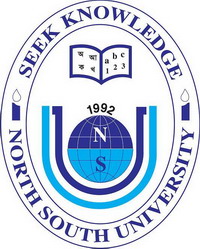
**Department of Electrical and Computer Engineering**

**North South University**

****

**Senior Design Project**

**Multiclass Classification Using Convolutional Neural Network**

**Submitted By**

**Md. Rifat Hossain**

**Id: 1621373042**

**Submitted To**

**Dr. Mahdy Rahman Chowdhury**

**Associate Professor, ECE**

**DECLARATION**

This is to certify that this Project is my original work. No part of this work has been submitted elsewhere partially or fully for the award of any other degree or diploma. Any material reproduced in this project has been properly acknowledged.

**Student’s name & Signature**

**Md. Rifat Hossain**

**APPROVAL**

The project entitled “**Multiclass Classification on pannuke dataset using Convolution Neural Network**” by Md. Rifat Hossain (ID#121373042), is approved in partial fulfillment of the requirement of the Degree of Bachelor of Science in Computer Science and Engineering on May 2022 and has been accepted as satisfactory.

|  |
| --- |
| **Supervisor’s Signature** |
| **Dr. Mahdy Rahman Chowdhury Associate Professor**  Department of Electrical and Computer Engineering North South University  Dhaka, Bangladesh. |
| **Department Chair’s Signature** |
| **Dr. Mohammad Rezaul Bari**  **Associate Professor**  Department of Electrical and Computer Engineering North South University  Dhaka, Bangladesh. |

**ACKNOWLEDGMENT**

First of all, I wish to express our gratitude to the Almighty for giving me the strength to perform our responsibilities and complete the report.

The capstone project program is very helpful to bridge the gap between theoretical knowledge and real-life experience as part of the Bachelor of Science (BSc) program. This report has been designed to have a practical experience through theoretical understanding.

I also acknowledge my profound sense of gratitude to all the teachers who have been instrumental in providing us with the technical knowledge and moral support to complete the project with full understanding.

It is imperative to show our appreciation for our honorable faculty member **Dr. Mahdy Rahman Chowdhury** for his undivided attention and help to achieve this milestone. Also, our gratefulness is divine to the ***North South University***, ***ECE*** department for providing us with a course such as **CSE499** in which I could really work on this project and materialize it the way I have dreamt of.

I thank and express gratitude my course instructor **Deponker Sarker Depto** who helped me in my project.

**ABSTRACT**

Deep Learning proposed by Hinton is a new learning algorithm of multi-layer neural network and is a type of machine learning. Deep Learning helps researchers analyze medical data to treat disease. It also helps in diagnosis of disease in early stages like cancer, Alzheimers disease and so on.

“Multiclass Classification using Convolutional Neural Network (CNN)” is a deep-learning-based classification process by which I trained a machine using cell images and classified cancer cells from it. ResNet50 is best for this classification and I used the model which helped the machine to learn from the images in order to provide better accuracy. In this research I have tried to implement ResNet50, VGG16, VGG19, DenseNet models on “Pannuke dataset” and tried to achieve better accuracy. I used different techniques like data preprocessing, data augmentation, and feature normalization to achieve better results. Data preprocessing is used for proper feature selection and feature extraction. Selecting important portions and removing   irrelevant portions of input features is the main aim of data preprocessing. Data Augmentation is used for all the original images that were transformed and augmented every epoch and then used for training to avoid overfitting. This allowed the model to be more robust and accurate, as it was trained on different variations of the same image. This method was preferred over random oversampling, which was one way of dealing with class imbalances. Random oversampling consisted of re-sampling less frequent samples to adjust their amount in comparison with predominant samples. However, the distribution of classes would change significantly, where the smaller classes would be much less variable, and the larger classes would have richer variations. Thus, I tried to find out how my model works on publicly available three datasets    Fold1, Fold2 and Fold2 and achieve better results in our project. I tried to classify these datasets and tried to overcome the obstacles I found during this process. And at the end of it I did a comparison of these models.

|  |  |  |  |
| --- | --- | --- | --- |
| Serial Number | Chapter | Content | Page Number |
| 1. | Chapter I | Overview |  |
| 2. |  | Introduction | 1-2 |
| 3. | CNN | 2-3 |
| 4. | Cancer Classification | 3-4 |
| 5. | Motivation | 4-5 |
| 6. | Chapter II | Dataset | 6-7 |
| 7. | Chapter III | Literature Review | 9 |
| 8. |  | Skin cancer Image Classification | 9 |
| 9. | Breast Cancer Classification | 9-10 |
| 10. | Brain Tumor Image Classification | 10-11 |
| 11. | Lung Cancer Image Classification | 11-12 |
| 12. | Chapter IV | System Requirements |  |
| 13. |  | Software Requirement | 13-14 |
| 14. | Hardware Requirement | 15-16 |
| 15. | Chapter V | Library Details | 17-26 |
| 16. | Chapter VI | Methodology |  |
| 17. |  | Workflow | 27 |
| 18. | Implemented models | 28-36 |
| 19. | Data processing | 36-37 |
| 20. | One Hot Encoding | 37-38 |
| 21. | Training-validation-testing Split | 38 |
| 22. | Data Augmentation | 39 |
|  | Optimizers | 39-41 |
|  | Loss function | 42 |

Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| Serial Number | Chapter | Content | Page Number |
| 23. |  | Activation Function | 43-45 |
| 24. | Hyper Parameters | 45 |
| 25. | The initial learning rate | 45-47 |
| 26. | Training and evaluating model criterion | 47-51 |
|  | Ensemble learning | 51-55 |
| 27. | Chapter VII | Problem | 56 |
| 28. | Chapter VIII | Results | 57-62 |
| 29. | Chapter IX | Discussion | 63-64 |
| 30. | Chapter X | Conclusion | 65 |
| 31. | Chapter XI | Reference | 65-67 |



CHAPTER 1: Overview

In this chapter, I am going to discuss why cancer classification is needed for the betterment of society. I am also going to discuss my project aim, objectives and motivation for doing this project.

## 1.1 INTRODUCTION

Deep learning is a sub field of machine learning. DL (Deep Learning) provides excellent opportunities in image processing, image classification, segmentation and detection. There are many Convolutional Neural Networks (CNN) which can be used for biomedical sector. A deep learning-based approach provides better results for classifying different cancer cells, so I used many different models in this purpose. Many researches are going on in this sector to classify different cancer cells but which architecture is more effective for classification is not proven yet. It has both encoder and decoder parts in the architecture. The encoder part consists of several convolutional and max pooling layers. This part mainly identifies the context of the image by labeling each pixel. The decoder part is a fully connected convolutional network. It does not learn any pixel of context of the image, rather it localizes the important portion of the image. It just classifies the important pixels and highlight it to do classification.

In the field of medical science, cancer examination is an important task for the physicians. Because of its high social impact, many approaches have been done in order to help diagnosis. In the past two decades many computer-based algorithms had been proposed for specific cancer types but many of them failed to achieve the state of art. It happened because of the limitation of the classification process. The handcrafted classification process is expensive as well as time consuming. But the development of Convolutional Neural Network (CNN) provides a brilliant way of feature learning. A CNN is a class of deep neural networks that use convolution in place of general matrix multiplication in at least one of their layers. It excels in analyzing visual imagery as they are fully connected (FC) feed forward neural networks that reduce the number of parameters very efficiently without losing out on the quality of models.

## 1.2 CNN

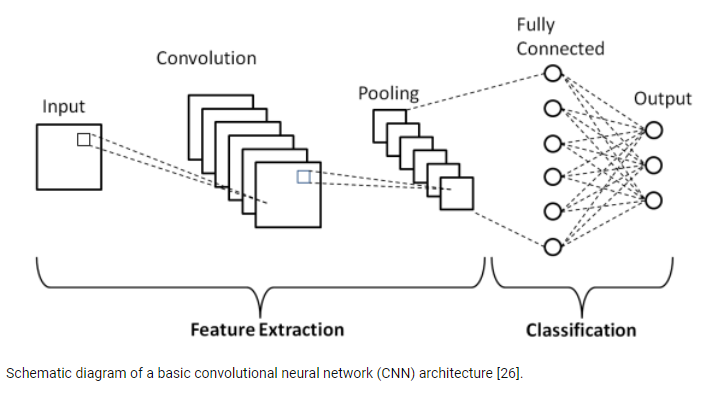
Remarkable progress has been made in image recognition, primarily due to the availability of large-scale annotated datasets and deep convolutional neural networks (CNNs). CNNs enable learning data-driven, highly representative, hierarchical image features from sufficient training data. A CNN is a class of deep neural networks that comprise of FC feed forward neural networks and utilize convolution in place of general matrix multiplication in at least one of their layers. This allows the network to reduce the number of parameters very efficiently without losing out on the quality of models, thriving in analyzing visual imagery. The network keeps on learning new higher dimensionality and more complex features with every layer. It comprises of several kinds of layers:

1) Convolutional Layer: Applies a filter that scans the whole image a couple of pixels at a time, producing a feature map for class probabilities predictions of each feature.

2) Pooling Layer: A layer that scales down the information produced from the convolution layer yet still maintains the most essential information. Different types of pooling include max pooling and average pooling.

3) FC Input Layer: Flattens previous layer’s outputs into one vector for use in the next FC layers.

4) FC Output Layer Determines the image class by generating the final probabilities.



## 1.3 CANCER CLASSIFICATION

Cancer is a disease in which some of the body's cells grow uncontrollably and spread to other parts of the body. Cancer can start almost anywhere in the human body, which is made up of trillions of cells. Normally, human cells grow and multiply (through a process called cell division) to form new cells as the body needs them. When cells grow old or become damaged, they die, and new cells take their place.

Sometimes this orderly process breaks down, and abnormal or damaged cells grow and multiply when they shouldn’t. These cells may form tumors, which are lumps of tissue. Tumors can be cancerous or not cancerous (benign).

Cancerous tumors spread into, or invade, nearby tissues and can travel to distant places in the body to form new tumors (a process called metastasis). Cancerous tumors may also be called malignant tumors. Many cancers form solid tumors, but cancers of the blood, such as leukemia, generally do not.

Benign tumors do not spread into, or invade, nearby tissues. When removed, benign tumors usually don’t grow back, whereas cancerous tumors sometimes do. Benign tumors can sometimes be quite large, however. Some can cause serious symptoms or be life threatening, such as benign tumors in the brain.

Cancer classification is a multiclass image classification task, with 19 different classes. By primary site of origin, cancers may be of specific types like breast cancer, lung cancer, prostate cancer, liver cancer, renal cell carcinoma (kidney cancer), oral cancer, brain cancer etc.

As cancer can be specified by the origin of cancer cells, identifying the position of the cancer cell is important. From the image of the cell it’s going to be identified, which type of cancer it is. From different types of cell images, it is going to be classified, which type of cancer it is.

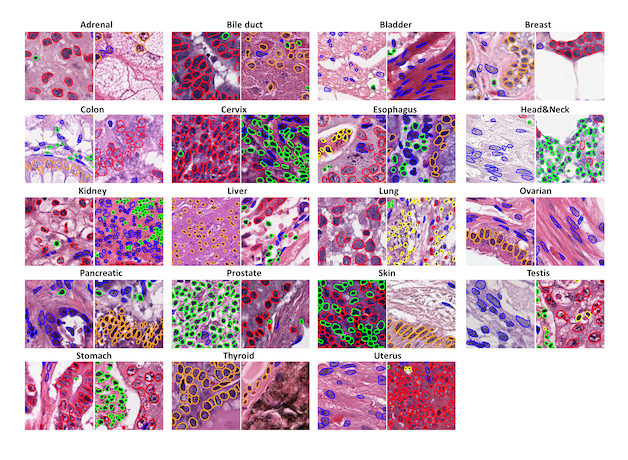
## 1.4 MOTIVATION

## I have chosen the project considering the huge social impact. Cancer is common in developed countries. People around the world are suffering from different types of Cancer like Lung, Pancreases, Breast, Skin etc. These diseases are severe and can cause permanent organ failure, even death as well if not treated. Most of them concentrate on the cancer cell image classification, because proper characterization of the cancer cells plays significant role in different cancer diagnostics. So, I wanted to do this project to help the biomedical sector. Physicians can use the classified images of cancer cells and treat their patients at an early stage. The better result of our project brings a better outcome to the society. Multi-class cancer classification using CNN will bring great change in the biomedical sector. High accuracy and performance of this project can replace the traditional way of taking a fundus image of cancer cells. This project can show the way of cancer cell image classification better than a skillful physician. Our aim of achieving higher accuracy rate and proper classification of cancer cells will help us to fulfill our goal. So, our main motivation behind this project lies in helping physicians as well as patients so that we can help people of every stage of the society. The need to help people who are suffering from cancer related diseases motivated us to choose this project.

