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 All published and unpublished source materials that have been used in this report have been acknowledged.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

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] 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.

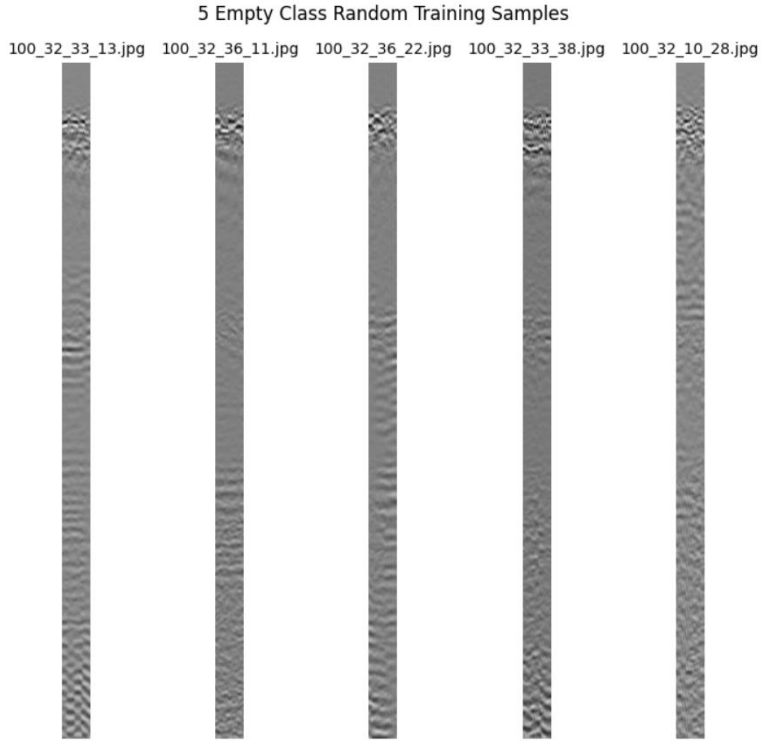
[1] Dataset - https://drive.google.com/drive/folders/1g5NS45zaoxXOran9CYM8J49tTyhZ_9Hd

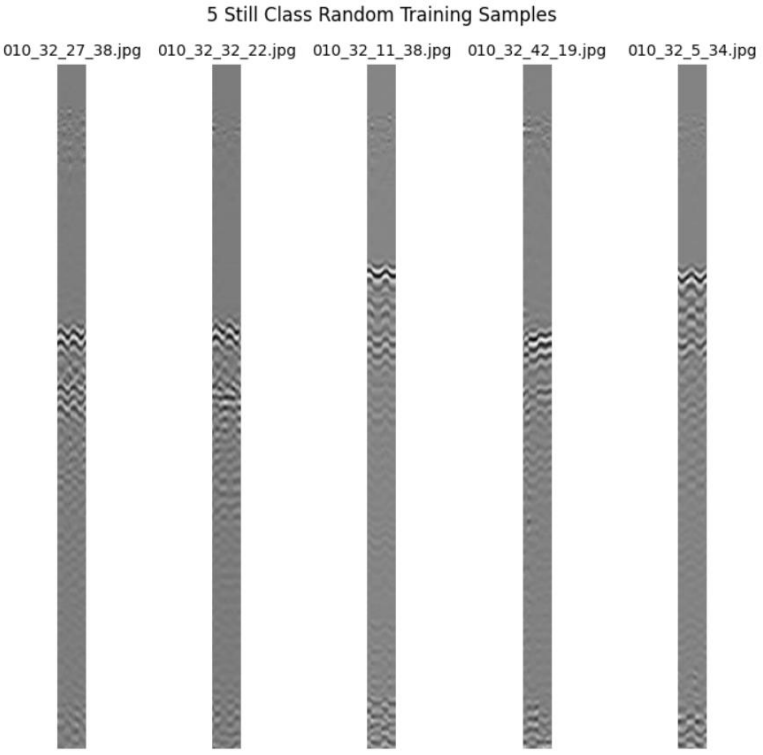
4. Designs and uses a CNN with 1 hidden layer and trains it on provided training data and creates a report on accuracy vs epoch (graphs), model details, total number of learnable parameters, training time per epoch etc.
5. Evaluates the above trained CNN model with the test data set and creates a report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

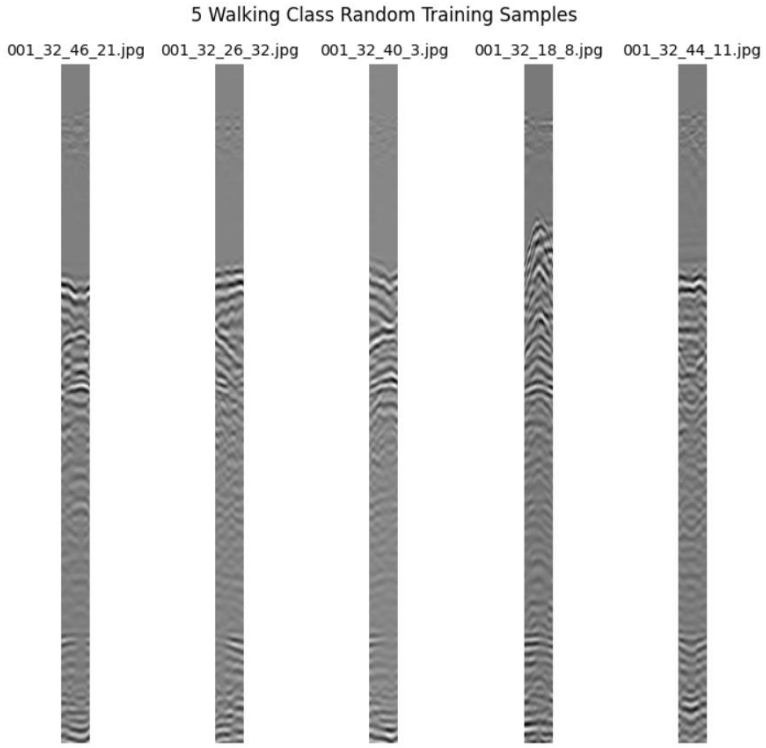
Manual Pattern Analysis

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.







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.

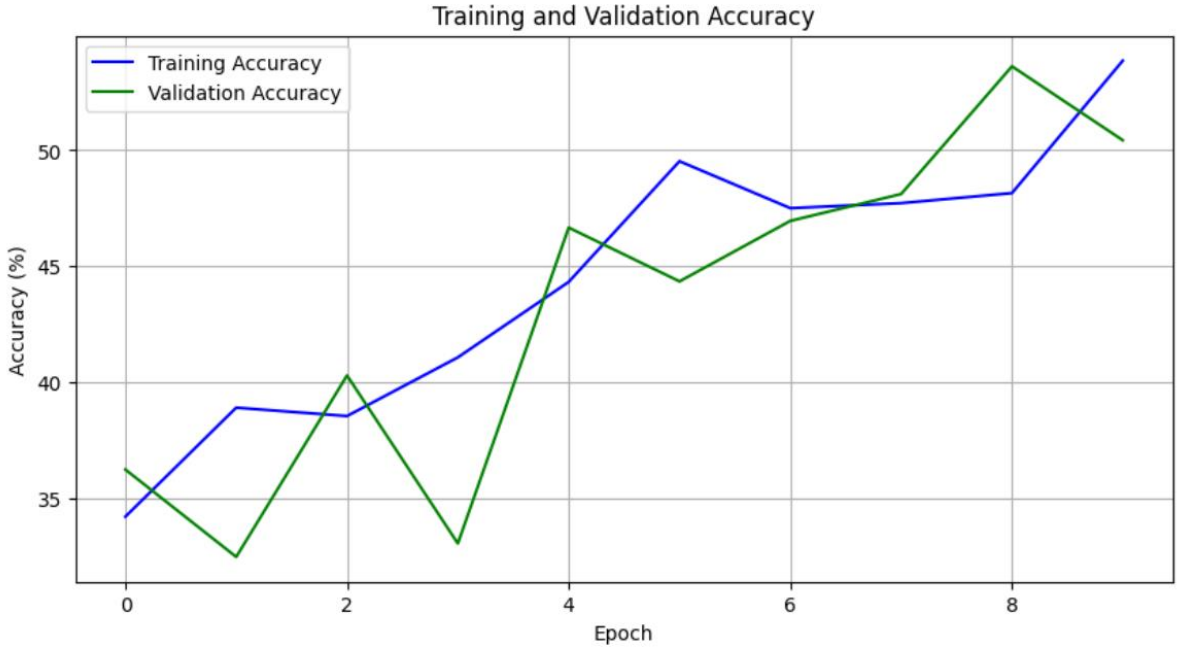
Image File Name	Identified Class	Actual Class	Difficulty Faced	Manual Classification Accuracy
100_32_30_15.jpg	Empty	Empty		93.33%
100_32_33_34.jpg	Empty	Empty		
100_32_34_14.jpg	Empty	Empty		
100_32_35_27.jpg	Empty	Empty		
100_32_3_35.jpg	Walking	Empty	Challenging to distinguish the patterns	
100_32_12_35.jpg	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		

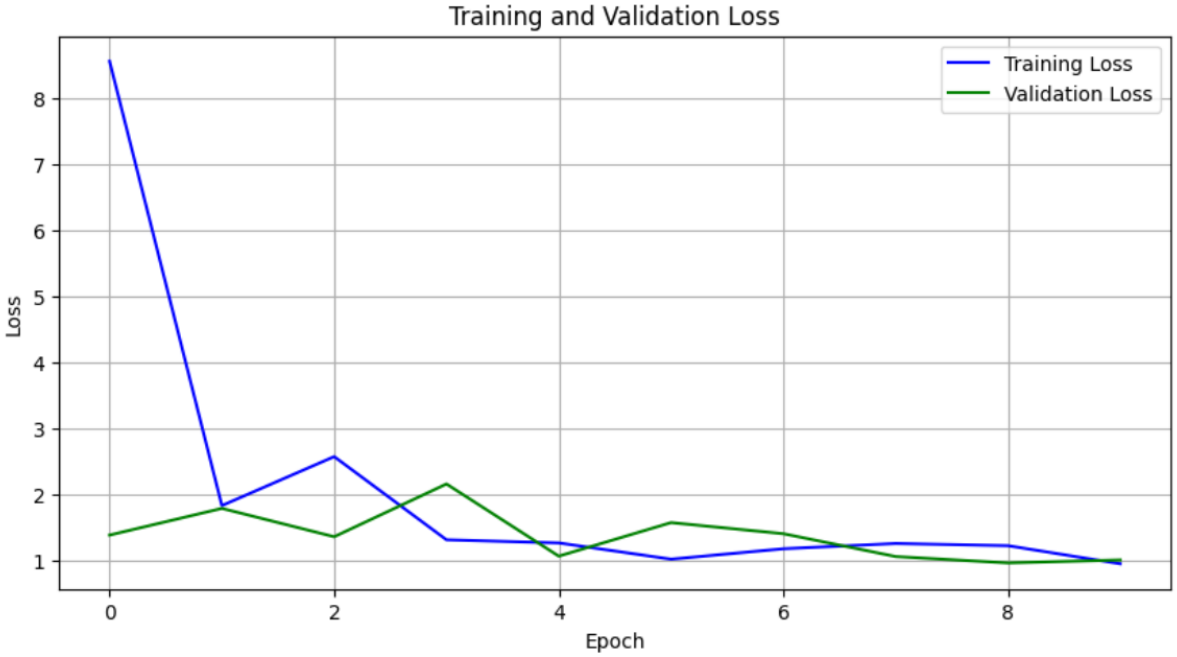
MLP-based Model-1 Training Accuracy

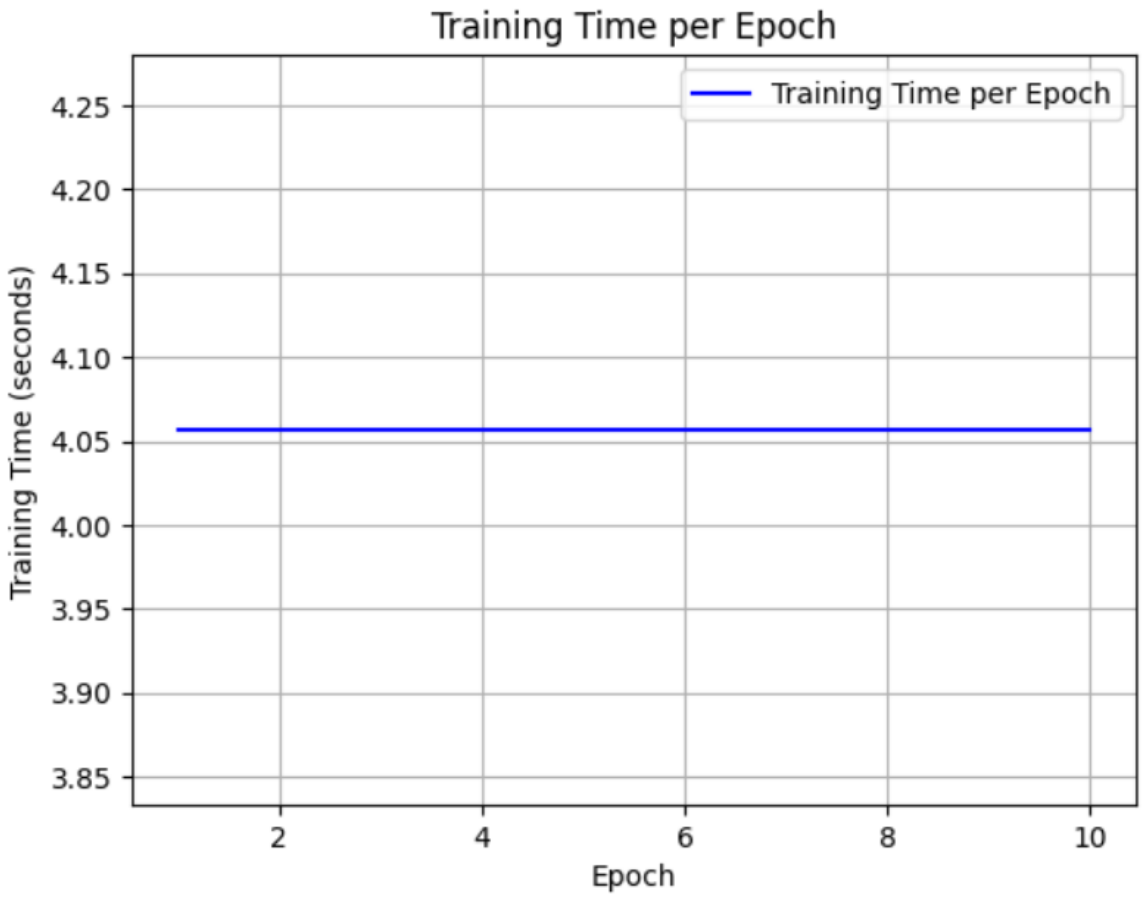
Deployed Google Colab runtime for this classification:

