EE904 Course Project Report

**Q3-2023-24**

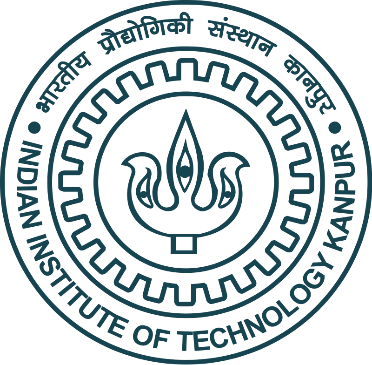
Detection of Objects Behind a Wall Using Deep Learning Techniques from RADAR Signal Imagery

Venkateswar Reddy Melachervu

A report submitted in part fulfilment of EE-904 course for

**Masters in Next Generation Wireless Technologies**

**Supervision:** Prof.Tushar Sandhan



Department of Electrical Engineering

IIT, Kanpur, India

April 02, 2024

**Declaration**

This report has been prepared based on my programming work to investigate the RADAR signal imagery for accurate detection of objects behind a wall. All published and unpublished source materials that have been used in this report have been acknowledged.

**Student Name: Venkateswar Reddy Melachervu**

**Date of Submission: 10th Mar 2024**

**Signature:**



**Abstract**

The project endeavours to explore the efficacy of RADAR signal analysis in accurately detecting objects situated behind a wall, with classifications including empty space, stationary objects, and moving entities using deep learning techniques. Through rigorous investigation, various methodologies are employed, ranging from manual classification to advanced deep learning techniques.

Initial analysis involves the examination of RADAR patterns extracted from samples of each class, aiding in the manual classification process. A subset of samples is randomly selected, mixed, and manually classified to assess human classification accuracy, thereby providing insights into potential challenges and ambiguities in discerning RADAR signals.

The project progresses to utilize machine learning models, starting with a multilayer perceptron (MLP) for training and testing. Detailed metrics including accuracy, training time, and accuracy graphs are reported, shedding light on the performance of the MLP approach.

Further exploration is conducted with the implementation of a convolutional neural network (CNN) architecture, initially with a single hidden layer. The training process is monitored with accuracy versus epoch graphs, alongside model details such as the total number of learnable parameters and training time per epoch. Evaluation of the trained CNN on a separate test dataset is performed, with accuracy metrics and confusion matrices generated to assess classification performance.

The experimentation extends to CNN architectures with multiple hidden layers, where the number of layers and kernels is systematically varied. Observations are meticulously recorded, providing insights into the impact of architectural choices on classification accuracy and computational efficiency.

Additionally, the project leverages a pretrained ResNet model to evaluate its suitability for RADAR signal analysis tasks. Results and observations obtained from this evaluation contribute to the broader understanding of the applicability of transfer learning in RADAR-based object detection.

This report contains the code developed for this project, screenshots, images, graphs, accuracy tables, confusion matrices, and summary of findings. Through this endeavour, a deeper understanding of RADAR signal analysis and its potential for object detection behind obstacles is attained, paving the way for future advancements in this domain.

Table of Contents

[Abstract 3](#_Toc162783125)

[Acknowledgements 5](#_Toc162783126)

[Project Objectives 6](#_Toc162783127)

[Manual Pattern Analysis 8](#_Toc162783128)

[Manual Classification Accuracy 10](#_Toc162783129)

[MLP-based Model-1 Training Accuracy 11](#_Toc162783130)

[MLP-based Model-2 Accuracy 14](#_Toc162783131)

[Single Hidden Layer CNN-based Model-1 Accuracy and Performance 17](#_Toc162783132)

[Single Hidden Layer CNN-based Model-2 Test Accuracy and Performance 20](#_Toc162783133)

[Multiple Hidden Layer CNN-based Model-1 Test Accuracy and Performance 23](#_Toc162783134)

[Multiple Hidden Layer CNN-based Model-2 Test Accuracy and Performance 26](#_Toc162783135)

[Multiple Hidden Layer CNN-based Model-3 Test Accuracy and Performance 29](#_Toc162783136)

[Multiple Hidden Layer CNN-based Model-4 Test Accuracy and Performance 32](#_Toc162783137)

[Pre-trained ResNet Model Test Accuracy and Performance 35](#_Toc162783138)

[Summary of Findings 47](#_Toc162783139)

[References and Bibliography 48](#_Toc162783140)

[Source Code – Google Colab 49](#_Toc162783141)

Acknowledgements

I am sincerely grateful to Professor Tushar Sandhan for his invaluable guidance, encouragement, and support throughout the duration of this project.

Professor Sandhan's expertise in the field of radar image classification has been instrumental in shaping the direction of our research and providing invaluable insights at every stage of the project.

His dedication to fostering intellectual curiosity and his unwavering commitment to excellence have inspired and motivated me to strive for the highest standards of academic and research excellence.

I am deeply appreciative of Professor Sandhan's mentorship, constructive feedback, and encouragement, which have been instrumental in the successful completion of this project.

I am also thankful for his willingness to share his knowledge and expertise generously, which has enriched my learning experience and contributed significantly to my personal and professional growth.

Project Objectives

The main purpose of this project is to investigate RADAR signal imagery (RADAR 2D patterns or crudely called as RADAR image) of objects present behind the wall (empty, still or walking) using deep learning techniques for accurate detection and write a detailed project report for the investigation conducted using deep learning techniques - including graphs, plots, snapshots, images, programming code and quantitative results obtained using the deep neural network models used.

This project uses the dataset[[1]](#footnote-1) that comprises of RADAR signal imagery samples collected through a concrete wall scan, aimed at analysing objects or events occurring behind obstacles. With profound implications in military operations and surveillance, this dataset serves as a valuable resource for investigating RADAR signal analysis. The dataset is organized into 'train' and 'test' folders, with each containing subfolders representing different scenarios:

* Empty: Samples collected when no object is present behind the wall.
* Still: Samples captured when a person is stationary behind the wall.
* Walking: Samples recorded when a person is walking behind the wall.

For the project's objectives, the 'train' folder samples are utilized for training deep neural network models, while the performance of these models is evaluated using the 'test' folder samples. This approach ensures a comprehensive assessment of the model's generalization capabilities beyond the training data.

This investigation uses Google Colab, a cloud-based Jupyter notebook environment provided by Google with Python language. Google Colab was utilized for its flexibility, accessibility, and compatibility with various programming languages such as Python, used in this project. Google Colab offers free access to computational resources, including GPU acceleration, which is particularly beneficial for training deep learning models efficiently. Moreover, Colab integrates seamlessly with Google Drive, allowing for easy storage and sharing of datasets, code, and project files. Its collaborative features enable multiple users to work simultaneously on a project, facilitating teamwork and knowledge sharing. Overall, Google Colab provides a convenient and powerful platform for conducting data analysis, implementing machine learning algorithms, and generating comprehensive project reports.

The investigation analyses the below related tasks and summarizes the findings in report in the ensuing sections.

1. Plots 5 random images from each class on a single row and details an intuitive human analysis of imagery patterns to manually classify any new sample into one of the three classes - Empty, Still, Walking
2. Selects 10 samples randomly from each class, mixes them all up for manual classification and creates a report on the correct classification and difficulties faced for in-correct classification.
3. Uses non-CNN MLP NN for training and testing and creates a report on accuracy, training time, accuracy graphs.
4. Designs and uses a CNN with 1 hidden layer and trains it on provided training data and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.
5. Evaluates the above trained CNN model with the test data set and creates a report on accuracy and plots confusion matrix.
6. Designs CNN with multiple hidden layers and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.
7. Changes number of hidden layers, number of kernels and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.
8. Uses pre-trained ResNet model and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.

Manual Pattern Analysis

This section presents an intuitive human pattern analysis of 5 random training samples from each class (Empty, Still, and Walking) to help manually classify any new sample into one of the classes – Empty, Still, and Walking.

Five random images of each object class (Empty, Still, Walking) from the training data set are presented below.

A group of gray rectangular objects

Description automatically generated with medium confidence

A group of gray and white lines

Description automatically generated with medium confidence

A screenshot of a test

Description automatically generated

Manual observations of the pattens for these image classes are detailed below.

* The empty class images have:
  + Pronounced smaller and random textures at the top and wanes down the in the centre and stars increasing in density further down to the bottom.
  + The textures slowly transform into ridges appearing dense at the bottom in some of these images.
  + The ridges appear to be like a mesh/grid pattern.
* The still class images have:
  + Sparse specks around the top with smoother uniform surface running down into the centre.
  + Repeated wavy textures around the centre waning down in intensity towards the bottom with abruptly increased intensity around the bottom.
* The walking class images have:
  + Sparse ridges just beneath the top with smoother uniform surface running down into the centre.
  + Repeated high intensity wavy textures around the centre waning down in intensity towards the bottom with prominent ridges with abruptly increased higher intensity around the bottom.

Manual Classification Accuracy

Ten random samples from each class are selected, mixed up and each of these images is manually classified into one of the three classes based on the manual pattern analysis observations of training samples from the previous section.