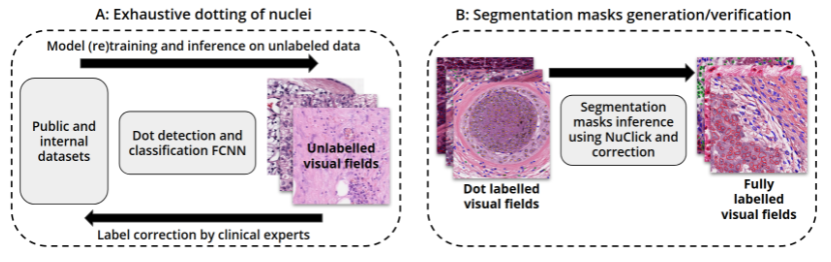


CHAPTER 2: Datasets

Dataset is an important part of deep learning procedure. In machine learning datasets are collections of numbers and values, which is needed to train the model. These numbers and values are essential for the algorithms of the model. Algorithms of the models use this collective data and train the model using it, so that the model can predict the result on another data which is used in the test period. But, in deep learning we usually use pictures or images to train the model. The dataset **contains** **three** different folders of the same type of data.



The dataset I am using is the “Pannuke-dataset”. It is a semi automatically generated nuclei instance segmentation and classification dataset with exhaustive nuclei labels across 19 different tissue types. The dataset consists of 481 visual fields, of which 312 are randomly sampled from more than 20K whole slide images at different magnifications, from multiple data sources. In total the dataset contains 205,343 labeled nuclei, each with an instance segmentation mask.



Models trained on pannuke can aid in whole slide image tissue type segmentation, and generalise to new tissues. PanNuke demonstrates one of the first successfully semi-automatically generated datasets.In short the dataset is:

* Pannuke dataset
* Data in 3 folds
* In each fold, there are images, types, masks
* For classification, we only need images and types.
* Here, we have 19 types of cancer images
* Almost 7,000 data in total of 3 folds

Link of the dataset: [PanNuke Dataset (jgamper.github.io)](https://jgamper.github.io/PanNukeDataset/)



CHAPTER 3: Literature Review

We searched the Google Scholar, IEEE Xplore and Web of Science databases for systematic reviews and original research articles published in English. Only papers that reported sufficient scientific proceedings are included in this review. There are many works on specific cancer classification like breast cancer, skin cancer, brain but there is no paper on combined cancer classification.

Image Processing techniques are extensively being used for the purpose. However due to limitations of Image Processing where certain parameters have to be sent manually, this approach for automated detection turns out practically to be inefficient. The need is to move towards an efficient and robust automated skin cancer classification system, which can provide highly accurate and speedy predictions. Thus, it would be apt to train the machine to identify the cancer areas without using complex algorithms, which in turn increases the accuracy and efficiency of the system/algorithms.

**3.1 Skin Cancer Image Classification**

Here are some of the projects’ literature review: used the MobileNet model from the HAM10000 dataset, and received an overall accuracy of 83.1%. Achieved a 92.9 percent accuracy with fast.ai and using ResNet34. This is 2.9% lower than the ISIC 2018 challenge obtained with a considerably more complex approach, employing a machine learning algorithm on top 18 different neural networks. With another project the accuracy was found to be 90.5% with ResNet model and 78% with VGG16 model, while with other models the accuracies were way lower around 65%. A modified VGGNet was also utilized, where the classification of melanoma versus nevi or lentigines was addressed using dermatoscopic images. The authors compared the classification accuracy of

A CNN trained from scratch,a nd a pre-trained CNN with transfer learning and finetuning of the weighting parameters.

All three configurations were tested with 379 images from the ISBI 2016 Challenge dataset, and the last-mentioned configuration achieved the highest accuracy of 81.33%. Mobile Net v2: 81.9%, Inception v3: 84.3%, ResNet50 v2: 81.3%, ResNet 152V2: 83.9% The top accuracy achieved on dilated versions of VGG16, VGG19, MobileNet, and InceptionV3 is 87.42%, 85.02%, 88.22%, and 89.81% respectively. 16 Dilated Inception V3 achieved the highest classification accuracy. Mobile Net also has high classification accuracy while having the lightest computational complexities.

**3.2 Breast Cancer Classification**

Now we are going to discuss literature review of breast cancer image classification. It’s about CNN classifiers based on salient features of mammography and bit-plane to improve recognition accuracy of breast cancer images. The classifier received overall accuracy of 64% - 92%. Some papers also extracted 25 feature subgraphs from the original image. Different feature subgraphs can provide the different characteristics of the original image features. Combining two salient features to improve recognition accuracy of breast cancer image. [29]. At the same time, some papers have fully tested the integration features of different bit-plane images. [29]

In a proposed breast cancer histopathology image classification by assembling multiple compact Convolutional Neural Networks (CNNs). [30]. First, a hybrid CNN architecture is designed, which contains a global model branch and a local model branch. By local voting and two-branch information merging, a hybrid model obtains stronger representation ability. Second, by embedding the proposed Squeeze-Excitation-Pruning (SEP) block into the hybrid model, the channel importance can be learned and the redundant channels are thus removed. The proposed channel pruning scheme can decrease the risk of overfitting and produce higher accuracy with the same model size. Finally, with different data partitions and composition, we build multiple models and assemble them together to further enhance the model generalization ability. [30]

**3.3 Brain Tumor Image Classification**

Brain tumor diagnosis and classification still rely on histopathological analysis of biopsy specimens today. The current method is invasive, time-consuming and prone to manual errors. [31]. These disadvantages show how essential it is to perform a fully automated method for multi-classification of brain tumors based on deep learning. This paper aims to make multi-classification of brain tumors for the early diagnosis purposes using convolutional neural networks (CNN). Three different CNN models are proposed for three different classification tasks. [31]. Brain tumor detection is achieved with 99.33% accuracy using the first CNN model. The second CNN model can classify the brain tumor into five brain tumor types as normal, glioma, meningioma, pituitary and metastatic with an accuracy of 92.66%. The third CNN model can classify the brain tumors into three grades as Grade II, Grade III and Grade IV with an accuracy of 98.14%. All the important hyper-parameters of CNN models are automatically designated using the grid search optimization algorithm. To the best of the author's knowledge, this is the first study for multi-classification of brain tumor MRI images using CNN whose almost all hyper-parameters are tuned by the grid search optimizer. [31]. The proposed CNN models are compared with other popular state-of-the-art CNN models such as AlexNet, Inceptionv3, ResNet-50, VGG-16 and GoogleNet. [31]. Satisfactory classification results are obtained using large and publicly available clinical datasets. The proposed CNN models can be employed to assist physicians and radiologists in validating their initial screening for brain tumor multi-classification purposes. [31]

**3.4 Lung Cancer Image Classification**

Here’s some other information about lung cancer prediction using deep learning. All models based on CNN-derived features were able to perform binary classification of tumor histology (ADC vs SCC). The 4096-D feature vector seemed to correlate with marginally better predictive performance with most machine learning classifiers. The kNN model had the highest performance (AUC = 0.71, p = 0.017). [32]. This was on par with or better than the CNN (AUC = 0.71, p = 0.018). Other classifiers also showed significant predictive power, with an AUC of 0.68 (p = 0.042) for SVC with linear kernel (c = 0.1), AUC of 0.64 (p = 0.107) for non-linear SVC classifiers. RF had the lowest predictive performance in all instances (AUC = 0.57, p = 0.423), although this improved to an AUC of 0.61 (p = 0.197) with the 512-D feature vector. All models had higher specificity than sensitivity, while accuracy was again highest with the kNN model. The VGG-16 based model achieved significant predictive performance differentiating between ADC and SCC on a held-out test set of 51 patients with AUC of 0.71 (p = 0.018). [32]. Similar fine-tuning and model evaluation was performed with another widely adopted ImageNet architecture, the ResNet50 network architecture49. There was no significant difference in its discriminative output and results from this analysis are included in the supplement. Discriminative performance of deep learning based radiomics models as represented by area under the ROC curve (AUC) scores. One model is tuned with a dataset containing adenocarcinoma (ADC) and squamous cell carcinoma (SCC) only, displayed an AUC of 0.71 for the 51 patient ADC vs SCC test set, and the other model was tuned with a dataset containing all histology types had AUC of 0.58 on a heterogenous test set of 83 patients (ADC vs SCC vs Other). Also shown are AUC scores for models combining deep learning derived feature maps with machine learning classifiers. When used on a 4096-D feature vector represented by the first fully connected layer in Model A with dimensionality reduction, the kNN model had an AUC of 0.71, Linear SVM model had AUC of 0.68, SVM model had AUC of 0.64, and RF had AUC of 0.57. When used on a 512-D feature vector, the kNN model had AUC of 0.64, Linear SVM model had AUC of 0.62, SVM model had AUC of 0.63, and RF had AUC of 0.61. As a comparison, univariate logistic regression models using clinical parameters yielded AUC of 0.64 (p = 0.118) with smoking status, AUC of 0.55 (p = 0.544) with age, and sex was the strongest predictor of histology in our cohort, with an AUC of 0.69 (p = 0.039).

Of note, these findings are consistent with what has been described in the literature, with female and non-smoker predominance in lung adenocarcinoma of young patients. [32]. First model also demonstrated predictive value with the independent validation dataset (Lung3), achieving AUC of 0.60 (p = 0.251). This dataset contained a sample of 49 patients, of which 30 (61%) had SCC and 19 (39%) had ADC, which is a different skew from the BLCS fine-tuning and test sets. The median age and survival for the Lung3 group was 67.9 years and 3.34 years, respectively. [32]. Their best performing model was able to detect adenocarcinoma with higher specificity than sensitivity, suggesting greater potential in computer assisted diagnosis, and limited value as a screening tool. Furthermore, there was a deterministic signal using this model to predict histology on an independent and different data set, again demonstrating the robustness of the model. The ability to non-invasively predict tumor histology has the potential to boost pathologist accuracy and productivity providing significant cost and time saving benefits.



CHAPTER 4: System Requirements

## Software Requirements

**Python:** Python is a dynamically semantic, interpreted, object-oriented high- level programming language [4]. Its high-level built-in data structures, together with dynamic typing and dynamic binding, making it ideal for Rapid Application Development and as a scripting or glue language for connecting existing components [4]. We use Python version **3.7.9**

**TensorFlow:** TensorFlow Federated (TFF) is an open-source framework for decentralized data machine learning and other computations [5]. TFF was created to promote open study and experimentation with Federated Learning (FL), a machine learning approach in which a single global model is taught across numerous clients who maintain their training data locally [5]. FL has been used to train prediction models for mobile keyboards without having to submit sensitive typing data to servers, for example [5]. I use TensorFlow version **2.1.0**

**GitHub:** Over 65 million developers collaborate on GitHub to create the future of software. Manage your Git repositories and contribute to the open-source community. GitHub keeps track of the numerous modifications made to each iteration of your source code projects in a variety of programming languages. As a result, the term "Git" refers to a version control system, which is a tool that allows programmers to keep track of their code's constant revisions.

**Google Colab Pro:** With Colab Pro your notebooks can stay connected for up to 24 hours, and idle timeouts are relatively lenient. Durations are not guaranteed, though, and idle timeouts may sometimes vary. In the free version of Colab notebooks can run for at most 12 hours, and idle timeouts are much stricter than in Colab Pro.

**Anaconda:** Anaconda is an open-source distribution of the Python and R programming languages for scientific computations that is publicly available.

Anaconda Navigator is a desktop graphical user interface (GUI) that comes with the Anaconda® distribution and allows you to run programs and manage conda packages, environments, and channels without having to use command-line commands [8]. Navigator can use Anaconda.org or a local Anaconda Repository to find packages [8]. This is the perfect area for us to conduct any machine learning or deep learning project. It contains a number of programs that will support us in the development of our machine learning and deep learning projects. These software’s feature a stunning graphical user interface, which makes our job a lot easier. We use it to run our Python script as well.

## 4.2 Hardware Requirements

**RAM**: A minimum of 16 GB of RAM is necessary, but I recommend utilizing 27GB if possible because training any algorithm requires a lot of heavy lifting. Multitasking can be difficult if our memory is less than 16 GB. I have used 25 GB of RAM in google colap pro.

**CPU**: Intel Corei7 7th Generation processors are recommended since they are more powerful and give high performance. As I did work on google colab pro Corei6 5th or 3rd Gen was okay.

**GPU**: This is the most crucial factor because Deep Learning, a sub-field of Machine Learning, relies on neural networks to function and is computationally expensive. Working with images or videos necessitates a large number of matrix calculations. GPUs make it possible to process these matrices in parallel. Without a GPU, the operation could take days or months to complete. Your Best Laptop for Machine Learning, on the other hand, can complete the same task in hours. My GPU was GTX 1650. We also use the GPU of Google Colab.