- 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

```
Image size used for training 32x768
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
Model: "Normal_MLP_NN_Without_Convolution_And_Original_Image_Resolution_Training_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 768, 32, 3)        0

 flatten_2 (Flatten)         (None, 73728)             0

 dense_6 (Dense)             (None, 128)               9437312

 dense_7 (Dense)             (None, 64)                8256

 dense_8 (Dense)             (None, 3)                 195

=================================================================
Total params: 9445763 (36.03 MB)
Trainable params: 9445763 (36.03 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________


Activation Functions:
rescaling_4 - No activation
flatten_4 - No activation
dense_9 - relu
dense_10 - relu
dense_11 - softmax
Epoch 1/10
44/44 [==============================] - 37s 731ms/step - loss: 8.5684 - accuracy: 0.3420 - val_loss: 1.3926 - val_accuracy: 0.3623
Epoch 2/10
44/44 [==============================] - 0s 9ms/step - loss: 1.8393 - accuracy: 0.3890 - val_loss: 1.7970 - val_accuracy: 0.3246
Epoch 3/10
44/44 [==============================] - 0s 8ms/step - loss: 2.5812 - accuracy: 0.3854 - val_loss: 1.3676 - val_accuracy: 0.4029
Epoch 4/10
44/44 [==============================] - 0s 8ms/step - loss: 1.3212 - accuracy: 0.4107 - val_loss: 2.1670 - val_accuracy: 0.3304
Epoch 5/10
44/44 [==============================] - 0s 8ms/step - loss: 1.2739 - accuracy: 0.4432 - val_loss: 1.0737 - val_accuracy: 0.4667
Epoch 6/10
44/44 [==============================] - 0s 8ms/step - loss: 1.0279 - accuracy: 0.4953 - val_loss: 1.5817 - val_accuracy: 0.4435
Epoch 7/10
44/44 [==============================] - 0s 8ms/step - loss: 1.1856 - accuracy: 0.4751 - val_loss: 1.4137 - val_accuracy: 0.4696
Epoch 8/10
44/44 [==============================] - 0s 8ms/step - loss: 1.2652 - accuracy: 0.4772 - val_loss: 1.0679 - val_accuracy: 0.4812
Epoch 9/10
44/44 [==============================] - 0s 8ms/step - loss: 1.2322 - accuracy: 0.4816 - val_loss: 0.9712 - val_accuracy: 0.5362
Epoch 10/10
44/44 [==============================] - 0s 8ms/step - loss: 0.9583 - accuracy: 0.5387 - val_loss: 1.0160 - val_accuracy: 0.5043
Training Time: 40.56749224662781 seconds
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy (%)
50
45
40
35
0
2
4
6
8
Epoch


Training and Validation Loss
Training Loss
Validation Loss
Loss
8
7
6
5
4
3
2
1
0
2
4
6
8
Epoch


Training Time per Epoch
Training Time per Epoch
Training Time (seconds)
4.25
4.20
4.15
4.10
4.05
4.00
3.95
3.90
3.85
2
4
6
8
10
Epoch

MLP-based Model-2 Accuracy

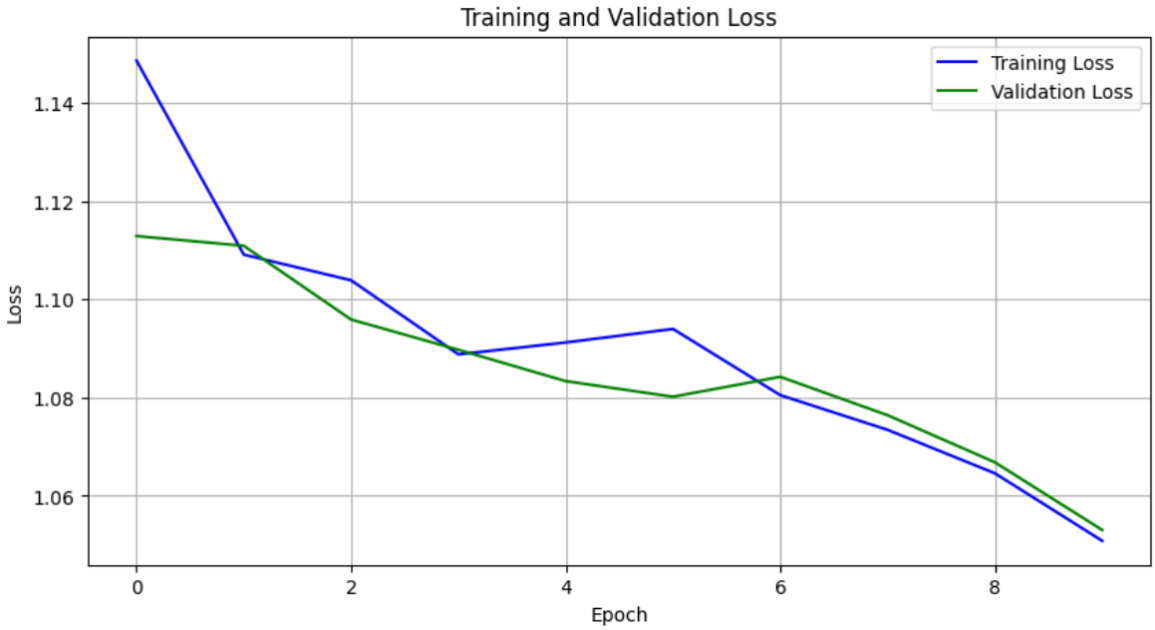
Deployed Google Colab runtime for this classification:

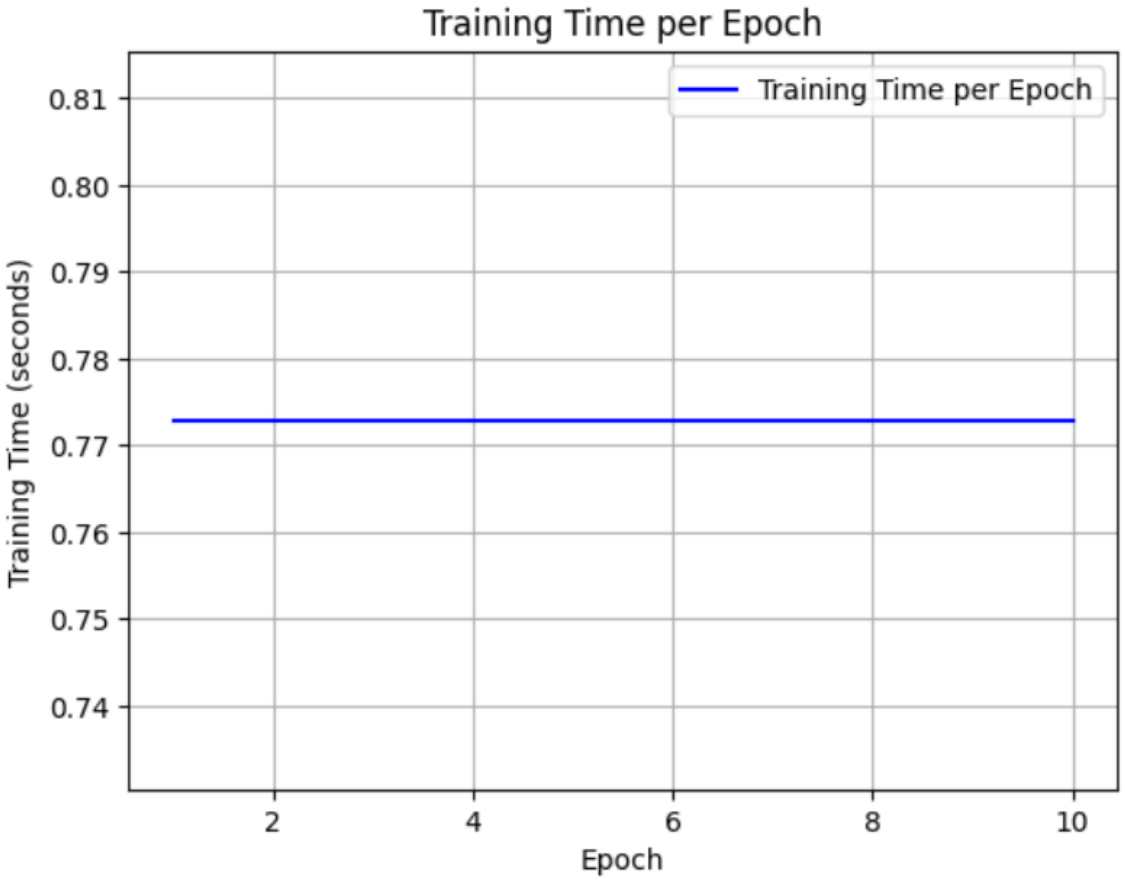
- Machine: T4 GPU – Google Cloud/Colab
- System RAM - Available - 12.7 GB
- GPU RAM Available – 15.0 GB
- Disk - Available – 78.2 GB
- Python 3 Google Compute Engine Backend

```
Preparing training data set...
Image size used for training 5x50
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

```
Model: "Normal_MLP_NN_Without_Convolution_And_Non_Original_Image_Resolution_Training_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 50, 5, 3)          0

 flatten_3 (Flatten)         (None, 750)               0

 dense_9 (Dense)             (None, 128)               96128

 dense_10 (Dense)            (None, 64)                8256

 dense_11 (Dense)            (None, 3)                 195

=================================================================
Total params: 104579 (408.51 KB)
Trainable params: 104579 (408.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Activation Functions:
rescaling_3 - No activation
flatten_3 - No activation
dense_9 - relu
dense_10 - relu
dense_11 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 6s 25ms/step - loss: 1.1486 - accuracy: 0.3196 - val_loss: 1.1129 - val_accuracy: 0.3304
Epoch 2/10
44/44 [==============================] - 0s 3ms/step - loss: 1.1091 - accuracy: 0.3398 - val_loss: 1.1109 - val_accuracy: 0.3246
Epoch 3/10
44/44 [==============================] - 0s 3ms/step - loss: 1.1039 - accuracy: 0.3449 - val_loss: 1.0959 - val_accuracy: 0.3478
Epoch 4/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0888 - accuracy: 0.3839 - val_loss: 1.0897 - val_accuracy: 0.3478
Epoch 5/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0912 - accuracy: 0.3608 - val_loss: 1.0834 - val_accuracy: 0.4406
Epoch 6/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0940 - accuracy: 0.3579 - val_loss: 1.0802 - val_accuracy: 0.4261
Epoch 7/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0805 - accuracy: 0.3854 - val_loss: 1.0842 - val_accuracy: 0.3652
Epoch 8/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0734 - accuracy: 0.3839 - val_loss: 1.0764 - val_accuracy: 0.3449
Epoch 9/10
44/44 [==============================] - 0s 3ms/step - loss: 1.0646 - accuracy: 0.4078 - val_loss: 1.0668 - val_accuracy: 0.4145
Epoch 10/10
44/44 [==============================] - 0s 4ms/step - loss: 1.0508 - accuracy: 0.4360 - val_loss: 1.0530 - val_accuracy: 0.4522
Training Time: 7.728684186935425 seconds
```




Training Time per Epoch
Training Time per Epoch
Training Time (seconds)
0.81
0.80
0.79
0.78
0.77
0.76
0.75
0.74
2
4
6
8
10
Epoch

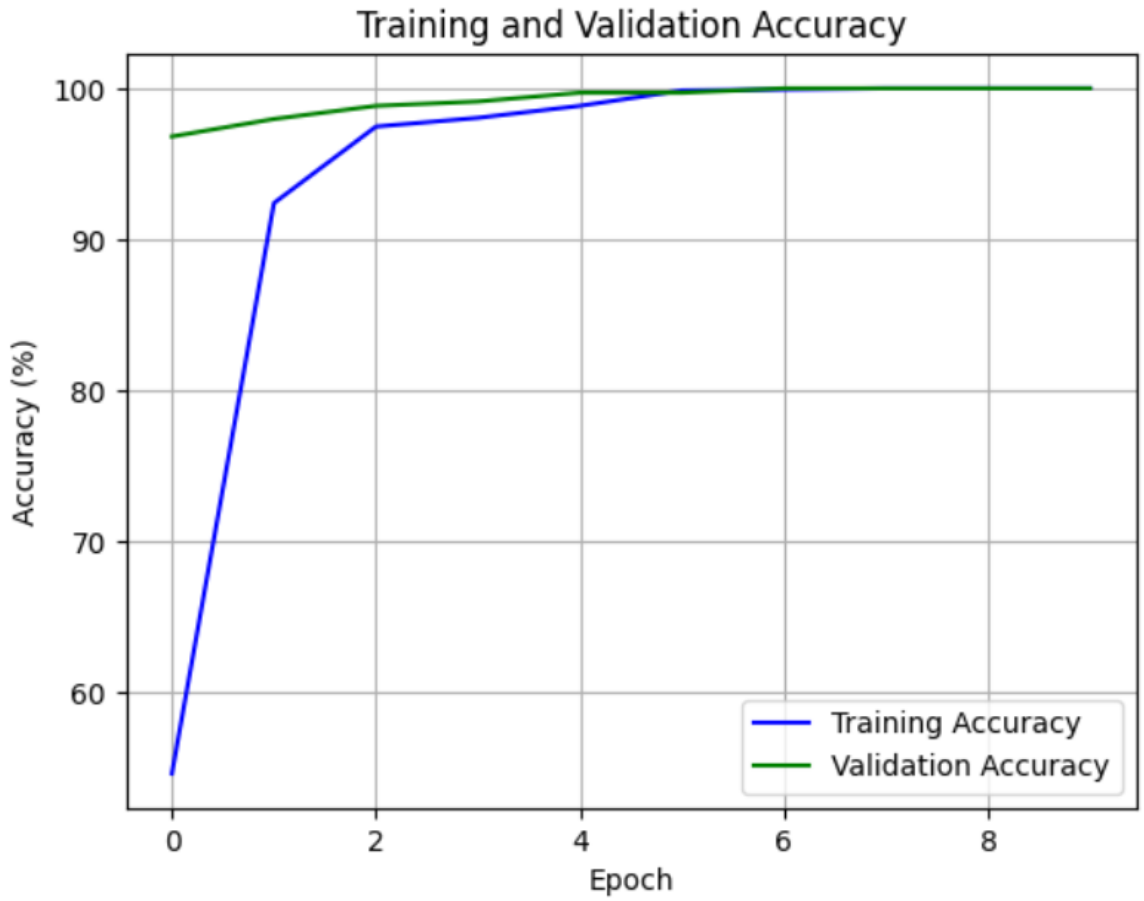
Single Hidden Layer CNN-based Model-1 Accuracy and Performance

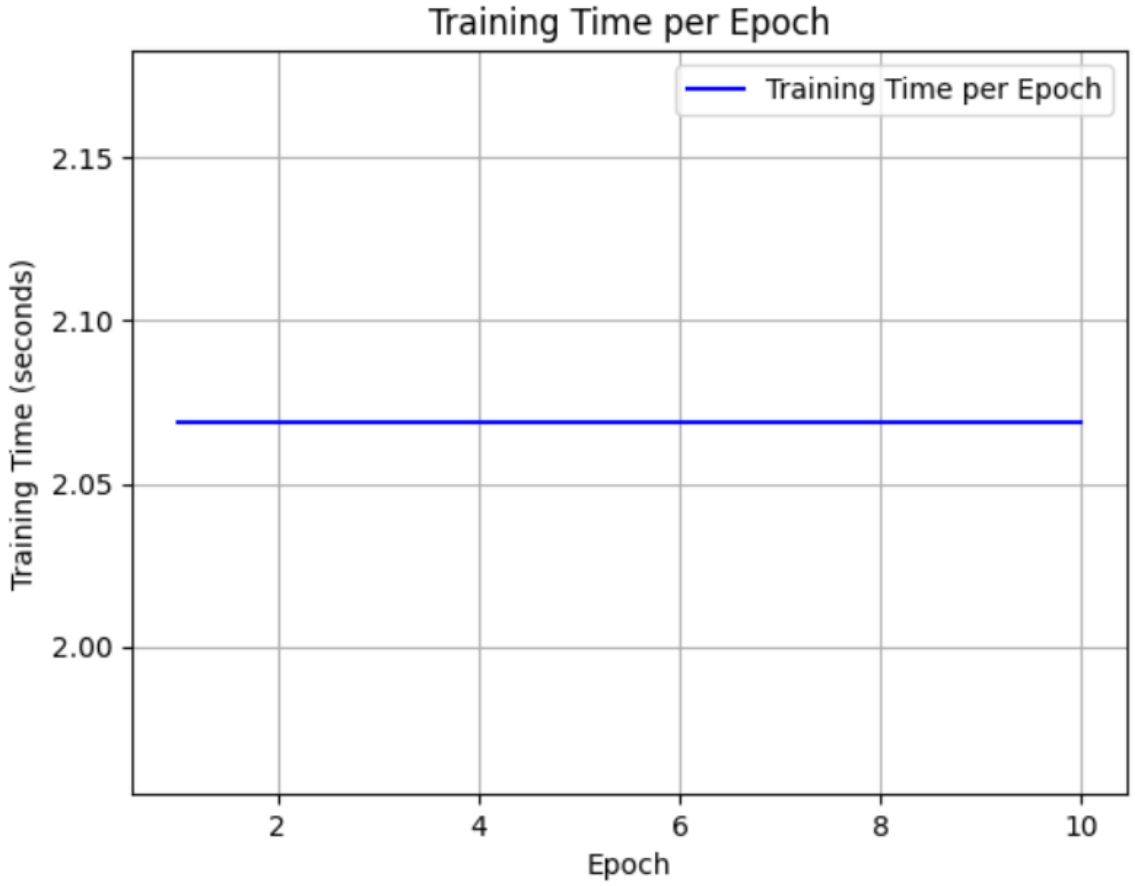
Deployed Google Colab runtime for this classification:

- 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

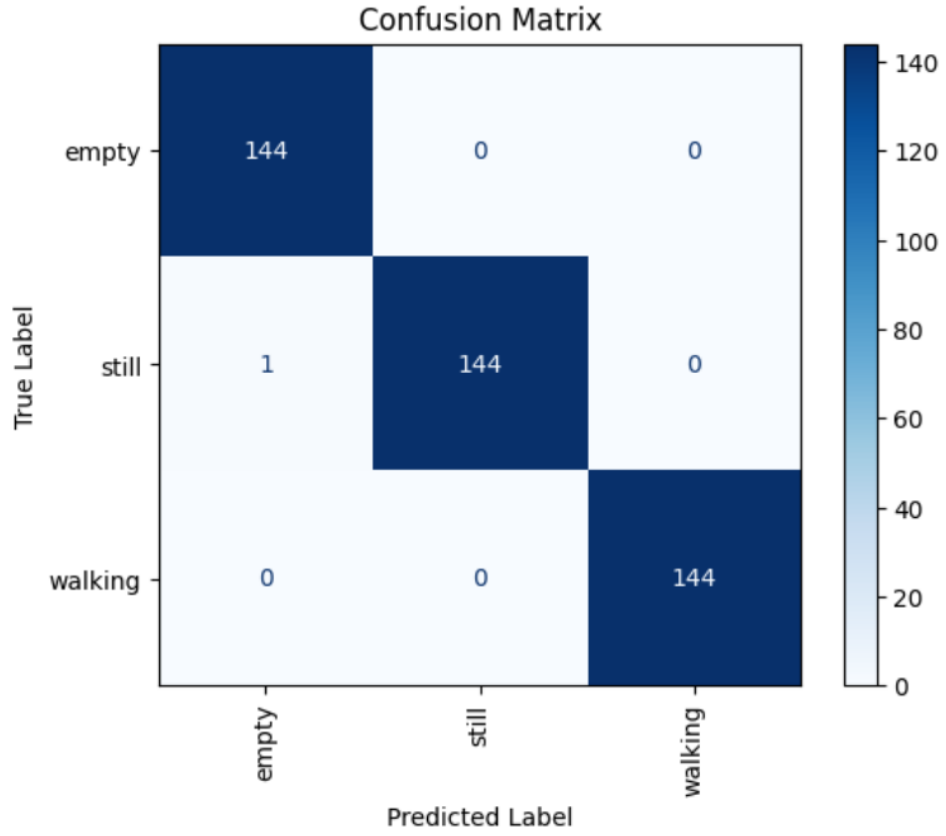
```
Preparing training data set...
Image size used for training 32x768
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
Model: "CNN_with_One_Hidden_Layer_RELU_Activation_with_Original_Image_Trainin_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 766, 30, 32)       896

 max_pooling2d (MaxPooling2  (None, 383, 15, 32)       0
 D)

 flatten_4 (Flatten)         (None, 183840)            0

 dense_12 (Dense)            (None, 128)               23531648

 dense_13 (Dense)            (None, 3)                 387

=================================================================
Total params: 23532931 (89.77 MB)
Trainable params: 23532931 (89.77 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Activation Functions:
conv2d - relu
max_pooling2d - No activation
flatten_4 - No activation
dense_12 - relu
dense_13 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 9s 48ms/step - loss: 538.7670 - accuracy: 0.5466 - val_loss: 0.1667 - val_accuracy: 0.9681
Epoch 2/10
44/44 [==============================] - 1s 20ms/step - loss: 1.2181 - accuracy: 0.9241 - val_loss: 0.4186 - val_accuracy: 0.9797
Epoch 3/10
44/44 [==============================] - 1s 19ms/step - loss: 0.2591 - accuracy: 0.9747 - val_loss: 0.2010 - val_accuracy: 0.9884
Epoch 4/10
44/44 [==============================] - 1s 22ms/step - loss: 0.2320 - accuracy: 0.9805 - val_loss: 0.0390 - val_accuracy: 0.9913
Epoch 5/10
44/44 [==============================] - 1s 21ms/step - loss: 0.0809 - accuracy: 0.9884 - val_loss: 0.0166 - val_accuracy: 0.9971
Epoch 6/10
44/44 [==============================] - 1s 21ms/step - loss: 0.0120 - accuracy: 0.9986 - val_loss: 0.0090 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 1s 20ms/step - loss: 0.0016 - accuracy: 0.9993 - val_loss: 5.2786e-04 - val_accuracy: 1.0000
Epoch 8/10
44/44 [==============================] - 1s 20ms/step - loss: 7.2231e-08 - accuracy: 1.0000 - val_loss: 7.7300e-04 - val_accuracy: 1.0000
Epoch 9/10
44/44 [==============================] - 1s 20ms/step - loss: 1.4222e-08 - accuracy: 1.0000 - val_loss: 8.0331e-04 - val_accuracy: 1.0000
Epoch 10/10
44/44 [==============================] - 1s 20ms/step - loss: 1.4136e-08 - accuracy: 1.0000 - val_loss: 8.0390e-04 - val_accuracy: 1.0000
Training Time Per Epoch: 2.0688756942749023 seconds
```






```
Evaluating the model accuracy with test folder data...
14/14 [==============================] - 87s 2s/step - loss: 0.0069 - accuracy: 0.9977
Test Accuracy: 99.76905584335327 %
14/14 [==============================] - 0s 6ms/step
<Figure size 1000x800 with 0 Axes>
```



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

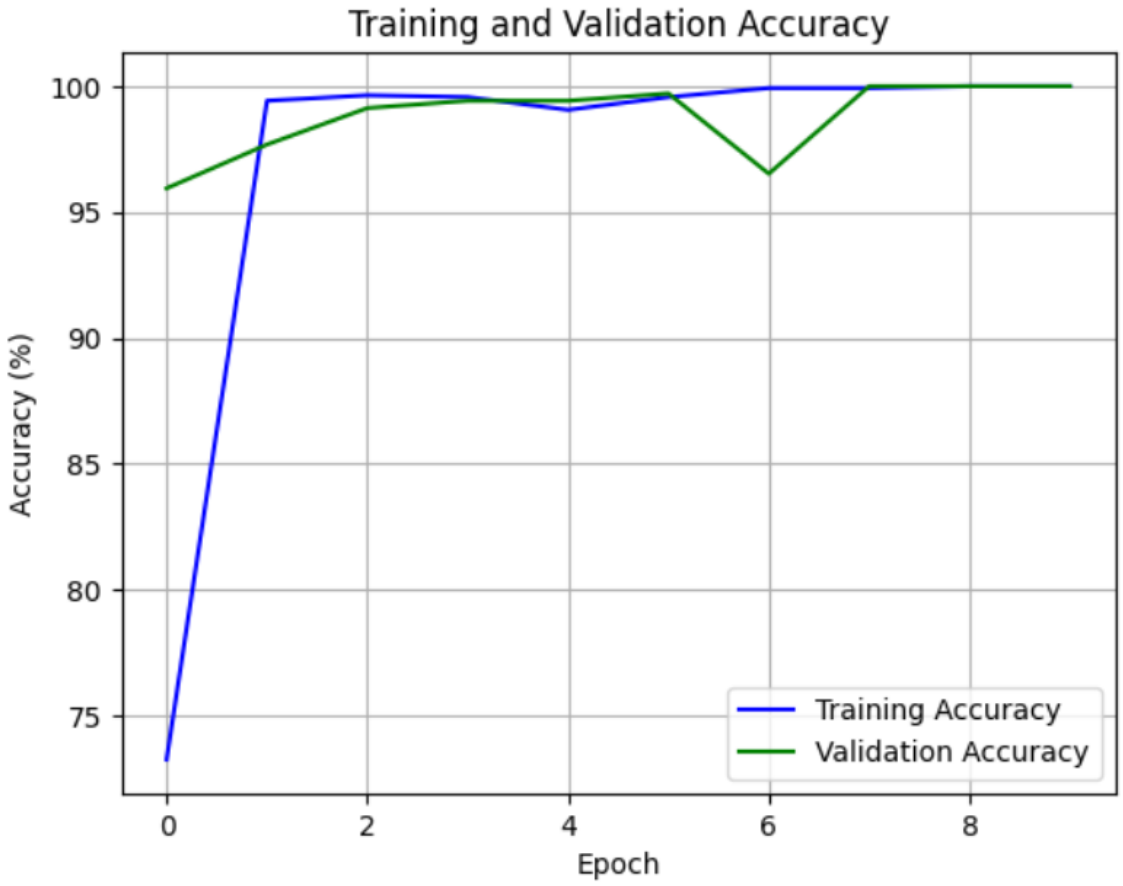
Deployed Google Colab runtime for this classification:

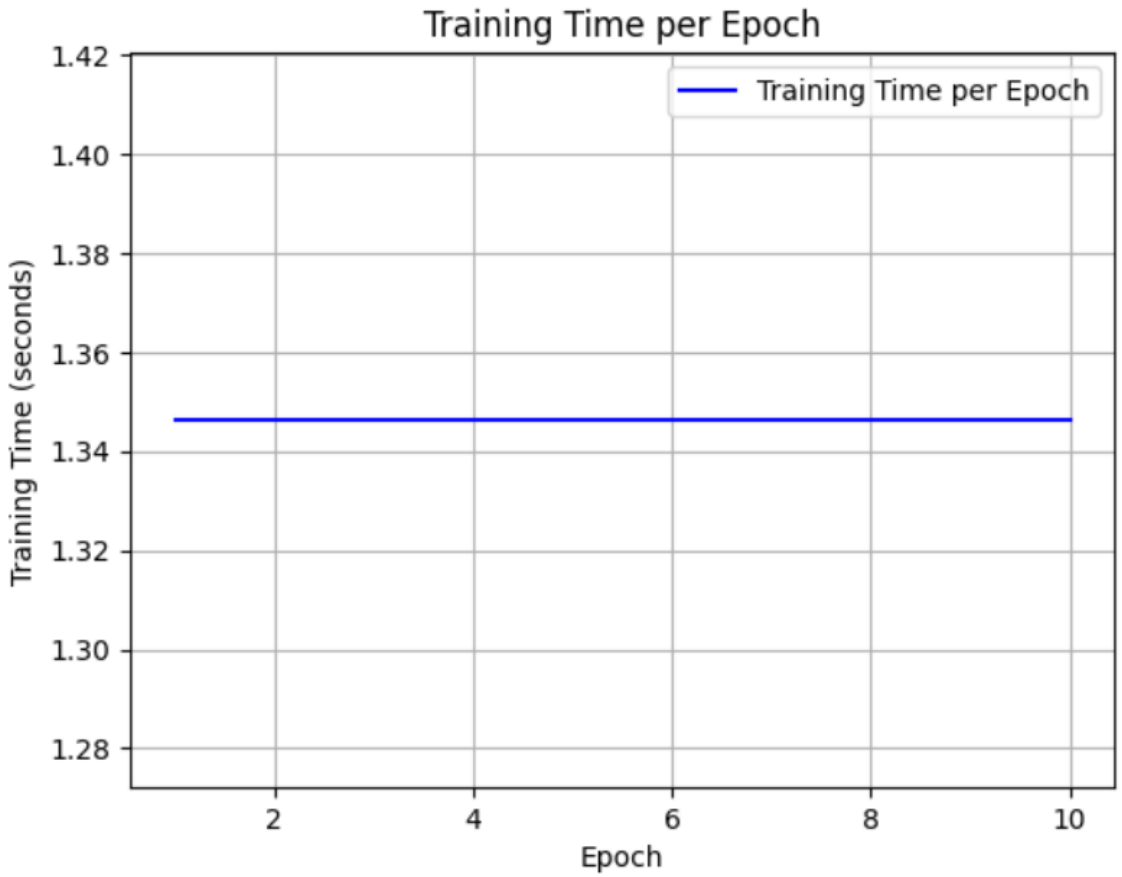
- Machine: T4 GPU – Google Cloud/Colab
- System RAM - Available - 12.7 GB
- GPU RAM Available – 15.0 GB
- Disk - Available – 78.2 GB
- Python 3 Google Compute Engine Backend

```
Preparing training data set...
Image size used for training 16x190
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

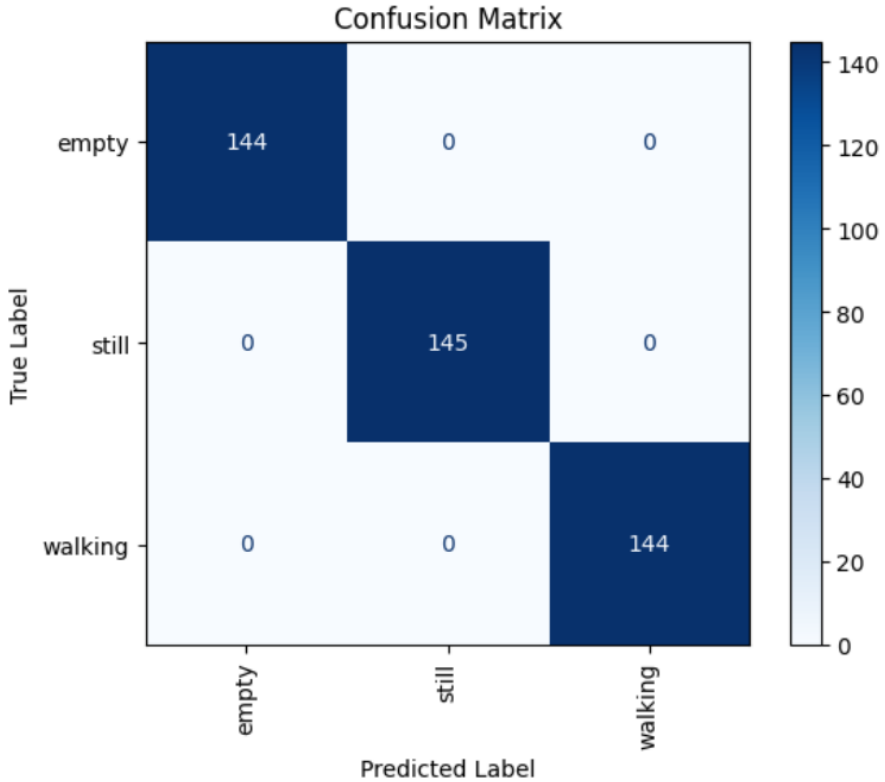
```
Model: "CNN_with_One_Hidden_Layer_RELU_Activation_with_Non_Original_Image_Training_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 188, 14, 32)       896