This exercise along with the results, accuracy of manual classification, and the difficulties faced are presented in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Image File Name | Identified Class | Actual Class | Difficulty Faced | Manual Classification Accuracy |
| 100\_32\_30\_15.jpg | Empty | Empty |  | 93.33% |
| 100\_32\_33\_34.jpg | Empty | Empty |  |
| 100\_32\_34\_14.jpg | Empty | Empty |  |
| 100\_32\_35\_27.jpg | Empty | Empty |  |
| 100\_32\_3\_35.jpg | Walking | Empty | Challenging to distinguish the patterns |
| 100\_32\_12\_35.jpg | Empty | Empty |  |
| 100\_32\_21\_28.jpg | Empty | Empty |  |
| 100\_32\_24\_33.jpg | Empty | Empty |  |
| 100\_32\_27\_37.jpg | Empty | Empty |  |
| 100\_32\_28\_35.jpg | Empty | Empty |  |
| 010\_32\_48\_20.jpg | Still | Still |  |
| 010\_32\_5\_6.jpg | Still | Still |  |
| 010\_32\_5\_7.jpg | Still | Still |  |
| 010\_32\_12\_5.jpg | Still | Still |  |
| 010\_32\_20\_30.jpg | Still | Still |  |
| 010\_32\_21\_33.jpg | Still | Still |  |
| 010\_32\_29\_25.jpg | Still | Still |  |
| 010\_32\_32\_23.jpg | Still | Still |  |
| 010\_32\_40\_26.jpg | Still | Still |  |
| 010\_32\_41\_24.jpg | Walking | Still | Challenging to distinguish the patterns |
| 001\_32\_48\_23.jpg | Walking | Walking |  |
| 001\_32\_1\_22.jpg | Walking | Walking |  |
| 001\_32\_8\_32.jpg | Walking | Walking |  |
| 001\_32\_15\_22.jpg | Walking | Walking |  |
| 001\_32\_16\_35.jpg | Walking | Walking |  |
| 001\_32\_23\_29.jpg | Walking | Walking |  |
| 001\_32\_31\_4.jpg | Walking | Walking |  |
| 001\_32\_32\_22.jpg | Walking | Walking |  |
| 001\_32\_38\_26.jpg | Walking | Walking |  |
| 001\_32\_42\_2.jpg | Walking | Walking |  |

MLP-based Model-1 Training Accuracy

Deployed Google Colab runtime for this classification:

* Machine: T4 GPU – Google Cloud/Colab
* System RAM - Available - 12.7 GB
* GPU RAM Available – 15.0 GB
* Disk - Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white screen with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with black text

Description automatically generated

A white background with black and white text

Description automatically generated

A graph with lines and text

Description automatically generated with medium confidence

A graph with lines and numbers

Description automatically generated

A graph with a line

Description automatically generated

MLP-based Model-2 Accuracy

Deployed Google Colab runtime for this classification:

* Machine: T4 GPU – Google Cloud/Colab
* System RAM - Available - 12.7 GB
* GPU RAM Available – 15.0 GB
* Disk - Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A group of text on a white background

Description automatically generated

A graph with lines and numbers

Description automatically generated with medium confidence

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

Single Hidden Layer CNN-based Model-1 Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: T4 GPU – Google Cloud/Colab
* System RAM - Available - 12.7 GB
* GPU RAM Available – 15.0 GB
* Disk - Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with black and white text

Description automatically generated

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a computer

Description automatically generated

Single Hidden Layer CNN-based Model-2 Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: T4 GPU – Google Cloud/Colab
* System RAM - Available - 12.7 GB
* GPU RAM Available – 15.0 GB
* Disk - Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A close up of a number

Description automatically generated with medium confidence

A graph with lines and numbers

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Multiple Hidden Layer CNN-based Model-1 Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: CPU
* System RAM - Available - 12.7 GB
* Disk - Available – 107.7GB
* Python 3 Google Compute Engine Backend

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A group of text on a white background

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a graph

Description automatically generated

Multiple Hidden Layer CNN-based Model-2 Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: CPU
* System RAM - Available - 12.7 GB
* Disk - Available – 107.7GB
* Python 3 Google Compute Engine Backend

A white text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a computer

Description automatically generated

Multiple Hidden Layer CNN-based Model-3 Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: V100
* System RAM - Available - 12.7 GB
* GPU RAM – available – 16 GB
* Disk Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a graph

Description automatically generated

Multiple Hidden Layer CNN-based Model-4 Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: V100
* System RAM - Available - 12.7 GB
* GPU RAM – available – 16 GB
* Disk Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a graph

Description automatically generated

Pre-trained ResNet Model Test Accuracy and Performance

Deployed Google Colab runtime for this classification:

* Machine: V100
* System RAM - Available - 12.7 GB
* GPU RAM – available – 16 GB
* Disk Available – 78.2 GB
* Python 3 Google Compute Engine Backend

A white text with black text

Description automatically generated

Length of the train data set is 44

Length of the validation data set is 44

Configuring ResNet50 with input shape of 32x768

Model: "ResNet50\_with\_Original\_Image\_Training\_GDV\_1"

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

Layer (type) Output Shape Param # Connected to

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

input\_6 (InputLayer) [(None, None, None, 3)] 0 []

conv1\_pad (ZeroPadding2D) (None, None, None, 3) 0 ['input\_6[0][0]']

conv1\_conv (Conv2D) (None, None, None, 64) 9472 ['conv1\_pad[0][0]']

conv1\_bn (BatchNormalizati (None, None, None, 64) 256 ['conv1\_conv[0][0]']

on)

conv1\_relu (Activation) (None, None, None, 64) 0 ['conv1\_bn[0][0]']

pool1\_pad (ZeroPadding2D) (None, None, None, 64) 0 ['conv1\_relu[0][0]']

pool1\_pool (MaxPooling2D) (None, None, None, 64) 0 ['pool1\_pad[0][0]']

conv2\_block1\_1\_conv (Conv2 (None, None, None, 64) 4160 ['pool1\_pool[0][0]']

D)

conv2\_block1\_1\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block1\_1\_conv[0][0]']

rmalization)

conv2\_block1\_1\_relu (Activ (None, None, None, 64) 0 ['conv2\_block1\_1\_bn[0][0]']

ation)

conv2\_block1\_2\_conv (Conv2 (None, None, None, 64) 36928 ['conv2\_block1\_1\_relu[0][0]']

D)

conv2\_block1\_2\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block1\_2\_conv[0][0]']

rmalization)

conv2\_block1\_2\_relu (Activ (None, None, None, 64) 0 ['conv2\_block1\_2\_bn[0][0]']

ation)

conv2\_block1\_0\_conv (Conv2 (None, None, None, 256) 16640 ['pool1\_pool[0][0]']

D)

conv2\_block1\_3\_conv (Conv2 (None, None, None, 256) 16640 ['conv2\_block1\_2\_relu[0][0]']

D)

conv2\_block1\_0\_bn (BatchNo (None, None, None, 256) 1024 ['conv2\_block1\_0\_conv[0][0]']

rmalization)

conv2\_block1\_3\_bn (BatchNo (None, None, None, 256) 1024 ['conv2\_block1\_3\_conv[0][0]']

rmalization)

conv2\_block1\_add (Add) (None, None, None, 256) 0 ['conv2\_block1\_0\_bn[0][0]',

'conv2\_block1\_3\_bn[0][0]']

conv2\_block1\_out (Activati (None, None, None, 256) 0 ['conv2\_block1\_add[0][0]']

on)

conv2\_block2\_1\_conv (Conv2 (None, None, None, 64) 16448 ['conv2\_block1\_out[0][0]']

D)

conv2\_block2\_1\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block2\_1\_conv[0][0]']

rmalization)

conv2\_block2\_1\_relu (Activ (None, None, None, 64) 0 ['conv2\_block2\_1\_bn[0][0]']

ation)

conv2\_block2\_2\_conv (Conv2 (None, None, None, 64) 36928 ['conv2\_block2\_1\_relu[0][0]']

D)

conv2\_block2\_2\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block2\_2\_conv[0][0]']

rmalization)

conv2\_block2\_2\_relu (Activ (None, None, None, 64) 0 ['conv2\_block2\_2\_bn[0][0]']

ation)

conv2\_block2\_3\_conv (Conv2 (None, None, None, 256) 16640 ['conv2\_block2\_2\_relu[0][0]']

D)

conv2\_block2\_3\_bn (BatchNo (None, None, None, 256) 1024 ['conv2\_block2\_3\_conv[0][0]']

rmalization)

conv2\_block2\_add (Add) (None, None, None, 256) 0 ['conv2\_block1\_out[0][0]',

'conv2\_block2\_3\_bn[0][0]']

conv2\_block2\_out (Activati (None, None, None, 256) 0 ['conv2\_block2\_add[0][0]']

on)

conv2\_block3\_1\_conv (Conv2 (None, None, None, 64) 16448 ['conv2\_block2\_out[0][0]']

D)

conv2\_block3\_1\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block3\_1\_conv[0][0]']

rmalization)

conv2\_block3\_1\_relu (Activ (None, None, None, 64) 0 ['conv2\_block3\_1\_bn[0][0]']

ation)

conv2\_block3\_2\_conv (Conv2 (None, None, None, 64) 36928 ['conv2\_block3\_1\_relu[0][0]']