**TPU**: Google Search, Google Street View, Google Photos, and Google Translate all use Google's accelerated neural network, commonly known as TPU. It is one of the most sophisticated deep learning training platforms available. TPU provides a 15-30x performance gain over current CPUs and GPUs, as well as a 30- 80x performance-per-watt ratio. The TPU is a 700MHz ASIC with a 12.5GB/s effective bandwidth that fits into a SATA hard drive slot and is connected to its host through a PCIe Gen3X16 connection.

**Storage**: Because the datasets are growing in size by the day, a minimum of 1TB HDD is necessary. If we have an SSD, a minimum of 256 GB is recommended. If you have limited capacity, however, Cloud Storage Options are an option. We can even acquire machines with powerful GPUs there. On google colab pro I used to have 64 GB of space.

**Operating System:** I use Windows 10 as my operating system. Because I have been using it for a long time and am familiar with it. However, while Linux is the most popular operating system, Windows and MacOS can both run Virtual Linux environments and we can work on both systems as well.



CHAPTER 5: Library Details

* 1. **Keras:** Keras is an opensource package that works well on both the CPU and the GPU. Deep learning, specifically neural networks, makes advantage of it. The popular ML library works with neural network construction blocks like:
     + Activation functions,
     + Layers,
     + Objectives, and
     + Optimizers.

Keras supports convolutional and recurrent neural networks in addition to regular neural networks. The ML library also includes a slew of tools for manipulating pictures and text graphics.

Highlights

* + - Can run on top of:
      * Microsoft Cognitive Toolkit (CNTK)
      * PlaidML
      * R
      * TensorFlow and
      * Theano.
    - Allows for rapid deep neural network research.
    - It provides a high-level, understandable set of abstractions to make deep learning model creation easier.
    - Superb community support.
    - Support available in TensorFlow’s core library.

**5.2 Matplotlib:** Matplotlib is a machine learning package that uses 2D plotting to create publication ready figures, pictures, and plots in a variety of formats. The Matplotlib library allows you to create rich, high-quality graphs with just a few lines of code.

* + - Bar charts,
    - Error charts,
    - Histograms,
    - Scatter plots, etc.

Although Matplotlib is intuitive, users who are used to the MATLAB interface will find it much more so, especially when utilizing the pyplot module. The ml library provides an object-oriented API for embedding graphs and plots in programs utilizing common GUI toolkits like as GTK+, Qt, and wxPython.

Highlights

* + - Ample documentation.
    - Excellent community support.
    - Functionality extension using many toolkits, including: o Cartopy
      * Excel tools
      * GTK tools and
      * Qt interface
    - The greater degree of customization.
    - SciPy uses Matplotlib.

**5.3 NumPy:**   The acronym NumPy stands for Numerical Python. It's evident from the name that this is a computation-oriented library. Developers can save a lot of time in scientific computations that need a lot of matrix operations by using the Python-based library.

NumPy arrays, a unique type of array used by the NumPy library, execute massive matrix-based calculations in milliseconds. This is made feasible by the C programming language's implementation of NumPy arrays.

NumPy has become one of the most popular libraries/packages for machine learning, particularly natural language processing, as a result of the aforementioned.

Highlights

* + - There is a comprehensive set of high-complexity mathematical functions available for processing massive multi-dimensional arrays and matrices.
    - Excellent for Fourier transformations, linear algebra, and random numbers.
    - Excellent community support.
    - TensorFlow uses this to manipulate tensors on the backend.
    - Out-of-the-box integration tools for C, C++, and Fortran code.

**5.4 Pandas:** When it comes to working with massive amounts of tabular data, pandas are the go-to machine learning library. Pandas is what Microsoft Excel is to Windows for Python. The ML library reduces the time and effort required for large, complex calculations to just a few lines of code.

In addition, pandas come with a huge list of pre-existing instructions that will save ML developers time by eliminating the need to write code for various mathematical processes. Apart from data manipulation, the Pandas library also assists with data transformation and visualization. The panda's library makes use of two different kinds of data structures:

* + - Series (1-dimensional), and
    - DataFrame (2-dimensional).

Developers may manage a wide range of data requirements and scenarios from engineering, finance, science, statistics, and other fields by using this duo in-line.

Pandas allow data scientists to focus less on creating boilerplate code and more on the real problem-solving connected with the tabular data they're working with.

Highlights

* + - Able to handle any type of statistical or observational dataset with ease:
* Data in an arbitrary matrix with homogeneous or heterogeneous data.
* Data in both ordered and unordered time series.
* Tabular data with heterogeneous data columns
  + - Outstanding community support.
* The library of choice for tackling real-world data analysis in Python, with exceptional efficiency in processing unequal time-series data.
  + - Supports data structures that are expressive, quick, and versatile, and can operate with both labeled and relational data.

**5.5 Scikit-Learn:** You name it, and Scikit-Learn has anything, whether it's decision trees, linear regression, logistical regression, or SVMs. It's one of the most widely used machine learning libraries for creating algorithms. Scikit-Learn can also:

1. Preprocess data and
2. Vectorize text using BOW, hashing vectorization, TF-IDF, and other methods.

Scikit-Learn, which is written in C and Python, has a growing and devoted community of programmers, machine learning hobbyists, and IT professionals all around the world. It is based on NumPy and SciPy, two of the most widely used machine learning frameworks for scientific computation.

The Scikit-learn library's only flaw is that it doesn't provide adequate support for distributed computing in large-scale production environments. Hopefully, it will be resolved in future releases of the famous machine learning library.

Highlights

* It has a wide range of supervised and unsupervised learning techniques and can be used for data analysis and data mining.
* Support from the mushrooming community.
* In addition to model management and preprocessing, the ML library can handle functions related to:
* Classification,
* Clustering,

**5.6 TensorFlow:** TensorFlow is one of the most powerful deep learning packages available. The ML library, created by Google, is a quick-start alternative for product- based businesses since it provides outstanding model prototyping, production, and everything in between.

The Tensorboard, a web-based visualization tool included with the TensorFlow framework, allows developers to see model parameters, gradients, and performance. TensorFlow Lite and TensorFlow Serving are two frameworks provided by the DL library for quickly deploying machine learning models.

Despite its many advantages, the machine learning library has been chastised for its poor graph implementation. This is due to the library's requirement that the graph is compiled first. Hopefully, with subsequent rollouts, we will see improvements in the aforementioned.

Highlights

* + 1. Google's support.
    2. Provides C++ and Python APIs that are extremely stable. It can also expose APIs for other programming languages

that are backward compatible. (However, these can be unstable.)

* + 1. There is a lot of documentation.
    2. Has a versatile architecture that lets it run on a variety of CPUs, GPUs, and TPUs (Tensor Processing Units).
    3. Supports a large range of toolkits for constructing ML models at various levels of abstraction; more than just a library, it's a popular computational framework for developing strong Machine learning models.
    4. A large, dependable community.

#### 5.7 Predefined classes and functions

We use Keras modules in our projects that they contain pre-defined classes, functions, and variables. In this section, we'll learn about the Keras modules.

**Keras.models:** The real neural network model is represented by the Keras model. Keras offers two ways to build neural network models. Sequential API's basic concept is to arrange Keras layers in a sequential manner. The Functional API has a more versatile and powerful Functional API for the same purpose, but it's not yet available.

The Sequential() API can be used to generate an ANN model, as seen below.

from keras.models import Sequential model = Sequential()

A functional API is a way to build more complicated models. You can define several inputs or outputs that share layers in a functional model . To access the model's input and output, we first build an instance for it and connect it to the layers of its layers.

**keras.optimizers:** Right optimizers are Classes or methods used to change the attributes of your machine/deep learning model such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster. There are many optimizers' algorithms we have in PyTorch and TensorFlow library.

TensorFlow primarily supports nine optimizer classes, including Adadelta, FTRL, NAdam, Adadelta, and many others.

* Adadelta: Adadelta is an optimizer that uses the Adadelta algorithm.
* Adagrad: an optimizer that uses the Adagrad algorithm.
* Adam: an optimizer that uses the Adam algorithm.
* Adamax: an optimizer that uses the Adamax algorithm.
* Ftrl: Ftrl is an optimizer that uses the FTRL algorithm.
* Nadam: The NAdam algorithm is implemented by this optimizer.
* Optimizer class: Keras optimizers' base class.
* RMSprop: This is an optimizer that uses the RMSprop algorithm.
* SGD: Optimizer for gradient descent (with momentum).

**keras.layers:** Keras layers are the most basic component of keras models. Layers are made with a variety of layer\_ functions, and they're usually put together by stacking calls to them with the pipe operator.

There are several different levels to choose from, including:

* [Core Layers](https://keras.rstudio.com/reference/index.html#section-core-layers)
* [Convolutional Layers](https://keras.rstudio.com/reference/index.html#section-convolutional-layers)
* [Pooling Layers](https://keras.rstudio.com/reference/index.html#section-pooling-layers)
* [Activation Layers](https://keras.rstudio.com/reference/index.html#section-activation-layers)
* [Dropout Layers](https://keras.rstudio.com/reference/index.html#section-dropout-layers)
* [Locally-connected Layers](https://keras.rstudio.com/reference/index.html#section-locally-connected-layers)
* [Recurrent Layers](https://keras.rstudio.com/reference/index.html#section-recurrent-layers)
* [Embedding Layers](https://keras.rstudio.com/reference/index.html#section-embedding-layers)
* [Normalization Layers](https://keras.rstudio.com/reference/index.html#section-normalization-layers)
* [Noise Layers](https://keras.rstudio.com/reference/index.html#section-noise-layers)
* [Merge Layers](https://keras.rstudio.com/reference/index.html#section-merge-layers)
* [Layer Wrappers](https://keras.rstudio.com/reference/index.html#section-layer-wrappers)

**keras.utils:** Keras includes the NumPy utility library, which includes routines for manipulating NumPy arrays. A NumPy array (or) a vector with integers that represent distinct categories can be transformed into a NumPy array (or) a matrix with binary values and columns equal to the number of categories in the data using the method to categorical().This function returns a binary vector matrix.

This function returns a binary values matrix (either'1' or '0'). It has the same number of rows as the input vector and the same number of columns as the number of classes

**keras.callbacks:** A call back is a set of functions that can be used at specific points during the training process. During training, you can use call-backs to acquire a view of the model's internal states and statistics. When you wish to automate some tasks after each training/epoch, you define and use a callback. This allows you to have more control over the training process. Stopping training when you attain a specified accuracy/loss score, saving your model as a checkpoint after each successful session, altering learning rates over time, and more are all examples of this.

**keras.applications:** The Keras applications module is used to deliver deep neural network models that have already been trained. Keras models are used for feature extraction, prediction, and fine tuning. Model Architecture and model Weights are the two aspects of a trained model. Because model weights are a huge file, we must download and extract them from the ImageNet database.

**Sklearn.model\_selection:** This approach divides arrays or matrices into random train and test subsets. When running a (supervised) machine learning experiment, it's typical to set aside a portion of the data as a test set. This allows us to assess how effectively the model will generalize to new data sets not observed during the training phase. In other words, this technique attempts to avoid the overfitting issue, in which the algorithm begins to memorize the labels from the training set but is unable to predict anything for the unknown test data. However, if we only do a train/test split, the findings may be influenced by a random selection for the pair of train/test sets.

**matplotlib.pyplot:**

Pyplot is a Python matplotlib API that essentially turns matplotlib into a viable open source alternative to MATLAB. Matplotlib is a data visualization package that generates plots, graphs, and charts. Pyplot adds two important features to matplotlib:

A MATLAB-style interface makes it easier for individuals who are already familiar with MATLAB to learn Python. Pyplot is stateful, which means it remembers the state of an object when you plot it for the first time. Until plt.close() is encountered in the code, this is required for use in the same loop or session state . When establishing multiple plots in a row, state is also significant.

The pyplot API is made up of a hierarchy of Python code objects that comprises a number of functions, the most important of which is matplotlib.pyplot. This stack can be divided into three layers, each of which is interdependent:

* Scripting layer – this layer is used to define a figure that has one or more plots with axes.
* Artist Layer – used to alter plot features such adding labels, drawing lines, and

so on.

* + Backend Layer – used to style a graphic for display in a target application like a Jupyter Notebook.



CHAPTER 6: Methodology

This chapter gives an overview of the different parts of the work chronologically. It mainly discusses the theories, techniques, and step by step workflow of the work.

## 6.1 Workflow

A complete workflow diagram of the proposed method is shown here.

**Data Pre-processing**

**Data Augmentation**

**Model Implementation**

The above diagram shows the complete picture for better understanding of the whole system.