 max_pooling2d_1 (MaxPoolin  (None, 94, 7, 32)         0
 g2D)

 flatten_5 (Flatten)         (None, 21056)             0

 dense_14 (Dense)            (None, 128)               2695296

 dense_15 (Dense)            (None, 3)                 387

=================================================================
Total params: 2696579 (10.29 MB)
Trainable params: 2696579 (10.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Activation Functions:
conv2d_1 - relu
max_pooling2d_1 - No activation
flatten_5 - No activation
dense_14 - relu
dense_15 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 6s 32ms/step - loss: 146.5767 - accuracy: 0.7325 - val_loss: 1.0857 - val_accuracy: 0.9594
Epoch 2/10
44/44 [==============================] - 0s 6ms/step - loss: 0.0686 - accuracy: 0.9942 - val_loss: 0.6501 - val_accuracy: 0.9768
Epoch 3/10
44/44 [==============================] - 0s 7ms/step - loss: 0.0315 - accuracy: 0.9964 - val_loss: 0.1797 - val_accuracy: 0.9913
Epoch 4/10
44/44 [==============================] - 0s 6ms/step - loss: 0.0297 - accuracy: 0.9957 - val_loss: 0.1326 - val_accuracy: 0.9942
Epoch 5/10
44/44 [==============================] - 0s 7ms/step - loss: 0.0757 - accuracy: 0.9906 - val_loss: 0.1183 - val_accuracy: 0.9942
Epoch 6/10
44/44 [==============================] - 0s 6ms/step - loss: 0.0395 - accuracy: 0.9957 - val_loss: 0.0278 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 0s 7ms/step - loss: 0.0027 - accuracy: 0.9993 - val_loss: 0.4784 - val_accuracy: 0.9652
Epoch 8/10
44/44 [==============================] - 0s 5ms/step - loss: 0.0057 - accuracy: 0.9993 - val_loss: 3.7427e-05 - val_accuracy: 1.0000
Epoch 9/10
44/44 [==============================] - 0s 6ms/step - loss: 2.0342e-08 - accuracy: 1.0000 - val_loss: 9.0232e-06 - val_accuracy: 1.0000
Epoch 10/10
44/44 [==============================] - 0s 5ms/step - loss: 3.3443e-08 - accuracy: 1.0000 - val_loss: 8.9201e-06 - val_accuracy: 1.0000
Training Time Per Epoch: 1.3463361024856568 seconds
```






```
Evaluating the model accuracy with test folder data...
14/14 [==============================] - 1s 37ms/step - loss: 1.5705e-06 - accuracy: 1.0000
Test Accuracy: 100.0 %
14/14 [==============================] - 0s 2ms/step
<Figure size 1000x800 with 0 Axes>
```



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

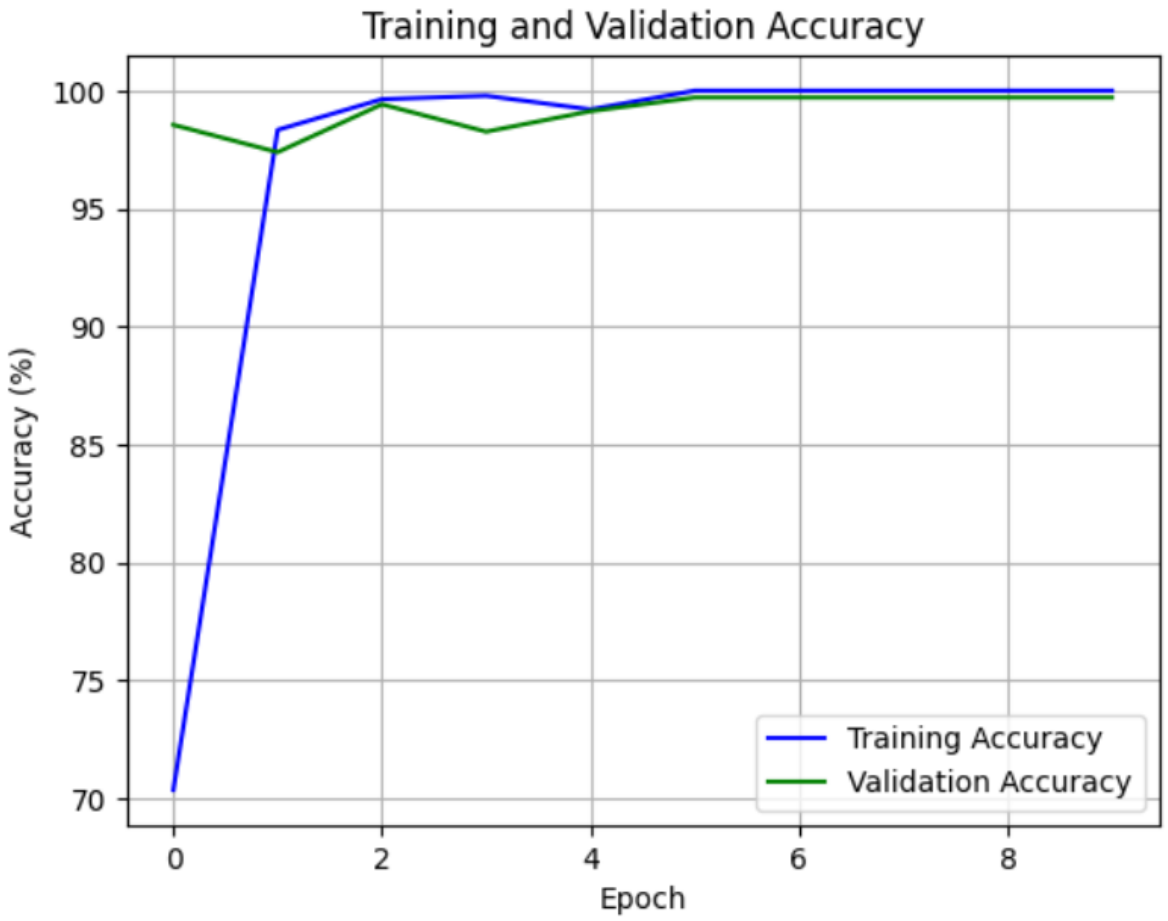
Deployed Google Colab runtime for this classification:

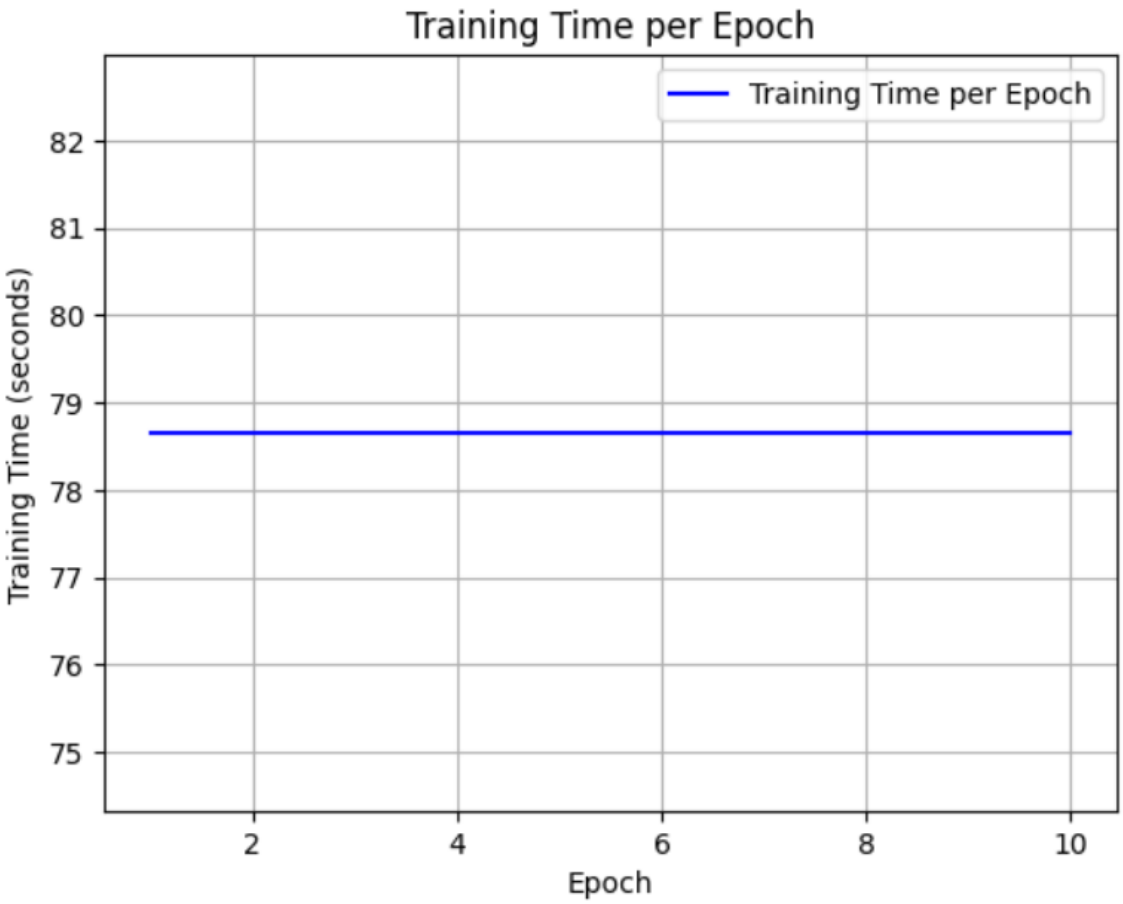
- Machine: CPU
- System RAM - Available - 12.7 GB
- Disk - Available – 107.7GB
- Python 3 Google Compute Engine Backend

```
Preparing training data set...
Image size used for training 32x768
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

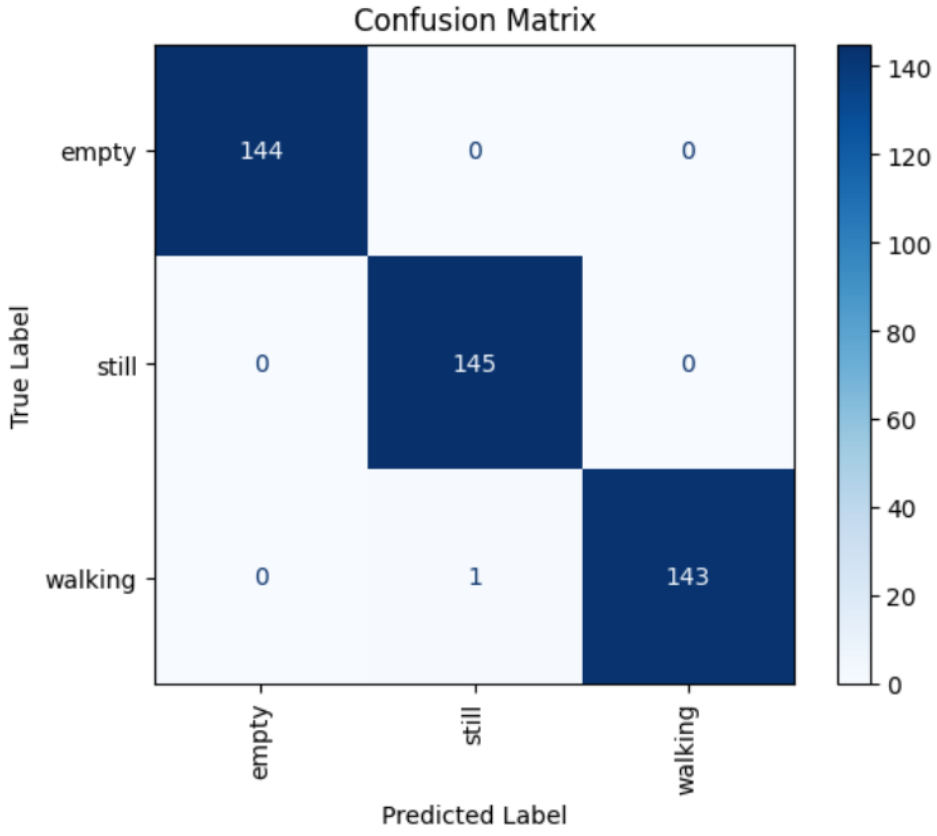
```
Model: "CNN_with_Multiple_Hidden_Layers_RELU_Activation_with_Original_Image_Training_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 768, 32, 3)        0

 conv2d_3 (Conv2D)           (None, 768, 32, 32)       896

 max_pooling2d_3 (MaxPoolin  (None, 384, 16, 32)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 382, 14, 64)       18496

 max_pooling2d_4 (MaxPoolin  (None, 191, 7, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 189, 5, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 94, 2, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 24064)             0

 dense_18 (Dense)            (None, 128)               3080320

 dense_19 (Dense)            (None, 3)                 387

=================================================================
Total params: 3173955 (12.11 MB)
Trainable params: 3173955 (12.11 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Activation Functions:
rescaling_4 - No activation
conv2d_3 - relu
max_pooling2d_3 - No activation
conv2d_4 - relu
max_pooling2d_4 - No activation
conv2d_5 - relu
max_pooling2d_5 - No activation
flatten_7 - No activation
dense_18 - relu
dense_19 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 82s 2s/step - loss: 0.6270 - accuracy: 0.7035 - val_loss: 0.0487 - val_accuracy: 0.9855
Epoch 2/10
44/44 [==============================] - 72s 2s/step - loss: 0.0632 - accuracy: 0.9834 - val_loss: 0.0896 - val_accuracy: 0.9739
Epoch 3/10
44/44 [==============================] - 71s 2s/step - loss: 0.0164 - accuracy: 0.9964 - val_loss: 0.0197 - val_accuracy: 0.9942
Epoch 4/10
44/44 [==============================] - 75s 2s/step - loss: 0.0046 - accuracy: 0.9978 - val_loss: 0.0922 - val_accuracy: 0.9826
Epoch 5/10
44/44 [==============================] - 69s 2s/step - loss: 0.0221 - accuracy: 0.9920 - val_loss: 0.0234 - val_accuracy: 0.9913
Epoch 6/10
44/44 [==============================] - 67s 2s/step - loss: 6.8929e-04 - accuracy: 1.0000 - val_loss: 0.0117 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 70s 2s/step - loss: 1.5981e-04 - accuracy: 1.0000 - val_loss: 0.0110 - val_accuracy: 0.9971
Epoch 8/10
44/44 [==============================] - 72s 2s/step - loss: 1.0387e-04 - accuracy: 1.0000 - val_loss: 0.0078 - val_accuracy: 0.9971
Epoch 9/10
44/44 [==============================] - 73s 2s/step - loss: 1.0189e-04 - accuracy: 1.0000 - val_loss: 0.0086 - val_accuracy: 0.9971
Epoch 10/10
44/44 [==============================] - 75s 2s/step - loss: 7.6940e-05 - accuracy: 1.0000 - val_loss: 0.0094 - val_accuracy: 0.9971
Training Time Per Epoch: 78.65565650463104 seconds
```






```
14/14 [==============================] - 9s 534ms/step - loss: 0.0043 - accuracy: 0.9977
Test Accuracy: 99.76905584335327 %
14/14 [==============================] - 5s 385ms/step
<Figure size 1000x800 with 0 Axes>
```



