 0.70 0.64 0.67 300

meningioma 0.59 0.44 0.50 306

notumor 0.82 0.84 0.83 405

pituitary 0.70 0.92 0.79 300

accuracy 0.72 1311

macro avg 0.70 0.71 0.70 1311

weighted avg 0.71 0.72 0.71 1311

import numpy as np

import matplotlib.pyplot as plt

import cv2

import os

import tensorflow as tf

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

image\_size = 224

image\_path = '/content/drive/MyDrive/Colab Notebooks/Testing/meningioma/Te-meTr\_0001.jpg'

check\_image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

check\_image = cv2.resize(check\_image, (image\_size, image\_size))

check\_image = np.expand\_dims(check\_image, axis=0) / 255.0

model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen2/Inceptionv3.h5'

model = tf.keras.models.load\_model(model\_path)

prediction = model.predict(check\_image)

predicted\_index = np.argmax(prediction[0])

predicted\_class = labels[predicted\_index]

confidence = prediction[0][predicted\_index]

print('Result: ', predicted\_class)

1/1 [==============================] - 1s 1s/step

Result: meningioma

**This is the code of second data amugmentation:**

#install tensorflow 2.13.0

!pip install tensorflow==2.13.0

Requirement already satisfied: tensorflow==2.13.0 in /usr/local/lib/python3.10/dist-packages (2.13.0)

Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.4.0)

Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.6.3)

Requirement already satisfied: flatbuffers>=23.1.21 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (24.3.25)

Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.4.0)

Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.2.0)

Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.64.1)

Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.11.0)

Requirement already satisfied: keras<2.14,>=2.13.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.1)

Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (18.1.1)

Requirement already satisfied: numpy<=1.24.3,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.24.3)

Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.3.0)

Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (24.1)

Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (3.20.3)

Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (71.0.4)

Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.16.0)

Requirement already satisfied: tensorboard<2.14,>=2.13 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.0)

Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.13.0)

Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (2.4.0)

Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (4.5.0)

Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (1.16.0)

Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.13.0) (0.37.1)

Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.13.0) (0.44.0)

Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.27.0)

Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (1.0.0)

Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.7)

Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.32.3)

Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.7.2)

Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.0.4)

Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (5.5.0)

Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.4.0)

Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (4.9)

Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (1.3.1)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.3.2)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.7)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.0.7)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2024.7.4)

Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (2.1.5)

Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (0.6.0)

Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow==2.13.0) (3.2.2)

#mount the google colab

from google.colab import drive

drive.mount('/content/drive')

Mounted at /content/drive

#importing the libraries

import os

import cv2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from sklearn.utils import shuffle

from sklearn.model\_selection import train\_test\_split as tts

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tqdm import tqdm

# Directory path of the training data

train\_dir = '/content/drive/MyDrive/Colab Notebooks/Training'

# Listing the files in each directory to count the number of training images for each type of tumor.

n\_train\_glioma = len(os.listdir(f'{train\_dir}/glioma'))

n\_train\_mening = len(os.listdir(f'{train\_dir}/meningioma'))

n\_train\_pit = len(os.listdir(f'{train\_dir}/pituitary'))

n\_train\_non = len(os.listdir(f'{train\_dir}/notumor'))

# Total length of the whole classes combined

total\_train = n\_train\_glioma + n\_train\_mening + n\_train\_pit + n\_train\_non

# Create a list of classes and label the total number of images.

count = [n\_train\_glioma, n\_train\_mening, n\_train\_pit, n\_train\_non, total\_train]

length = ['glioma', ' meningioma', 'pituitary', 'no tumor', 'total images']

# Create a loop to print the number of images belongs to each class

for i, sum in zip(length, count):

  print(f'Total counts of {i} is {sum}')

  print(100\*'-')

Total counts of glioma is 1317

----------------------------------------------------------------------------------------------------

Total counts of meningioma is 1331

----------------------------------------------------------------------------------------------------

Total counts of pituitary is 1457

----------------------------------------------------------------------------------------------------

Total counts of no tumor is 1595

----------------------------------------------------------------------------------------------------

Total counts of total images is 5700

# Define the name of all classes

class\_labels = ['Glioma', 'Meningioma', 'Pituitary', 'No Tumor']

# Creatng a list count for the number of images belong to each class

class\_counts = [n\_train\_glioma, n\_train\_mening, n\_train\_pit, n\_train\_non]

# Create a countplot

# Create a white grid background for the plot

sns.set(style="whitegrid")

# Give the figure size for the plot

plt.figure(figsize=(8, 6))

# Create a bar plot which correspond the class to the number of images

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Annotate the counts on top of the bars

for i, count in enumerate(class\_counts):

    ax.text(i, count + 2, str(count), ha='center', va='bottom', fontsize=12)

# Add titles and labels for the graph

plt.title("Distribution of Classes in Training Data")

plt.xlabel("Class Label")

plt.ylabel("Count")

plt.show()

<ipython-input-7-ea309d3e31b6>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Directory path of the test data

test\_dir="/content/drive/MyDrive/Colab Notebooks/Testing"

# Listing the files in each directory to count the number of testing images for each type of tumor.

n\_test\_glioma = len(os.listdir(f'{test\_dir}/glioma'))

n\_test\_mening = len(os.listdir(f'{test\_dir}/meningioma'))

n\_test\_pit = len(os.listdir(f'{test\_dir}/pituitary'))

n\_test\_non = len(os.listdir(f'{test\_dir}/notumor'))

# Total length of the whole classes combined

total\_test = n\_test\_glioma + n\_test\_mening + n\_test\_pit + n\_test\_non

# Create a list of classes and label the total number of images.

count = [n\_test\_glioma, n\_test\_mening, n\_test\_pit, n\_test\_non, total\_test]

length = ['glioma', ' meningioma', 'pituitary', 'no tumor', 'total images']

# Create a loop to print the number of images belongs to each class

for i, sum in zip(length, count):

  print(f'Total counts of {i} is {sum}')

  print(100\*'-')

Total counts of glioma is 300

----------------------------------------------------------------------------------------------------

Total counts of meningioma is 306

----------------------------------------------------------------------------------------------------

Total counts of pituitary is 300

----------------------------------------------------------------------------------------------------

Total counts of no tumor is 405

----------------------------------------------------------------------------------------------------

Total counts of total images is 1311

# Create a countplot

# Create a white grid background for the plot

sns.set(style="whitegrid")

plt.figure(figsize=(8, 6))

# Create a bar plot which correspond the class to the number of images

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Annotate the counts on top of the bars

for i, count in enumerate(class\_counts):

    ax.text(i, count + 2, str(count), ha='center', va='bottom', fontsize=12)

# Add titles and labels for the graph

plt.title("Distribution of Classes in Testing Data")

plt.xlabel("Class Label")

plt.ylabel("Count")

plt.show()

<ipython-input-10-bcf986257a76>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

ax = sns.barplot(x=class\_labels, y=class\_counts, palette="viridis")

# Create a Data generator on training and testing set with data augmentation

  train\_datagen=ImageDataGenerator(rescale=1./255,

                                  rotation\_range=20,

                                  width\_shift\_range=0.2,

                                  height\_shift\_range=0.2,

                                  zoom\_range=0.2,

                                  shear\_range=0.2,

                                  horizontal\_flip=True,

                                  vertical\_flip=True,

                                  fill\_mode='reflect')

# Apply only rescaling for the test data

  test\_datagen=ImageDataGenerator(rescale=1./255)

#Creating Training set and validation set

 training\_set=train\_datagen.flow\_from\_directory(train\_dir,

                                               target\_size=(224,224),

                                               batch\_size=16,

                                               class\_mode='categorical')

test\_set=test\_datagen.flow\_from\_directory(test\_dir,

                                          target\_size=(224,224),

                                          batch\_size=16,

                                          class\_mode="categorical")

Found 5700 images belonging to 4 classes.

Found 1311 images belonging to 4 classes.

