Final Term Project – Machine Learning

Car Object Detection

Authors: Hemangi Kinger and Ishan Kuchroo

4/25/2022

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Final Term Project – Machine Learning

Table of Contents

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Final Term Project – Machine Learning

Acknowledgment

We thank Prof. Yuxiao Huang, GWU for guiding us and providing us with utilities to run Deep Learning models smoothly. We thank our classmates for keeping the Spring session interactive.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Final Term Project – Machine Learning

Abstract

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.

The final project introduces application of one of the subsets of Machine learning – **Deep Learning** – which was inspired by the structure and function of the brain. The idea is that with advancement in technology, faster computers, and substantial amount of data, we can build and scale large networks that can mimic human brain.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Final Term Project – Machine Learning

Introduction

Final project will talk about application of one of the Deep Learning methods called *Convolutional Neural Networks (or CNN)*. CNN is widely used in the field of computer vision wherein we train computers to derive meaningful information from the visual world.

One such application is “**Object Detection**” which is using digital images from cameras and videos and deep learning models so machines can accurately identify and classify objects — and then react to what they “see.” Object Detection is widely used by organizations working on self-driving vehicles or automatic cruise control, like Tesla.

We’ll be using CNN to analyze images of cars and train a model to accurately identify and classify if the object in an image is a car or not.

Methods – Theory and Procedure

Data Extraction/Preparation

1. **Car Object Data**:

Data Source – https://www.kaggle.com/datasets/sshikamaru/car-object-detection

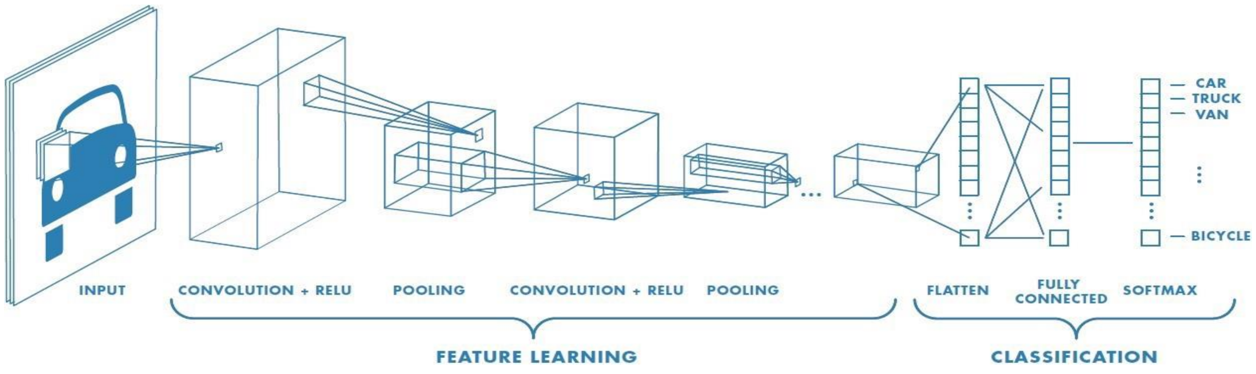
The dataset contains images of cars in all views. We have following data subsets:

- **Training Images** – Set of 1000 files
- **Testing images** – Set of 175 files
- **Train_solution_bounding_box** – Contains name of images and the dimensions which has the object "Car", in the training set.

Data Analysis – Methods

1. **Convolutional Neural Networks**:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. A CNN can capture the Spatial and Temporal intricacies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.



Reference: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

2. **TensorFlow**:

TensorFlow is an open-source collection of tools and libraries that allows researchers to build and deploy cutting-edge ML models. We'll be using various methods from TensorFlow like "image.resize", "cast" etc.

Reference: https://www.tensorflow.org/

3. **Keras**:

Keras is an open-source high-level neural network library that runs on top of TensorFlow. It provides a Python interface for artificial neural networks. We'll use Keras to import and run a pre-trained Neural Network called "ResNet50".