## Implemented Model:

## We used several famous models, so that we could use Transfer Learning.

## 

## 

## 6.2.1 VGG16

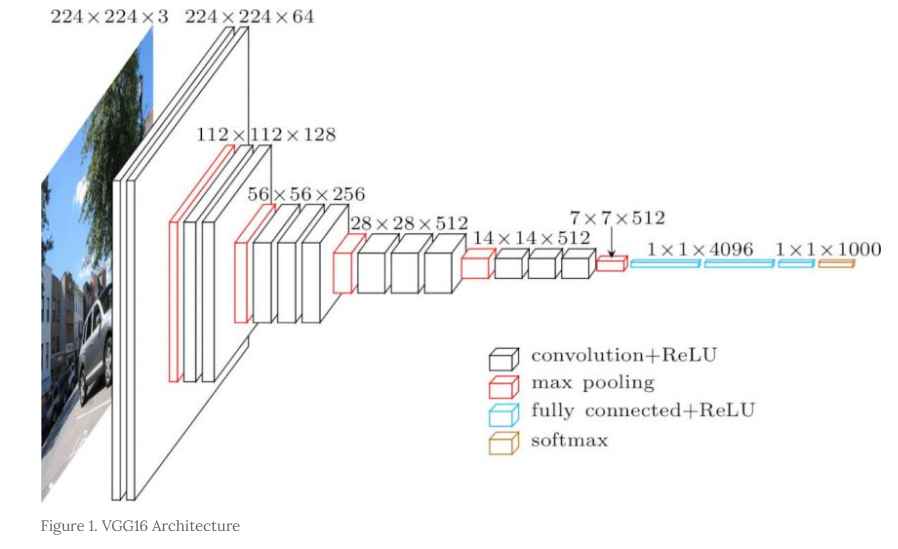
## VGG-16 is a convolutional neural network that is 16 layers deep. We can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

## Now let’s get to the detail implementation of VGG16. I will:

* Show the overview of VGG16

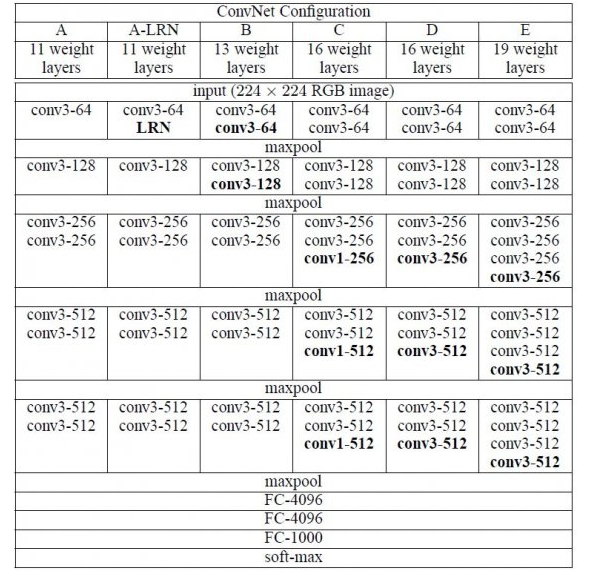
### Overview

The architecture depicted below is VGG16.



The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv.  layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixels window, with stride 2. [1]

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.



All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.

**6.2.2 VGG19**

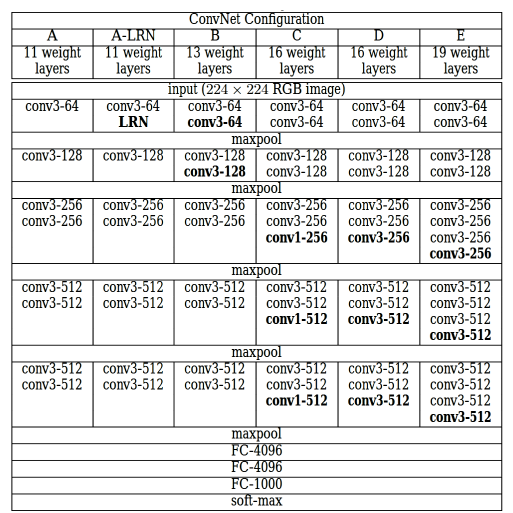
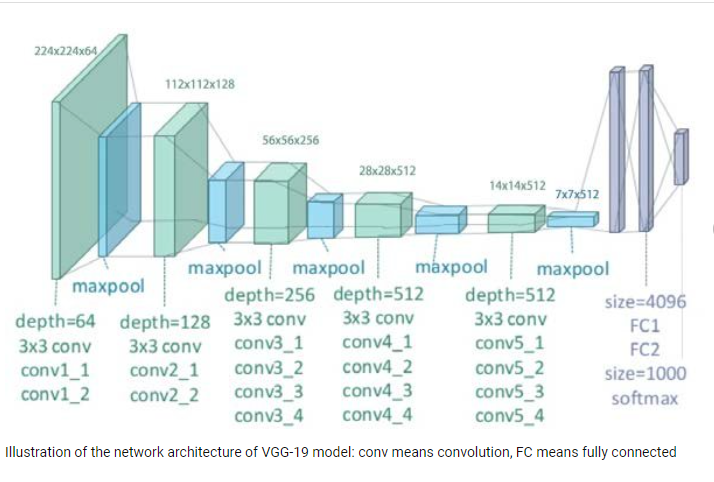
VGG-19 is a convolutional neural network that is 19 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.[2]

## Now let’s get to the detail implementation of VGG19. I will:

* Show the overview of VGG19

**Overview**

* A fixed size of (224 \* 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
* The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
* Used kernels of (3 \* 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
* spatial padding was used to preserve the spatial resolution of the image.
* max pooling was performed over a 2 \* 2 pixel windows with sride 2.
* this was followed by Rectified linear unit(ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
* implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way *ILSVRC* classification and the final layer is a softmax function.



The column E in the table is for VGG19 (other columns are for other variants of VGG models)

**6.2.3 ResNet**

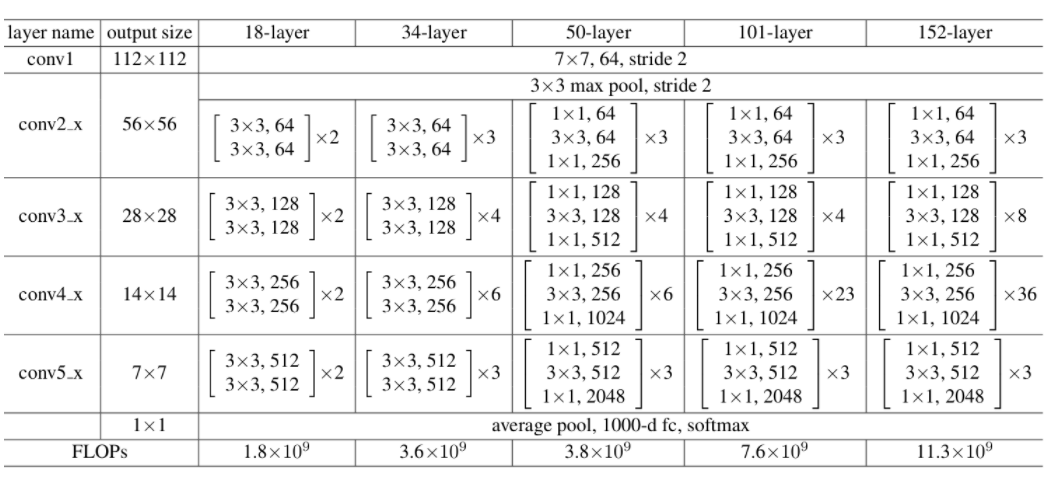
ResNet network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into the residual network.

## Now let’s get to the detail implementation of ResNet50, ResNet101, ResNet152.

I will show the overview of ResNet50, ResNet101, ResNet152

**Overview**

Each ResNet block is either two layers deep (used in small networks like ResNet 18, 34) or 3 layers deep (ResNet 50, 101, 152).



**50-layer ResNet:** Each 2-layer block is replaced in the 34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet (see above table). They use option 2 for increasing dimensions. This model has 3.8 billion FLOPs.

**101-layer and 152-layer ResNets:** They construct 101-layer and 152-layer ResNets by using more 3-layer blocks (above table). Even after the depth is increased, the 152-layer ResNet (11.3 billion FLOPs) has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs).

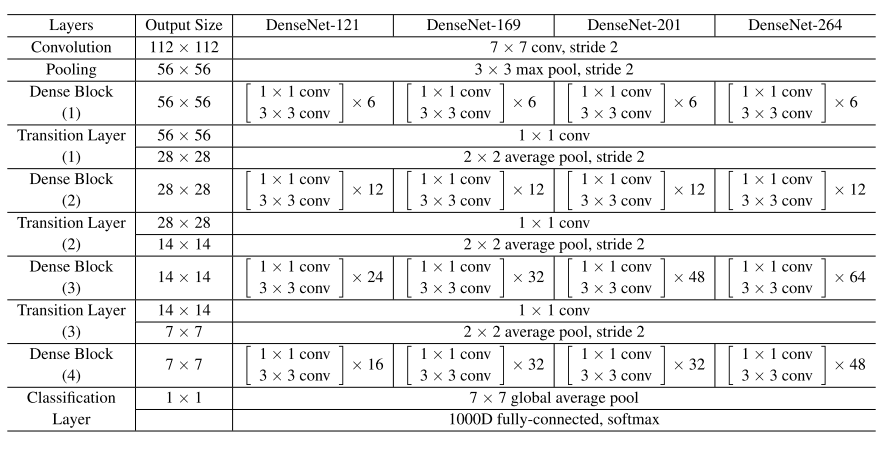
**6.2.4 DenseNet169**

A DenseNet169 is a type of convolutional neural network that utilizes dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other.

**Overview**

A summarization of the various architectures implemented for the ImageNet database have been provided in the table above. Stride is the number of pixels shifts over the input matrix. A stride of 'n' (default value being 1), indicates that the filters are moved 'n' pixels at a time.

Using the DenseNet-169 architecture to understand the table, we can see that every dense block has varying number of layers (repetitions) featuring two convolutions each; a 1x1 sized kernel as the bottleneck layer and 3x3 kernel to perform the convolution operation.



Also, each transition layer has a 1x1 convolutional layer and a 2x2 average pooling layer with a stride of 2. Thus, the layers present are as follows:

1. Basic convolution layer with 64 filters of size 7X7 and a stride of 2
2. Basic pooling layer with 3x3 max pooling and a stride of 2
3. Dense Block 1 with 2 convolutions repeated 6 times
4. Transition layer 1 (1 Conv + 1 AvgPool)
5. Dense Block 2 with 2 convolutions repeated 12 times
6. Transition layer 2 (1 Conv + 1 AvgPool)
7. Dense Block 3 with 2 convolutions repeated 24 times
8. Transition layer 3 (1 Conv + 1 AvgPool)
9. Dense Block 4 with 2 convolutions repeated 16 times
10. Global Average Pooling layer- accepts all the feature maps of the network to perform classification
11. Output layer

Therefore, DenseNet-121 has the following layers:

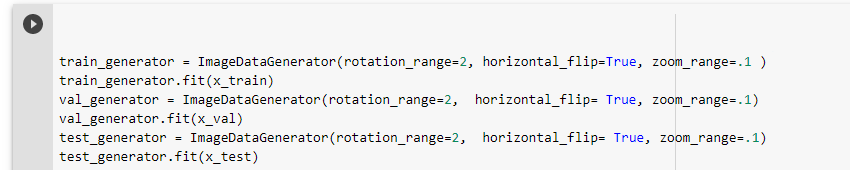
* 1 7x7 Convolution
* 58 3x3 Convolution
* 61 1x1 Convolution
* 4 AvgPool
* 1 Fully Connected Layer
* In short, DenseNet-121 has 120 Convolutions and 4 AvgPool.
* All layers i.e.those within the same dense block and transition layers, spread their weights over multiple inputs which allows deeper layers to use features extracted early on.
* Since transition layers outputs many redundant features, the layers in the second and third dense block assign the least weights to the output of the transition layers.
* Also, even though the weights of the entire dense block are used by the final layers, there still may be more high-level features generated deeper into the model as there seemed to be a higher concentration towards final feature maps in experiments.

**6.3 Data Preprocessing**

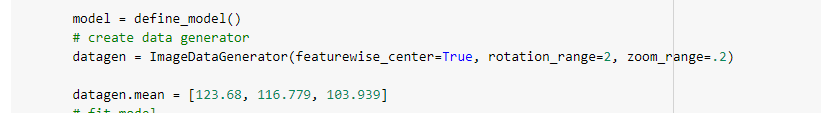
Preprocessing data is an important task and is very important for accuracy. This is because, data is mostly noisy. It sometimes has missing values and also false values. So, feature selection and feature extractions are two major methods of Preprocessing which directly impact the accuracy of the model. Feature selection is a method of selecting some features out of the data and discarding the irrelevant ones. Feature extraction is the method which converts M attributes of the data to N attributes. The original number of attributes may be greater than the new number of attributes. They may be less or equal to as well. In our Drive and Stare datasets each image can be made up of thousands of pixels. In that case, the algorithm we train with it may turn out to be inefficient. So, changing pixels can be an important part of data preprocessing. The feature extraction techniques in images are mostly dependent on the data. In our dataset, our data was not level. So, we use ‘One-Hot-Encoding’ to level our data.