# Apply a batch size for the test and training of data

batch\_size=16

# Define a colour channel

color\_channel=3

# Define the image shape for the imout images

image\_shape=(224,224,3)

#import some more libraries for the model development

import tensorflow as tf

from tensorflow.keras.applications import InceptionV3, ResNet101

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input, LSTM, Conv2D, MaxPooling2D, MaxPool2D, Flatten, Dense, Dropout, Reshape, BatchNormalization,GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, LearningRateScheduler

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import regularizers

# Load the base pre-trained model for the ResNet101, excluding the top layer

base\_model = ResNet101(weights='imagenet', include\_top=False, input\_shape=image\_shape)

# Freezing all the layers of the model for prevent from the weights during the training

for layer in base\_model.layers:

    layer.trainable = False

# Customize the top layer of the model

model = Sequential([

  base\_model,

  tf.keras.layers.GlobalAveragePooling2D(),

  tf.keras.layers.Dense(4, activation='softmax')

])

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet101_weights_tf_dim_ordering_tf_kernels_notop.h5>

171446536/171446536 [==============================] - 1s 0us/step

# Call the summary of the model

model.summary()

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

resnet101 (Functional) (None, 7, 7, 2048) 42658176

global\_average\_pooling2d\_1 (None, 2048) 0

(GlobalAveragePooling2D)

dense\_1 (Dense) (None, 4) 8196

=================================================================

Total params: 42666372 (162.76 MB)

Trainable params: 8196 (32.02 KB)

Non-trainable params: 42658176 (162.73 MB)

# Compile the moodel

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# For saving the best model a directory is given

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen2/Resnet101.h5', monitor='val\_accuracy', save\_best\_only=True)

# Model training and validation

history = model.fit(

    training\_set,

    epochs=50,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/50

357/357 [==============================] - ETA: 0s - loss: 1.1819 - accuracy: 0.4626

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

357/357 [==============================] - 831s 2s/step - loss: 1.1819 - accuracy: 0.4626 - val\_loss: 1.2119 - val\_accuracy: 0.4729

Epoch 2/50

357/357 [==============================] - 468s 1s/step - loss: 1.1011 - accuracy: 0.5267 - val\_loss: 1.1481 - val\_accuracy: 0.5584

Epoch 3/50

357/357 [==============================] - 472s 1s/step - loss: 1.0660 - accuracy: 0.5474 - val\_loss: 1.1382 - val\_accuracy: 0.5271

Epoch 4/50

357/357 [==============================] - 469s 1s/step - loss: 1.0482 - accuracy: 0.5589 - val\_loss: 1.1027 - val\_accuracy: 0.5584

Epoch 5/50

357/357 [==============================] - 472s 1s/step - loss: 1.0240 - accuracy: 0.5754 - val\_loss: 1.0718 - val\_accuracy: 0.5828

Epoch 6/50

357/357 [==============================] - 475s 1s/step - loss: 1.0098 - accuracy: 0.5851 - val\_loss: 1.0578 - val\_accuracy: 0.5774

Epoch 7/50

357/357 [==============================] - 465s 1s/step - loss: 1.0044 - accuracy: 0.5840 - val\_loss: 1.0707 - val\_accuracy: 0.5736

Epoch 8/50

357/357 [==============================] - 467s 1s/step - loss: 0.9954 - accuracy: 0.5832 - val\_loss: 1.0944 - val\_accuracy: 0.5469

Epoch 9/50

357/357 [==============================] - 485s 1s/step - loss: 0.9825 - accuracy: 0.5923 - val\_loss: 1.0447 - val\_accuracy: 0.5873

Epoch 10/50

357/357 [==============================] - 483s 1s/step - loss: 0.9807 - accuracy: 0.5835 - val\_loss: 1.0154 - val\_accuracy: 0.5973

Epoch 11/50

357/357 [==============================] - 487s 1s/step - loss: 0.9733 - accuracy: 0.5918 - val\_loss: 1.0027 - val\_accuracy: 0.5995

Epoch 12/50

357/357 [==============================] - 486s 1s/step - loss: 0.9602 - accuracy: 0.6012 - val\_loss: 1.0302 - val\_accuracy: 0.6110

Epoch 13/50

357/357 [==============================] - 485s 1s/step - loss: 0.9532 - accuracy: 0.6016 - val\_loss: 1.0016 - val\_accuracy: 0.6133

Epoch 14/50

357/357 [==============================] - 485s 1s/step - loss: 0.9465 - accuracy: 0.6123 - val\_loss: 1.0047 - val\_accuracy: 0.5973

Epoch 15/50

357/357 [==============================] - 486s 1s/step - loss: 0.9532 - accuracy: 0.6004 - val\_loss: 0.9946 - val\_accuracy: 0.6194

Epoch 16/50

357/357 [==============================] - 486s 1s/step - loss: 0.9413 - accuracy: 0.6072 - val\_loss: 0.9932 - val\_accuracy: 0.6041

Epoch 17/50

357/357 [==============================] - 490s 1s/step - loss: 0.9323 - accuracy: 0.6075 - val\_loss: 1.0016 - val\_accuracy: 0.6095

Epoch 18/50

357/357 [==============================] - 473s 1s/step - loss: 0.9411 - accuracy: 0.6025 - val\_loss: 1.0203 - val\_accuracy: 0.6011

Epoch 19/50

357/357 [==============================] - 470s 1s/step - loss: 0.9249 - accuracy: 0.6170 - val\_loss: 0.9584 - val\_accuracy: 0.6285

Epoch 20/50

357/357 [==============================] - 475s 1s/step - loss: 0.9328 - accuracy: 0.6060 - val\_loss: 0.9865 - val\_accuracy: 0.6209

Epoch 21/50

357/357 [==============================] - 473s 1s/step - loss: 0.9250 - accuracy: 0.6119 - val\_loss: 1.0219 - val\_accuracy: 0.6003

Epoch 22/50

357/357 [==============================] - 477s 1s/step - loss: 0.9207 - accuracy: 0.6174 - val\_loss: 1.0471 - val\_accuracy: 0.6110

Epoch 23/50

357/357 [==============================] - 471s 1s/step - loss: 0.9108 - accuracy: 0.6246 - val\_loss: 1.0087 - val\_accuracy: 0.6255

Epoch 24/50

357/357 [==============================] - 468s 1s/step - loss: 0.9077 - accuracy: 0.6232 - val\_loss: 1.0177 - val\_accuracy: 0.6087

Epoch 25/50

357/357 [==============================] - 473s 1s/step - loss: 0.9133 - accuracy: 0.6258 - val\_loss: 1.0280 - val\_accuracy: 0.5789

Epoch 26/50

357/357 [==============================] - 470s 1s/step - loss: 0.8996 - accuracy: 0.6311 - val\_loss: 0.9560 - val\_accuracy: 0.6247

Epoch 27/50

357/357 [==============================] - 464s 1s/step - loss: 0.9167 - accuracy: 0.6272 - val\_loss: 0.9696 - val\_accuracy: 0.6255

Epoch 28/50

357/357 [==============================] - 467s 1s/step - loss: 0.9074 - accuracy: 0.6254 - val\_loss: 0.9509 - val\_accuracy: 0.6331

Epoch 29/50

357/357 [==============================] - 474s 1s/step - loss: 0.9066 - accuracy: 0.6296 - val\_loss: 0.9423 - val\_accuracy: 0.6255

Epoch 30/50

357/357 [==============================] - 464s 1s/step - loss: 0.9015 - accuracy: 0.6277 - val\_loss: 0.9565 - val\_accuracy: 0.6331

Epoch 31/50

357/357 [==============================] - 461s 1s/step - loss: 0.9030 - accuracy: 0.6282 - val\_loss: 0.9648 - val\_accuracy: 0.6049

Epoch 32/50

357/357 [==============================] - 461s 1s/step - loss: 0.8948 - accuracy: 0.6296 - val\_loss: 0.9501 - val\_accuracy: 0.6178

Epoch 33/50

357/357 [==============================] - 466s 1s/step - loss: 0.8919 - accuracy: 0.6372 - val\_loss: 1.0305 - val\_accuracy: 0.6018

Epoch 34/50

357/357 [==============================] - 465s 1s/step - loss: 0.8950 - accuracy: 0.6298 - val\_loss: 0.9925 - val\_accuracy: 0.6209

Epoch 35/50

357/357 [==============================] - 469s 1s/step - loss: 0.8885 - accuracy: 0.6332 - val\_loss: 1.0023 - val\_accuracy: 0.6110