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

```
Preparing training data set...
Image size used for training 16x390
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

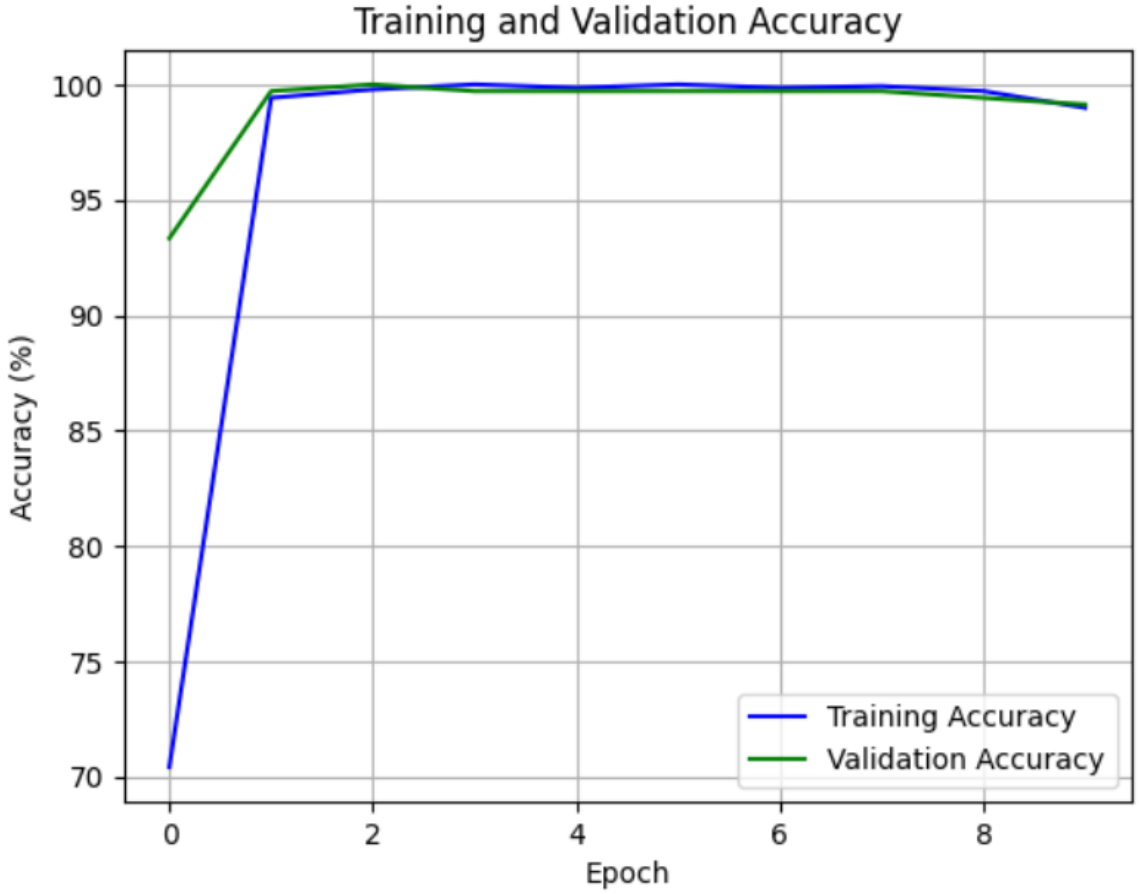
Model: "CNN_with_Multiple_Hidden_Layers_RELU_Activation_with_Non_Original_Image_Training_2"

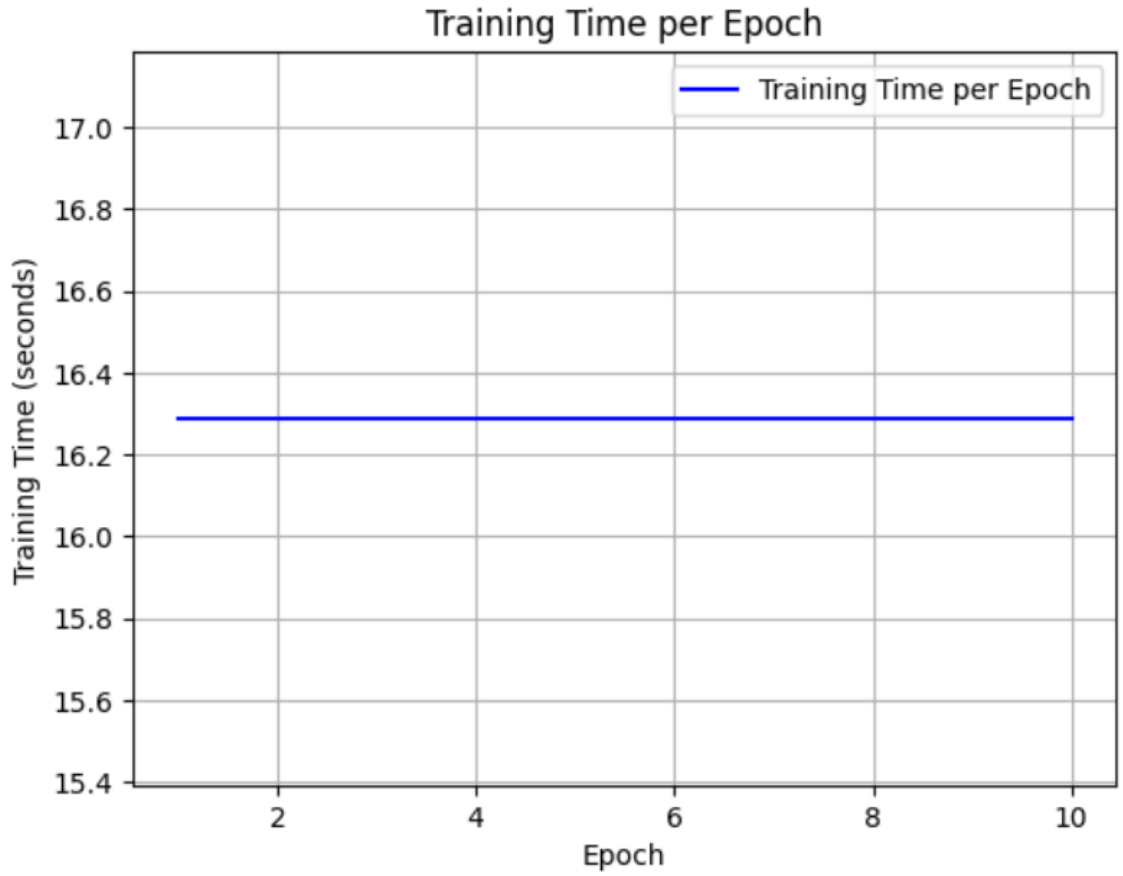
Layer (type)	Output Shape	Param #
rescaling_2 (Rescaling)	(None, 390, 16, 3)	0
conv2d_6 (Conv2D)	(None, 390, 16, 12)	336
max_pooling2d_6 (MaxPooling2D)	(None, 195, 8, 12)	0
conv2d_7 (Conv2D)	(None, 195, 8, 24)	2616
max_pooling2d_7 (MaxPooling2D)	(None, 98, 4, 24)	0
conv2d_8 (Conv2D)	(None, 96, 2, 48)	10416
max_pooling2d_8 (MaxPooling2D)	(None, 48, 1, 48)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_4 (Dense)	(None, 128)	295040
dense_5 (Dense)	(None, 3)	387

Total params: 308795 (1.18 MB)
Trainable params: 308795 (1.18 MB)
Non-trainable params: 0 (0.00 Byte)

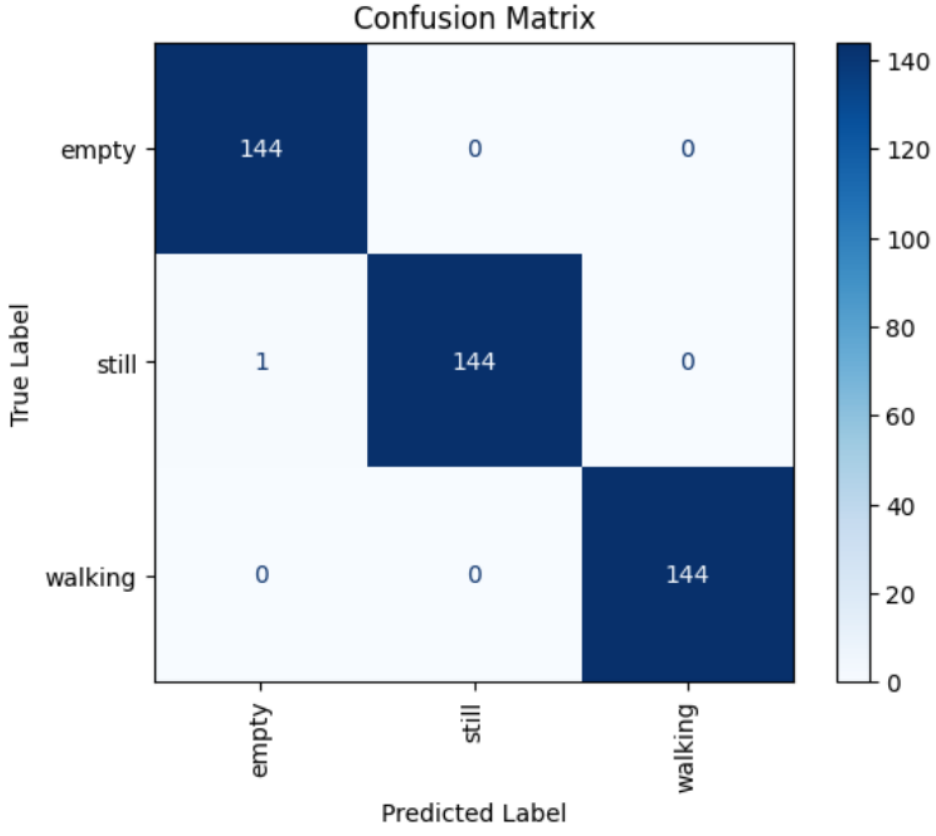
```
Activation Functions:
rescaling_2 - No activation
conv2d_6 - relu
max_pooling2d_6 - No activation
conv2d_7 - relu
max_pooling2d_7 - No activation
conv2d_8 - relu
max_pooling2d_8 - No activation
flatten_2 - No activation
dense_4 - relu
dense_5 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 54s 1s/step - loss: 0.6550 - accuracy: 0.7043 - val_loss: 0.1805 - val_accuracy: 0.9333
Epoch 2/10
44/44 [==============================] - 8s 182ms/step - loss: 0.0431 - accuracy: 0.9942 - val_loss: 0.0096 - val_accuracy: 0.9971
Epoch 3/10
44/44 [==============================] - 6s 146ms/step - loss: 0.0099 - accuracy: 0.9978 - val_loss: 0.0093 - val_accuracy: 1.0000
Epoch 4/10
44/44 [==============================] - 8s 184ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.0101 - val_accuracy: 0.9971
Epoch 5/10
44/44 [==============================] - 8s 179ms/step - loss: 0.0037 - accuracy: 0.9986 - val_loss: 0.0209 - val_accuracy: 0.9971
Epoch 6/10
44/44 [==============================] - 6s 145ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.0088 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 9s 195ms/step - loss: 0.0024 - accuracy: 0.9986 - val_loss: 0.0098 - val_accuracy: 0.9971
Epoch 8/10
44/44 [==============================] - 6s 146ms/step - loss: 0.0045 - accuracy: 0.9993 - val_loss: 0.0144 - val_accuracy: 0.9971
Epoch 9/10
44/44 [==============================] - 8s 186ms/step - loss: 0.0064 - accuracy: 0.9971 - val_loss: 0.0250 - val_accuracy: 0.9942
Epoch 10/10
44/44 [==============================] - 8s 175ms/step - loss: 0.0324 - accuracy: 0.9899 - val_loss: 0.0125 - val_accuracy: 0.9913
Training Time Per Epoch: 16.288434004783632 seconds
```