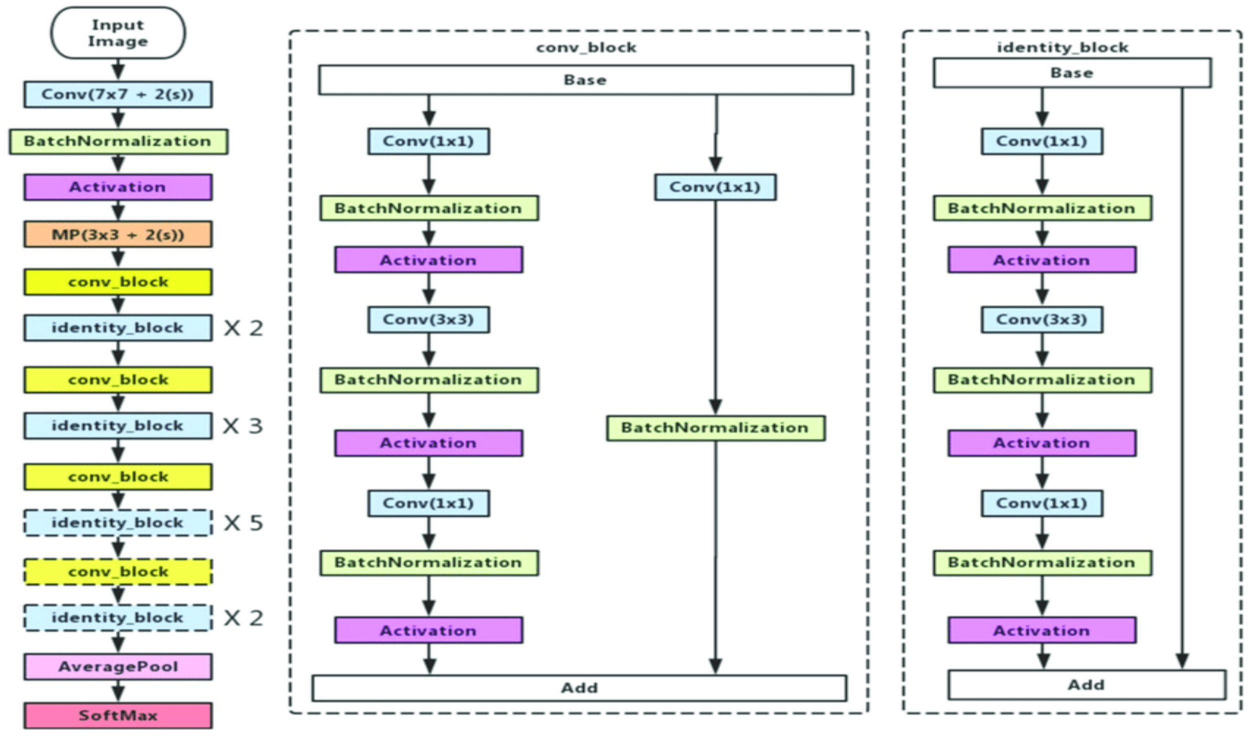
Reference: https://keras.io/

4. **ResNet50**:

ResNet or Residual Network is an innovative neural network that was first introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper titled 'Deep Residual Learning for Image Recognition'.

Resnet50 is one of the many ResNet variants that can work with 50 neural network layers.

Reference: https://viso.ai/deep-learning/resnet-residual-neural-network/



Model Training & Results

In this section, we'll talk about data preprocessing over the RAW data, importing ResNet50, selecting the best out of Adam or SGD optimizers by studying the validation loss and accuracy.

Pre-processing

1. Read Train_solution_bounding_box file in a dataframe. The dataframe contains image name, x-axis and y-axis dimensions of the "car" in the image.