Epoch 36/50

357/357 [==============================] - 471s 1s/step - loss: 0.8876 - accuracy: 0.6311 - val\_loss: 0.9459 - val\_accuracy: 0.6186

Epoch 37/50

357/357 [==============================] - 474s 1s/step - loss: 0.8866 - accuracy: 0.6379 - val\_loss: 0.9387 - val\_accuracy: 0.6278

Epoch 38/50

357/357 [==============================] - 469s 1s/step - loss: 0.8852 - accuracy: 0.6312 - val\_loss: 0.9203 - val\_accuracy: 0.6369

Epoch 39/50

357/357 [==============================] - 468s 1s/step - loss: 0.8885 - accuracy: 0.6282 - val\_loss: 0.9613 - val\_accuracy: 0.6316

Epoch 40/50

357/357 [==============================] - 473s 1s/step - loss: 0.8830 - accuracy: 0.6372 - val\_loss: 0.9388 - val\_accuracy: 0.6201

Epoch 41/50

357/357 [==============================] - 474s 1s/step - loss: 0.8806 - accuracy: 0.6342 - val\_loss: 0.8971 - val\_accuracy: 0.6270

Epoch 42/50

357/357 [==============================] - 488s 1s/step - loss: 0.8751 - accuracy: 0.6433 - val\_loss: 0.9255 - val\_accuracy: 0.6392

Epoch 43/50

357/357 [==============================] - 489s 1s/step - loss: 0.8778 - accuracy: 0.6400 - val\_loss: 0.8968 - val\_accuracy: 0.6415

Epoch 44/50

357/357 [==============================] - 488s 1s/step - loss: 0.8808 - accuracy: 0.6372 - val\_loss: 0.9072 - val\_accuracy: 0.6392

Epoch 45/50

357/357 [==============================] - 488s 1s/step - loss: 0.8681 - accuracy: 0.6414 - val\_loss: 0.9364 - val\_accuracy: 0.6423

Epoch 46/50

357/357 [==============================] - 486s 1s/step - loss: 0.8668 - accuracy: 0.6391 - val\_loss: 0.9488 - val\_accuracy: 0.6316

Epoch 47/50

357/357 [==============================] - 488s 1s/step - loss: 0.8730 - accuracy: 0.6351 - val\_loss: 0.9272 - val\_accuracy: 0.6407

Epoch 48/50

357/357 [==============================] - 485s 1s/step - loss: 0.8716 - accuracy: 0.6326 - val\_loss: 0.9081 - val\_accuracy: 0.6384

Epoch 49/50

357/357 [==============================] - 482s 1s/step - loss: 0.8573 - accuracy: 0.6442 - val\_loss: 0.9527 - val\_accuracy: 0.6362

Epoch 50/50

357/357 [==============================] - 486s 1s/step - loss: 0.8741 - accuracy: 0.6489 - val\_loss: 0.9049 - val\_accuracy: 0.6453

# Plot the training graph of the ResNet101 model

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Developmet of the VGG16 model from scratch

# Define the model

model = Sequential()

# Block 1

model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input\_shape=image\_shape))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 2

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 3

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

# Block 4

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(BatchNormalization())

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(len(class\_labels), activation='softmax'))

model.summary()

Model: "sequential\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_20 (Conv2D) (None, 224, 224, 64) 1792

conv2d\_21 (Conv2D) (None, 224, 224, 64) 36928

max\_pooling2d\_8 (MaxPoolin (None, 112, 112, 64) 0

g2D)

batch\_normalization\_8 (Bat (None, 112, 112, 64) 256

chNormalization)

conv2d\_22 (Conv2D) (None, 112, 112, 128) 73856

conv2d\_23 (Conv2D) (None, 112, 112, 128) 147584

max\_pooling2d\_9 (MaxPoolin (None, 56, 56, 128) 0

g2D)

batch\_normalization\_9 (Bat (None, 56, 56, 128) 512

chNormalization)

conv2d\_24 (Conv2D) (None, 56, 56, 256) 295168

conv2d\_25 (Conv2D) (None, 56, 56, 256) 590080

conv2d\_26 (Conv2D) (None, 56, 56, 256) 590080

max\_pooling2d\_10 (MaxPooli (None, 28, 28, 256) 0

ng2D)

batch\_normalization\_10 (Ba (None, 28, 28, 256) 1024

tchNormalization)

conv2d\_27 (Conv2D) (None, 28, 28, 512) 1180160

conv2d\_28 (Conv2D) (None, 28, 28, 512) 2359808

conv2d\_29 (Conv2D) (None, 28, 28, 512) 2359808

max\_pooling2d\_11 (MaxPooli (None, 14, 14, 512) 0

ng2D)

batch\_normalization\_11 (Ba (None, 14, 14, 512) 2048

tchNormalization)

flatten\_2 (Flatten) (None, 100352) 0

dense\_6 (Dense) (None, 512) 51380736

dropout\_4 (Dropout) (None, 512) 0

dense\_7 (Dense) (None, 512) 262656

dropout\_5 (Dropout) (None, 512) 0

dense\_8 (Dense) (None, 4) 2052

=================================================================

Total params: 59284548 (226.15 MB)

Trainable params: 59282628 (226.15 MB)

Non-trainable params: 1920 (7.50 KB)

# compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen2/VGG16.h5', monitor='val\_accuracy', save\_best\_only=True)

# Train the model

history1 = model.fit(

    training\_set,

    epochs=10,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/10

357/357 [==============================] - ETA: 0s - loss: 2.9799 - accuracy: 0.4616

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

357/357 [==============================] - 2374s 7s/step - loss: 2.9799 - accuracy: 0.4616 - val\_loss: 2.4172 - val\_accuracy: 0.2288

Epoch 2/10

357/357 [==============================] - 2379s 7s/step - loss: 1.1155 - accuracy: 0.5272 - val\_loss: 1.2083 - val\_accuracy: 0.5133

Epoch 3/10

357/357 [==============================] - 2320s 6s/step - loss: 1.0039 - accuracy: 0.5693 - val\_loss: 1.4118 - val\_accuracy: 0.5332

Epoch 4/10

357/357 [==============================] - 2375s 7s/step - loss: 0.9124 - accuracy: 0.6193 - val\_loss: 1.0801 - val\_accuracy: 0.6095

Epoch 5/10

357/357 [==============================] - 2392s 7s/step - loss: 0.8877 - accuracy: 0.6270 - val\_loss: 2.1643 - val\_accuracy: 0.4027

Epoch 6/10

357/357 [==============================] - 2425s 7s/step - loss: 0.8257 - accuracy: 0.6746 - val\_loss: 0.8491 - val\_accuracy: 0.6621

Epoch 7/10

357/357 [==============================] - 2447s 7s/step - loss: 0.8109 - accuracy: 0.6768 - val\_loss: 2.1530 - val\_accuracy: 0.4439

Epoch 8/10

357/357 [==============================] - 2373s 7s/step - loss: 0.7580 - accuracy: 0.6947 - val\_loss: 0.6724 - val\_accuracy: 0.7155

Epoch 9/10

357/357 [==============================] - 2327s 7s/step - loss: 0.7424 - accuracy: 0.7046 - val\_loss: 0.9335 - val\_accuracy: 0.6178

Epoch 10/10

357/357 [==============================] - 2355s 7s/step - loss: 0.6958 - accuracy: 0.7226 - val\_loss: 1.4081 - val\_accuracy: 0.6034

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history1.history['accuracy'], label='Training Accuracy')

plt.plot(history1.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history1.history['loss'], label='Training Loss')

plt.plot(history1.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Load the base model pre-trained model of Inception V3

base\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=image\_shape)

#  Freeze each layer in the base model to halt weight changes during training.

for layer in base\_model.layers:

    layer.trainable = False

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dropout(0.5)(x)

x = Dense(1024, activation='relu')(x)

predictions = Dense(len(class\_labels), activation='softmax')(x)

# Create the complete model

model = Model(inputs=base\_model.input, outputs=predictions)