D)

conv2\_block3\_2\_bn (BatchNo (None, None, None, 64) 256 ['conv2\_block3\_2\_conv[0][0]']

rmalization)

conv2\_block3\_2\_relu (Activ (None, None, None, 64) 0 ['conv2\_block3\_2\_bn[0][0]']

ation)

conv2\_block3\_3\_conv (Conv2 (None, None, None, 256) 16640 ['conv2\_block3\_2\_relu[0][0]']

D)

conv2\_block3\_3\_bn (BatchNo (None, None, None, 256) 1024 ['conv2\_block3\_3\_conv[0][0]']

rmalization)

conv2\_block3\_add (Add) (None, None, None, 256) 0 ['conv2\_block2\_out[0][0]',

'conv2\_block3\_3\_bn[0][0]']

conv2\_block3\_out (Activati (None, None, None, 256) 0 ['conv2\_block3\_add[0][0]']

on)

conv3\_block1\_1\_conv (Conv2 (None, None, None, 128) 32896 ['conv2\_block3\_out[0][0]']

D)

conv3\_block1\_1\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block1\_1\_conv[0][0]']

rmalization)

conv3\_block1\_1\_relu (Activ (None, None, None, 128) 0 ['conv3\_block1\_1\_bn[0][0]']

ation)

conv3\_block1\_2\_conv (Conv2 (None, None, None, 128) 147584 ['conv3\_block1\_1\_relu[0][0]']

D)

conv3\_block1\_2\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block1\_2\_conv[0][0]']

rmalization)

conv3\_block1\_2\_relu (Activ (None, None, None, 128) 0 ['conv3\_block1\_2\_bn[0][0]']

ation)

conv3\_block1\_0\_conv (Conv2 (None, None, None, 512) 131584 ['conv2\_block3\_out[0][0]']

D)

conv3\_block1\_3\_conv (Conv2 (None, None, None, 512) 66048 ['conv3\_block1\_2\_relu[0][0]']

D)

conv3\_block1\_0\_bn (BatchNo (None, None, None, 512) 2048 ['conv3\_block1\_0\_conv[0][0]']

rmalization)

conv3\_block1\_3\_bn (BatchNo (None, None, None, 512) 2048 ['conv3\_block1\_3\_conv[0][0]']

rmalization)

conv3\_block1\_add (Add) (None, None, None, 512) 0 ['conv3\_block1\_0\_bn[0][0]',

'conv3\_block1\_3\_bn[0][0]']

conv3\_block1\_out (Activati (None, None, None, 512) 0 ['conv3\_block1\_add[0][0]']

on)

conv3\_block2\_1\_conv (Conv2 (None, None, None, 128) 65664 ['conv3\_block1\_out[0][0]']

D)

conv3\_block2\_1\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block2\_1\_conv[0][0]']

rmalization)

conv3\_block2\_1\_relu (Activ (None, None, None, 128) 0 ['conv3\_block2\_1\_bn[0][0]']

ation)

conv3\_block2\_2\_conv (Conv2 (None, None, None, 128) 147584 ['conv3\_block2\_1\_relu[0][0]']

D)

conv3\_block2\_2\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block2\_2\_conv[0][0]']

rmalization)

conv3\_block2\_2\_relu (Activ (None, None, None, 128) 0 ['conv3\_block2\_2\_bn[0][0]']

ation)

conv3\_block2\_3\_conv (Conv2 (None, None, None, 512) 66048 ['conv3\_block2\_2\_relu[0][0]']

D)

conv3\_block2\_3\_bn (BatchNo (None, None, None, 512) 2048 ['conv3\_block2\_3\_conv[0][0]']

rmalization)

conv3\_block2\_add (Add) (None, None, None, 512) 0 ['conv3\_block1\_out[0][0]',

'conv3\_block2\_3\_bn[0][0]']

conv3\_block2\_out (Activati (None, None, None, 512) 0 ['conv3\_block2\_add[0][0]']

on)

conv3\_block3\_1\_conv (Conv2 (None, None, None, 128) 65664 ['conv3\_block2\_out[0][0]']

D)

conv3\_block3\_1\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block3\_1\_conv[0][0]']

rmalization)

conv3\_block3\_1\_relu (Activ (None, None, None, 128) 0 ['conv3\_block3\_1\_bn[0][0]']

ation)

conv3\_block3\_2\_conv (Conv2 (None, None, None, 128) 147584 ['conv3\_block3\_1\_relu[0][0]']

D)

conv3\_block3\_2\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block3\_2\_conv[0][0]']

rmalization)

conv3\_block3\_2\_relu (Activ (None, None, None, 128) 0 ['conv3\_block3\_2\_bn[0][0]']

ation)

conv3\_block3\_3\_conv (Conv2 (None, None, None, 512) 66048 ['conv3\_block3\_2\_relu[0][0]']

D)

conv3\_block3\_3\_bn (BatchNo (None, None, None, 512) 2048 ['conv3\_block3\_3\_conv[0][0]']

rmalization)

conv3\_block3\_add (Add) (None, None, None, 512) 0 ['conv3\_block2\_out[0][0]',

'conv3\_block3\_3\_bn[0][0]']

conv3\_block3\_out (Activati (None, None, None, 512) 0 ['conv3\_block3\_add[0][0]']

on)

conv3\_block4\_1\_conv (Conv2 (None, None, None, 128) 65664 ['conv3\_block3\_out[0][0]']

D)

conv3\_block4\_1\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block4\_1\_conv[0][0]']

rmalization)

conv3\_block4\_1\_relu (Activ (None, None, None, 128) 0 ['conv3\_block4\_1\_bn[0][0]']

ation)

conv3\_block4\_2\_conv (Conv2 (None, None, None, 128) 147584 ['conv3\_block4\_1\_relu[0][0]']

D)

conv3\_block4\_2\_bn (BatchNo (None, None, None, 128) 512 ['conv3\_block4\_2\_conv[0][0]']

rmalization)

conv3\_block4\_2\_relu (Activ (None, None, None, 128) 0 ['conv3\_block4\_2\_bn[0][0]']

ation)

conv3\_block4\_3\_conv (Conv2 (None, None, None, 512) 66048 ['conv3\_block4\_2\_relu[0][0]']

D)

conv3\_block4\_3\_bn (BatchNo (None, None, None, 512) 2048 ['conv3\_block4\_3\_conv[0][0]']

rmalization)

conv3\_block4\_add (Add) (None, None, None, 512) 0 ['conv3\_block3\_out[0][0]',

'conv3\_block4\_3\_bn[0][0]']

conv3\_block4\_out (Activati (None, None, None, 512) 0 ['conv3\_block4\_add[0][0]']

on)

conv4\_block1\_1\_conv (Conv2 (None, None, None, 256) 131328 ['conv3\_block4\_out[0][0]']

D)

conv4\_block1\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block1\_1\_conv[0][0]']

rmalization)

conv4\_block1\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block1\_1\_bn[0][0]']

ation)

conv4\_block1\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block1\_1\_relu[0][0]']

D)

conv4\_block1\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block1\_2\_conv[0][0]']

rmalization)

conv4\_block1\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block1\_2\_bn[0][0]']

ation)

conv4\_block1\_0\_conv (Conv2 (None, None, None, 1024) 525312 ['conv3\_block4\_out[0][0]']

D)

conv4\_block1\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block1\_2\_relu[0][0]']

D)

conv4\_block1\_0\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block1\_0\_conv[0][0]']

rmalization)

conv4\_block1\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block1\_3\_conv[0][0]']

rmalization)

conv4\_block1\_add (Add) (None, None, None, 1024) 0 ['conv4\_block1\_0\_bn[0][0]',

'conv4\_block1\_3\_bn[0][0]']

conv4\_block1\_out (Activati (None, None, None, 1024) 0 ['conv4\_block1\_add[0][0]']

on)

conv4\_block2\_1\_conv (Conv2 (None, None, None, 256) 262400 ['conv4\_block1\_out[0][0]']

D)

conv4\_block2\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block2\_1\_conv[0][0]']

rmalization)

conv4\_block2\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block2\_1\_bn[0][0]']

ation)

conv4\_block2\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block2\_1\_relu[0][0]']

D)

conv4\_block2\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block2\_2\_conv[0][0]']

rmalization)

conv4\_block2\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block2\_2\_bn[0][0]']

ation)

conv4\_block2\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block2\_2\_relu[0][0]']

D)

conv4\_block2\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block2\_3\_conv[0][0]']

rmalization)

conv4\_block2\_add (Add) (None, None, None, 1024) 0 ['conv4\_block1\_out[0][0]',

'conv4\_block2\_3\_bn[0][0]']

conv4\_block2\_out (Activati (None, None, None, 1024) 0 ['conv4\_block2\_add[0][0]']

on)

conv4\_block3\_1\_conv (Conv2 (None, None, None, 256) 262400 ['conv4\_block2\_out[0][0]']

D)

conv4\_block3\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block3\_1\_conv[0][0]']

rmalization)

conv4\_block3\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block3\_1\_bn[0][0]']

ation)

conv4\_block3\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block3\_1\_relu[0][0]']

D)

conv4\_block3\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block3\_2\_conv[0][0]']

rmalization)

conv4\_block3\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block3\_2\_bn[0][0]']

ation)

conv4\_block3\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block3\_2\_relu[0][0]']