```
14/14 [==============================] - 54s 1s/step - loss: 0.0139 - accuracy: 0.9977
Test Accuracy: 99.76905584335327 %
14/14 [==============================] - 1s 45ms/step
<Figure size 1000x800 with 0 Axes>
```



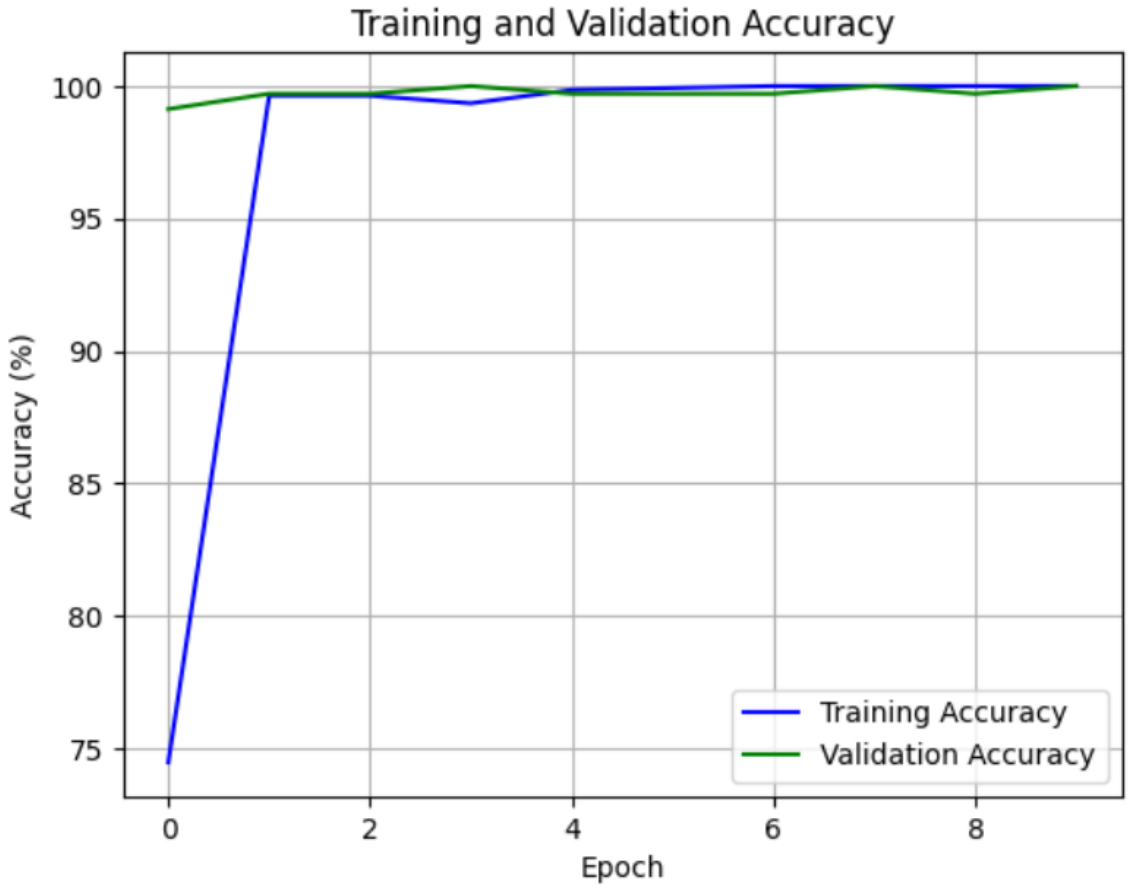
Multiple Hidden Layer CNN-based Model-3 Test Accuracy and Performance

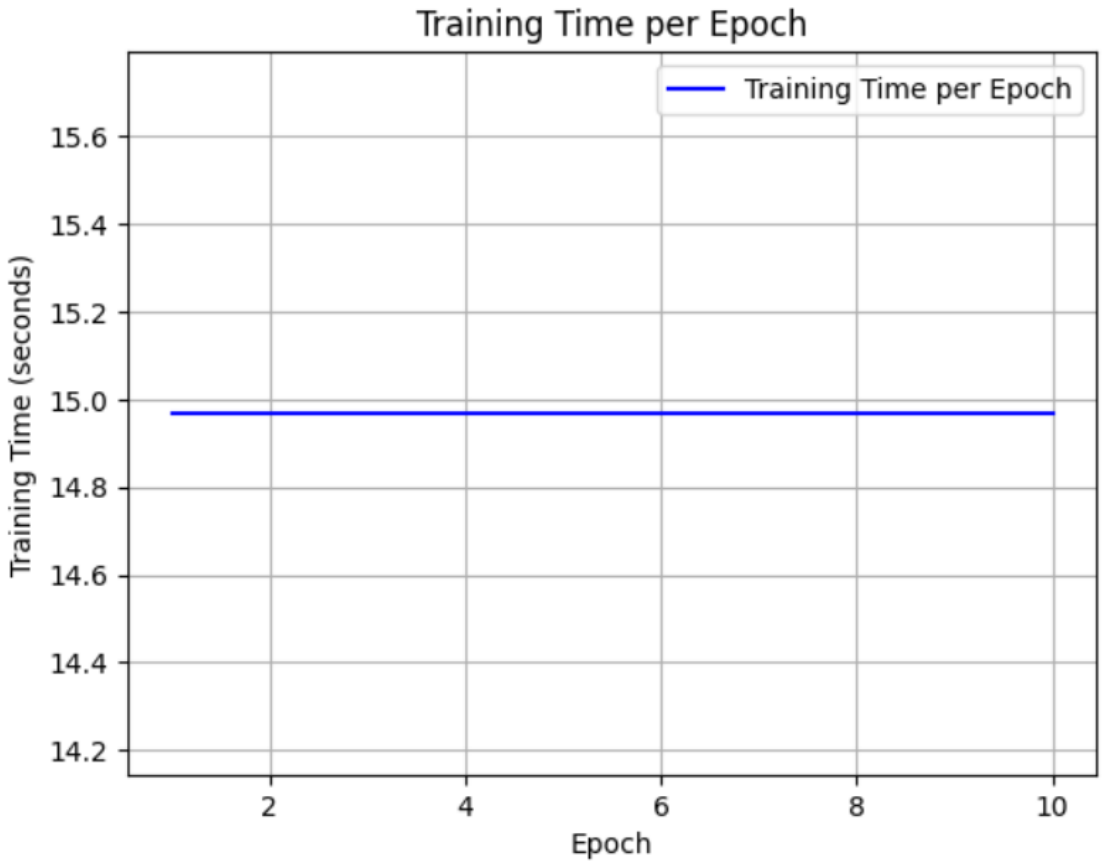
Deployed Google Colab runtime for this classification:

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

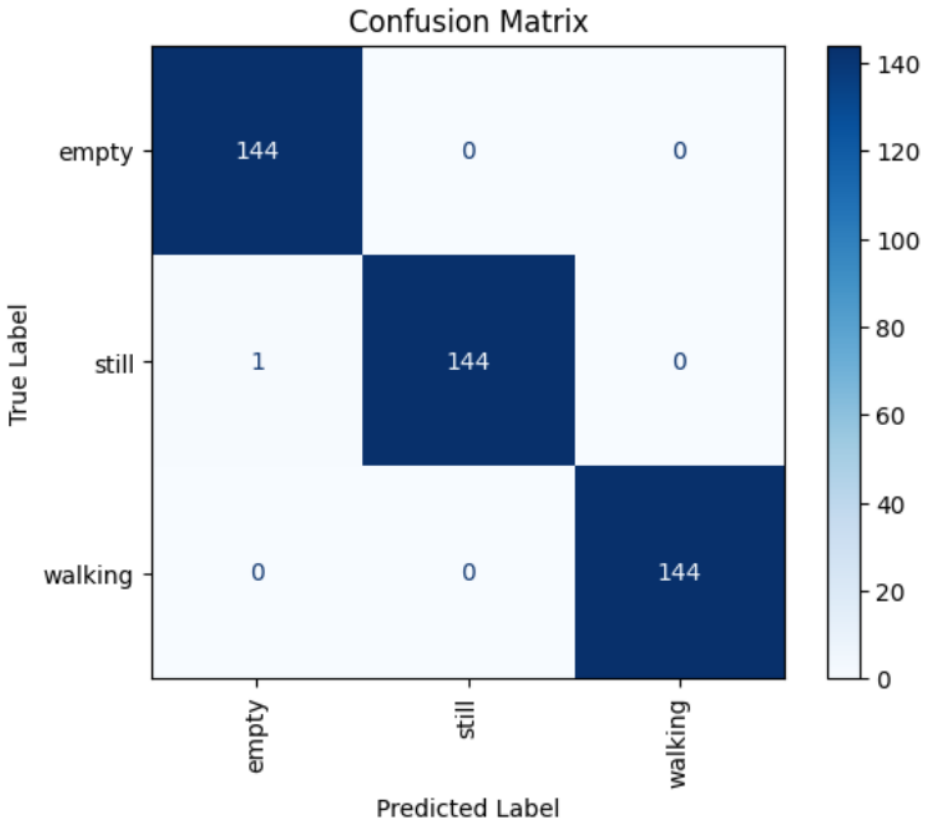
```
Image size used for training 16x390
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
Model: "CNN_with_Multiple_Hidden_Layers_RELU_Activation_with_Non_Original_Image_Training_GDV_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 390, 16, 3)        0

 conv2d (Conv2D)             (None, 390, 16, 12)       336

 max_pooling2d (MaxPooling2D  (None, 195, 8, 12)       0
 )

 conv2d_1 (Conv2D)           (None, 195, 8, 24)        2616

 max_pooling2d_1 (MaxPooling  (None, 98, 4, 24)        0
 2D)

 conv2d_2 (Conv2D)           (None, 96, 2, 48)         10416

 max_pooling2d_2 (MaxPooling  (None, 48, 1, 48)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense_2 (Dense)             (None, 128)               295040

 dense_3 (Dense)             (None, 3)                 387

=================================================================
Total params: 308,795
Trainable params: 308,795
Non-trainable params: 0
_________________________________________________________________
Activation Functions:
rescaling - No activation
conv2d - relu
max_pooling2d - No activation
conv2d_1 - relu
max_pooling2d_1 - No activation
conv2d_2 - relu
max_pooling2d_2 - No activation
flatten - No activation
dense_2 - relu
dense_3 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 16s 250ms/step - loss: 0.5370 - accuracy: 0.7448 - val_loss: 0.0775 - val_accuracy: 0.9913
Epoch 2/10
44/44 [==============================] - 8s 174ms/step - loss: 0.0215 - accuracy: 0.9964 - val_loss: 0.0086 - val_accuracy: 0.9971
Epoch 3/10
44/44 [==============================] - 11s 258ms/step - loss: 0.0137 - accuracy: 0.9964 - val_loss: 0.0099 - val_accuracy: 0.9971
Epoch 4/10
44/44 [==============================] - 12s 286ms/step - loss: 0.0184 - accuracy: 0.9935 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 5/10
44/44 [==============================] - 10s 240ms/step - loss: 0.0058 - accuracy: 0.9986 - val_loss: 0.0153 - val_accuracy: 0.9971
Epoch 6/10
44/44 [==============================] - 10s 234ms/step - loss: 0.0047 - accuracy: 0.9993 - val_loss: 0.0086 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 7s 169ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0053 - val_accuracy: 0.9971
Epoch 8/10
44/44 [==============================] - 10s 232ms/step - loss: 4.4150e-04 - accuracy: 1.0000 - val_loss: 0.0049 - val_accuracy: 1.0000
Epoch 9/10
44/44 [==============================] - 8s 179ms/step - loss: 1.5604e-04 - accuracy: 1.0000 - val_loss: 0.0029 - val_accuracy: 0.9971
Epoch 10/10
44/44 [==============================] - 10s 220ms/step - loss: 1.1555e-04 - accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Training Time Per Epoch: 14.968262076377869 seconds
```






```
14/14 [==============================] - 2s 84ms/step - loss: 0.0071 - accuracy: 0.9977
Test Accuracy: 99.76905584335327 %
14/14 [==============================] - 1s 60ms/step
<Figure size 1000x800 with 0 Axes>
```



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

Deployed Google Colab runtime for this classification:

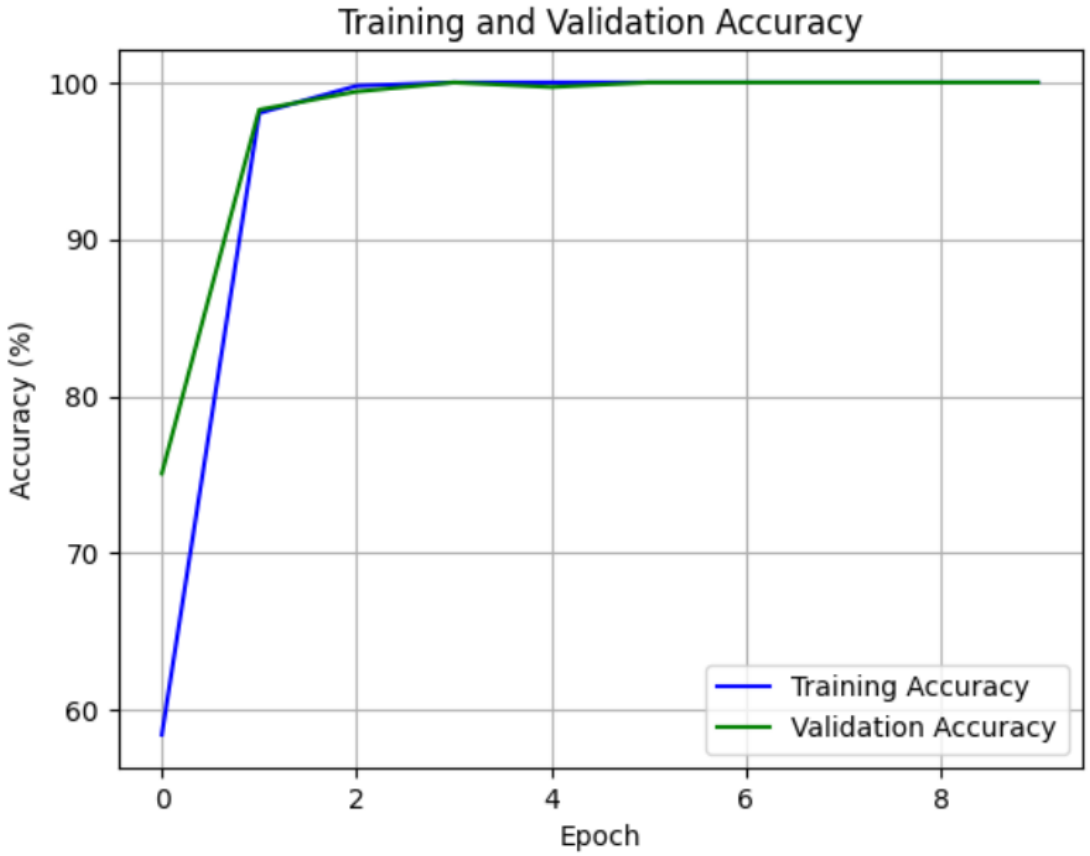
- Machine: V100
- System RAM - Available - 12.7 GB
- GPU RAM – available – 16 GB
- Disk Available – 78.2 GB
- Python 3 Google Compute Engine Backend

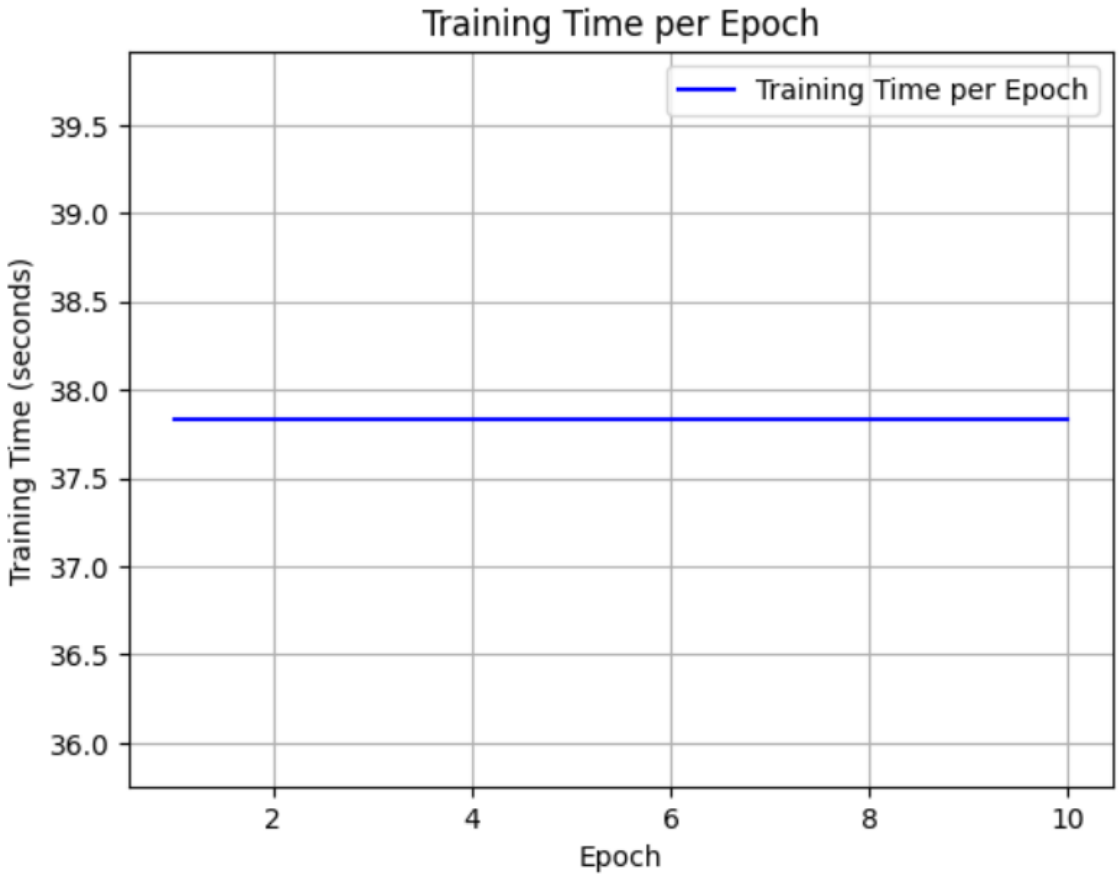
```
Preparing training data set...
Image size used for training 32x768
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

```
Model: "CNN_with_Multiple_Hidden_Layers_RELU_Activation_with_Original_Image_Size_Training_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 768, 32, 3)        0

 conv2d_3 (Conv2D)           (None, 768, 32, 12)       336

 max_pooling2d_3 (MaxPooling  (None, 384, 16, 12)      0
 2D)

 conv2d_4 (Conv2D)           (None, 382, 14, 24)       2616

 max_pooling2d_4 (MaxPooling  (None, 191, 7, 24)       0
 2D)

 flatten_1 (Flatten)         (None, 32088)             0

 dense_4 (Dense)             (None, 128)               4107392

 dense_5 (Dense)             (None, 3)                 387

=================================================================
Total params: 4,110,731
Trainable params: 4,110,731
Non-trainable params: 0
_________________________________________________________________
```