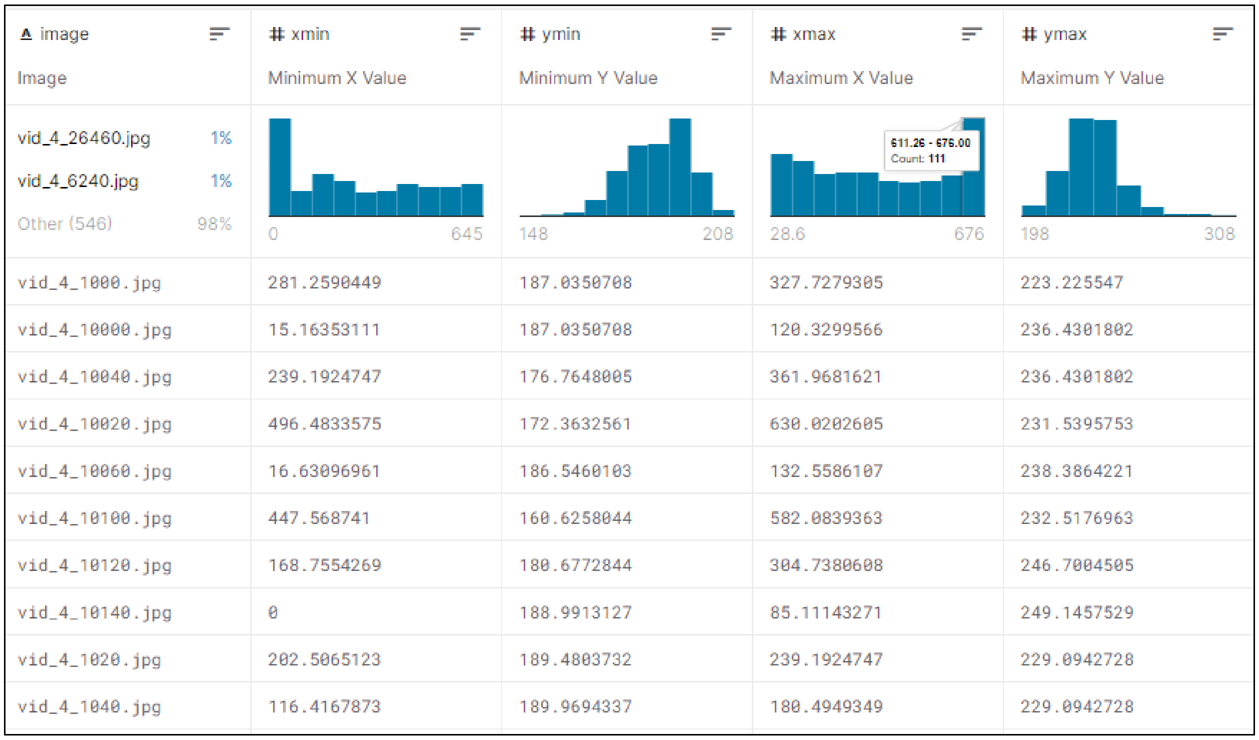


image	xmin	ymin	xmax	ymax
Image	Minimum X Value	Minimum Y Value	Maximum X Value	Maximum Y Value
vid_4_1000.jpg	281.2590449	187.0350708	327.7279305	223.225547
vid_4_10000.jpg	15.16353111	187.0350708	120.3299566	236.4301802
vid_4_10040.jpg	239.1924747	176.7648005	361.9681621	236.4301802
vid_4_10020.jpg	496.4833575	172.3632561	630.0202605	231.5395753
vid_4_10060.jpg	16.63096961	186.5460103	132.5586107	238.3864221
vid_4_10100.jpg	447.568741	160.6258044	582.0839363	232.5176963
vid_4_10120.jpg	168.7554269	180.6772844	304.7380608	246.7004505
vid_4_10140.jpg	0	188.9913127	85.11143271	249.1457529
vid_4_1020.jpg	202.5065123	189.4803732	239.1924747	229.0942728
vid_4_1040.jpg	116.4167873	189.9694337	180.4949349	229.0942728

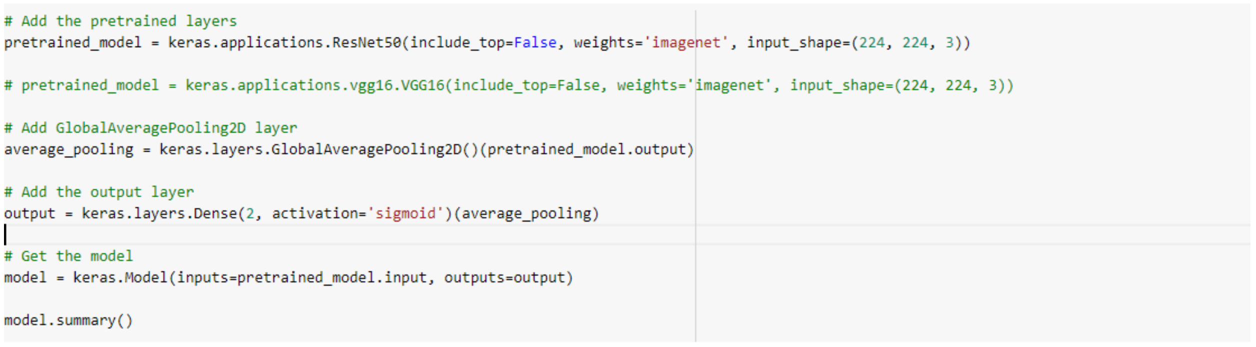
2. Using the above data, we've created a new folder which has the training dataset into two sub-folders – "**Car**" and "**No Car**". This is done because our dataset had no target or label which required to classify the image.

3. We use the parent folder of Split the data into train and validation dataset using Keras utility "**image_dataset_from_directory**"

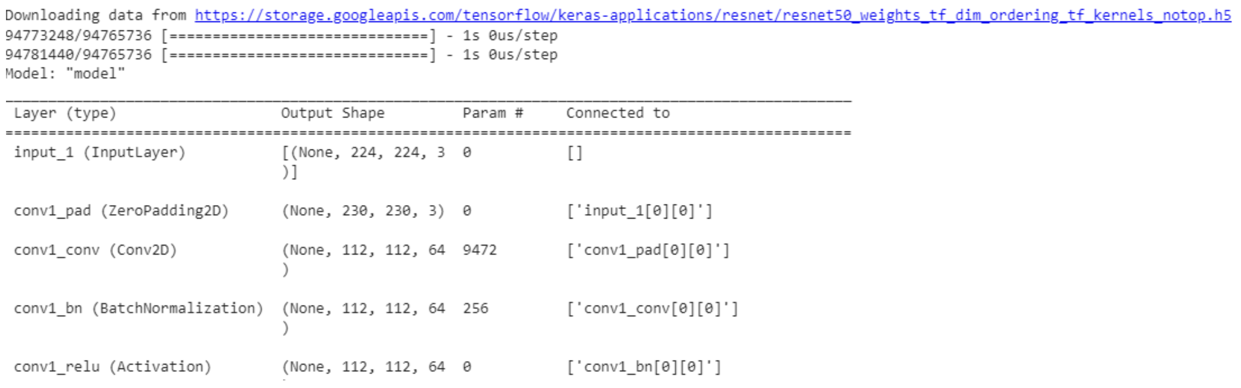
4. Set the default input size for the pretrained model, i.e., (224, 224).

Pre-trained Model (Transfer Learning)

1. Import the pre-trained ResNet50 model using Keras.



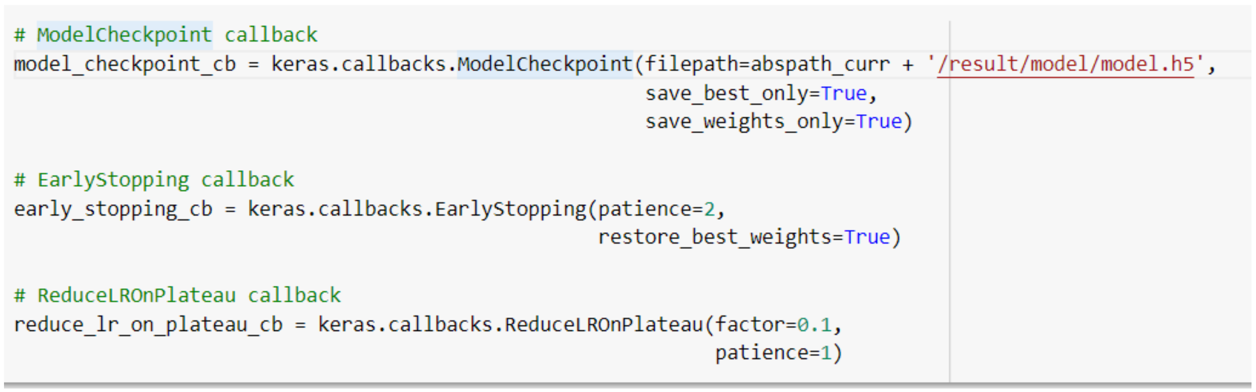
```
# Add the pretrained layers
pretrained_model = keras.applications.ResNet50(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# pretrained_model = keras.applications.vgg16.VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Add GlobalAveragePooling2D layer
average_pooling = keras.layers.GlobalAveragePooling2D()(pretrained_model.output)

# Add the output layer
output = keras.layers.Dense(2, activation='sigmoid')(average_pooling)

# Get the model
model = keras.Model(inputs=pretrained_model.input, outputs=output)

model.summary()
```