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5>

87910968/87910968 [==============================] - 1s 0us/step

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 224, 224, 3)] 0 []

conv2d (Conv2D) (None, 111, 111, 32) 864 ['input\_1[0][0]']

batch\_normalization (Batch (None, 111, 111, 32) 96 ['conv2d[0][0]']

Normalization)

activation (Activation) (None, 111, 111, 32) 0 ['batch\_normalization[0][0]']

conv2d\_1 (Conv2D) (None, 109, 109, 32) 9216 ['activation[0][0]']

batch\_normalization\_1 (Bat (None, 109, 109, 32) 96 ['conv2d\_1[0][0]']

chNormalization)

activation\_1 (Activation) (None, 109, 109, 32) 0 ['batch\_normalization\_1[0][0]'

]

conv2d\_2 (Conv2D) (None, 109, 109, 64) 18432 ['activation\_1[0][0]']

batch\_normalization\_2 (Bat (None, 109, 109, 64) 192 ['conv2d\_2[0][0]']

chNormalization)

activation\_2 (Activation) (None, 109, 109, 64) 0 ['batch\_normalization\_2[0][0]'

]

max\_pooling2d (MaxPooling2 (None, 54, 54, 64) 0 ['activation\_2[0][0]']

D)

conv2d\_3 (Conv2D) (None, 54, 54, 80) 5120 ['max\_pooling2d[0][0]']

batch\_normalization\_3 (Bat (None, 54, 54, 80) 240 ['conv2d\_3[0][0]']

chNormalization)

activation\_3 (Activation) (None, 54, 54, 80) 0 ['batch\_normalization\_3[0][0]'

]

conv2d\_4 (Conv2D) (None, 52, 52, 192) 138240 ['activation\_3[0][0]']

batch\_normalization\_4 (Bat (None, 52, 52, 192) 576 ['conv2d\_4[0][0]']

chNormalization)

activation\_4 (Activation) (None, 52, 52, 192) 0 ['batch\_normalization\_4[0][0]'

]

max\_pooling2d\_1 (MaxPoolin (None, 25, 25, 192) 0 ['activation\_4[0][0]']

g2D)

conv2d\_8 (Conv2D) (None, 25, 25, 64) 12288 ['max\_pooling2d\_1[0][0]']

batch\_normalization\_8 (Bat (None, 25, 25, 64) 192 ['conv2d\_8[0][0]']

chNormalization)

activation\_8 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_8[0][0]'

]

conv2d\_6 (Conv2D) (None, 25, 25, 48) 9216 ['max\_pooling2d\_1[0][0]']

conv2d\_9 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_8[0][0]']

batch\_normalization\_6 (Bat (None, 25, 25, 48) 144 ['conv2d\_6[0][0]']

chNormalization)

batch\_normalization\_9 (Bat (None, 25, 25, 96) 288 ['conv2d\_9[0][0]']

chNormalization)

activation\_6 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_6[0][0]'

]

activation\_9 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_9[0][0]'

]

average\_pooling2d (Average (None, 25, 25, 192) 0 ['max\_pooling2d\_1[0][0]']

Pooling2D)

conv2d\_5 (Conv2D) (None, 25, 25, 64) 12288 ['max\_pooling2d\_1[0][0]']

conv2d\_7 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_6[0][0]']

conv2d\_10 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_9[0][0]']

conv2d\_11 (Conv2D) (None, 25, 25, 32) 6144 ['average\_pooling2d[0][0]']

batch\_normalization\_5 (Bat (None, 25, 25, 64) 192 ['conv2d\_5[0][0]']

chNormalization)

batch\_normalization\_7 (Bat (None, 25, 25, 64) 192 ['conv2d\_7[0][0]']

chNormalization)

batch\_normalization\_10 (Ba (None, 25, 25, 96) 288 ['conv2d\_10[0][0]']

tchNormalization)

batch\_normalization\_11 (Ba (None, 25, 25, 32) 96 ['conv2d\_11[0][0]']

tchNormalization)

activation\_5 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_5[0][0]'

]

activation\_7 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_7[0][0]'

]

activation\_10 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_10[0][0]

']

activation\_11 (Activation) (None, 25, 25, 32) 0 ['batch\_normalization\_11[0][0]

']

mixed0 (Concatenate) (None, 25, 25, 256) 0 ['activation\_5[0][0]',

'activation\_7[0][0]',

'activation\_10[0][0]',

'activation\_11[0][0]']

conv2d\_15 (Conv2D) (None, 25, 25, 64) 16384 ['mixed0[0][0]']

batch\_normalization\_15 (Ba (None, 25, 25, 64) 192 ['conv2d\_15[0][0]']

tchNormalization)

activation\_15 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_15[0][0]

']

conv2d\_13 (Conv2D) (None, 25, 25, 48) 12288 ['mixed0[0][0]']

conv2d\_16 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_15[0][0]']

batch\_normalization\_13 (Ba (None, 25, 25, 48) 144 ['conv2d\_13[0][0]']

tchNormalization)

batch\_normalization\_16 (Ba (None, 25, 25, 96) 288 ['conv2d\_16[0][0]']

tchNormalization)

activation\_13 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_13[0][0]

']

activation\_16 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_16[0][0]

']

average\_pooling2d\_1 (Avera (None, 25, 25, 256) 0 ['mixed0[0][0]']

gePooling2D)

conv2d\_12 (Conv2D) (None, 25, 25, 64) 16384 ['mixed0[0][0]']

conv2d\_14 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_13[0][0]']

conv2d\_17 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_16[0][0]']

conv2d\_18 (Conv2D) (None, 25, 25, 64) 16384 ['average\_pooling2d\_1[0][0]']

batch\_normalization\_12 (Ba (None, 25, 25, 64) 192 ['conv2d\_12[0][0]']

tchNormalization)

batch\_normalization\_14 (Ba (None, 25, 25, 64) 192 ['conv2d\_14[0][0]']

tchNormalization)

batch\_normalization\_17 (Ba (None, 25, 25, 96) 288 ['conv2d\_17[0][0]']

tchNormalization)

batch\_normalization\_18 (Ba (None, 25, 25, 64) 192 ['conv2d\_18[0][0]']

tchNormalization)

activation\_12 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_12[0][0]

']

activation\_14 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_14[0][0]

']

activation\_17 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_17[0][0]

']

activation\_18 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_18[0][0]

']

mixed1 (Concatenate) (None, 25, 25, 288) 0 ['activation\_12[0][0]',

'activation\_14[0][0]',

'activation\_17[0][0]',

'activation\_18[0][0]']

conv2d\_22 (Conv2D) (None, 25, 25, 64) 18432 ['mixed1[0][0]']

batch\_normalization\_22 (Ba (None, 25, 25, 64) 192 ['conv2d\_22[0][0]']

tchNormalization)

activation\_22 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_22[0][0]

']

conv2d\_20 (Conv2D) (None, 25, 25, 48) 13824 ['mixed1[0][0]']

conv2d\_23 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_22[0][0]']

batch\_normalization\_20 (Ba (None, 25, 25, 48) 144 ['conv2d\_20[0][0]']

tchNormalization)

batch\_normalization\_23 (Ba (None, 25, 25, 96) 288 ['conv2d\_23[0][0]']

tchNormalization)

activation\_20 (Activation) (None, 25, 25, 48) 0 ['batch\_normalization\_20[0][0]

']

activation\_23 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_23[0][0]

']

average\_pooling2d\_2 (Avera (None, 25, 25, 288) 0 ['mixed1[0][0]']

gePooling2D)

conv2d\_19 (Conv2D) (None, 25, 25, 64) 18432 ['mixed1[0][0]']

conv2d\_21 (Conv2D) (None, 25, 25, 64) 76800 ['activation\_20[0][0]']

conv2d\_24 (Conv2D) (None, 25, 25, 96) 82944 ['activation\_23[0][0]']

conv2d\_25 (Conv2D) (None, 25, 25, 64) 18432 ['average\_pooling2d\_2[0][0]']

batch\_normalization\_19 (Ba (None, 25, 25, 64) 192 ['conv2d\_19[0][0]']

tchNormalization)

batch\_normalization\_21 (Ba (None, 25, 25, 64) 192 ['conv2d\_21[0][0]']

tchNormalization)

batch\_normalization\_24 (Ba (None, 25, 25, 96) 288 ['conv2d\_24[0][0]']

tchNormalization)

batch\_normalization\_25 (Ba (None, 25, 25, 64) 192 ['conv2d\_25[0][0]']

tchNormalization)

activation\_19 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_19[0][0]