Below data processing for different models are shown:

For resNet50



For Vgg19 and Vgg16



**6.4 One-Hot-Encoding**

One-Hot-Encoding is a technique for converting categorical variables into a format that can be fed into machine learning algorithms to improve prediction accuracy. A technique for representing categorical variables as binary vectors is also known as One Hot Encoding. The categorical variables must be converted to integer values for this to work. The binary 0 and 1 will be represented using the integer values. This is a difficult situation. This is implemented in the SciKit-Sklearn (One Hot Encoder) library used in Python.

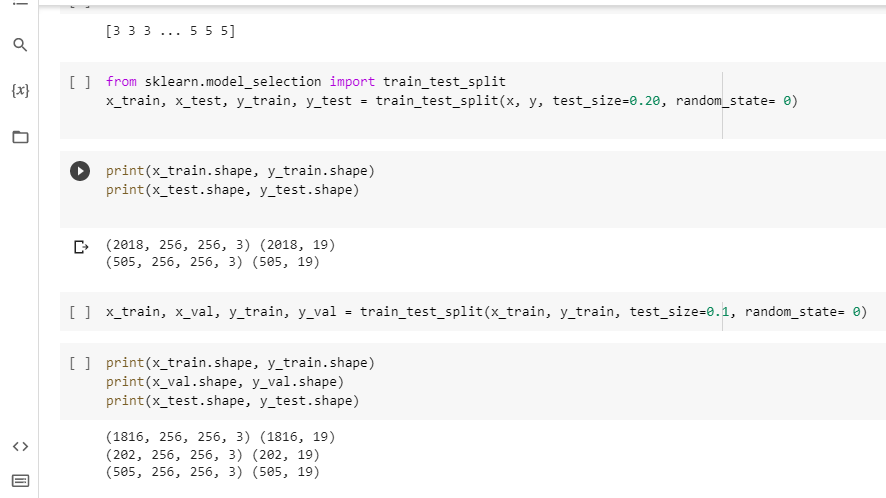
When using categorical data in linear and SVM models, this is critical. Because many Machine Learning models can't work with categorical data directly, this is part of the data pre-processing stage. The data must first be converted to binary vectors. The encoder can find the unique values per feature in a two-feature dataset and convert the data to a binary one-hot encoding.

In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column.



**6.5 Training-Validation-Testing Split:**

After using one hot encoder I split the dataset into three parts including training, testing and validation part. Within 100% of the data 70% data were taken for training purpose, 20% data were taken for testing and 10% of the data of were taken for validation. As we have three folders. I did the same split for rest of the two folders.



## 6.6 Data Augmentation

## Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [18]. It acts as a regularize and helps reduce overfitting when training a machine learning model. Data augmentation is very useful if you would like to augment your data or increase the amount of training or validation data. Now if you have to 2000 images let’s say and you would like to get 5,000 or 10,000 of those then this can be very useful. But if you only have 5 or 10 images don’t expect to get 2,000 or 20,000 images sort of data augmentation a still be able to get decent results out of your deep learning. Because, Deep learning will be highly biased towards these 5 or 10 images that you’re augmenting.

## In augmentation part, we did rotation, zoom, centering, standard normalization, changing brightness/height/weight, horizontal/vertical flipping, rescaling etc.

### 6.7 Optimizer

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster. There are lots of Optimizer in image processing. But we used Adam Optimizer.

## Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data

### 6.7.1 How Does Adam Work?

Adam is not the same as traditional stochastic gradient descent. For all weight updates, stochastic gradient descent maintains a single learning rate (called alpha), which does not change during training. Each network weight (parameter) has its own learning rate, which is adjusted separately as learning progresses.

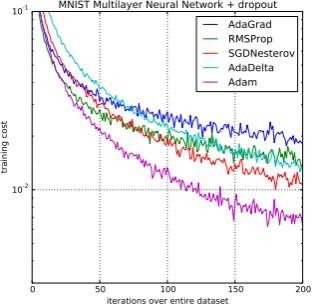
“The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [20].”

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent [20]. Specifically:

·        Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g., natural language and computer vision problems) [20].

·        Root Mean Square Propagation (RMSProp) that also maintains per- parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g., how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g., noisy).

*Using large models and datasets, we demonstrate Adam can efficiently solve practical deep learning problems.*

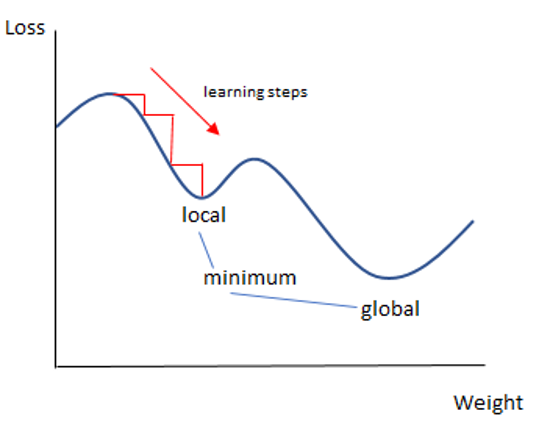


Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Taken from Adam: A Method for Stochastic Optimization, 2015

### 6.7.2 How Does SGD Work?

The name Stochastic Gradient Descent - Classifier (SGD-Classifier) might mislead some user to think that SGD is a classifier. But that’s not the case! SGD Classifier is a linear classifier (SVM, logistic regression, a.o.) optimized by the SGD. These are two different concepts. While SGD is a optimization method, Logistic Regression or linear Support Vector Machine is a machine learning algorithm/model. You can think of that a machine learning model defines a loss function, and the optimization method minimizes/maximizes it.



### Loss Function

### One of the most important components of Neural Networks is the loss function. Loss is nothing more than a Neural Net prediction error. Loss Function is the method for calculating the loss.

### The error between the algorithm's output and a given target value is quantified by a loss function, cost function, or error function.

### The loss function binary cross entropy tells you how inaccurate your model's predictions are. It's used to make yes-or-no decisions. Consider multi-label classification issues. Where an example can belong to multiple classes at once, the model tries to determine whether the example belongs to that class or not for each class.

### Cross-entropy is the default loss function to use for binary classification problems.

### It is intended for use with binary classification where the target values are in the set {0, 1}.

### Under the maximum likelihood inference framework, it is the preferred loss function mathematically. It is the loss function that should be evaluated first, and only changed if there is a compelling reason to do so.

### For predicting class 1, cross-entropy will compute a score that summarizes the average difference between the actual and predicted probability distributions. The score is minimized and a perfect cross-entropy value is 0.

### Cross-entropy can be specified as the loss function in Keras by specifying

### ‘categorical\_crossentropy‘ when compiling the model.

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

### The function requires that the output layer is configured with a single node and a ‘sigmoid ‘activation in order to predict the probability for classes.

### Activation Function

### The output of a neural network is determined by activation functions, which are mathematical equations. The function is attached to each neuron in the network and determines whether it should be activated or not based on whether each neuron's input is relevant for the model’s prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between minus 1 and 1. Additionally, because activation functions are calculated across thousands or even millions of neurons for each data sample, they must be computationally efficient. Modern neural networks use a technique called back propagation to train the model which places an increased computational strain on the activation function and its derivative function. In a neural network numeric data points called inputs are fed into the neurons in the inputs layer. Each neuron has a weight and multiplying the input number with the weight gives the output of the neuron which is transferred to the next layer.

### The activation function is a mathematical gate that connects the current neuron's input to its output, which goes to the next layer. It can be as simple as a step function that turns the neuron output on and off depending on a ruler threshold. Or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function. Increasingly neural networks use nonlinear activation functions which can help the network learn complex data computation and learn almost any function representing a question and provide accurate predictions.

### 6.9.1 Softmax

### The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

### The softmax function is sometimes called the soft argmax function, or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used for multi-class classification, and its formula is very similar to the sigmoid function which is used for logistic regression. The softmax function can be used in a classifier only when the classes are mutually exclusive.

### Many multi-layer neural networks end in a penultimate layer which outputs real-valued scores that are not conveniently scaled and which may be difficult to work with. Here the softmax is very useful because it converts the scores to a normalized probability distribution, which can be displayed to a user or used as input to other systems. For this reason, it is usual to append a softmax function as the final layer of the neural network.

### Softmax Formula:

### The softmax formula is as follows: https://lh6.googleusercontent.com/3vcfJ5hJhsMZAMFIbQOEycfVW1t6rh1CXt62DeMk8RPPXVzV4vCcURNm_z_F7618uAeSHT7qT7wE_UiK5Ic0b-Eeuunn6iTGeHWbpAaUAP6-G2ePubeGWCb4_TmSapeaimZqvuUs

### *Mathematical definition of the softmax function*

### 

### where all the zi values are the elements of the input vector and can take any real value. The term on the bottom of the formula is the normalization term which ensures that all the output values of the function will sum to 1, thus constituting a valid probability distribution.

### Hyper-Parameters

In machine learning, a hyperparameter is a parameter whose value is used to control the learning process and whose value is given by the Programmer where the values of other parameters are derived via training. In Deep Learning algorithms, there are so many hyper-parameters whose values need to be adjusted like knobs which makes it really annoying.

Choosing hyper-parameter values is formally equivalent to the question of model selection. We define a hyper-parameter for a learning algorithm A as a variable to be set prior to the actual application of A to the data, one that is not directly selected by the learning algorithm itself. It is basically an outside control knob. The hyper-parameters can be fixed by hand or tuned by an algorithm, but their value has to be selected. The value of some hyperparameters can be selected based on the performance of A on its training data, but most cannot. For any hyper-parameter that has an impact on the effective capacity of a learner, it makes more sense to select its value based on out-of-sample data (outside the training set), e.g., a validation set performance, online error, or cross-validation error.

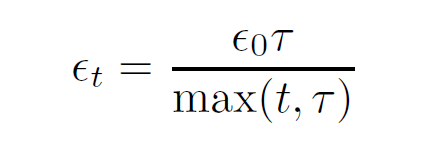
**6.11 The initial learning rate:**

This is often the single most important hyperparameter and one should always make sure that it has been tuned (up to approximately a factor of 2). Typical values for a neural network with standardized inputs (or inputs mapped to the (0,1) interval) are less than 1 and greater than 10^−6 but these should not be taken as strict ranges and greatly depend on the parametrization of the model.

We started the initial learning rate with 0.001 and then started tuning. And we saw that, as the learning rate decreases, the model performs better. So, the final learning rate, we decided on was 0.0001.

**6.11.1 Learning rate scheduler:**

The choice of strategy for decreasing or adapting the learning rate schedule (with hyperparameters such as the time constant τ in the Equation below). The default value of τ → 1 means that the learning rate is constant over training iterations. In many cases the benefit of choosing this default value is small. An example of O(1/t) learning rate schedule, used in Bergstra and Bengio (2012) is

****

which keeps the learning rate constant for the first τ steps and then decreases it in https://lh5.googleusercontent.com/EvG5FzQ_r0GL8zaeQEaI-uL9eIlF8Ee-Gz-wQxIXw9kSmVBZcLOecw6PoGsEQNVTjZ8AFJvNhxoRTb6B8VsW7IWe-c-ZSRtxckJES_aLjtLzFlNFKCKO7v5xVVftt_T_uRJyJ2Fq, with traditional recommendations suggesting α = 1.[26]

An adaptive and heuristic way of automatically setting τ above is to keep https://lh3.googleusercontent.com/2o3LhRi586pcwZjRTG3Et7bdq-MznngV2JD2ZIUVL732iAEfKCZQVpI5X8xm9PxhnXD_xh6yRkx_w19eL5lsLdhoqet27qrigQjgGCuIOxubf5C2QAAnDXkH6BYNT9JVeIaU4One constant until the training criterion stops decreasing significantly from epoch to epoch. An alternative to a fixed schedule with a couple of (global) free hyper-parameters like in the above formula is the use of an adaptive learning rate heuristic at regular intervals during training, using a fixed small subset of the training set (what matters is only the number of examples used, not what fraction of the whole training set it represents), continue training with N different choices of learning rate (all in parallel), and keep the value that gave the best results until the next re-estimation of the optimal learning rate.

In our case, we used to decrease the learning rate after 8 epochs if the validation accuracy was not increasing, with a factor of 0.6 until we reach the minimum learning rate of 5×10^–5.

**6.11.2 mini-batch size:**

The Mini batch size is typically chosen between 1 and a few hundred.