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
```

2. Freeze the layers in the model which capture the basic metadata about an image.

3. Set up callbacks for saving the best model, early stopping to avoid over-fitting and to reduce learning rate when a metric has stopped improving



```
# ModelCheckpoint callback
model_checkpoint_cb = keras.callbacks.ModelCheckpoint(filepath=abspath_curr + '/result/model/model.h5',
                                                      save_best_only=True,
                                                      save_weights_only=True)

# EarlyStopping callback
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
                                                  restore_best_weights=True)

# ReduceLROnPlateau callback
reduce_lr_on_plateau_cb = keras.callbacks.ReduceLROnPlateau(factor=0.1,
                                                            patience=1)
```

Compiling Model

1. **Stochastic Gradient Descent (SGD) Optimizer**:

SGD algorithm is an extension of the Gradient Descent, and it overcomes some of the disadvantages of the GD algorithm. Gradient Descent has a disadvantage that it requires a lot of memory to load the entire dataset of n-points at a time to compute the derivative of the loss function. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

2. **ADAM Optimizer**

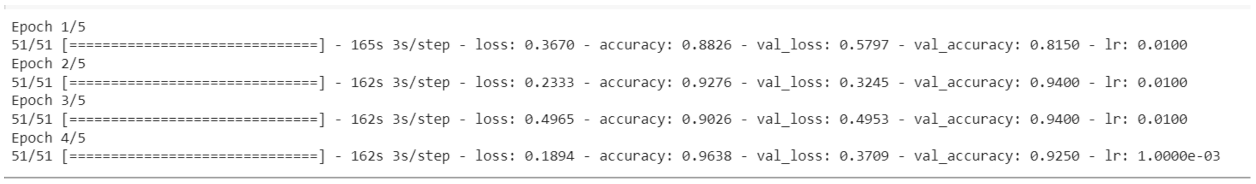
Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. The method computes individual adaptive learning rates for different parameters (network weight) from estimates of first and second moments of the gradients.

Model Results

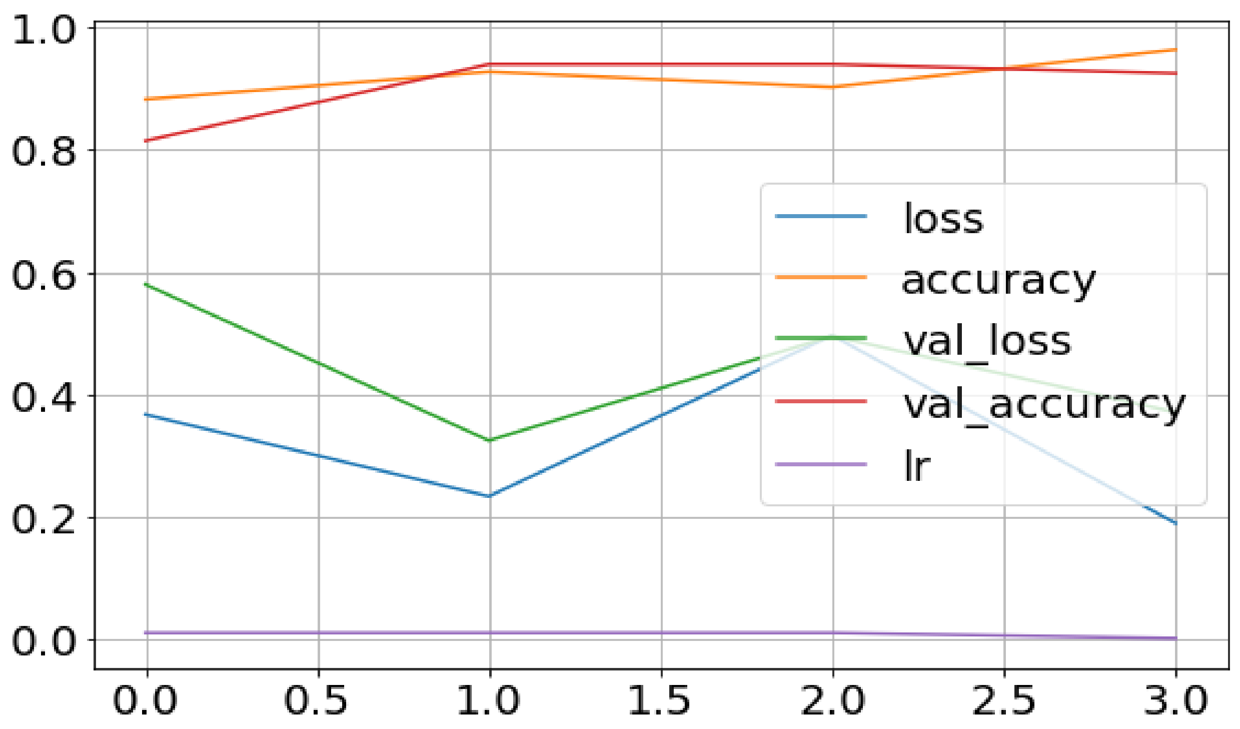
ADAM Optimizer Training Results

ADAM - Iteration 1

Freeze Layers – epoch = 5, learning rate = 0.01, Batch Size = 16

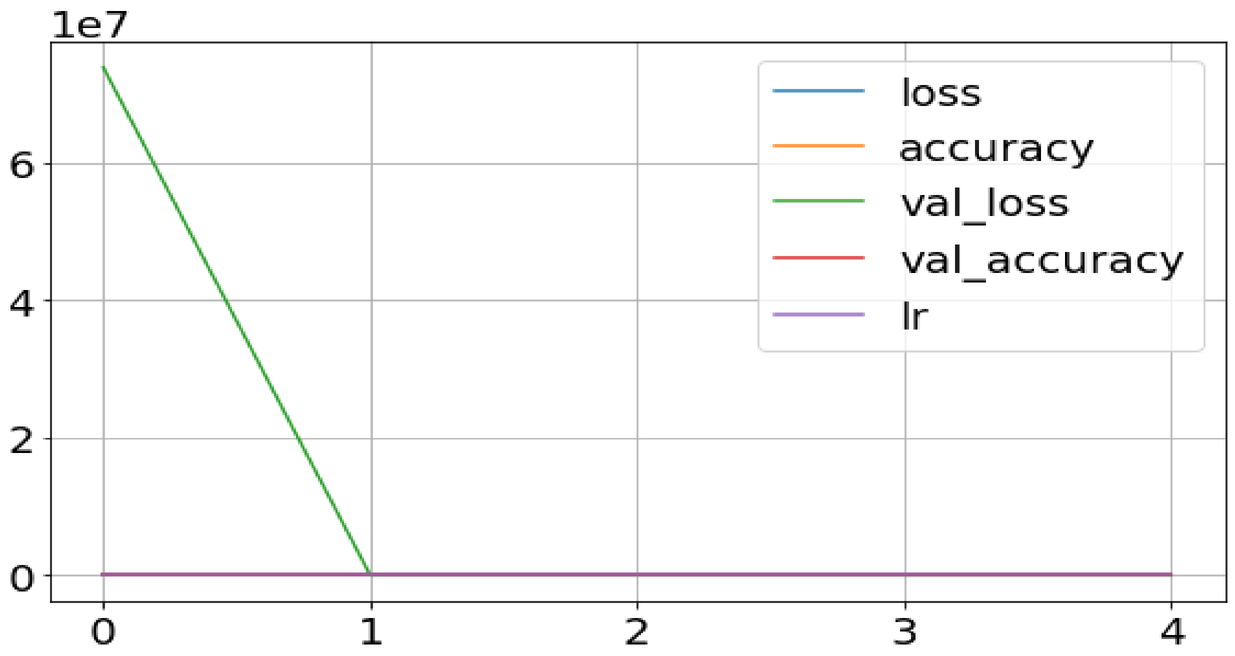