']

activation\_21 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_21[0][0]

']

activation\_24 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_24[0][0]

']

activation\_25 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_25[0][0]

']

mixed2 (Concatenate) (None, 25, 25, 288) 0 ['activation\_19[0][0]',

'activation\_21[0][0]',

'activation\_24[0][0]',

'activation\_25[0][0]']

conv2d\_27 (Conv2D) (None, 25, 25, 64) 18432 ['mixed2[0][0]']

batch\_normalization\_27 (Ba (None, 25, 25, 64) 192 ['conv2d\_27[0][0]']

tchNormalization)

activation\_27 (Activation) (None, 25, 25, 64) 0 ['batch\_normalization\_27[0][0]

']

conv2d\_28 (Conv2D) (None, 25, 25, 96) 55296 ['activation\_27[0][0]']

batch\_normalization\_28 (Ba (None, 25, 25, 96) 288 ['conv2d\_28[0][0]']

tchNormalization)

activation\_28 (Activation) (None, 25, 25, 96) 0 ['batch\_normalization\_28[0][0]

']

conv2d\_26 (Conv2D) (None, 12, 12, 384) 995328 ['mixed2[0][0]']

conv2d\_29 (Conv2D) (None, 12, 12, 96) 82944 ['activation\_28[0][0]']

batch\_normalization\_26 (Ba (None, 12, 12, 384) 1152 ['conv2d\_26[0][0]']

tchNormalization)

batch\_normalization\_29 (Ba (None, 12, 12, 96) 288 ['conv2d\_29[0][0]']

tchNormalization)

activation\_26 (Activation) (None, 12, 12, 384) 0 ['batch\_normalization\_26[0][0]

']

activation\_29 (Activation) (None, 12, 12, 96) 0 ['batch\_normalization\_29[0][0]

']

max\_pooling2d\_2 (MaxPoolin (None, 12, 12, 288) 0 ['mixed2[0][0]']

g2D)

mixed3 (Concatenate) (None, 12, 12, 768) 0 ['activation\_26[0][0]',

'activation\_29[0][0]',

'max\_pooling2d\_2[0][0]']

conv2d\_34 (Conv2D) (None, 12, 12, 128) 98304 ['mixed3[0][0]']

batch\_normalization\_34 (Ba (None, 12, 12, 128) 384 ['conv2d\_34[0][0]']

tchNormalization)

activation\_34 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_34[0][0]

']

conv2d\_35 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_34[0][0]']

batch\_normalization\_35 (Ba (None, 12, 12, 128) 384 ['conv2d\_35[0][0]']

tchNormalization)

activation\_35 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_35[0][0]

']

conv2d\_31 (Conv2D) (None, 12, 12, 128) 98304 ['mixed3[0][0]']

conv2d\_36 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_35[0][0]']

batch\_normalization\_31 (Ba (None, 12, 12, 128) 384 ['conv2d\_31[0][0]']

tchNormalization)

batch\_normalization\_36 (Ba (None, 12, 12, 128) 384 ['conv2d\_36[0][0]']

tchNormalization)

activation\_31 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_31[0][0]

']

activation\_36 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_36[0][0]

']

conv2d\_32 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_31[0][0]']

conv2d\_37 (Conv2D) (None, 12, 12, 128) 114688 ['activation\_36[0][0]']

batch\_normalization\_32 (Ba (None, 12, 12, 128) 384 ['conv2d\_32[0][0]']

tchNormalization)

batch\_normalization\_37 (Ba (None, 12, 12, 128) 384 ['conv2d\_37[0][0]']

tchNormalization)

activation\_32 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_32[0][0]

']

activation\_37 (Activation) (None, 12, 12, 128) 0 ['batch\_normalization\_37[0][0]

']

average\_pooling2d\_3 (Avera (None, 12, 12, 768) 0 ['mixed3[0][0]']

gePooling2D)

conv2d\_30 (Conv2D) (None, 12, 12, 192) 147456 ['mixed3[0][0]']

conv2d\_33 (Conv2D) (None, 12, 12, 192) 172032 ['activation\_32[0][0]']

conv2d\_38 (Conv2D) (None, 12, 12, 192) 172032 ['activation\_37[0][0]']

conv2d\_39 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_3[0][0]']

batch\_normalization\_30 (Ba (None, 12, 12, 192) 576 ['conv2d\_30[0][0]']

tchNormalization)

batch\_normalization\_33 (Ba (None, 12, 12, 192) 576 ['conv2d\_33[0][0]']

tchNormalization)

batch\_normalization\_38 (Ba (None, 12, 12, 192) 576 ['conv2d\_38[0][0]']

tchNormalization)

batch\_normalization\_39 (Ba (None, 12, 12, 192) 576 ['conv2d\_39[0][0]']

tchNormalization)

activation\_30 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_30[0][0]

']

activation\_33 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_33[0][0]

']

activation\_38 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_38[0][0]

']

activation\_39 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_39[0][0]

']

mixed4 (Concatenate) (None, 12, 12, 768) 0 ['activation\_30[0][0]',

'activation\_33[0][0]',

'activation\_38[0][0]',

'activation\_39[0][0]']

conv2d\_44 (Conv2D) (None, 12, 12, 160) 122880 ['mixed4[0][0]']

batch\_normalization\_44 (Ba (None, 12, 12, 160) 480 ['conv2d\_44[0][0]']

tchNormalization)

activation\_44 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_44[0][0]

']

conv2d\_45 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_44[0][0]']

batch\_normalization\_45 (Ba (None, 12, 12, 160) 480 ['conv2d\_45[0][0]']

tchNormalization)

activation\_45 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_45[0][0]

']

conv2d\_41 (Conv2D) (None, 12, 12, 160) 122880 ['mixed4[0][0]']

conv2d\_46 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_45[0][0]']

batch\_normalization\_41 (Ba (None, 12, 12, 160) 480 ['conv2d\_41[0][0]']

tchNormalization)

batch\_normalization\_46 (Ba (None, 12, 12, 160) 480 ['conv2d\_46[0][0]']

tchNormalization)

activation\_41 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_41[0][0]

']

activation\_46 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_46[0][0]

']

conv2d\_42 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_41[0][0]']

conv2d\_47 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_46[0][0]']

batch\_normalization\_42 (Ba (None, 12, 12, 160) 480 ['conv2d\_42[0][0]']

tchNormalization)

batch\_normalization\_47 (Ba (None, 12, 12, 160) 480 ['conv2d\_47[0][0]']

tchNormalization)

activation\_42 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_42[0][0]

']

activation\_47 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_47[0][0]

']

average\_pooling2d\_4 (Avera (None, 12, 12, 768) 0 ['mixed4[0][0]']

gePooling2D)

conv2d\_40 (Conv2D) (None, 12, 12, 192) 147456 ['mixed4[0][0]']

conv2d\_43 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_42[0][0]']

conv2d\_48 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_47[0][0]']

conv2d\_49 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_4[0][0]']

batch\_normalization\_40 (Ba (None, 12, 12, 192) 576 ['conv2d\_40[0][0]']

tchNormalization)

batch\_normalization\_43 (Ba (None, 12, 12, 192) 576 ['conv2d\_43[0][0]']

tchNormalization)

batch\_normalization\_48 (Ba (None, 12, 12, 192) 576 ['conv2d\_48[0][0]']

tchNormalization)

batch\_normalization\_49 (Ba (None, 12, 12, 192) 576 ['conv2d\_49[0][0]']

tchNormalization)

activation\_40 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_40[0][0]

']

activation\_43 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_43[0][0]

']

activation\_48 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_48[0][0]

']

activation\_49 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_49[0][0]