D)

conv4\_block3\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block3\_3\_conv[0][0]']

rmalization)

conv4\_block3\_add (Add) (None, None, None, 1024) 0 ['conv4\_block2\_out[0][0]',

'conv4\_block3\_3\_bn[0][0]']

conv4\_block3\_out (Activati (None, None, None, 1024) 0 ['conv4\_block3\_add[0][0]']

on)

conv4\_block4\_1\_conv (Conv2 (None, None, None, 256) 262400 ['conv4\_block3\_out[0][0]']

D)

conv4\_block4\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block4\_1\_conv[0][0]']

rmalization)

conv4\_block4\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block4\_1\_bn[0][0]']

ation)

conv4\_block4\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block4\_1\_relu[0][0]']

D)

conv4\_block4\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block4\_2\_conv[0][0]']

rmalization)

conv4\_block4\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block4\_2\_bn[0][0]']

ation)

conv4\_block4\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block4\_2\_relu[0][0]']

D)

conv4\_block4\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block4\_3\_conv[0][0]']

rmalization)

conv4\_block4\_add (Add) (None, None, None, 1024) 0 ['conv4\_block3\_out[0][0]',

'conv4\_block4\_3\_bn[0][0]']

conv4\_block4\_out (Activati (None, None, None, 1024) 0 ['conv4\_block4\_add[0][0]']

on)

conv4\_block5\_1\_conv (Conv2 (None, None, None, 256) 262400 ['conv4\_block4\_out[0][0]']

D)

conv4\_block5\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block5\_1\_conv[0][0]']

rmalization)

conv4\_block5\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block5\_1\_bn[0][0]']

ation)

conv4\_block5\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block5\_1\_relu[0][0]']

D)

conv4\_block5\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block5\_2\_conv[0][0]']

rmalization)

conv4\_block5\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block5\_2\_bn[0][0]']

ation)

conv4\_block5\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block5\_2\_relu[0][0]']

D)

conv4\_block5\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block5\_3\_conv[0][0]']

rmalization)

conv4\_block5\_add (Add) (None, None, None, 1024) 0 ['conv4\_block4\_out[0][0]',

'conv4\_block5\_3\_bn[0][0]']

conv4\_block5\_out (Activati (None, None, None, 1024) 0 ['conv4\_block5\_add[0][0]']

on)

conv4\_block6\_1\_conv (Conv2 (None, None, None, 256) 262400 ['conv4\_block5\_out[0][0]']

D)

conv4\_block6\_1\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block6\_1\_conv[0][0]']

rmalization)

conv4\_block6\_1\_relu (Activ (None, None, None, 256) 0 ['conv4\_block6\_1\_bn[0][0]']

ation)

conv4\_block6\_2\_conv (Conv2 (None, None, None, 256) 590080 ['conv4\_block6\_1\_relu[0][0]']

D)

conv4\_block6\_2\_bn (BatchNo (None, None, None, 256) 1024 ['conv4\_block6\_2\_conv[0][0]']

rmalization)

conv4\_block6\_2\_relu (Activ (None, None, None, 256) 0 ['conv4\_block6\_2\_bn[0][0]']

ation)

conv4\_block6\_3\_conv (Conv2 (None, None, None, 1024) 263168 ['conv4\_block6\_2\_relu[0][0]']

D)

conv4\_block6\_3\_bn (BatchNo (None, None, None, 1024) 4096 ['conv4\_block6\_3\_conv[0][0]']

rmalization)

conv4\_block6\_add (Add) (None, None, None, 1024) 0 ['conv4\_block5\_out[0][0]',

'conv4\_block6\_3\_bn[0][0]']

conv4\_block6\_out (Activati (None, None, None, 1024) 0 ['conv4\_block6\_add[0][0]']

on)

conv5\_block1\_1\_conv (Conv2 (None, None, None, 512) 524800 ['conv4\_block6\_out[0][0]']

D)

conv5\_block1\_1\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block1\_1\_conv[0][0]']

rmalization)

conv5\_block1\_1\_relu (Activ (None, None, None, 512) 0 ['conv5\_block1\_1\_bn[0][0]']

ation)

conv5\_block1\_2\_conv (Conv2 (None, None, None, 512) 2359808 ['conv5\_block1\_1\_relu[0][0]']

D)

conv5\_block1\_2\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block1\_2\_conv[0][0]']

rmalization)

conv5\_block1\_2\_relu (Activ (None, None, None, 512) 0 ['conv5\_block1\_2\_bn[0][0]']

ation)

conv5\_block1\_0\_conv (Conv2 (None, None, None, 2048) 2099200 ['conv4\_block6\_out[0][0]']

D)

conv5\_block1\_3\_conv (Conv2 (None, None, None, 2048) 1050624 ['conv5\_block1\_2\_relu[0][0]']

D)

conv5\_block1\_0\_bn (BatchNo (None, None, None, 2048) 8192 ['conv5\_block1\_0\_conv[0][0]']

rmalization)

conv5\_block1\_3\_bn (BatchNo (None, None, None, 2048) 8192 ['conv5\_block1\_3\_conv[0][0]']

rmalization)

conv5\_block1\_add (Add) (None, None, None, 2048) 0 ['conv5\_block1\_0\_bn[0][0]',

'conv5\_block1\_3\_bn[0][0]']

conv5\_block1\_out (Activati (None, None, None, 2048) 0 ['conv5\_block1\_add[0][0]']

on)

conv5\_block2\_1\_conv (Conv2 (None, None, None, 512) 1049088 ['conv5\_block1\_out[0][0]']

D)

conv5\_block2\_1\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block2\_1\_conv[0][0]']

rmalization)

conv5\_block2\_1\_relu (Activ (None, None, None, 512) 0 ['conv5\_block2\_1\_bn[0][0]']

ation)

conv5\_block2\_2\_conv (Conv2 (None, None, None, 512) 2359808 ['conv5\_block2\_1\_relu[0][0]']

D)

conv5\_block2\_2\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block2\_2\_conv[0][0]']

rmalization)

conv5\_block2\_2\_relu (Activ (None, None, None, 512) 0 ['conv5\_block2\_2\_bn[0][0]']

ation)

conv5\_block2\_3\_conv (Conv2 (None, None, None, 2048) 1050624 ['conv5\_block2\_2\_relu[0][0]']

D)

conv5\_block2\_3\_bn (BatchNo (None, None, None, 2048) 8192 ['conv5\_block2\_3\_conv[0][0]']

rmalization)

conv5\_block2\_add (Add) (None, None, None, 2048) 0 ['conv5\_block1\_out[0][0]',

'conv5\_block2\_3\_bn[0][0]']

conv5\_block2\_out (Activati (None, None, None, 2048) 0 ['conv5\_block2\_add[0][0]']

on)

conv5\_block3\_1\_conv (Conv2 (None, None, None, 512) 1049088 ['conv5\_block2\_out[0][0]']

D)

conv5\_block3\_1\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block3\_1\_conv[0][0]']

rmalization)

conv5\_block3\_1\_relu (Activ (None, None, None, 512) 0 ['conv5\_block3\_1\_bn[0][0]']

ation)

conv5\_block3\_2\_conv (Conv2 (None, None, None, 512) 2359808 ['conv5\_block3\_1\_relu[0][0]']

D)

conv5\_block3\_2\_bn (BatchNo (None, None, None, 512) 2048 ['conv5\_block3\_2\_conv[0][0]']

rmalization)

conv5\_block3\_2\_relu (Activ (None, None, None, 512) 0 ['conv5\_block3\_2\_bn[0][0]']

ation)

conv5\_block3\_3\_conv (Conv2 (None, None, None, 2048) 1050624 ['conv5\_block3\_2\_relu[0][0]']

D)

conv5\_block3\_3\_bn (BatchNo (None, None, None, 2048) 8192 ['conv5\_block3\_3\_conv[0][0]']

rmalization)

conv5\_block3\_add (Add) (None, None, None, 2048) 0 ['conv5\_block2\_out[0][0]',

'conv5\_block3\_3\_bn[0][0]']

conv5\_block3\_out (Activati (None, None, None, 2048) 0 ['conv5\_block3\_add[0][0]']

on)

global\_average\_pooling2d\_5 (None, 2048) 0 ['conv5\_block3\_out[0][0]']

(GlobalAveragePooling2D)

dense\_10 (Dense) (None, 1024) 2098176 ['global\_average\_pooling2d\_5[0

][0]']

dense\_11 (Dense) (None, 3) 3075 ['dense\_10[0][0]']

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

Total params: 25688963 (98.00 MB)

Trainable params: 2101251 (8.02 MB)

Non-trainable params: 23587712 (89.98 MB)