```
Epoch 1/5
51/51 [==============================] - 165s 3s/step - loss: 0.3670 - accuracy: 0.8826 - val_loss: 0.5797 - val_accuracy: 0.8150 - lr: 0.0100
Epoch 2/5
51/51 [==============================] - 162s 3s/step - loss: 0.2333 - accuracy: 0.9276 - val_loss: 0.3245 - val_accuracy: 0.9400 - lr: 0.0100
Epoch 3/5
51/51 [==============================] - 162s 3s/step - loss: 0.4965 - accuracy: 0.9026 - val_loss: 0.4953 - val_accuracy: 0.9400 - lr: 0.0100
Epoch 4/5
51/51 [==============================] - 162s 3s/step - loss: 0.1894 - accuracy: 0.9638 - val_loss: 0.3709 - val_accuracy: 0.9250 - lr: 1.0000e-03
```



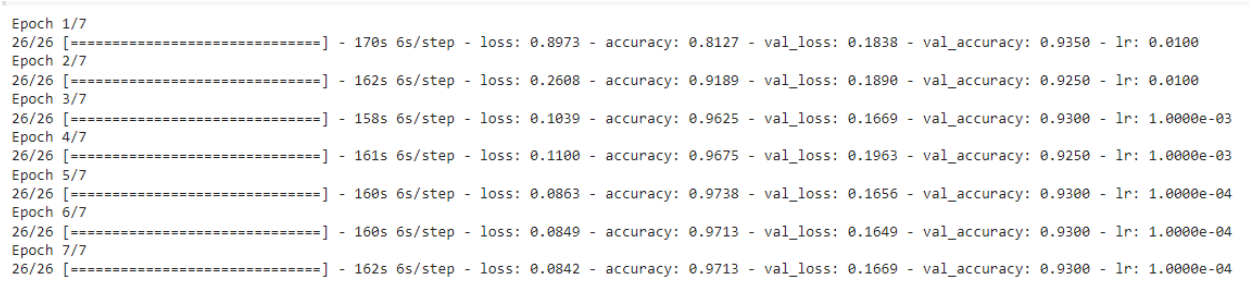
Unfreeze Layers – epoch – 5, learning rate = 0.01, Batch Size = 16