']

mixed5 (Concatenate) (None, 12, 12, 768) 0 ['activation\_40[0][0]',

'activation\_43[0][0]',

'activation\_48[0][0]',

'activation\_49[0][0]']

conv2d\_54 (Conv2D) (None, 12, 12, 160) 122880 ['mixed5[0][0]']

batch\_normalization\_54 (Ba (None, 12, 12, 160) 480 ['conv2d\_54[0][0]']

tchNormalization)

activation\_54 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_54[0][0]

']

conv2d\_55 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_54[0][0]']

batch\_normalization\_55 (Ba (None, 12, 12, 160) 480 ['conv2d\_55[0][0]']

tchNormalization)

activation\_55 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_55[0][0]

']

conv2d\_51 (Conv2D) (None, 12, 12, 160) 122880 ['mixed5[0][0]']

conv2d\_56 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_55[0][0]']

batch\_normalization\_51 (Ba (None, 12, 12, 160) 480 ['conv2d\_51[0][0]']

tchNormalization)

batch\_normalization\_56 (Ba (None, 12, 12, 160) 480 ['conv2d\_56[0][0]']

tchNormalization)

activation\_51 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_51[0][0]

']

activation\_56 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_56[0][0]

']

conv2d\_52 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_51[0][0]']

conv2d\_57 (Conv2D) (None, 12, 12, 160) 179200 ['activation\_56[0][0]']

batch\_normalization\_52 (Ba (None, 12, 12, 160) 480 ['conv2d\_52[0][0]']

tchNormalization)

batch\_normalization\_57 (Ba (None, 12, 12, 160) 480 ['conv2d\_57[0][0]']

tchNormalization)

activation\_52 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_52[0][0]

']

activation\_57 (Activation) (None, 12, 12, 160) 0 ['batch\_normalization\_57[0][0]

']

average\_pooling2d\_5 (Avera (None, 12, 12, 768) 0 ['mixed5[0][0]']

gePooling2D)

conv2d\_50 (Conv2D) (None, 12, 12, 192) 147456 ['mixed5[0][0]']

conv2d\_53 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_52[0][0]']

conv2d\_58 (Conv2D) (None, 12, 12, 192) 215040 ['activation\_57[0][0]']

conv2d\_59 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_5[0][0]']

batch\_normalization\_50 (Ba (None, 12, 12, 192) 576 ['conv2d\_50[0][0]']

tchNormalization)

batch\_normalization\_53 (Ba (None, 12, 12, 192) 576 ['conv2d\_53[0][0]']

tchNormalization)

batch\_normalization\_58 (Ba (None, 12, 12, 192) 576 ['conv2d\_58[0][0]']

tchNormalization)

batch\_normalization\_59 (Ba (None, 12, 12, 192) 576 ['conv2d\_59[0][0]']

tchNormalization)

activation\_50 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_50[0][0]

']

activation\_53 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_53[0][0]

']

activation\_58 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_58[0][0]

']

activation\_59 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_59[0][0]

']

mixed6 (Concatenate) (None, 12, 12, 768) 0 ['activation\_50[0][0]',

'activation\_53[0][0]',

'activation\_58[0][0]',

'activation\_59[0][0]']

conv2d\_64 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

batch\_normalization\_64 (Ba (None, 12, 12, 192) 576 ['conv2d\_64[0][0]']

tchNormalization)

activation\_64 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_64[0][0]

']

conv2d\_65 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_64[0][0]']

batch\_normalization\_65 (Ba (None, 12, 12, 192) 576 ['conv2d\_65[0][0]']

tchNormalization)

activation\_65 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_65[0][0]

']

conv2d\_61 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

conv2d\_66 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_65[0][0]']

batch\_normalization\_61 (Ba (None, 12, 12, 192) 576 ['conv2d\_61[0][0]']

tchNormalization)

batch\_normalization\_66 (Ba (None, 12, 12, 192) 576 ['conv2d\_66[0][0]']

tchNormalization)

activation\_61 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_61[0][0]

']

activation\_66 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_66[0][0]

']

conv2d\_62 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_61[0][0]']

conv2d\_67 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_66[0][0]']

batch\_normalization\_62 (Ba (None, 12, 12, 192) 576 ['conv2d\_62[0][0]']

tchNormalization)

batch\_normalization\_67 (Ba (None, 12, 12, 192) 576 ['conv2d\_67[0][0]']

tchNormalization)

activation\_62 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_62[0][0]

']

activation\_67 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_67[0][0]

']

average\_pooling2d\_6 (Avera (None, 12, 12, 768) 0 ['mixed6[0][0]']

gePooling2D)

conv2d\_60 (Conv2D) (None, 12, 12, 192) 147456 ['mixed6[0][0]']

conv2d\_63 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_62[0][0]']

conv2d\_68 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_67[0][0]']

conv2d\_69 (Conv2D) (None, 12, 12, 192) 147456 ['average\_pooling2d\_6[0][0]']

batch\_normalization\_60 (Ba (None, 12, 12, 192) 576 ['conv2d\_60[0][0]']

tchNormalization)

batch\_normalization\_63 (Ba (None, 12, 12, 192) 576 ['conv2d\_63[0][0]']

tchNormalization)

batch\_normalization\_68 (Ba (None, 12, 12, 192) 576 ['conv2d\_68[0][0]']

tchNormalization)

batch\_normalization\_69 (Ba (None, 12, 12, 192) 576 ['conv2d\_69[0][0]']

tchNormalization)

activation\_60 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_60[0][0]

']

activation\_63 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_63[0][0]

']

activation\_68 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_68[0][0]

']

activation\_69 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_69[0][0]

']

mixed7 (Concatenate) (None, 12, 12, 768) 0 ['activation\_60[0][0]',

'activation\_63[0][0]',

'activation\_68[0][0]',

'activation\_69[0][0]']

conv2d\_72 (Conv2D) (None, 12, 12, 192) 147456 ['mixed7[0][0]']

batch\_normalization\_72 (Ba (None, 12, 12, 192) 576 ['conv2d\_72[0][0]']

tchNormalization)

activation\_72 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_72[0][0]

']

conv2d\_73 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_72[0][0]']

batch\_normalization\_73 (Ba (None, 12, 12, 192) 576 ['conv2d\_73[0][0]']

tchNormalization)

activation\_73 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_73[0][0]

']

conv2d\_70 (Conv2D) (None, 12, 12, 192) 147456 ['mixed7[0][0]']

conv2d\_74 (Conv2D) (None, 12, 12, 192) 258048 ['activation\_73[0][0]']

batch\_normalization\_70 (Ba (None, 12, 12, 192) 576 ['conv2d\_70[0][0]']

tchNormalization)

batch\_normalization\_74 (Ba (None, 12, 12, 192) 576 ['conv2d\_74[0][0]']

tchNormalization)

activation\_70 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_70[0][0]

']

activation\_74 (Activation) (None, 12, 12, 192) 0 ['batch\_normalization\_74[0][0]

']

conv2d\_71 (Conv2D) (None, 5, 5, 320) 552960 ['activation\_70[0][0]']

conv2d\_75 (Conv2D) (None, 5, 5, 192) 331776 ['activation\_74[0][0]']

batch\_normalization\_71 (Ba (None, 5, 5, 320) 960 ['conv2d\_71[0][0]']

tchNormalization)

batch\_normalization\_75 (Ba (None, 5, 5, 192) 576 ['conv2d\_75[0][0]']

tchNormalization)

activation\_71 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_71[0][0]

']

activation\_75 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_75[0][0]

']

max\_pooling2d\_3 (MaxPoolin (None, 5, 5, 768) 0 ['mixed7[0][0]']

g2D)

mixed8 (Concatenate) (None, 5, 5, 1280) 0 ['activation\_71[0][0]',

'activation\_75[0][0]',

'max\_pooling2d\_3[0][0]']

conv2d\_80 (Conv2D) (None, 5, 5, 448) 573440 ['mixed8[0][0]']

batch\_normalization\_80 (Ba (None, 5, 5, 448) 1344 ['conv2d\_80[0][0]']

tchNormalization)

activation\_80 (Activation) (None, 5, 5, 448) 0 ['batch\_normalization\_80[0][0]