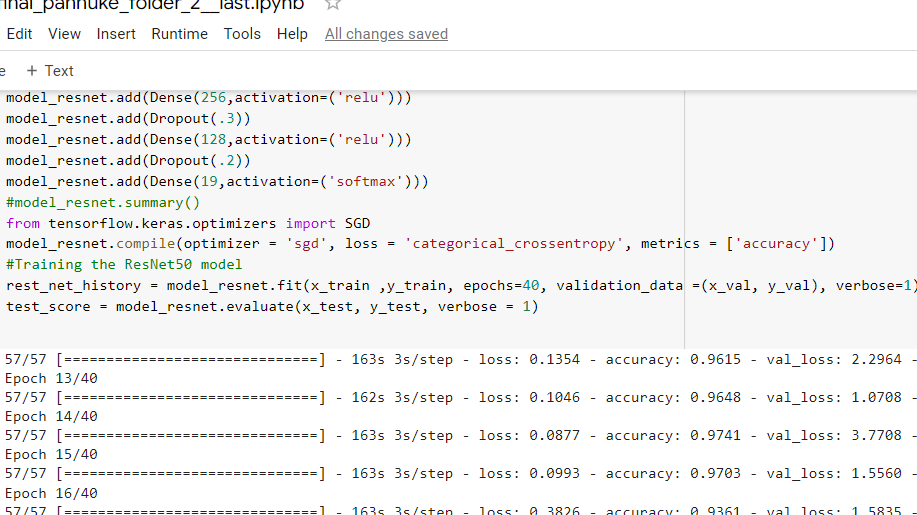
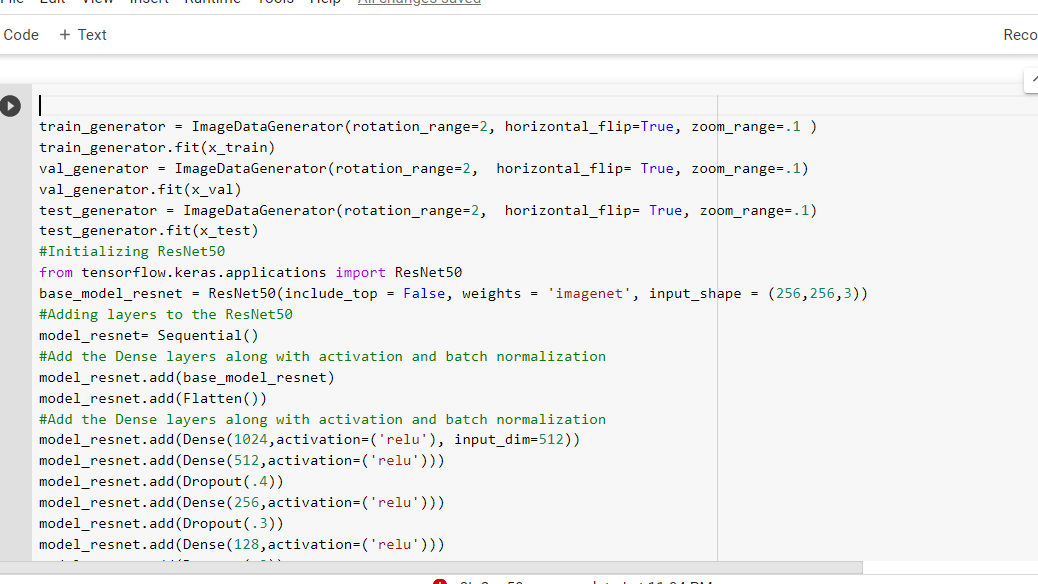
B = 32 is a good default value that we used in our case.

B with values above 10 takes advantage of the speed-up of matrix-matrix products over matrix-vector products. The impact of B is mostly computational, i.e., larger B yields faster computation (with appropriate implementations) but requires visiting more examples in order to reach the same error, since there are fewer updates per epoch.

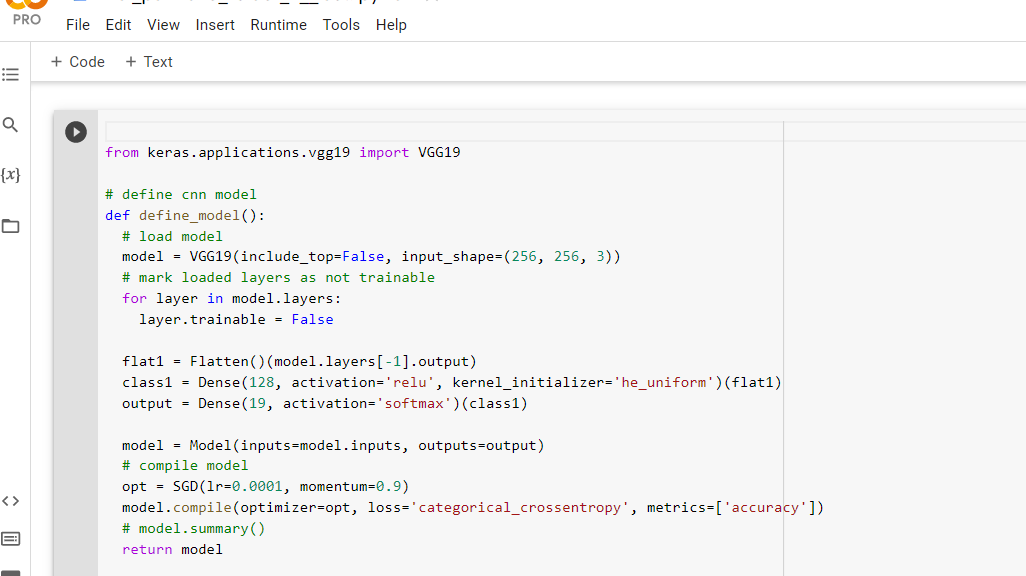
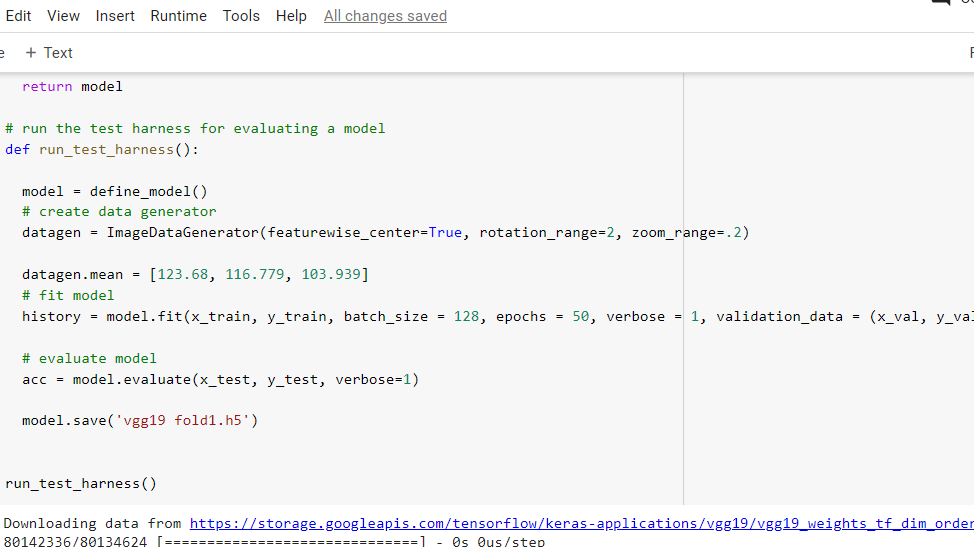
**6.12 Training and evaluating model Criterion**

The approaches were used for all the three folders of the pannuke dataset.

**6.12.1 ResNet50**: I generated a training data generator for x \_test, x\_val, x\_train . Here rotation\_range was 2 , horizontal \_flip was true and zoom range was .02.I used weights of “imagenet” and my shape for training was (256,256,3).After that I added some dense layers along with activation and batch normalization. The activation was used “relu” and the optimizer was ‘sgd’ . After some decent epoch I got the results which are discussed and noted in the result section. The same procedure was followed for the rest of the two folders.



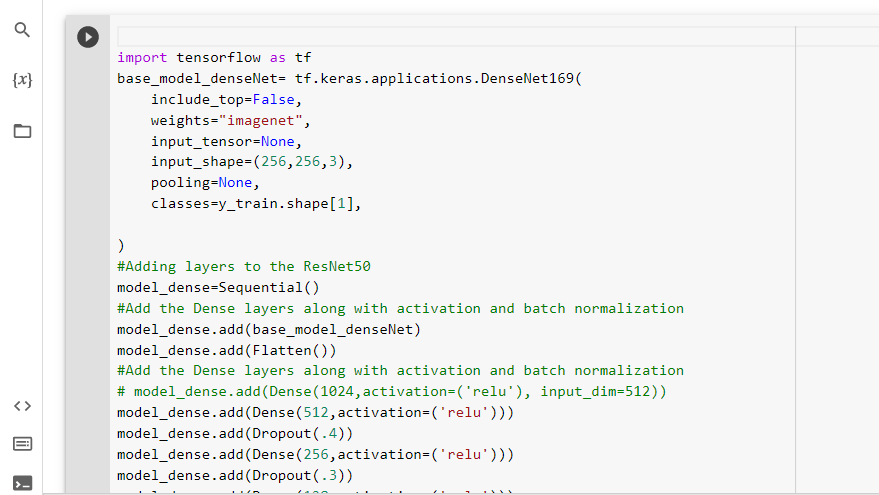
**6.12.2 Vgg19**: while defining vgg19 from keras I included the top false. My shape was (256,256,3) . I kept all the layers trainable =false. Then I flattened the layers and used the activation “relu “ and the activation was softmax. The learning rate for SGD was 0.0001 and the momentum was 0.9. I used the loss as categorical \_crossentropy as it was a multiclass classification for 19 type tissue cells. While generating the ImageDataGenerator the featurewise\_center was true. The rotation range was 2 and the zoom\_range was .2. After some decent epochs the I got the result.

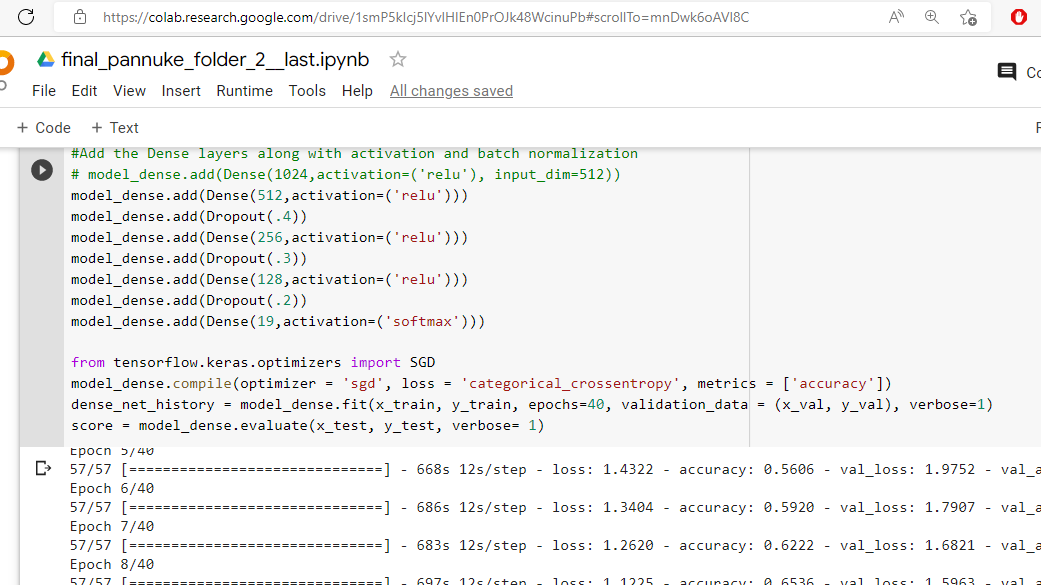


**6.12.3 Vgg16**: While defining vgg16 from keras I included the top false. My shape was (256,256,3) . I kept all the layers trainable =false. Then I flattened the layers and used the activation “relu“ and the activation was softmax. The learning rate for SGD was 0.0001 and the momentum was 0.9. I used the loss as categorical \_crossentropy as it was a multiclass classification for 19 type tissue cells. While generating the ImageDataGenerator the featurewise\_center was true. The rotation range was 2 and the zoom\_range was .2. After some decent epoch I got the result.



**6.12.4 DenseNet169**: While taking the model from keras I used the weight of the “imagenet”, include\_top was false, and the input shape was (256,256,3). The I flatten the layers and use activation and dropout. I used “relu” as activation. And the last layer activation was softmax. The optimizer was “SGD”; the loss was categorical \_crossentopy. After some decent epoch I got the result and evaluated the model.