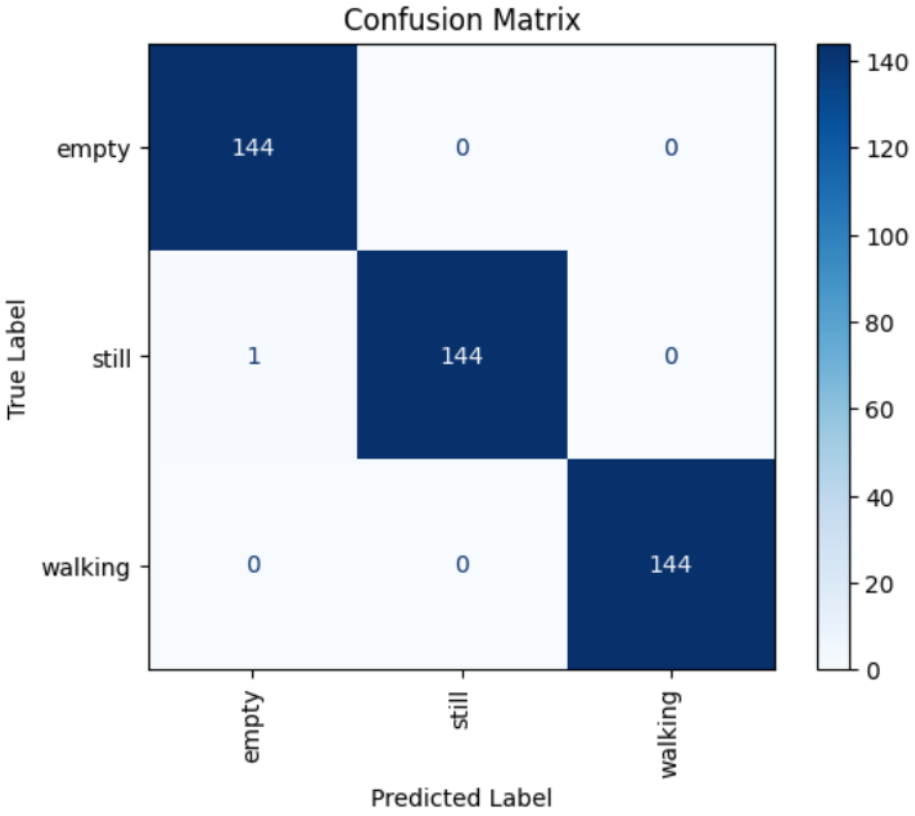
```
Activation Functions:
rescaling_1 - No activation
conv2d_3 - relu
max_pooling2d_3 - No activation
conv2d_4 - relu
max_pooling2d_4 - No activation
flatten_1 - No activation
dense_4 - relu
dense_5 - softmax
```

```
Epoch 1/10
44/44 [==============================] - 39s 790ms/step - loss: 1.3831 - accuracy: 0.5842 - val_loss: 0.4246 - val_accuracy: 0.7507
Epoch 2/10
44/44 [==============================] - 33s 751ms/step - loss: 0.1349 - accuracy: 0.9805 - val_loss: 0.0582 - val_accuracy: 0.9826
Epoch 3/10
44/44 [==============================] - 37s 832ms/step - loss: 0.0169 - accuracy: 0.9978 - val_loss: 0.0240 - val_accuracy: 0.9942
Epoch 4/10
44/44 [==============================] - 33s 739ms/step - loss: 0.0065 - accuracy: 1.0000 - val_loss: 0.0064 - val_accuracy: 1.0000
Epoch 5/10
44/44 [==============================] - 33s 733ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.0087 - val_accuracy: 0.9971
Epoch 6/10
44/44 [==============================] - 33s 761ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.0065 - val_accuracy: 1.0000
Epoch 7/10
44/44 [==============================] - 32s 739ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0042 - val_accuracy: 1.0000
Epoch 8/10
44/44 [==============================] - 35s 785ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0061 - val_accuracy: 1.0000
Epoch 9/10
44/44 [==============================] - 33s 755ms/step - loss: 5.9261e-04 - accuracy: 1.0000 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 10/10
44/44 [==============================] - 33s 740ms/step - loss: 4.2200e-04 - accuracy: 1.0000 - val_loss: 0.0039 - val_accuracy: 1.0000
Training Time Per Epoch: 37.83412370681763 seconds
```






```
14/14 [==============================] - 6s 372ms/step - loss: 0.0062 - accuracy: 0.9977
Test Accuracy: 99.76905584335327 %
14/14 [==============================] - 3s 235ms/step
<Figure size 1000x800 with 0 Axes>
```



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

```
Preparing training data set...
Image size used for training 32x768
Validation split of the training samples 20.0%
Found 1728 files belonging to 3 classes.
Using 1383 files for training.
Done - Preparing training data set.
Preparing validation data set...
Found 1728 files belonging to 3 classes.
Using 345 files for validation.
Done - Preparing validation data set.
Preparing test data set...
Found 433 files belonging to 3 classes.
Done - Preparing test data set.
List of all training classes: ['empty', 'still', 'walking']
```

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 (BatchNormalization)	(None, None, None, 64)	256	['conv1_conv[0][0]']
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 (Conv2D)	(None, None, None, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchNormalization)	(None, None, None, 64)	256	['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)	(None, None, None, 64)	0	['conv2_block1_1_bn[0][0]']
conv2_block1_2_conv (Conv2D)	(None, None, None, 64)	36928	['conv2_block1_1_relu[0][0]']
conv2_block1_2_bn (BatchNormalization)	(None, None, None, 64)	256	['conv2_block1_2_conv[0][0]']

conv2_block1_2_relu (Activ ation) (None, None, None, 64) 0 ['conv2_block1_2_bn[0][0]']

conv2_block1_0_conv (Conv2D) (None, None, None, 256) 16640 ['pool1_pool[0][0]']

conv2_block1_3_conv (Conv2D) (None, None, None, 256) 16640 ['conv2_block1_2_relu[0][0]']

conv2_block1_0_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv2_block1_0_conv[0][0]']

conv2_block1_3_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv2_block1_3_conv[0][0]']

conv2_block1_add (Add) (None, None, None, 256) 0 ['conv2_block1_0_bn[0][0]', 'conv2_block1_3_bn[0][0]']

conv2_block1_out (Activation) (None, None, None, 256) 0 ['conv2_block1_add[0][0]']

conv2_block2_1_conv (Conv2D) (None, None, None, 64) 16448 ['conv2_block1_out[0][0]']

conv2_block2_1_bn (BatchNormalization) (None, None, None, 64) 256 ['conv2_block2_1_conv[0][0]']

conv2_block2_1_relu (Activation) (None, None, None, 64) 0 ['conv2_block2_1_bn[0][0]']

conv2_block2_2_conv (Conv2D) (None, None, None, 64) 36928 ['conv2_block2_1_relu[0][0]']

conv2_block2_2_bn (BatchNormalization) (None, None, None, 64) 256 ['conv2_block2_2_conv[0][0]']

conv2_block2_2_relu (Activation) (None, None, None, 64) 0 ['conv2_block2_2_bn[0][0]']

conv2_block2_3_conv (Conv2D) (None, None, None, 256) 16640 ['conv2_block2_2_relu[0][0]']

conv2_block2_3_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv2_block2_3_conv[0][0]']

conv2_block2_add (Add) (None, None, None, 256) 0 ['conv2_block1_out[0][0]', 'conv2_block2_3_bn[0][0]']

conv2_block2_out (Activation) (None, None, None, 256) 0 ['conv2_block2_add[0][0]']

conv2_block3_1_conv (Conv2D) (None, None, None, 64) 16448 ['conv2_block2_out[0][0]']

conv2_block3_1_bn (BatchNormalization) (None, None, None, 64) 256 ['conv2_block3_1_conv[0][0]']

conv2_block3_1_relu (Activation) (None, None, None, 64) 0 ['conv2_block3_1_bn[0][0]']

conv2_block3_2_conv (Conv2D) (None, None, None, 64) 36928 ['conv2_block3_1_relu[0][0]']

conv2_block3_2_bn (BatchNormalization) (None, None, None, 64) 256 ['conv2_block3_2_conv[0][0]']

conv2_block3_2_relu (Activation) (None, None, None, 64) 0 ['conv2_block3_2_bn[0][0]']

conv2_block3_3_conv (Conv2D) (None, None, None, 256) 16640 ['conv2_block3_2_relu[0][0]']

conv2_block3_3_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv2_block3_3_conv[0][0]']

conv2_block3_add (Add) (None, None, None, 256) 0 ['conv2_block2_out[0][0]', 'conv2_block3_3_bn[0][0]']

conv2_block3_out (Activation) (None, None, None, 256) 0 ['conv2_block3_add[0][0]']

conv3_block1_1_conv (Conv2D) (None, None, None, 128) 32896 ['conv2_block3_out[0][0]']

conv3_block1_1_bn (BatchNormalization) (None, None, None, 128) 512 ['conv3_block1_1_conv[0][0]']

conv3_block1_1_relu (Activation) (None, None, None, 128) 0 ['conv3_block1_1_bn[0][0]']

conv3_block1_2_conv (Conv2D) (None, None, None, 128) 147584 ['conv3_block1_1_relu[0][0]']

conv3_block1_2_bn (BatchNormalization) (None, None, None, 128) 512 ['conv3_block1_2_conv[0][0]']

conv3_block1_2_relu (Activation) (None, None, None, 128) 0 ['conv3_block1_2_bn[0][0]']

conv3_block1_0_conv (Conv2D) (None, None, None, 512) 131584 ['conv2_block3_out[0][0]']

conv3_block1_3_conv (Conv2D) (None, None, None, 512) 66048 ['conv3_block1_2_relu[0][0]']

conv3_block1_0_bn (BatchNormalization) (None, None, None, 512) 2048 ['conv3_block1_0_conv[0][0]']

conv3_block1_3_bn (BatchNormalization) (None, None, None, 512) 2048 ['conv3_block1_3_conv[0][0]']

conv3_block1_add (Add) (None, None, None, 512) 0 ['conv3_block1_0_bn[0][0]', 'conv3_block1_3_bn[0][0]']

conv3_block1_out (Activation) (None, None, None, 512) 0 ['conv3_block1_add[0][0]']

conv3_block2_1_conv (Conv2D) (None, None, None, 128) 65664 ['conv3_block1_out[0][0]']

conv3_block2_1_bn (BatchNormalization) (None, None, None, 128) 512 ['conv3_block2_1_conv[0][0]']

conv3_block2_1_relu (Activation) (None, None, None, 128) 0 ['conv3_block2_1_bn[0][0]']

conv3_block2_2_conv (Conv2D) (None, None, None, 128) 147584 ['conv3_block2_1_relu[0][0]']

conv3_block2_2_bn (BatchNormalization) (None, None, None, 128) 512 ['conv3_block2_2_conv[0][0]']

conv3_block2_2_relu (Activation) (None, None, None, 128) 0 ['conv3_block2_2_bn[0][0]']

conv3_block2_3_conv (Conv2D) (None, None, None, 512) 66048 ['conv3_block2_2_relu[0][0]']

conv3_block2_3_bn (BatchNormalization) (None, None, None, 512) 2048 ['conv3_block2_3_conv[0][0]']

conv3_block2_add (Add) (None, None, None, 512) 0 ['conv3_block1_out[0][0]', 'conv3_block2_3_bn[0][0]']

conv3_block2_out (Activation) (None, None, None, 512) 0 ['conv3_block2_add[0][0]']

conv3_block3_1_conv (Conv2D) (None, None, None, 128) 65664 ['conv3_block2_out[0][0]']

conv3_block3_1_bn (BatchNo rmalization) (None, None, None, 128) 512 ['conv3_block3_1_conv[0][0]']

conv3_block3_1_relu (Activ ation) (None, None, None, 128) 0 ['conv3_block3_1_bn[0][0]']

conv3_block3_2_conv (Conv2 D) (None, None, None, 128) 147584 ['conv3_block3_1_relu[0][0]']

conv3_block3_2_bn (BatchNo rmalization) (None, None, None, 128) 512 ['conv3_block3_2_conv[0][0]']

conv3_block3_2_relu (Activ ation) (None, None, None, 128) 0 ['conv3_block3_2_bn[0][0]']

conv3_block3_3_conv (Conv2 D) (None, None, None, 512) 66048 ['conv3_block3_2_relu[0][0]']

conv3_block3_3_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv3_block3_3_conv[0][0]']

conv3_block3_add (Add) (None, None, None, 512) 0 ['conv3_block2_out[0][0]', 'conv3_block3_3_bn[0][0]']

conv3_block3_out (Activati on) (None, None, None, 512) 0 ['conv3_block3_add[0][0]']

conv3_block4_1_conv (Conv2 D) (None, None, None, 128) 65664 ['conv3_block3_out[0][0]']

conv3_block4_1_bn (BatchNo rmalization) (None, None, None, 128) 512 ['conv3_block4_1_conv[0][0]']

conv3_block4_1_relu (Activ ation) (None, None, None, 128) 0 ['conv3_block4_1_bn[0][0]']

conv3_block4_2_conv (Conv2 D) (None, None, None, 128) 147584 ['conv3_block4_1_relu[0][0]']

conv3_block4_2_bn (BatchNo rmalization) (None, None, None, 128) 512 ['conv3_block4_2_conv[0][0]']

conv3_block4_2_relu (Activ ation) (None, None, None, 128) 0 ['conv3_block4_2_bn[0][0]']

conv3_block4_3_conv (Conv2 D) (None, None, None, 512) 66048 ['conv3_block4_2_relu[0][0]']

conv3_block4_3_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv3_block4_3_conv[0][0]']

conv3_block4_add (Add) (None, None, None, 512) 0 ['conv3_block3_out[0][0]', 'conv3_block4_3_bn[0][0]']

conv3_block4_out (Activati on) (None, None, None, 512) 0 ['conv3_block4_add[0][0]']

conv4_block1_1_conv (Conv2 D) (None, None, None, 256) 131328 ['conv3_block4_out[0][0]']

conv4_block1_1_bn (BatchNo rmalization) (None, None, None, 256) 1024 ['conv4_block1_1_conv[0][0]']

conv4_block1_1_relu (Activ ation) (None, None, None, 256) 0 ['conv4_block1_1_bn[0][0]']

conv4_block1_2_conv (Conv2 D) (None, None, None, 256) 590080 ['conv4_block1_1_relu[0][0]']

conv4_block1_2_bn (BatchNo rmalization) (None, None, None, 256) 1024 ['conv4_block1_2_conv[0][0]']

conv4_block1_2_relu (Activ ation) (None, None, None, 256) 0 ['conv4_block1_2_bn[0][0]']

conv4_block1_0_conv (Conv2D) (None, None, None, 1024) 525312 ['conv3_block4_out[0][0]']

conv4_block1_3_conv (Conv2D) (None, None, None, 1024) 263168 ['conv4_block1_2_relu[0][0]']

conv4_block1_0_bn (BatchNormalization) (None, None, None, 1024) 4096 ['conv4_block1_0_conv[0][0]']

conv4_block1_3_bn (BatchNormalization) (None, None, None, 1024) 4096 ['conv4_block1_3_conv[0][0]']

conv4_block1_add (Add) (None, None, None, 1024) 0 ['conv4_block1_0_bn[0][0]', 'conv4_block1_3_bn[0][0]']

conv4_block1_out (Activation) (None, None, None, 1024) 0 ['conv4_block1_add[0][0]']

conv4_block2_1_conv (Conv2D) (None, None, None, 256) 262400 ['conv4_block1_out[0][0]']

conv4_block2_1_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv4_block2_1_conv[0][0]']

conv4_block2_1_relu (Activation) (None, None, None, 256) 0 ['conv4_block2_1_bn[0][0]']

conv4_block2_2_conv (Conv2D) (None, None, None, 256) 590080 ['conv4_block2_1_relu[0][0]']

conv4_block2_2_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv4_block2_2_conv[0][0]']

conv4_block2_2_relu (Activation) (None, None, None, 256) 0 ['conv4_block2_2_bn[0][0]']

conv4_block2_3_conv (Conv2D) (None, None, None, 1024) 263168 ['conv4_block2_2_relu[0][0]']

conv4_block2_3_bn (BatchNormalization) (None, None, None, 1024) 4096 ['conv4_block2_3_conv[0][0]']

conv4_block2_add (Add) (None, None, None, 1024) 0 ['conv4_block1_out[0][0]', 'conv4_block2_3_bn[0][0]']

conv4_block2_out (Activation) (None, None, None, 1024) 0 ['conv4_block2_add[0][0]']

conv4_block3_1_conv (Conv2D) (None, None, None, 256) 262400 ['conv4_block2_out[0][0]']

conv4_block3_1_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv4_block3_1_conv[0][0]']

conv4_block3_1_relu (Activation) (None, None, None, 256) 0 ['conv4_block3_1_bn[0][0]']

conv4_block3_2_conv (Conv2D) (None, None, None, 256) 590080 ['conv4_block3_1_relu[0][0]']

conv4_block3_2_bn (BatchNormalization) (None, None, None, 256) 1024 ['conv4_block3_2_conv[0][0]']

conv4_block3_2_relu (Activation) (None, None, None, 256) 0 ['conv4_block3_2_bn[0][0]']

conv4_block3_3_conv (Conv2D) (None, None, None, 1024) 263168 ['conv4_block3_2_relu[0][0]']

conv4_block3_3_bn (BatchNormalization) (None, None, None, 1024) 4096 ['conv4_block3_3_conv[0][0]']