']

conv2d\_77 (Conv2D) (None, 5, 5, 384) 491520 ['mixed8[0][0]']

conv2d\_81 (Conv2D) (None, 5, 5, 384) 1548288 ['activation\_80[0][0]']

batch\_normalization\_77 (Ba (None, 5, 5, 384) 1152 ['conv2d\_77[0][0]']

tchNormalization)

batch\_normalization\_81 (Ba (None, 5, 5, 384) 1152 ['conv2d\_81[0][0]']

tchNormalization)

activation\_77 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_77[0][0]

']

activation\_81 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_81[0][0]

']

conv2d\_78 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_77[0][0]']

conv2d\_79 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_77[0][0]']

conv2d\_82 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_81[0][0]']

conv2d\_83 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_81[0][0]']

average\_pooling2d\_7 (Avera (None, 5, 5, 1280) 0 ['mixed8[0][0]']

gePooling2D)

conv2d\_76 (Conv2D) (None, 5, 5, 320) 409600 ['mixed8[0][0]']

batch\_normalization\_78 (Ba (None, 5, 5, 384) 1152 ['conv2d\_78[0][0]']

tchNormalization)

batch\_normalization\_79 (Ba (None, 5, 5, 384) 1152 ['conv2d\_79[0][0]']

tchNormalization)

batch\_normalization\_82 (Ba (None, 5, 5, 384) 1152 ['conv2d\_82[0][0]']

tchNormalization)

batch\_normalization\_83 (Ba (None, 5, 5, 384) 1152 ['conv2d\_83[0][0]']

tchNormalization)

conv2d\_84 (Conv2D) (None, 5, 5, 192) 245760 ['average\_pooling2d\_7[0][0]']

batch\_normalization\_76 (Ba (None, 5, 5, 320) 960 ['conv2d\_76[0][0]']

tchNormalization)

activation\_78 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_78[0][0]

']

activation\_79 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_79[0][0]

']

activation\_82 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_82[0][0]

']

activation\_83 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_83[0][0]

']

batch\_normalization\_84 (Ba (None, 5, 5, 192) 576 ['conv2d\_84[0][0]']

tchNormalization)

activation\_76 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_76[0][0]

']

mixed9\_0 (Concatenate) (None, 5, 5, 768) 0 ['activation\_78[0][0]',

'activation\_79[0][0]']

concatenate (Concatenate) (None, 5, 5, 768) 0 ['activation\_82[0][0]',

'activation\_83[0][0]']

activation\_84 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_84[0][0]

']

mixed9 (Concatenate) (None, 5, 5, 2048) 0 ['activation\_76[0][0]',

'mixed9\_0[0][0]',

'concatenate[0][0]',

'activation\_84[0][0]']

conv2d\_89 (Conv2D) (None, 5, 5, 448) 917504 ['mixed9[0][0]']

batch\_normalization\_89 (Ba (None, 5, 5, 448) 1344 ['conv2d\_89[0][0]']

tchNormalization)

activation\_89 (Activation) (None, 5, 5, 448) 0 ['batch\_normalization\_89[0][0]

']

conv2d\_86 (Conv2D) (None, 5, 5, 384) 786432 ['mixed9[0][0]']

conv2d\_90 (Conv2D) (None, 5, 5, 384) 1548288 ['activation\_89[0][0]']

batch\_normalization\_86 (Ba (None, 5, 5, 384) 1152 ['conv2d\_86[0][0]']

tchNormalization)

batch\_normalization\_90 (Ba (None, 5, 5, 384) 1152 ['conv2d\_90[0][0]']

tchNormalization)

activation\_86 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_86[0][0]

']

activation\_90 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_90[0][0]

']

conv2d\_87 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_86[0][0]']

conv2d\_88 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_86[0][0]']

conv2d\_91 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_90[0][0]']

conv2d\_92 (Conv2D) (None, 5, 5, 384) 442368 ['activation\_90[0][0]']

average\_pooling2d\_8 (Avera (None, 5, 5, 2048) 0 ['mixed9[0][0]']

gePooling2D)

conv2d\_85 (Conv2D) (None, 5, 5, 320) 655360 ['mixed9[0][0]']

batch\_normalization\_87 (Ba (None, 5, 5, 384) 1152 ['conv2d\_87[0][0]']

tchNormalization)

batch\_normalization\_88 (Ba (None, 5, 5, 384) 1152 ['conv2d\_88[0][0]']

tchNormalization)

batch\_normalization\_91 (Ba (None, 5, 5, 384) 1152 ['conv2d\_91[0][0]']

tchNormalization)

batch\_normalization\_92 (Ba (None, 5, 5, 384) 1152 ['conv2d\_92[0][0]']

tchNormalization)

conv2d\_93 (Conv2D) (None, 5, 5, 192) 393216 ['average\_pooling2d\_8[0][0]']

batch\_normalization\_85 (Ba (None, 5, 5, 320) 960 ['conv2d\_85[0][0]']

tchNormalization)

activation\_87 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_87[0][0]

']

activation\_88 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_88[0][0]

']

activation\_91 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_91[0][0]

']

activation\_92 (Activation) (None, 5, 5, 384) 0 ['batch\_normalization\_92[0][0]

']

batch\_normalization\_93 (Ba (None, 5, 5, 192) 576 ['conv2d\_93[0][0]']

tchNormalization)

activation\_85 (Activation) (None, 5, 5, 320) 0 ['batch\_normalization\_85[0][0]

']

mixed9\_1 (Concatenate) (None, 5, 5, 768) 0 ['activation\_87[0][0]',

'activation\_88[0][0]']

concatenate\_1 (Concatenate (None, 5, 5, 768) 0 ['activation\_91[0][0]',

) 'activation\_92[0][0]']

activation\_93 (Activation) (None, 5, 5, 192) 0 ['batch\_normalization\_93[0][0]

']

mixed10 (Concatenate) (None, 5, 5, 2048) 0 ['activation\_85[0][0]',

'mixed9\_1[0][0]',

'concatenate\_1[0][0]',

'activation\_93[0][0]']

global\_average\_pooling2d ( (None, 2048) 0 ['mixed10[0][0]']

GlobalAveragePooling2D)

dropout (Dropout) (None, 2048) 0 ['global\_average\_pooling2d[0][

0]']

dense (Dense) (None, 1024) 2098176 ['dropout[0][0]']

dense\_1 (Dense) (None, 4) 4100 ['dense[0][0]']

==================================================================================================

Total params: 23905060 (91.19 MB)

Trainable params: 2102276 (8.02 MB)

Non-trainable params: 21802784 (83.17 MB)

# Define a path for saving the model

checkpoint = ModelCheckpoint('/content/drive/MyDrive/Colab Notebooks/Datagen2/Inceptionv3.h5', monitor='val\_accuracy', save\_best\_only=True)

# Model traing and the validation

history = model.fit(

    training\_set,

    epochs=25,

    validation\_data=test\_set,

    callbacks=[checkpoint]

)

Epoch 1/25

357/357 [==============================] - ETA: 0s - loss: 0.8394 - accuracy: 0.6965

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

357/357 [==============================] - 1446s 4s/step - loss: 0.8394 - accuracy: 0.6965 - val\_loss: 0.5384 - val\_accuracy: 0.7956

Epoch 2/25

357/357 [==============================] - 226s 632ms/step - loss: 0.6032 - accuracy: 0.7675 - val\_loss: 0.5103 - val\_accuracy: 0.7979

Epoch 3/25

357/357 [==============================] - 225s 631ms/step - loss: 0.5591 - accuracy: 0.7868 - val\_loss: 0.4784 - val\_accuracy: 0.8108

Epoch 4/25

357/357 [==============================] - 226s 634ms/step - loss: 0.5455 - accuracy: 0.7865 - val\_loss: 0.4864 - val\_accuracy: 0.8169

Epoch 5/25

357/357 [==============================] - 226s 632ms/step - loss: 0.5263 - accuracy: 0.7979 - val\_loss: 0.4462 - val\_accuracy: 0.8322

Epoch 6/25

357/357 [==============================] - 224s 628ms/step - loss: 0.5009 - accuracy: 0.8096 - val\_loss: 0.4631 - val\_accuracy: 0.8314