**6.13 Ensemble learning:**

The ensemble methods in machine learning combine the insights obtained from multiple learning models to facilitate accurate and improved decisions. These methods follow the same principle as the example of buying an air-conditioner cited above.

In learning models, noise, variance, and bias are the major sources of error. The ensemble methods in machine learning help minimize these error-causing factors, thereby ensuring the accuracy and stability of machine learning (ML) algorithms.

Mode: In statistical terminology, "mode" is the number or value that most often appears in a dataset of numbers or values. In this ensemble technique, machine learning professionals use a number of models for making predictions about each data point. The predictions made by different models are taken as separate votes. Subsequently, the prediction made by most models is treated as the ultimate prediction.

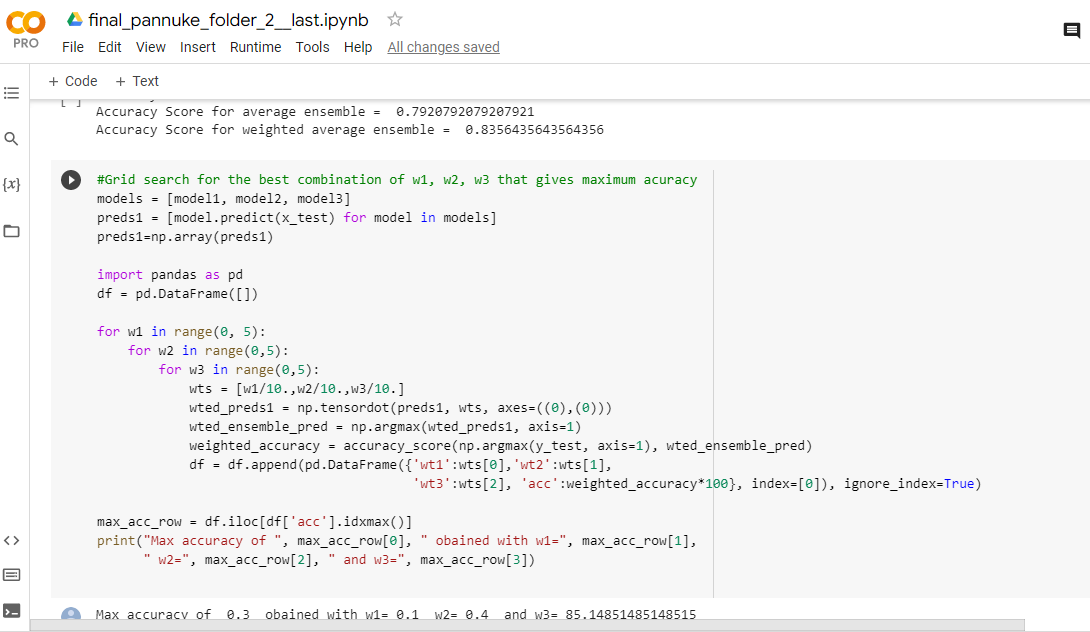
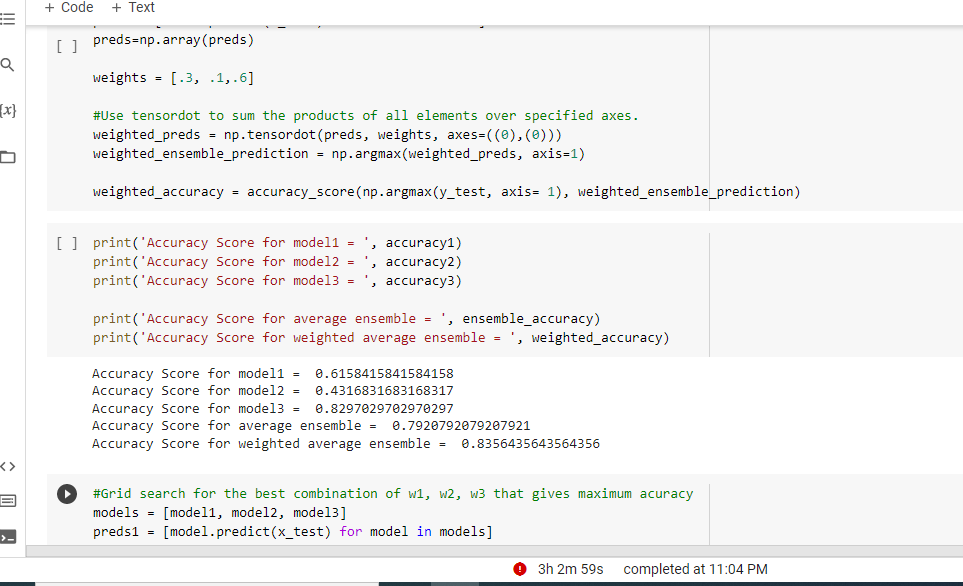
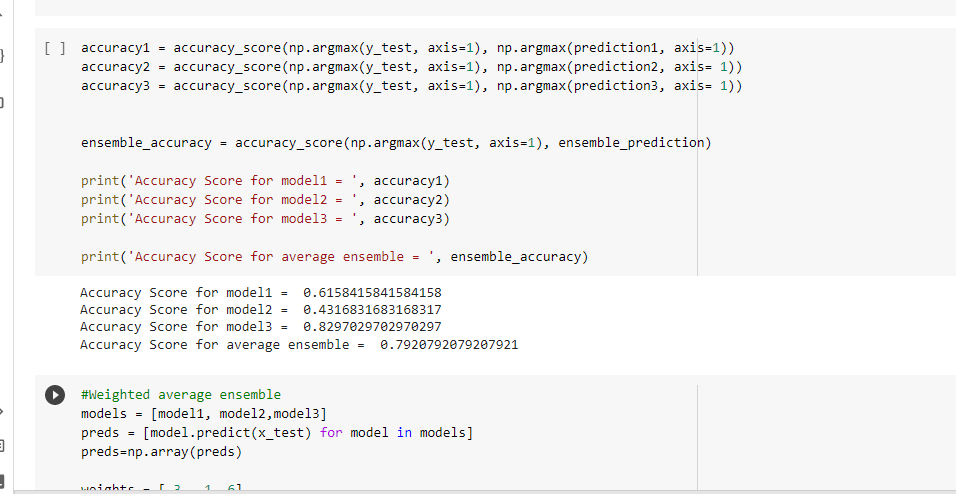
The Mean/Average: In the mean/average ensemble technique, data analysts take the average predictions made by all models into account when making the ultimate prediction.

Let's take, for instance, one hundred people rated the beta release of your travel and tourism app on a scale of 1 to 5, where 15 people gave a rating of 1, 28 people gave a rating of 2, 37 people gave a rating of 3, 12 people gave a rating of 4, and 8 people gave a rating of 5.

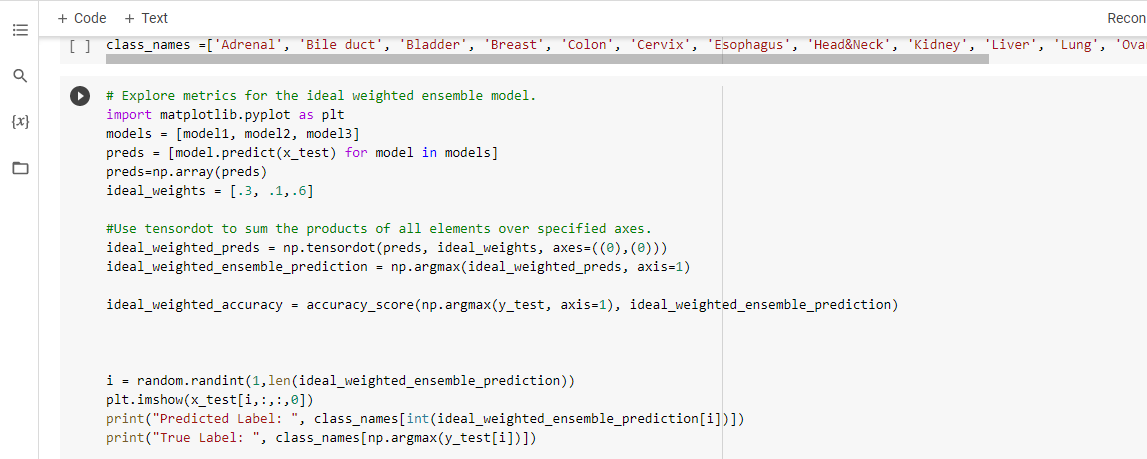
The average in this case is - (1 \* 15) + (2 \* 28) + (3 \* 37) + (4 \* 12) + (5 \* 8) / 100 = 2.7

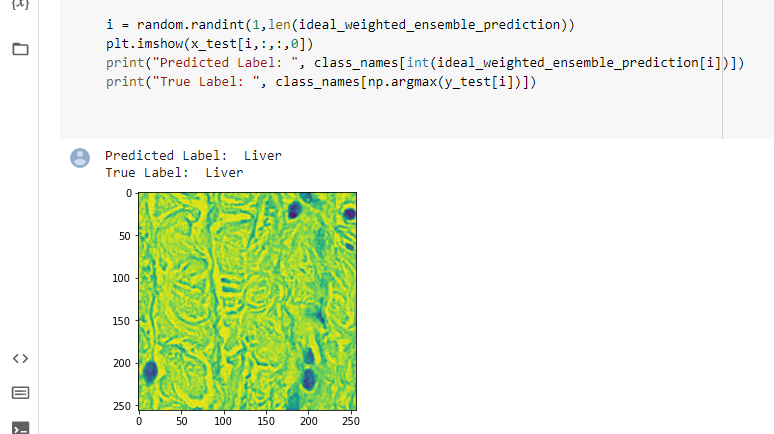
The Weighted Average: In the weighted average ensemble method, data scientists assign different weights to all the models in order to make a prediction, where the assigned weight defines the relevance of each model. As an example, let's assume that out of 100 people who gave feedback for your travel app, 70 are professional app developers, while the other 30 have no experience in app development. In this scenario, the weighted average ensemble technique will give more weight to the feedback of app developers compared to others.

Ensemble used for three models:



After ensembling I tried to predict an output:







CHAPTER 7: Problems

In this project while working with our dataset I faced Overfitting problem. In this chapter I will discuss it.

**Problem**

Overfitting is a concept in data science, which occurs when a statistical model fits exactly against its training data. When this happens, the model does extremely well against the training data. but unfortunately, cannot perform accurately later against unseen data.

It is recognized by:

Low error rates and a high variance are good indicators of overfitting.

K-fold cross-validation is one of the most popular techniques to assess accuracy of the model.

Another problem with deep learning algorithms is that they’re very good at mapping inputs to outputs but not so much at understanding the context of the data they’re handling. In fact, the word “deep” in deep learning is much more a reference to the architecture of the technology and the number of hidden layers it contains rather than an allusion to its deep understanding of what it does.

**How I solved it**

As our dataset in each fold is small (2.5 thousand), OVERFITTING would be a very common problem here.

To solve this problem, we needed Data Regularization.

We used:

* 1. More neurons in the fully connected Layer
  2. Dropout Layer (Regularization)
  3. Batch normalization
  4. Data Augmentation Increase the number of data



CHAPTER 8: Results

As I trained several deep learning models for my dataset, different models threw different results. A comparison and output are shown below.

For folder 1 the results are shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation  Accuracy | Testing  Accuracy | Epoch |
| ResNet50 | 99.90 | 82.16 | 76.88 | 50 |
| Vgg19 | 99.95 | 51.64 | 50.56 | 50 |
| Vgg16 | 99.95 | 57.75 | 51.69 | 50 |
| DenseNet169 | 99.53 | 82.63 | 83.08 | 50 |

After taking 70% of training data, 20% of testing data and 10 % of validation data we can see that the ResNet50 model gives us 99.90% accuracy for training and 82.16% validation accuracy. After 50epoch testing accuracy was 76.88% while evaluating the ResNet50 model.

Then I tried the vgg19 model. It gives 99.95% training accuracy, 51.64% validation accuracy after 50 epochs. While evaluating the model 51.64% accuracy was found. Which is less than the ResNet50 model.

Then I tried the vgg16 model. It gives 99.95% training accuracy, 57.75% validation accuracy after 50 epochs. While evaluating the model 51.69% accuracy was found. Which is less than the ResNet50 model.

Then I tried the DenseNet169 model. It gives 98.48% training accuracy, 78.40% validation accuracy after 50 epochs. While evaluating the model 78.20% accuracy was found. Which is little less than the ResNet50 model.

Then I tried to split the dataset into almost three equal parts. The percentage of the training, validation and testing data was 70, 10 and 20 percent. I took the weight of the three models after training from it. Then applied an ensemble method. Got the average and weighted avg. ensemble accuracy for testing data.