```
Epoch 1/5
51/51 [==============================] - 582s 11s/step - loss: 1.3685 - accuracy: 0.7690 - val_loss: 73775552.0000 - val_accuracy: 0.3550 - lr: 0.0010
Epoch 2/5
51/51 [==============================] - 572s 11s/step - loss: 0.3215 - accuracy: 0.8789 - val_loss: 607.2209 - val_accuracy: 0.3300 - lr: 0.0010
Epoch 3/5
51/51 [==============================] - 566s 11s/step - loss: 0.2073 - accuracy: 0.9401 - val_loss: 1144.5415 - val_accuracy: 0.3550 - lr: 0.0010
Epoch 4/5
51/51 [==============================] - 567s 11s/step - loss: 0.1182 - accuracy: 0.9638 - val_loss: 80.2307 - val_accuracy: 0.4400 - lr: 1.0000e-04
Epoch 5/5
51/51 [==============================] - 569s 11s/step - loss: 0.0468 - accuracy: 0.9850 - val_loss: 4.1853 - val_accuracy: 0.7200 - lr: 1.0000e-04
```

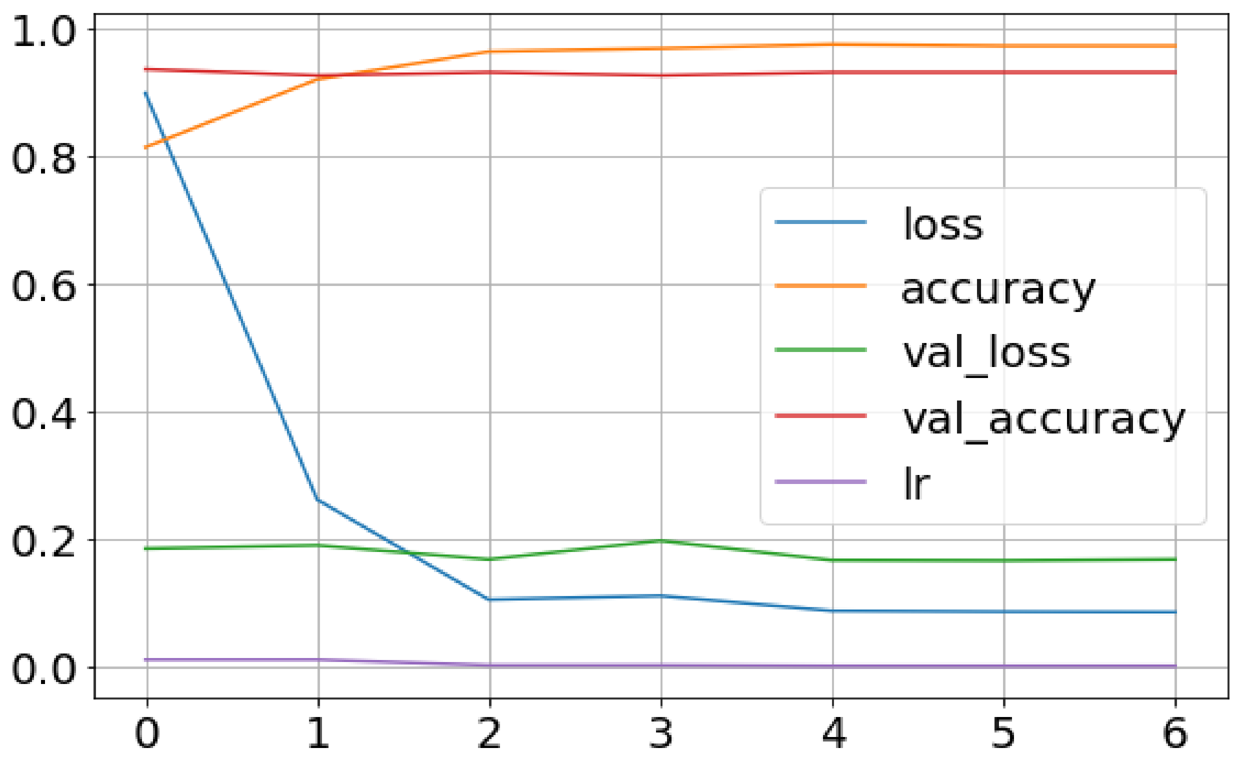


ADAM - Iteration 2

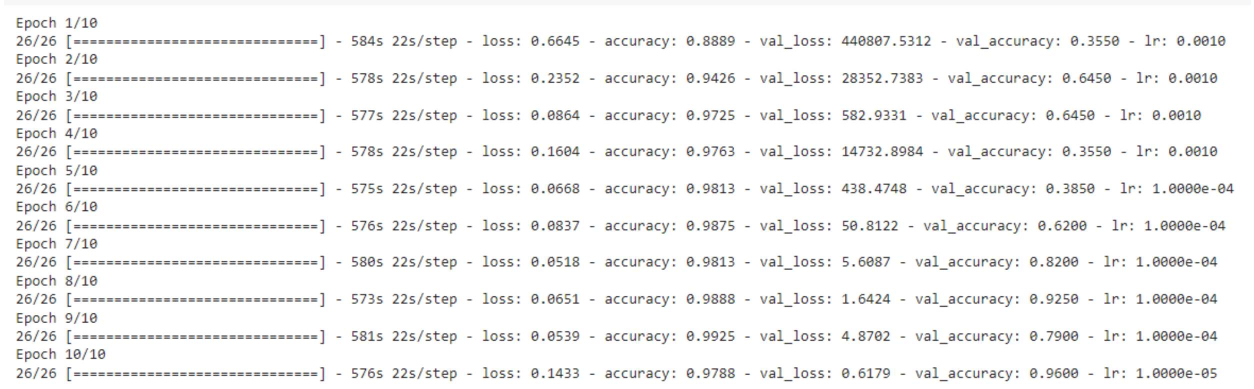
Freeze Layers – epoch – 7, learning rate = 0.01, Batch size = 32



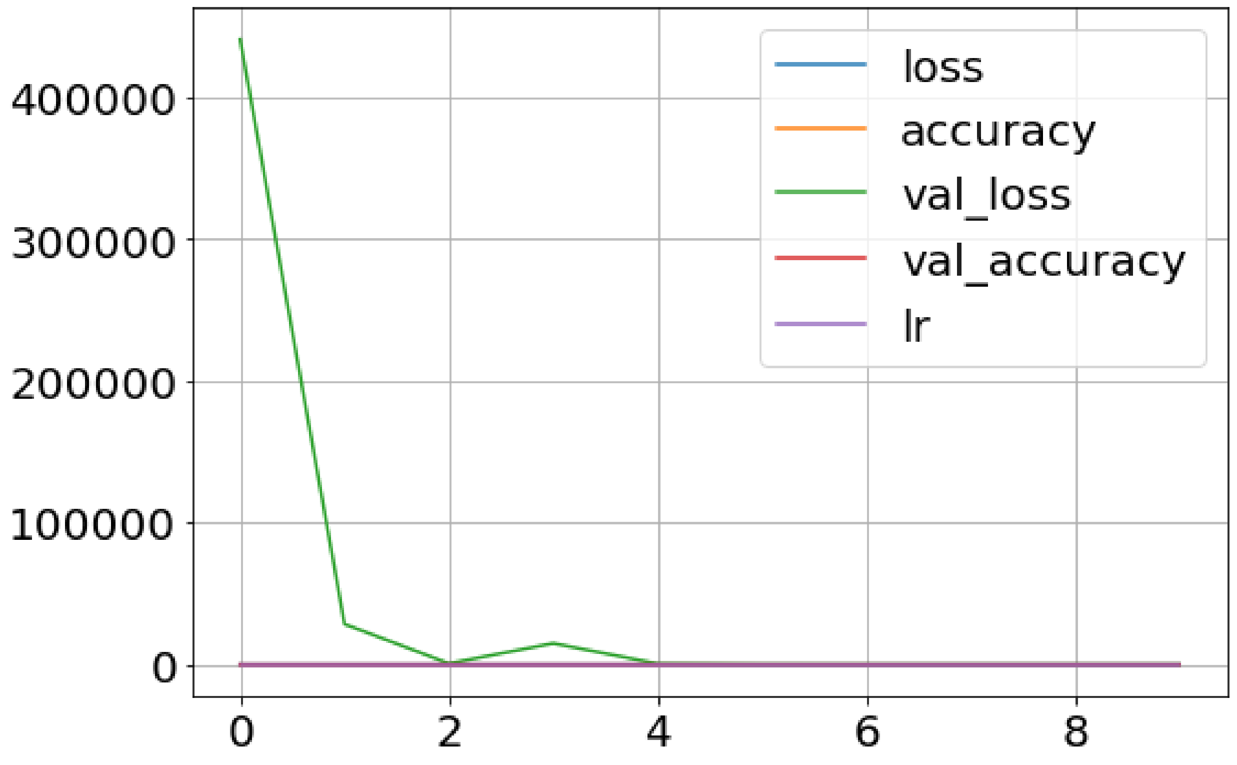
```
Epoch 1/7