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

Activation Functions:

input\_6 - No activation

conv1\_pad - No activation

conv1\_conv - linear

conv1\_bn - No activation

conv1\_relu - relu

pool1\_pad - No activation

pool1\_pool - No activation

conv2\_block1\_1\_conv - linear

conv2\_block1\_1\_bn - No activation

conv2\_block1\_1\_relu - relu

conv2\_block1\_2\_conv - linear

conv2\_block1\_2\_bn - No activation

conv2\_block1\_2\_relu - relu

conv2\_block1\_0\_conv - linear

conv2\_block1\_3\_conv - linear

conv2\_block1\_0\_bn - No activation

conv2\_block1\_3\_bn - No activation

conv2\_block1\_add - No activation

conv2\_block1\_out - relu

conv2\_block2\_1\_conv - linear

conv2\_block2\_1\_bn - No activation

conv2\_block2\_1\_relu - relu

conv2\_block2\_2\_conv - linear

conv2\_block2\_2\_bn - No activation

conv2\_block2\_2\_relu - relu

conv2\_block2\_3\_conv - linear

conv2\_block2\_3\_bn - No activation

conv2\_block2\_add - No activation

conv2\_block2\_out - relu

conv2\_block3\_1\_conv - linear

conv2\_block3\_1\_bn - No activation

conv2\_block3\_1\_relu - relu

conv2\_block3\_2\_conv - linear

conv2\_block3\_2\_bn - No activation

conv2\_block3\_2\_relu - relu

conv2\_block3\_3\_conv - linear

conv2\_block3\_3\_bn - No activation

conv2\_block3\_add - No activation

conv2\_block3\_out - relu

conv3\_block1\_1\_conv - linear

conv3\_block1\_1\_bn - No activation

conv3\_block1\_1\_relu - relu

conv3\_block1\_2\_conv - linear

conv3\_block1\_2\_bn - No activation

conv3\_block1\_2\_relu - relu

conv3\_block1\_0\_conv - linear

conv3\_block1\_3\_conv - linear

conv3\_block1\_0\_bn - No activation

conv3\_block1\_3\_bn - No activation

conv3\_block1\_add - No activation

conv3\_block1\_out - relu

conv3\_block2\_1\_conv - linear

conv3\_block2\_1\_bn - No activation

conv3\_block2\_1\_relu - relu

conv3\_block2\_2\_conv - linear

conv3\_block2\_2\_bn - No activation

conv3\_block2\_2\_relu - relu

conv3\_block2\_3\_conv - linear

conv3\_block2\_3\_bn - No activation

conv3\_block2\_add - No activation

conv3\_block2\_out - relu

conv3\_block3\_1\_conv - linear

conv3\_block3\_1\_bn - No activation

conv3\_block3\_1\_relu - relu

conv3\_block3\_2\_conv - linear

conv3\_block3\_2\_bn - No activation

conv3\_block3\_2\_relu - relu

conv3\_block3\_3\_conv - linear

conv3\_block3\_3\_bn - No activation

conv3\_block3\_add - No activation

conv3\_block3\_out - relu

conv3\_block4\_1\_conv - linear

conv3\_block4\_1\_bn - No activation

conv3\_block4\_1\_relu - relu

conv3\_block4\_2\_conv - linear

conv3\_block4\_2\_bn - No activation

conv3\_block4\_2\_relu - relu

conv3\_block4\_3\_conv - linear

conv3\_block4\_3\_bn - No activation

conv3\_block4\_add - No activation

conv3\_block4\_out - relu

conv4\_block1\_1\_conv - linear

conv4\_block1\_1\_bn - No activation

conv4\_block1\_1\_relu - relu

conv4\_block1\_2\_conv - linear

conv4\_block1\_2\_bn - No activation

conv4\_block1\_2\_relu - relu

conv4\_block1\_0\_conv - linear

conv4\_block1\_3\_conv - linear

conv4\_block1\_0\_bn - No activation

conv4\_block1\_3\_bn - No activation

conv4\_block1\_add - No activation

conv4\_block1\_out - relu

conv4\_block2\_1\_conv - linear

conv4\_block2\_1\_bn - No activation

conv4\_block2\_1\_relu - relu

conv4\_block2\_2\_conv - linear

conv4\_block2\_2\_bn - No activation

conv4\_block2\_2\_relu - relu

conv4\_block2\_3\_conv - linear

conv4\_block2\_3\_bn - No activation

conv4\_block2\_add - No activation

conv4\_block2\_out - relu

conv4\_block3\_1\_conv - linear

conv4\_block3\_1\_bn - No activation

conv4\_block3\_1\_relu - relu

conv4\_block3\_2\_conv - linear

conv4\_block3\_2\_bn - No activation

conv4\_block3\_2\_relu - relu

conv4\_block3\_3\_conv - linear

conv4\_block3\_3\_bn - No activation

conv4\_block3\_add - No activation

conv4\_block3\_out - relu

conv4\_block4\_1\_conv - linear

conv4\_block4\_1\_bn - No activation

conv4\_block4\_1\_relu - relu

conv4\_block4\_2\_conv - linear

conv4\_block4\_2\_bn - No activation

conv4\_block4\_2\_relu - relu

conv4\_block4\_3\_conv - linear

conv4\_block4\_3\_bn - No activation

conv4\_block4\_add - No activation

conv4\_block4\_out - relu

conv4\_block5\_1\_conv - linear

conv4\_block5\_1\_bn - No activation

conv4\_block5\_1\_relu - relu

conv4\_block5\_2\_conv - linear

conv4\_block5\_2\_bn - No activation

conv4\_block5\_2\_relu - relu

conv4\_block5\_3\_conv - linear

conv4\_block5\_3\_bn - No activation

conv4\_block5\_add - No activation

conv4\_block5\_out - relu

conv4\_block6\_1\_conv - linear

conv4\_block6\_1\_bn - No activation

conv4\_block6\_1\_relu - relu

conv4\_block6\_2\_conv - linear

conv4\_block6\_2\_bn - No activation

conv4\_block6\_2\_relu - relu

conv4\_block6\_3\_conv - linear

conv4\_block6\_3\_bn - No activation

conv4\_block6\_add - No activation

conv4\_block6\_out - relu

conv5\_block1\_1\_conv - linear

conv5\_block1\_1\_bn - No activation

conv5\_block1\_1\_relu - relu

conv5\_block1\_2\_conv - linear

conv5\_block1\_2\_bn - No activation

conv5\_block1\_2\_relu - relu

conv5\_block1\_0\_conv - linear

conv5\_block1\_3\_conv - linear

conv5\_block1\_0\_bn - No activation

conv5\_block1\_3\_bn - No activation

conv5\_block1\_add - No activation

conv5\_block1\_out - relu

conv5\_block2\_1\_conv - linear

conv5\_block2\_1\_bn - No activation

conv5\_block2\_1\_relu - relu

conv5\_block2\_2\_conv - linear

conv5\_block2\_2\_bn - No activation

conv5\_block2\_2\_relu - relu

conv5\_block2\_3\_conv - linear

conv5\_block2\_3\_bn - No activation

conv5\_block2\_add - No activation

conv5\_block2\_out - relu

conv5\_block3\_1\_conv - linear

conv5\_block3\_1\_bn - No activation

conv5\_block3\_1\_relu - relu

conv5\_block3\_2\_conv - linear

conv5\_block3\_2\_bn - No activation

conv5\_block3\_2\_relu - relu

conv5\_block3\_3\_conv - linear

conv5\_block3\_3\_bn - No activation

conv5\_block3\_add - No activation

conv5\_block3\_out - relu

global\_average\_pooling2d\_5 - No activation

dense\_10 - relu

dense\_11 - softmax

A white background with text

Description automatically generated with medium confidence

A graph with a line and a line

Description automatically generated

A graph with a line

Description automatically generated

A screenshot of a graph

Description automatically generated

Summary of Findings

The accuracy results along with model hyper and performance parameters for all the designed models for this project’s objectives is tabulated below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Training Accuracy | Validation Accuracy | Test Accuracy | Comment |
| Manual Classification Accuracy | NA | NA | 93.33% | Image size used 32x768 |
| MLP-Based Model-1 Training Accuracy | 53.87% | 50.43% |  | Image size used 32x768 |
| MLP-Based Model-2 Accuracy | 43.60% | 45.22% |  | Image size used 5x50 |
| Single Hidden Layer CNN-Based Model-1 Accuracy and Performance | 100% | 100% | 99.77% | Image size used 32x768 |
| Single Hidden Layer CNN-Based Model-2 Test Accuracy and Performance | 100% | 100% | 100% | Image size used 16x390 |
| Multiple Hidden Layer CNN-Based Model-1 Test Accuracy and Performance | 100% | 99.71% | 99.71% | Image size used 32x768 |
| Multiple Hidden Layer CNN-Based Model-2 Test Accuracy and Performance | 98.99% | 99.13% | 99.13% | Image size used 16x390 |
| Multiple Hidden Layer CNN-Based Model-3 Test Accuracy and Performance | 100% | 100% | 99.77% | Image size used 16x390 |
| Multiple Hidden Layer CNN-Based Model-4 Test Accuracy and Performance | 100% | 100% | 99.77% | Image size used 32x768 |
| Pre-Trained Resnet Model Test Accuracy and Performance | 100% | 99.71% | 99.31% | Image size used 32x768 |

Key findings:

* CNN models performed better than MLP NN models for the accuracies – training and validation for this limited et of training data.
* Within CNN, hidden multiple non-pre-trained models performed better in terms of training, validation, and test
* ResNet’s accuracy of 99.71% is not better than hidden multi-layer CNN models – 99.77% - 100%. This could be due to:
  + Model complexity
  + Data volume and diversity for the classifier section of ResNet
  + Deep NNs are prone to over-fitting
  + Bias of pre-trained weights of ResNet
  + Feature representation – Resnet may be not well-suited for this type of RADAR tall images