The table below shows the performance of the ensemble method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation Accuracy | Testing Accuracy | epoch |
| ResNet50 | 99.06 | 59.72 | 64.04 | 50 |
| Vgg19 | 100.00 | 38.20 | 39.75 | 45 |
| Vgg16 | 100.00 | 54.55 | 47.47 | 45 |

Ensemble ResNet50, Vgg19, Vgg16

|  |  |  |
| --- | --- | --- |
| Model Ensembled | Avg Ensemble | Weighted Avg. Ensemble |
| ResNet50, Vgg19, Vgg16 | 83.83 | 86.46 |
| ResNet50, Vgg19 | 59.96 | 64.47 |
| ResNet50, Vgg16 | 87.21 | 79.13 |
| Vgg19, Vgg16 | 85.52 | 86.27 |
|  |  |  |

We can clearly see that in most of the cases the weighted avg. ensemble increased the accuracy.

For folder 2 the results are shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation  Accuracy | Testing  Accuracy | Epoch |
| ResNet50 | 99.89 | 72.28 | 76.44 | 40 |
| Vgg19 | 99.83 | 40.10 | 49.31 | 50 |
| Vgg16 | 99.89 | 43.07 | 49.11 | 50 |
| DenseNet | 99.06 | 76.73 | 80.99 | 40 |

After taking 70% of training data, 20% of testing data and 10 % of validation data we can see that the ResNet50 model gives us 99.89% accuracy for training and 72.28% validation accuracy. After 40 epoch testing accuracy was 76.44% while evaluating the ResNet50 model.

Then I tried the vgg19 model. It gives 99.83% training accuracy, 40.10% validation accuracy after 50 epochs. While evaluating the model 49.31% accuracy was found. Which is less than the ResNet50 model.

Then I tried the vgg16 model. It gives 99.89% training accuracy 43.07% validation accuracy after 50 epochs. While evaluating the model 49.11% accuracy was found. Which is less than the ResNet50 model.

Then I tried the DenseNet model. It gives 99.06% training accuracy, 76.73% validation accuracy after 50 epochs. While evaluating the model 80.99% accuracy was found. Which is a little better than the ResNet50 model.

Then I tried to split the dataset into almost three equal parts. The percentage of the training, validation and testing data was 70, 10 and 20 percent. I took the weight of the three models after training from it. Then applied an ensemble method. Got the average and weighted avg. ensemble accuracy for testing data.

The table below shows the performance of the ensemble method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation Accuracy | Testing Accuracy | epoch |
| ResNet50 | 98.85 | 52.94 | 63.31 | 50 |
| Vgg19 | 100.00 | 43.53 | 40.79 | 45 |
| Vgg16 | 99.83 | 45.24 | 45.33 | 45 |

Ensemble ResNet50, Vgg19, Vgg16

|  |  |  |
| --- | --- | --- |
| Model Ensembled | Avg Ensemble | Weighted Avg. Ensemble |
| ResNet50, Vgg19, Vgg16 | 79.20 | 82.97 |
| ResNet50, Vgg19 | 57.62 | 60.19 |
| ResNet50, Vgg16 | 83.56 | 83.36 |
| Vgg19, Vgg16 | 81.18 | 82.97 |

We can clearly see that in most of the cases the weighted avg. ensemble increased the accuracy.

For folder 3 the results are shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation  Accuracy | Testing  Accuracy | Epoch |
| ResNet50 | 99.29 | 78.90 | 73.58 | 40 |
| Vgg19 | 99.54 | 47.25 | 49.91 | 50 |
| Vgg16 | 99.85 | 50.00 | 48.62 | 50 |
| DenseNet | 98.47 | 83.49 | 80.92 | 50 |

After taking 70% of training data, 20% of testing data and 10 % of validation data we can see that the ResNet50 model gives us 99.29% accuracy for training and 78.90% validation accuracy. After 40 epoch testing accuracy was 73.58% while evaluating the ResNet50 model.

Then I tried the vgg19 model. It gives 99.54% training accuracy, 47.25% validation accuracy after 50 epochs. While evaluating the model 49.91% accuracy was found. Which is less than the ResNet50 model.

Then I tried the vgg16 model. It gives 99.85% training accuracy 50.00% validation accuracy after 50 epochs. While evaluating the model 48.62% accuracy was found. Which is less than the ResNet50 model.

Then I tried the DenseNet model. It gives 98.52% training accuracy, 79.27% validation accuracy after 50 epochs. While evaluating the model 79.27% accuracy was found. Which is a little better than the ResNet50 model.

Then I tried to split the dataset into almost three equal parts. The percentage of the training, validation and testing data was 70, 10 and 20 percent. I took the weight of the three models after training from it. Then applied an ensemble method. Got the average and weighted avg. ensemble accuracy for testing data.

The table below shows the performance of the ensemble method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation Accuracy | Testing Accuracy | epoch |
| ResNet50 | 99.54 | 60.27 | 66.67 | 50 |
| Vgg19 | 100.00 | 41.30 | 41.46 | 45 |
| Vgg16 | 99.85 | 44.44 | 45.06 | 45 |

Ensemble ResNet50, Vgg19, Vgg16

|  |  |  |
| --- | --- | --- |
| Model Ensembled | Avg Ensemble | Weighted Avg. Ensemble |
| ResNet50, Vgg19, Vgg16 | 82.75 | 85.13 |
| ResNet50, Vgg19 | 60.00 | 61.28 |
| ResNet50, Vgg16 | 86.42 | 74.31 |
| Vgg19, Vgg16 | 84.58 | 84.95 |

We can clearly see that in most of the cases the weighted avg. ensemble increased the accuracy.

CHAPTER 9: Discussion

My project was on general cancer classification. Till now, I saw lots of papers on different types of specific cancer classification like breast, lung, brain, skin. But my project is a start of Combination of 19 different types of cancer in one dataset. As my dataset was divided into 3 folds, I trained 3 folds of data separately. At the beginning, I got really low validation accuracy which indicated overfitting. Overfitting means our data was high in variance. So, I needed to find the sweet spot between high variance and high bias which I did through data augmentation. I also tried batch normalization, tuned hyper parameters. So, this confirms that more data can provide better accuracy. I observed that the accuracy metrics SGD on our dataset performs better. In the accuracy curve, the training accuracy curve was static but the validation accuracy curve was quite chaotic because of some noise in it.

As it was a generalized cancer project, I have to go through different types of individual cancer classification for research. They were skin, breast, brain, and lung. In our project the dataset we got was in a numpy file, I resized the file, and the classification was done on that file. But previously in most of the classification problems, the classification was done on images from directory in jpg/png/mpeg/jepg etc format.

In my results, I only concentrated on accuracy to evaluate our results. But in case of classification, accuracy is not everything. There are some other criteria on which models are evaluated like precision, recall, F1 score, AUC curve, confusion matrix etc. Precision and recall are two extremely important evaluation metrics. Precision means the percentage of the results which are relevant. On the other hand, recall refers to the percentage of total relevant results correctly classified by the algorithm. I could either give a higher priority to maximizing precision, or recall but not both. Because, increasing one, decreases the other. There is one balanced spot between precision and recall which is F1 score which is a harmonic mean of precision and recall. Confusion matrices demonstrate visual display of how correctly the model has classified the classes. With this I can see how many samples are classified classes correctly and what and how many classes are misclassified as which classes. As the size of the dataset was quite large, it was really difficult to train the dataset on personal level computer systems. I used the default batch size while training in google colab pro.

For further research on our project, the semantic segmentation part can be implemented. Semantic segmentation is the process of classifying each pixel belonging to a particular class/label. It doesn't different across different instances of the same object. With this, it can be pointed out the exact place in the image where cancer happened. Our results can also be implemented for practice purposes. For example, if an app is developed which takes in suspected cancer images for general people as users, they can get the initial idea if they have cancer or not. If the app gives positive results for cancer, then they can consult professional doctors for further medical treatment. But if the result comes out negative, then they don't have to spend money and time only to find negative results.

For further implementation, we have few more plans-

1. When I have lots of data, then the neural network generalizes well. It’s better to have more data. When we get more data, I will try to increase accuracy more.

2. I can also move our work to segmentation.

3. I may also try to implement our project in an android application and web application.

4. I can do the AB testing for better performance. It will help the algorithm to fetch real data and experience.

N.B: A/B Testing is a widely used concept in most industries nowadays, and data scientists are at the forefront of implementing it. A/B testing is a basic randomized control experiment. It is a way to compare the two versions of a variable to find out which performs better in a controlled environment.



Chapter 10: Conclusion

In this project I wanted to classify multi-class cancer cells using convolution neural networks. I have achieved a better score using the ResNet50 model on a given dataset. Mainly in this project I observed how a deep learning model works, the ins and outs of the multi-class classification as well as architecture of VGG16, VGG19, ResNet50 and DenseNet159. This knowledge helps me to do further work in this sector

Now, my future plan for this project.

1. Collecting more data.

2. Secondly, I can also generalize the training data for better performance.

3. I can also move the work to segmentation. And, finally

4. The A/B testing

I have equally worked on this project, and tried my best to get the outcome of the task and achieved a good performance. I hope to achieve better results on remaining work in future.

Chapter 11: References

|  |  |
| --- | --- |
| [1] | "VGG16 – Convolutional Network for Classification and Detection" [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/ |
| [2] | "Understanding the VGG19 Architecture" [Online]. Available: https://iq.opengenus.org/vgg19-architecture/ |
| [3] | "Inceptionv3" [Online]. Available: https://en.wikipedia.org/wiki/Inceptionv3 |
| [4] | https:/[/www.python.org/doc/essays/blurb/.](http://www.python.org/doc/essays/blurb/) [Accessed 16 July 2021]. |
| [5] | "tensorflow," [Online]. Available: https://[www.tensorflow.org/federated.](http://www.tensorflow.org/federated) [Accessed 16 July 2021]. |
| [6] | E. Novoseltseva, "apiumhub," 30 November 2017. [Online]. Available: https://apiumhub.com/tech- blog-barcelona/using-github/. [Accessed 16 July 2021]. |
| [7] | M. Driscoll, "realpython," [Online]. Available: https://realpython.com/jupyter-notebook- introduction/#reader-comments. [Accessed 16 July 2021]. |
| [8] | "anaconda," [Online]. Available: https://docs.anaconda.com/anaconda/navigator/index.html. [Accessed 16 July 2021]. |
| [9] | "tutorialspoint," [Online]. Available: https://[www.tutorialspoint.com/keras/keras\_models.htm.](http://www.tutorialspoint.com/keras/keras_models.htm) [Accessed 16 July 2021]. |
| [10] | M. MAITHANI, "analyticsindiamag," 18 January 2021. [Online]. Available: https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/. [Accessed 16 July 2021]. |
| [11] | J. A. F. C. Daniel Falbel, "keras.rstudio," [Online]. Available: https://keras.rstudio.com/articles/about\_keras\_layers.html. |
| [12] | adnanirshad158, "geeksforgeeks," 23 June 2021. [Online]. Available: https:/[/www.geeksforgeeks.org/python-keras-keras-utils-to\_categorical/.](http://www.geeksforgeeks.org/python-keras-keras-utils-to_categorical/) [Accessed 16 July 2021]. |
| [13] | A. Duong, "kdnuggets," August 2019. [Online]. Available: https://[www.kdnuggets.com/2019/08/keras-callbacks-explained-three-minutes.html.](http://www.kdnuggets.com/2019/08/keras-callbacks-explained-three-minutes.html) |
| [14] | "tutorialspoint," [Online]. Available: https:/[/www.tutorialspoint.com/keras/keras\_applications.htm.](http://www.tutorialspoint.com/keras/keras_applications.htm) [Accessed 16 July 2021]. |
| [15] | "[http://ethen8181.github.io,](http://ethen8181.github.io/)" [Online]. Available: <http://ethen8181.github.io/machine-> learning/model\_selection/model\_selection.html. Accessed 16 July 2021]. |
| [16] | "activestate," [Online]. Available: https://[www.activestate.com/resources/quick-reads/what-is-](http://www.activestate.com/resources/quick-reads/what-is-) pyplot-in-matplotlib/. [Accessed 16 July 2021]. |
| [17] | J. Zhang, "towardsdatascience," U-net, 18 Oct 2019. [Online]. Available: https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5. [Accessed 17 July 2021]. |
| [18] | JSTOR, "Wikipedia," Data Augmentation, 01 September 2020. [Online]. Available: https://en.wikipedia.org/wiki/Data\_augmentation. [Accessed 17 July 2021]. |
| [19] | S. Sinha, "Geeksforgeeks," OpenCV, 29 April 2019. [Online]. Available: https:/[/www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/.](http://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/) [Accessed 17 July 2021]. |
| [20] | J. Brownlee, "Machine Learning Mastery," Deep Learning Performance, 3 July 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed 17 July 2021]. |