Epoch 7/25

357/357 [==============================] - 226s 632ms/step - loss: 0.5037 - accuracy: 0.8016 - val\_loss: 0.4407 - val\_accuracy: 0.8337

Epoch 8/25

357/357 [==============================] - 225s 628ms/step - loss: 0.4931 - accuracy: 0.8054 - val\_loss: 0.4537 - val\_accuracy: 0.8154

Epoch 9/25

357/357 [==============================] - 224s 626ms/step - loss: 0.4715 - accuracy: 0.8239 - val\_loss: 0.4655 - val\_accuracy: 0.8108

Epoch 10/25

357/357 [==============================] - 224s 627ms/step - loss: 0.4763 - accuracy: 0.8156 - val\_loss: 0.4290 - val\_accuracy: 0.8230

Epoch 11/25

357/357 [==============================] - 224s 628ms/step - loss: 0.4646 - accuracy: 0.8253 - val\_loss: 0.4329 - val\_accuracy: 0.8192

Epoch 12/25

357/357 [==============================] - 225s 630ms/step - loss: 0.4531 - accuracy: 0.8254 - val\_loss: 0.3923 - val\_accuracy: 0.8406

Epoch 13/25

357/357 [==============================] - 224s 627ms/step - loss: 0.4423 - accuracy: 0.8284 - val\_loss: 0.4522 - val\_accuracy: 0.8101

Epoch 14/25

357/357 [==============================] - 225s 630ms/step - loss: 0.4477 - accuracy: 0.8235 - val\_loss: 0.3906 - val\_accuracy: 0.8474

Epoch 15/25

357/357 [==============================] - 229s 640ms/step - loss: 0.4267 - accuracy: 0.8347 - val\_loss: 0.3713 - val\_accuracy: 0.8551

Epoch 16/25

357/357 [==============================] - 229s 642ms/step - loss: 0.4294 - accuracy: 0.8375 - val\_loss: 0.3770 - val\_accuracy: 0.8421

Epoch 17/25

357/357 [==============================] - 227s 636ms/step - loss: 0.4239 - accuracy: 0.8354 - val\_loss: 0.3857 - val\_accuracy: 0.8467

Epoch 18/25

357/357 [==============================] - 225s 630ms/step - loss: 0.4266 - accuracy: 0.8337 - val\_loss: 0.3749 - val\_accuracy: 0.8520

Epoch 19/25

357/357 [==============================] - 223s 625ms/step - loss: 0.4134 - accuracy: 0.8421 - val\_loss: 0.3768 - val\_accuracy: 0.8459

Epoch 20/25

357/357 [==============================] - 223s 626ms/step - loss: 0.4256 - accuracy: 0.8321 - val\_loss: 0.3628 - val\_accuracy: 0.8490

Epoch 21/25

357/357 [==============================] - 226s 632ms/step - loss: 0.4011 - accuracy: 0.8474 - val\_loss: 0.3637 - val\_accuracy: 0.8574

Epoch 22/25

357/357 [==============================] - 224s 626ms/step - loss: 0.4038 - accuracy: 0.8430 - val\_loss: 0.3595 - val\_accuracy: 0.8566

Epoch 23/25

357/357 [==============================] - 224s 626ms/step - loss: 0.4025 - accuracy: 0.8470 - val\_loss: 0.4301 - val\_accuracy: 0.8268

Epoch 24/25

357/357 [==============================] - 226s 632ms/step - loss: 0.4052 - accuracy: 0.8484 - val\_loss: 0.3279 - val\_accuracy: 0.8642

Epoch 25/25

357/357 [==============================] - 223s 625ms/step - loss: 0.4021 - accuracy: 0.8488 - val\_loss: 0.3490 - val\_accuracy: 0.8535

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Import libraries

import os

from tensorflow.keras.models import load\_model

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score,f1\_score

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

image\_size = 224

# Initialze empty cell to store the test images anf their corresponding labels

x\_test = []

y\_test = []

# To analyze the photos in the testing dataset, iterate over each class label.

for label in labels:

    testingpath = os.path.join('/content/drive/MyDrive/Colab Notebooks/Testing', label)

      # Build a path to the test images for the given label.

    for file in os.listdir(testingpath):

        image = cv2.imread(os.path.join(testingpath, file), cv2.IMREAD\_COLOR)

        image = cv2.resize(image, (image\_size, image\_size))

        x\_test.append(image)

        y\_test.append(labels.index(label))

# Normalize the pixel values and convert the list of images to a numpy array.

x\_test = np.array(x\_test) / 255.0

y\_test = np.array(y\_test)

y\_test\_categorical = tf.keras.utils.to\_categorical(y\_test, num\_classes=len(labels))

# Give specific locations for the saved model weights files for different architectures.

vgg\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen2/VGG16.h5'

inception\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen2/Inceptionv3.h5'

resnet\_model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen2/Resnet101.h5'

# Load the pre-trained models from the specified file paths

vgg\_model = tf.keras.models.load\_model(vgg\_model\_path)

inception\_model = tf.keras.models.load\_model(inception\_model\_path)

resnet\_model = tf.keras.models.load\_model(resnet\_model\_path)

# Function to evaluate model

def evaluate\_model(model, x\_test, y\_test, y\_test\_categorical, labels):

    predictions = model.predict(x\_test)

    y\_pred = np.argmax(predictions, axis=1)

    y\_true = np.argmax(y\_test\_categorical, axis=1)

    # Classification report

    print("Classification Report:")

    print(classification\_report(y\_true, y\_pred, target\_names=labels))

    # Confusion matrix

    cm = confusion\_matrix(y\_true, y\_pred)

    plt.figure(figsize=(10, 7))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

    plt.xlabel('Predicted')

    plt.ylabel('Actual')

    plt.title('Confusion Matrix')

    plt.show()

# Evaluate model

print("VGG Model Evaluation:")

evaluate\_model(vgg\_model, x\_test, y\_test, y\_test\_categorical, labels)

VGG Model Evaluation:

41/41 [==============================] - 146s 4s/step

Classification Report:

precision recall f1-score support

glioma 0.58 0.93 0.71 300

meningioma 0.55 0.13 0.21 306

notumor 0.82 0.90 0.86 405

pituitary 0.78 0.81 0.80 300

accuracy 0.71 1311

macro avg 0.68 0.69 0.64 1311

weighted avg 0.69 0.71 0.66 1311

evaluate\_model(inception\_model, x\_test, y\_test, y\_test\_categorical, labels)

41/41 [==============================] - 31s 736ms/step

Classification Report:

precision recall f1-score support

glioma 0.85 0.83 0.84 300

meningioma 0.77 0.68 0.72 306

notumor 0.95 0.96 0.95 405

pituitary 0.86 0.96 0.91 300

accuracy 0.86 1311

macro avg 0.85 0.86 0.85 1311

weighted avg 0.86 0.86 0.86 1311

evaluate\_model(resnet\_model, x\_test, y\_test, y\_test\_categorical, labels)

41/41 [==============================] - 93s 2s/step

Classification Report:

precision recall f1-score support

glioma 0.57 0.54 0.55 300

meningioma 0.50 0.41 0.45 306

notumor 0.77 0.75 0.76 405

pituitary 0.64 0.80 0.71 300

accuracy 0.64 1311

macro avg 0.62 0.63 0.62 1311

weighted avg 0.63 0.64 0.63 1311

import numpy as np

import matplotlib.pyplot as plt

import cv2

import os

import tensorflow as tf

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

image\_size = 224

image\_path = '/content/drive/MyDrive/Colab Notebooks/Testing/meningioma/Te-meTr\_0001.jpg'

check\_image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

check\_image = cv2.resize(check\_image, (image\_size, image\_size))

check\_image = np.expand\_dims(check\_image, axis=0) / 255.0

model\_path = '/content/drive/MyDrive/Colab Notebooks/Datagen2/Inceptionv3.h5'

model = tf.keras.models.load\_model(model\_path)

prediction = model.predict(check\_image)

predicted\_index = np.argmax(prediction[0])

predicted\_class = labels[predicted\_index]

confidence = prediction[0][predicted\_index]

print('Result: ', predicted\_class)

1/1 [==============================] - 1s 1s/step

Result: meningioma