References and Bibliography

1. Google. (n.d.). Colaboratory. Retrieved March 10, 2024, from <https://colab.research.google.com/>
2. Jupyter Development Team. (n.d.). Jupyter Notebook. Retrieved March 10, 2024, from <https://jupyter.org/>
3. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016
4. 2. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006
5. 3. R.-S. He and Z.-G. Ding, Applications of Machine Learning in Wireless Communications, IET, 2019
6. <https://en.wikipedia.org/wiki/Artificial_intelligence>

Source Code – Google Colab

Course Project - Object Detection Using CNNs in RADAR Signal Imagery

Course Name: EE904 - Deep Learning For Wireless Communication

Professor: Prof. Tushar Sandhan

Academic Quarter: Q3-2023-2024

Author and Student Name: Venkateswar Reddy Melachervu

Roll No: 23156022

Email: vmela23@iitk.ac.in

Project Description

The principal objective of this project is to classify, using deep neural network, a given/unseen RADAR signal imagery obtained from the RADAR scan of a wall accurately into:

Empty: No person behind the wall

Still: A person is present behind the wall in stand-still position

Walking: A person is walking behind the wall

This scenario finds crucial application in military endeavors and surveillance.

Training Dataset

Training data set of RADAR scan imagery taken through a concrete wall is provided as input to this project and was made available at this link. The dataset contains train folder with sample imagery for training the deep network and test folder with test magery to evaluate the performance of the developed network model.

Program Tasks

The main task is to investigate/classify the RADAR signal imagery accurately for the object behind the wall - empty, person standing still or walking. Additionally, this project investigates higher accuracy CNN model, using trial and error method, for this classification while addressing the below requirements of the project assignment.

Plots 5 random images from each class on a single row and details an intuitive human analysis of imagery patterns to manually classify any new sample into one of the three classes - Empty, Still, Walking

Selects 10 samples randomly from each class, mixes them all up for manual classification and creates a report on the correct classification and difficulties faced for in-correct classification.

Uses non-CNN MLP NN for training and testing and creates a report on accuracy, training time, accuracy graphs

Designs and uses a CNN with 1 hidden layer and trains it on provided training data and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.

Evaluates the above trained CNN model with the test data set and creates a report on accuracy and plots confusion matrix.

Designs CNN with multiple hidden layers and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.

Changes number of hidden layers, number of kernels and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.

Uses pre-trained ResNet model and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.

Dependencies

This project leverages the below libraries:

1 Tensorflow

2 Keras

3 ResNet

4 gdown

5 PIL

6 numpy

7 google drive for training test data set etc.

# let's do the imports and initializations and installs

import matplotlib.pyplot as plt

import numpy as np

import PIL

import tensorflow as tf

import time

import numpy as np

import os

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from tensorflow import keras

from tensorflow.keras import layers, models

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.optimizers import Adam

from google.colab import drive

# Mount Google Drive

drive.mount('/content/drive', force\_remount=True)

# configure the train and test data directory

train\_dir = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/train'

train\_dir\_empty = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/train/empty'

train\_dir\_still = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/train/still'

train\_dir\_walking = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/train/walking'

val\_dir = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/val/'

tests\_dir = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/tests/'

tests\_dir\_empty = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/tests/empty'

tests\_dir\_still = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/tests/still'

tests\_dir\_walking = '/content/drive/MyDrive/EE-904-GPR-CNN-Project/dataset/tests/walking'

# load the training samples from training data dir

import pathlib

data\_dir = pathlib.Path(train\_dir)

tests\_dir = pathlib.Path(tests\_dir)

val\_dir = pathlib.Path(val\_dir)

# sanity checks for test/validation folder contents

files = os.listdir(train\_dir)

print('Files/dirs in '+ str(train\_dir) + ':' + str(files))

files = os.listdir(tests\_dir)

print('Files/dirs in '+ str(tests\_dir) + ':' + str(files))

# sanity check for training data

images = list(data\_dir.glob('\*/\*.jpg'))

image\_count = len(images)

print('Count of images in ' + str(data\_dir) + ':' + str(image\_count))

empty = list(data\_dir.glob('empty/\*'))

empty\_image\_count = len(empty)

print('\"empty\" class training image sample count' + ':' + str(empty\_image\_count))

print('Displaying a training image in \"empty\" class...')

PIL.Image.open(str(empty[0]))

# sanity check for training data

still = list(data\_dir.glob('still/\*'))

still\_image\_count = len(still)

print('\"still\" class training image sample count' + ':' + str(still\_image\_count))

print('Displaying a training image in \"still\" class...')

PIL.Image.open(str(still[0]))

# sanity check for training data

walking = list(data\_dir.glob('walking/\*'))

walking\_image\_count = len(still)

print('\"walking\" class training image sample count' + ':' + str(walking\_image\_count))

print('Displaying a training image in \"walking\" class...')

PIL.Image.open(str(walking[0]))

# sanity check for tests data

images = list(tests\_dir.glob('\*/\*.jpg'))

image\_count = len(images)

print('Count of all images in ' + str(tests\_dir) + ':' + str(image\_count))

empty = list(tests\_dir.glob('empty/\*'))

empty\_image\_count = len(empty)

print('\"empty\" class test image sample count' + ':' + str(empty\_image\_count))

print('Displaying a test image in \"empty\" class...')

PIL.Image.open(str(empty[0]))

# sanity check for validation data

images = list(val\_dir.glob('\*/\*.jpg'))

image\_count = len(images)

print('Count of images in ' + str(val\_dir) + ':' + str(image\_count))

empty = list(val\_dir.glob('empty/\*'))

empty\_image\_count = len(empty)

print('\"empty\" class validation image sample count' + ':' + str(empty\_image\_count))

print('Displaying a validation image in \"empty\" class...')

PIL.Image.open(str(empty[0]))

# Displaying 5 random images of Empty class in a row from training data set

from PIL import Image

import os

import random

import matplotlib.pyplot as plt

def display\_random\_images(directory, image\_width, image\_height):

plt.figure(figsize=(image\_width, image\_height))

plt.suptitle('5 Empty Class Random Training Samples', fontsize=12)

image\_files = os.listdir(directory)

random.shuffle(image\_files)

for i in range(5):

img\_path = os.path.join(directory, image\_files[i])

img = Image.open(img\_path)

plt.subplot(1, 5, i+1)

plt.imshow(img, cmap='gray') # Specify colormap as 'gray' for grayscale images

plt.title(image\_files[i], fontsize=10)

plt.axis('off')

plt.tight\_layout() # Adjust layout to prevent overlap

plt.show()

# Display from "Empty" training directory

directory = train\_dir\_empty

image\_width = 8 # Width of each image in inches

image\_height = 7.0 # Height of each image in inches

display\_random\_images(directory, image\_width, image\_height)

# Displaying 5 random images of Still class in a row from training data set

from PIL import Image

import os

import random

import matplotlib.pyplot as plt

def display\_random\_images(directory, image\_width, image\_height):

plt.figure(figsize=(image\_width, image\_height))

plt.suptitle('5 Still Class Random Training Samples', fontsize=12)

image\_files = os.listdir(directory)

random.shuffle(image\_files)

for i in range(5):

img\_path = os.path.join(directory, image\_files[i])

img = Image.open(img\_path)

plt.subplot(1, 5, i+1)

plt.imshow(img, cmap='gray') # Specify colormap as 'gray' for grayscale images

plt.title(image\_files[i], fontsize=10)

plt.axis('off')

plt.tight\_layout() # Adjust layout to prevent overlap

plt.show()

# Display from "Still" training directory

directory = train\_dir\_still

image\_width = 8 # Width of each image in inches

image\_height = 7.0 # Height of each image in inches

display\_random\_images(directory, image\_width, image\_height)

# Displaying 5 random images of Walking class in a row from training data set

from PIL import Image

import os

import random

import matplotlib.pyplot as plt

def display\_random\_images(directory, image\_width, image\_height):

plt.figure(figsize=(image\_width, image\_height))

plt.suptitle('5 Walking Class Random Training Samples', fontsize=12)

image\_files = os.listdir(directory)

random.shuffle(image\_files)

for i in range(5):

img\_path = os.path.join(directory, image\_files[i])

img = Image.open(img\_path)

plt.subplot(1, 5, i+1)

plt.imshow(img, cmap='gray') # Specify colormap as 'gray' for grayscale images

plt.title(image\_files[i], fontsize=10)

plt.axis('off')

plt.tight\_layout() # Adjust layout to prevent overlap

plt.show()

# Display from "Walking" training directory

directory = train\_dir\_walking

image\_width = 8 # Width of each image in inches

image\_height = 7.0 # Height of each image in inches

display\_random\_images(directory, image\_width, image\_height)

[2]

# define reusable data pipeline function

def setup\_data\_pipeline(train\_dir, test\_dir, validation\_split=0.2, batch\_size=32, img\_height=768, img\_width=32, seed=12081970):

# Training dataset configuration

print('Preparing training data set...')

print('Image size used for training ' + str(img\_width) + 'x' + str(img\_height))

print('Validation split of the training samples ' + str(validation\_split\*100) + '%')

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

train\_dir,

validation\_split=validation\_split,

subset="training",

seed=seed,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size

)

print('Done - Preparing training data set.')