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 rmalization) (None, None, None, 256) 1024 ['conv4_block6_2_conv[0][0]']

conv4_block6_2_relu (Activ ation) (None, None, None, 256) 0 ['conv4_block6_2_bn[0][0]']

conv4_block6_3_conv (Conv2 D) (None, None, None, 1024) 263168 ['conv4_block6_2_relu[0][0]']

conv4_block6_3_bn (BatchNo rmalization) (None, None, None, 1024) 4096 ['conv4_block6_3_conv[0][0]']

conv4_block6_add (Add) (None, None, None, 1024) 0 ['conv4_block5_out[0][0]', 'conv4_block6_3_bn[0][0]']

conv4_block6_out (Activati on) (None, None, None, 1024) 0 ['conv4_block6_add[0][0]']

conv5_block1_1_conv (Conv2 D) (None, None, None, 512) 524800 ['conv4_block6_out[0][0]']

conv5_block1_1_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv5_block1_1_conv[0][0]']

conv5_block1_1_relu (Activ ation) (None, None, None, 512) 0 ['conv5_block1_1_bn[0][0]']

conv5_block1_2_conv (Conv2 D) (None, None, None, 512) 2359808 ['conv5_block1_1_relu[0][0]']

conv5_block1_2_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv5_block1_2_conv[0][0]']

conv5_block1_2_relu (Activ ation) (None, None, None, 512) 0 ['conv5_block1_2_bn[0][0]']

conv5_block1_0_conv (Conv2 D) (None, None, None, 2048) 2099200 ['conv4_block6_out[0][0]']

conv5_block1_3_conv (Conv2 D) (None, None, None, 2048) 1050624 ['conv5_block1_2_relu[0][0]']

conv5_block1_0_bn (BatchNo rmalization) (None, None, None, 2048) 8192 ['conv5_block1_0_conv[0][0]']

conv5_block1_3_bn (BatchNo rmalization) (None, None, None, 2048) 8192 ['conv5_block1_3_conv[0][0]']

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 on) (None, None, None, 2048) 0 ['conv5_block1_add[0][0]']

conv5_block2_1_conv (Conv2 D) (None, None, None, 512) 1049088 ['conv5_block1_out[0][0]']

conv5_block2_1_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv5_block2_1_conv[0][0]']

conv5_block2_1_relu (Activ ation) (None, None, None, 512) 0 ['conv5_block2_1_bn[0][0]']

conv5_block2_2_conv (Conv2 D) (None, None, None, 512) 2359808 ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNo rmalization) (None, None, None, 512) 2048 ['conv5_block2_2_conv[0][0]']

conv5_block2_2_relu (Activ ation) (None, None, None, 512) 0 ['conv5_block2_2_bn[0][0]']

conv5_block2_3_conv (Conv2 D) (None, None, None, 2048) 1050624 ['conv5_block2_2_relu[0][0]']

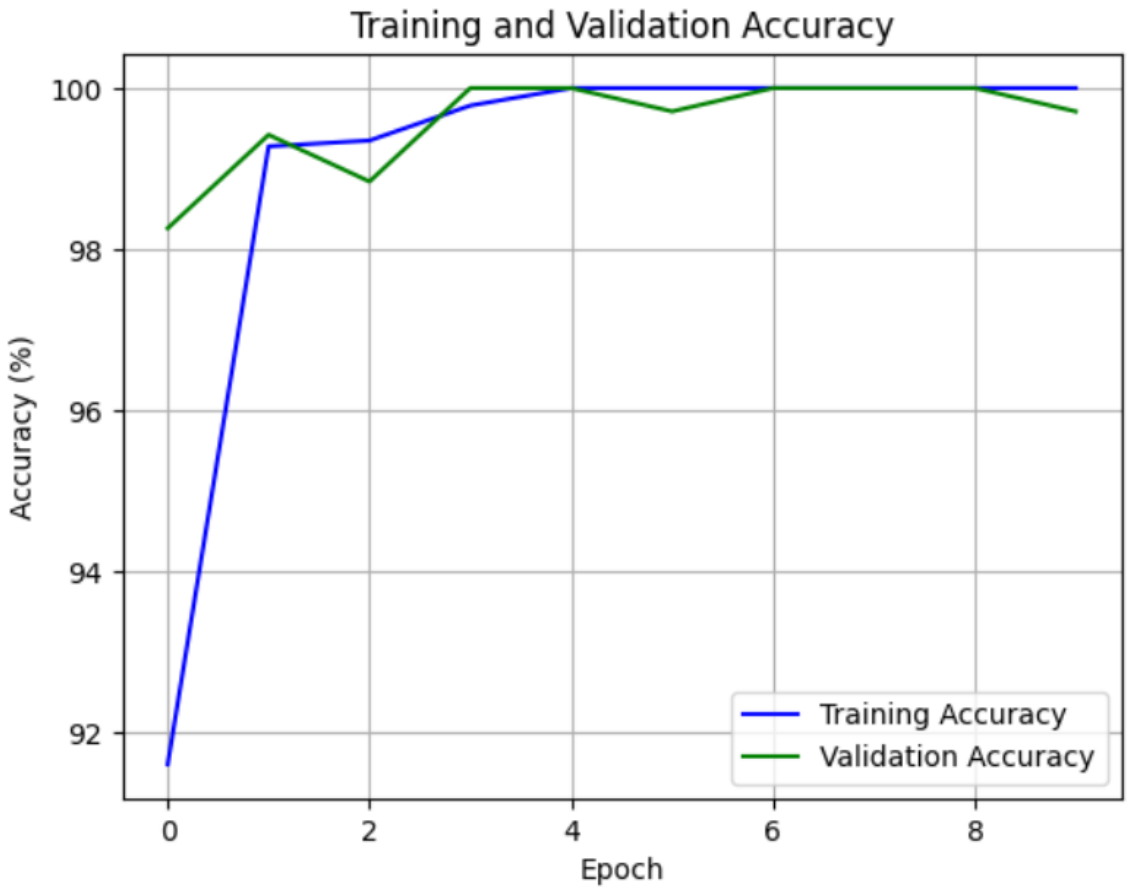
```
conv5_block2_3_bn (BatchNo  (None, None, None, 2048)    8192     ['conv5_block2_3_conv[0][0]']
rmalization)

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
```

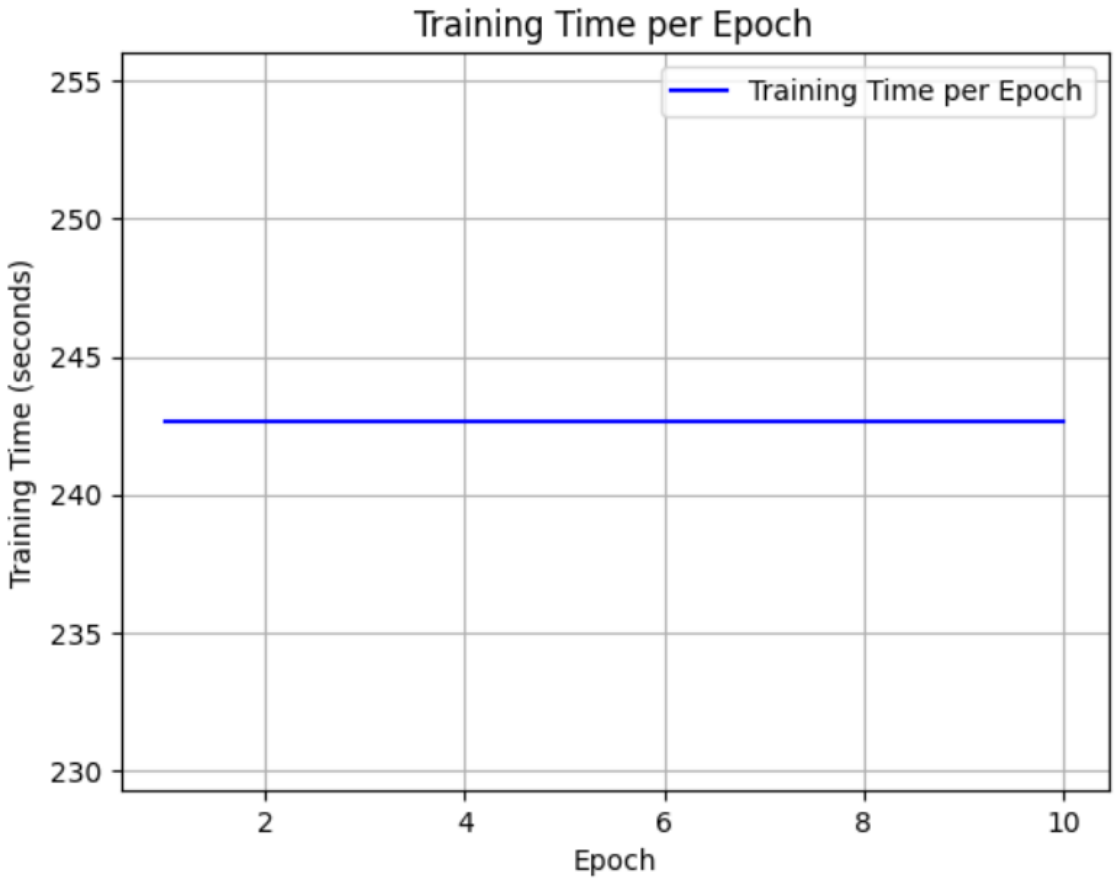
conv2_block2_2_conv - linear
conv2_block2_2_bn - No activation
conv2_block2_2_relu - relu
conv2_block2_3_conv - linear
conv2_block2_3_bn - No activation
conv2_block2_add - No activation
conv2_block2_out - relu
conv2_block3_1_conv - linear
conv2_block3_1_bn - No activation
conv2_block3_1_relu - relu
conv2_block3_2_conv - 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

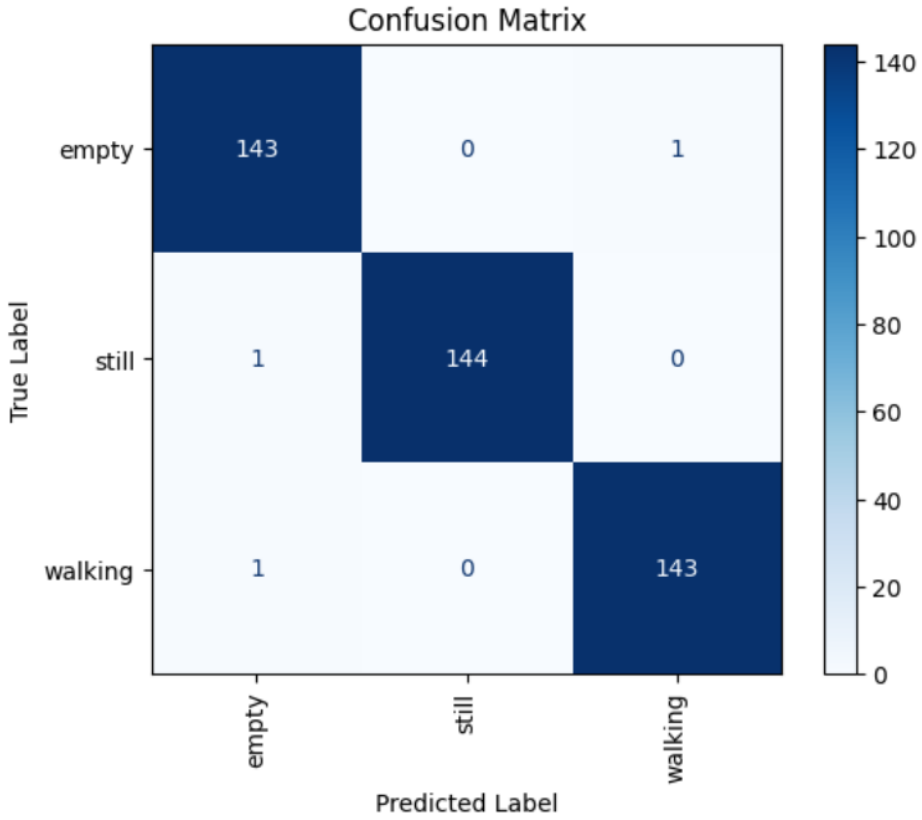
conv5_block3_3_bn - No activation
conv5_block3_add - No activation
conv5_block3_out - relu
global_average_pooling2d_5 - No activation
dense_10 - relu
dense_11 - softmax

```
Epoch 1/10
44/44 [==============================] - 313s 7s/step - loss: 0.4248 - accuracy: 0.9161 - val_loss: 0.0439 - val_accuracy: 0.9826
Epoch 2/10
44/44 [==============================] - 243s 6s/step - loss: 0.0255 - accuracy: 0.9928 - val_loss: 0.0112 - val_accuracy: 0.9942
Epoch 3/10
44/44 [==============================] - 245s 6s/step - loss: 0.0147 - accuracy: 0.9935 - val_loss: 0.0234 - val_accuracy: 0.9884
Epoch 4/10
44/44 [==============================] - 248s 6s/step - loss: 0.0095 - accuracy: 0.9978 - val_loss: 0.0042 - val_accuracy: 1.0000
Epoch 5/10
44/44 [==============================] - 244s 6s/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 6/10
44/44 [==============================] - 201s 5s/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0051 - val_accuracy: 0.9971
Epoch 7/10
44/44 [==============================] - 246s 6s/step - loss: 9.1407e-04 - accuracy: 1.0000 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 8/10
44/44 [==============================] - 197s 5s/step - loss: 9.0623e-04 - accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 9/10
44/44 [==============================] - 242s 6s/step - loss: 7.0356e-04 - accuracy: 1.0000 - val_loss: 0.0030 - val_accuracy: 1.0000
Epoch 10/10
44/44 [==============================] - 243s 6s/step - loss: 6.3628e-04 - accuracy: 1.0000 - val_loss: 0.0048 - val_accuracy: 0.9971
Done - Training the ResNet50 with training data.
Training Time Per Epoch: 242.65863223075866 seconds
```






```
14/14 [==============================] - 118s 4s/step - loss: 0.0151 - accuracy: 0.9931
Test Accuracy: 99.30716156959534 %
14/14 [==============================] - 32s 2s/step
<Figure size 1000x800 with 0 Axes>
```



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
```