26/26 [==============================] - 170s 6s/step - loss: 0.8973 - accuracy: 0.8127 - val_loss: 0.1838 - val_accuracy: 0.9350 - lr: 0.0100
Epoch 2/7
26/26 [==============================] - 162s 6s/step - loss: 0.2608 - accuracy: 0.9189 - val_loss: 0.1890 - val_accuracy: 0.9250 - lr: 0.0100
Epoch 3/7
26/26 [==============================] - 158s 6s/step - loss: 0.1039 - accuracy: 0.9625 - val_loss: 0.1669 - val_accuracy: 0.9300 - lr: 1.0000e-03
Epoch 4/7
26/26 [==============================] - 161s 6s/step - loss: 0.1100 - accuracy: 0.9675 - val_loss: 0.1963 - val_accuracy: 0.9250 - lr: 1.0000e-03
Epoch 5/7
26/26 [==============================] - 160s 6s/step - loss: 0.0863 - accuracy: 0.9738 - val_loss: 0.1656 - val_accuracy: 0.9300 - lr: 1.0000e-04
Epoch 6/7
26/26 [==============================] - 160s 6s/step - loss: 0.0849 - accuracy: 0.9713 - val_loss: 0.1649 - val_accuracy: 0.9300 - lr: 1.0000e-04
Epoch 7/7
26/26 [==============================] - 162s 6s/step - loss: 0.0842 - accuracy: 0.9713 - val_loss: 0.1669 - val_accuracy: 0.9300 - lr: 1.0000e-04
```



Unfreeze Layers – epoch – 10, learning rate = 0.01, Batch Size = 32