# Validation dataset configuration

print('Preparing validation data set...')

val\_ds = tf.keras.utils.image\_dataset\_from\_directory(

train\_dir,

validation\_split=validation\_split,

subset="validation",

seed=seed,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size

)

print('Done - Preparing validation data set.')

# Test dataset configuration

print('Preparing test data set...')

test\_ds = tf.keras.utils.image\_dataset\_from\_directory(

test\_dir,

seed=seed,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size

)

print('Done - Preparing test data set.')

# List of inferred class names

class\_names = train\_ds.class\_names

print('List of all training classes:', class\_names)

# training batch set info

print('Training set batches information:')

for image\_batch, labels\_batch in train\_ds:

print(image\_batch.shape)

print(labels\_batch.shape)

# perforance settings - buffered prefetching

print('Setting buffered prefetching the data/images for better execution performance of the program...')

# Performance settings - buffered prefetching

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=AUTOTUNE)

val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

test\_ds = test\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

print('Done - Setting buffered pre-fetch.')

return train\_ds, val\_ds, test\_ds, class\_names

# tensorflow dashboard for gradient descent analytics

!pip install ngrok

!pip install pyngrok

import ngrok

from pyngrok import ngrok

# Set Ngrok authtoken

ngrok.set\_auth\_token("2dgXfPL1pbdqUcPXyS5axzhmnrB\_4Ux9eRzJVVyUPJD12NEqy")

# Start ngrok

# Establish a tunnel to the TensorBoard port

public\_url = ngrok.connect(addr="6006", bind\_tls=True).public\_url

print("TensorBoard URL:", public\_url)

ngrok\_tunnel = ngrok.connect(addr="6006", bind\_tls=True)

# Print the public URL where TensorBoard is exposed

print("TensorBoard URL:", ngrok\_tunnel.public\_url)

%load\_ext tensorboard

log\_dir = "logs/fit"

%tensorboard --logdir $log\_dir --bind\_all

# Normal MLP NN model without any convolution operation

# hyper paramters definition

batch\_size = 32 # to run back propagation epoch

img\_height = 768

img\_width = 32

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

# design the model

normal\_mlp\_nn\_model\_1 = Sequential([

layers.Rescaling(1./255, input\_shape=input\_shape),

Flatten(input\_shape=input\_shape),

Dense(128, activation='relu'),

Dense(64, activation='relu'),

Dense(num\_classes, activation='softmax')

])

normal\_mlp\_nn\_model\_1.\_name = 'Normal\_MLP\_NN\_Without\_Convolution\_And\_Original\_Image\_Resolution\_Training\_1'

# compile the model

normal\_mlp\_nn\_model\_1.compile(

# Adam optimizer

optimizer='adam',

# # Loss function for classification tasks

loss='sparse\_categorical\_crossentropy',

# Metric to monitor during training (e.g., accuracy)

metrics=['accuracy'])

# let's print the summary of the cnn basic model we built

normal\_mlp\_nn\_model\_1.summary()

# Print activation functions

print("Activation Functions:")

for layer in normal\_mlp\_nn\_model\_1.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# let's train the model - 10 epochs

import time

start\_time = time.time()

history = normal\_mlp\_nn\_model\_1.fit(

train\_ds,

validation\_data = val\_ds,

epochs = epochs

)

end\_time = time.time()

training\_time = end\_time - start\_time

print("Training Time:", training\_time, "seconds")

# let's plot all graphs to visually understand and evaluate the performance for nomal MLP model

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# plot training and validation accuracy

plt.figure(figsize=(10, 5))

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training and validation loss

plt.figure(figsize=(10, 5))

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

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

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.grid(True)

plt.show()

# training time per epoch graph

import matplotlib.pyplot as plt

# Calculate training time per epoch

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Normal MLP NN model without any convolution operation - non-original size image training

# hyper paramters definition

batch\_size = 32 # to run back propagation epoch

# img\_height = 768

# img\_width = 32

img\_height = 50

img\_width = 5

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

# design the model

normal\_mlp\_nn\_model\_2 = Sequential([

layers.Rescaling(1./255, input\_shape=input\_shape),

Flatten(input\_shape=input\_shape),

Dense(128, activation='relu'),

Dense(64, activation='relu'),

Dense(num\_classes, activation='softmax')

])

normal\_mlp\_nn\_model\_2.\_name = 'Normal\_MLP\_NN\_Without\_Convolution\_And\_Non\_Original\_Image\_Resolution\_Training\_2'

# compile the model

normal\_mlp\_nn\_model\_2.compile(

# Adam optimizer

optimizer='adam',

# # Loss function for classification tasks

loss='sparse\_categorical\_crossentropy',

# Metric to monitor during training (e.g., accuracy)

metrics=['accuracy'])

# let's print the summary of the cnn basic model we built

normal\_mlp\_nn\_model\_2.summary()

# Print activation functions

print("Activation Functions:")

for layer in normal\_mlp\_nn\_model\_2.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# let's train the model - 10 epochs

epochs = 10

import time

start\_time = time.time()

history = normal\_mlp\_nn\_model\_2.fit(

train\_ds,

validation\_data = val\_ds,

epochs = epochs

)

end\_time = time.time()

training\_time = end\_time - start\_time

print("Training Time:", training\_time, "seconds")

# let's plot all graphs to visually understand and evaluate the performance

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# plot training and validation accuracy

plt.figure(figsize=(10, 5))

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training and validation loss

plt.figure(figsize=(10, 5))

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

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

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.grid(True)

plt.show()

# training time per epoch graph

import matplotlib.pyplot as plt

# Calculate training time per epoch

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# CNN with one hidden layer and RELU activation

batch\_size = 32 # to run back propagation epoch

img\_height = 768

img\_width = 32

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

cnn\_model\_with\_1hl = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_1hl.\_name = 'CNN\_with\_One\_Hidden\_Layer\_RELU\_Activation\_with\_Original\_Image\_Trainin\_1'

# Compile the model

cnn\_model\_with\_1hl.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_1hl.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_1hl.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_1hl.fit(train\_ds, validation\_data=val\_ds, epochs=epochs)

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_one\_hidden\_layer on the test data available in val directory dataset

print('Evaluating the model accuracy with test folder data...')

test\_loss, test\_accuracy = cnn\_model\_with\_1hl.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_1hl.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# CNN with one hidden layer and RELU activation with non-original image size

batch\_size = 32 # to run back propagation epoch

img\_height = 190

img\_width = 16

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

cnn\_model\_with\_1hl\_2 = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_1hl\_2.\_name = 'CNN\_with\_One\_Hidden\_Layer\_RELU\_Activation\_with\_Non\_Original\_Image\_Training\_2'

# Compile the model

cnn\_model\_with\_1hl\_2.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_1hl\_2.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_1hl\_2.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_1hl\_2.fit(train\_ds, validation\_data=val\_ds, epochs=epochs)

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_one\_hidden\_layer on the test data available in val directory dataset

print('Evaluating the model accuracy with test folder data...')

test\_loss, test\_accuracy = cnn\_model\_with\_1hl\_2.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_1hl\_2.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# CNN with multiple hidden layer and RELU activation, 32 filter kernels and visual plots of gradient descent

batch\_size = 32 # to run back propagation epoch

img\_height = 768

img\_width = 32

validation\_split=0.2

seed=12081970

epochs = 10

from tensorflow.keras.callbacks import TensorBoard

import datetime

# Let's define log directory for TensorBoard

log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Let's create TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

cnn\_model\_with\_multiple\_hidden\_layers\_1 = models.Sequential([

layers.Rescaling(1./255, input\_shape=input\_shape),

layers.Conv2D(32, 3, padding='same', activation='relu'), # filters 32, kernel size 3

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_multiple\_hidden\_layers\_1.\_name = 'CNN\_with\_Multiple\_Hidden\_Layers\_RELU\_Activation\_with\_Original\_Image\_Training\_1'

# Compile the model

cnn\_model\_with\_multiple\_hidden\_layers\_1.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_multiple\_hidden\_layers\_1.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_multiple\_hidden\_layers\_1.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_multiple\_hidden\_layers\_1.fit(train\_ds, validation\_data=val\_ds, epochs=epochs, callbacks=[tensorboard\_callback])

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_multiple\_hidden\_layers on the test data available in val directory dataset

test\_loss, test\_accuracy = cnn\_model\_with\_multiple\_hidden\_layers\_1.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_multiple\_hidden\_layers\_1.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# CNN with multiple hidden layer, RELU activation

batch\_size = 32 # to run back propagation epoch

# img\_height = 768

# img\_width = 32

img\_height = 390

img\_width = 16

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

cnn\_model\_with\_multiple\_hidden\_layers\_2 = models.Sequential([

layers.Rescaling(1./255, input\_shape=(img\_height, img\_width,3)),

layers.Conv2D(12, 3, padding='same', activation='relu'), # filters 12, kernel size 3

layers.MaxPooling2D((2, 2),strides=(2, 2), padding='same'),

layers.Conv2D(24, 3, padding='same', activation='relu'),

layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'),

layers.Conv2D(48, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_multiple\_hidden\_layers\_2.\_name = 'CNN\_with\_Multiple\_Hidden\_Layers\_RELU\_Activation\_with\_Non\_Original\_Image\_Training\_2'

# Compile the model

cnn\_model\_with\_multiple\_hidden\_layers\_2.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_multiple\_hidden\_layers\_2.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_multiple\_hidden\_layers\_2.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_multiple\_hidden\_layers\_2.fit(train\_ds, validation\_data=val\_ds, epochs=epochs)

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_multiple\_hidden\_layers on the test data available in val directory dataset

test\_loss, test\_accuracy = cnn\_model\_with\_multiple\_hidden\_layers\_2.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_multiple\_hidden\_layers\_2.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# CNN with multiple hidden layer, RELU activation with gradient descent data visualization

# For visualizing gradient descent data we will use TensorBoard

from tensorflow.keras.callbacks import TensorBoard

import datetime

# Let's define a log directory for TensorBoard

log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Let's create TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

batch\_size = 32 # to run back propagation epoch

# img\_height = 768

# img\_width = 32

img\_height = 390

img\_width = 16

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = len(class\_names)

input\_shape = (img\_height,img\_width, 3)

cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3 = models.Sequential([

layers.Rescaling(1./255, input\_shape=(img\_height, img\_width,3)),

layers.Conv2D(12, 3, padding='same', activation='relu'), # filters 12, kernel size 3

layers.MaxPooling2D((2, 2),strides=(2, 2), padding='same'),

layers.Conv2D(24, 3, padding='same', activation='relu'),

layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'),

layers.Conv2D(48, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.\_name = 'CNN\_with\_Multiple\_Hidden\_Layers\_RELU\_Activation\_with\_Non\_Original\_Image\_Training\_GDV\_3'

# Compile the model

cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.fit(train\_ds, epochs=epochs, validation\_data=val\_ds, callbacks=[tensorboard\_callback])

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_multiple\_hidden\_layers on the test data available in val directory dataset

test\_loss, test\_accuracy = cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_multiple\_hidden\_layers\_gdv\_3.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# CNN with multiple hidden layer, RELU activation, and non-original image size for training set with gdv

from tensorflow.keras.callbacks import TensorBoard

import datetime

# Let's define a log directory for TensorBoard

log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Let's create TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# let's train with original image sizes

i\_height = 768

i\_width = 32

val\_split = 0.2

seed=12081970

b\_size = 32

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, val\_split, i\_width, i\_height, b\_size, seed)

num\_classes = len(class\_names)

input\_shape = (i\_height,i\_width, 3)

epochs = 10

cnn\_model\_with\_multiple\_hidden\_layers\_4 = models.Sequential([

layers.Rescaling(1./255, input\_shape=input\_shape),

layers.Conv2D(12, 3, padding='same', activation='relu'), # filters 12, kernel size 3

layers.MaxPooling2D((2, 2)),

layers.Conv2D(24, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn\_model\_with\_multiple\_hidden\_layers\_4.\_name = 'CNN\_with\_Multiple\_Hidden\_Layers\_RELU\_Activation\_with\_Original\_Image\_Size\_Training\_4'

# Compile the model

cnn\_model\_with\_multiple\_hidden\_layers\_4.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Model details

cnn\_model\_with\_multiple\_hidden\_layers\_4.summary()

# Print activation functions

print("Activation Functions:")

for layer in cnn\_model\_with\_multiple\_hidden\_layers\_4.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Train the model

start\_time = time.time()

history = cnn\_model\_with\_multiple\_hidden\_layers\_4.fit(train\_ds, validation\_data=val\_ds, epochs=epochs, callbacks=[tensorboard\_callback])

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_multiple\_hidden\_layers on the test data available in val directory dataset

test\_loss, test\_accuracy = cnn\_model\_with\_multiple\_hidden\_layers\_4.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = cnn\_model\_with\_multiple\_hidden\_layers\_4.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# Pre-trained ResNet model with gradient descent data visualization

# install necessary modules

!pip install tensorflow==2.12.0

!pip install tensorflow-datasets==4.8.0

# imports for Resnet50

import tensorflow\_datasets as tfds

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D, BatchNormalization, GlobalAveragePooling2D

from tensorflow.keras.applications.resnet50 import preprocess\_input, decode\_predictions

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.preprocessing import image

# For visualizing gradient descent data we will use TensorBoard

from tensorflow.keras.callbacks import TensorBoard

import datetime

# Let's define a log directory for TensorBoard

log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Let's create TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# Hyper parameters

batch\_size = 32 # to run back propagation epoch

img\_height = 768

img\_width = 32

# img\_height = 390

# img\_width = 16

validation\_split=0.2

seed=12081970

epochs = 10

# prepare up train and validation data sets

train\_ds, val\_ds, test\_ds, class\_names = setup\_data\_pipeline(train\_dir, val\_dir, validation\_split, batch\_size, img\_height, img\_width, seed)

num\_classes = 3

input\_shape = (img\_height,img\_width, 3)

# Due to repeat of data for training, we need to set the number of steps manually

total\_samples = len(list(train\_ds))

train\_steps\_per\_epoch = total\_samples # this is batch size

val\_steps\_per\_epoch = total\_samples # for valaidation

print('Length of the train data set is ' + str(train\_steps\_per\_epoch))

print('Length of the validation data set is ' + str(val\_steps\_per\_epoch))

# Load ResNet50 pretrained model without the top classification layer

print('Configuring ResNet50 with input shape of ' + str(img\_width) + 'x' + str(img\_height))

base\_model = ResNet50(weights='imagenet', include\_top=False)

# Freeze the base model layers

base\_model.trainable = False

x = base\_model.output

x = GlobalAveragePooling2D()(x)

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

predictions = Dense(num\_classes, activation='softmax')(x)

# Create the full model

gpr\_object\_classify\_resnet50\_model\_1 = Model(inputs=base\_model.input, outputs=predictions)

# Compile the model

for layer in base\_model.layers:

layer.trainable = False

gpr\_object\_classify\_resnet50\_model\_1.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

gpr\_object\_classify\_resnet50\_model\_1.compile(

optimizer = 'adam',

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=False),

metrics = ['accuracy']

)

gpr\_object\_classify\_resnet50\_model\_1.\_name = 'ResNet50\_with\_Original\_Image\_Training\_GDV\_1'

# adapted resnet50 model summary

gpr\_object\_classify\_resnet50\_model\_1.summary()

print("Activation Functions:")

for layer in gpr\_object\_classify\_resnet50\_model\_1.layers:

if hasattr(layer, 'activation'):

print(layer.name, "-", layer.activation.\_\_name\_\_)

else:

print(layer.name, "- No activation")

# Set up TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir='./logs', histogram\_freq=1)

# Train the model

# let's ensure not to run out of training data

train\_ds = train\_ds.repeat()

val\_ds = val\_ds.repeat()

print('Training the ResNet50 with training data...')

start\_time = time.time()

history = gpr\_object\_classify\_resnet50\_model\_1.fit(

train\_ds,

steps\_per\_epoch=train\_steps\_per\_epoch,

validation\_data = val\_ds,

validation\_steps=val\_steps\_per\_epoch,

epochs = epochs,

callbacks=[tensorboard\_callback])

end\_time = time.time()

training\_time\_per\_epoch = (end\_time - start\_time) / epochs

print('Done - Training the ResNet50 with training data.')

print("Training Time Per Epoch:", training\_time\_per\_epoch, "seconds")

# Convert accuracy values to percentage

train\_accuracy\_percentage = [acc \* 100 for acc in history.history['accuracy']]

val\_accuracy\_percentage = [acc \* 100 for acc in history.history['val\_accuracy']]

# Plot accuracy versus epoch in percentage

plt.plot(train\_accuracy\_percentage, label='Training Accuracy', color='blue')

plt.plot(val\_accuracy\_percentage, label='Validation Accuracy', color='green')

plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot training time per epoch

plt.plot(range(1, epochs + 1), [training\_time\_per\_epoch] \* epochs, label='Training Time per Epoch', color='blue')

plt.xlabel('Epoch')

plt.ylabel('Training Time (seconds)')

plt.title('Training Time per Epoch')

plt.legend()

plt.grid(True)

plt.show()

# Evaluate the cnn\_model\_with\_multiple\_hidden\_layers on the test data available in val directory dataset

test\_loss, test\_accuracy = gpr\_object\_classify\_resnet50\_model\_1.evaluate(test\_ds)

# Plot accuracy in percentage

print("Test Accuracy:", test\_accuracy \* 100, "%")

# Predict labels for the test dataset

test\_predictions = gpr\_object\_classify\_resnet50\_model\_1.predict(test\_ds)

test\_labels = np.concatenate([labels for \_, labels in test\_ds], axis=0)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(test\_labels, np.argmax(test\_predictions, axis=1))

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=class\_names)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

%tensorboard --logdir logs

1. Dataset - <https://drive.google.com/drive/folders/1g5NS45zaoxXOran9CYM8J49tTyhZ_9Hd> [↑](#footnote-ref-1)