```
Epoch 1/10
26/26 [==============================] - 584s 22s/step - loss: 0.6645 - accuracy: 0.8889 - val_loss: 440807.5312 - val_accuracy: 0.3550 - lr: 0.0010
Epoch 2/10
26/26 [==============================] - 578s 22s/step - loss: 0.2352 - accuracy: 0.9426 - val_loss: 28352.7383 - val_accuracy: 0.6450 - lr: 0.0010
Epoch 3/10
26/26 [==============================] - 577s 22s/step - loss: 0.0864 - accuracy: 0.9725 - val_loss: 582.9331 - val_accuracy: 0.6450 - lr: 0.0010
Epoch 4/10
26/26 [==============================] - 578s 22s/step - loss: 0.1604 - accuracy: 0.9763 - val_loss: 14732.8984 - val_accuracy: 0.3550 - lr: 0.0010
Epoch 5/10
26/26 [==============================] - 575s 22s/step - loss: 0.0668 - accuracy: 0.9813 - val_loss: 438.4748 - val_accuracy: 0.3850 - lr: 1.0000e-04
Epoch 6/10
26/26 [==============================] - 576s 22s/step - loss: 0.0837 - accuracy: 0.9875 - val_loss: 50.8122 - val_accuracy: 0.6200 - lr: 1.0000e-04
Epoch 7/10
26/26 [==============================] - 580s 22s/step - loss: 0.0518 - accuracy: 0.9813 - val_loss: 5.6087 - val_accuracy: 0.8200 - lr: 1.0000e-04
Epoch 8/10
26/26 [==============================] - 573s 22s/step - loss: 0.0651 - accuracy: 0.9888 - val_loss: 1.6424 - val_accuracy: 0.9250 - lr: 1.0000e-04
Epoch 9/10
26/26 [==============================] - 581s 22s/step - loss: 0.0539 - accuracy: 0.9925 - val_loss: 4.8702 - val_accuracy: 0.7900 - lr: 1.0000e-04
Epoch 10/10
26/26 [==============================] - 576s 22s/step - loss: 0.1433 - accuracy: 0.9788 - val_loss: 0.6179 - val_accuracy: 0.9600 - lr: 1.0000e-05
```

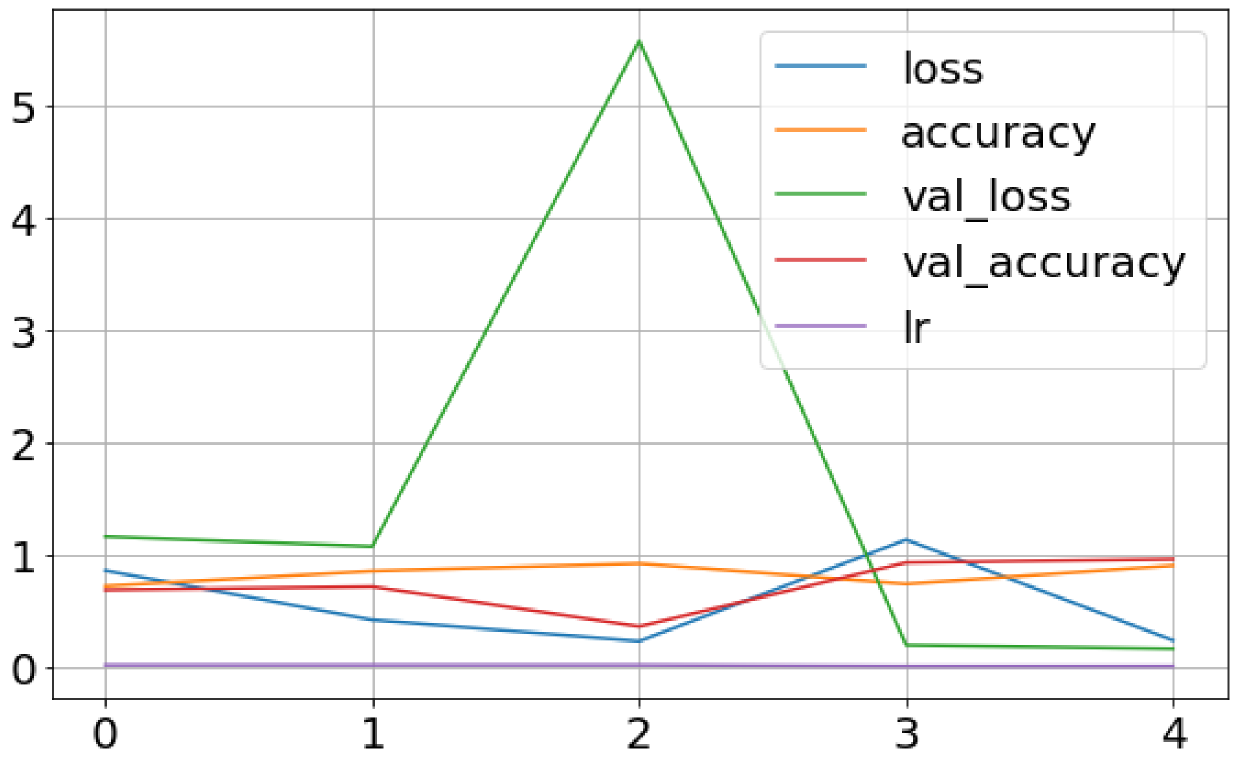


SGD Optimizer Training Results

SGD - Iteration 1

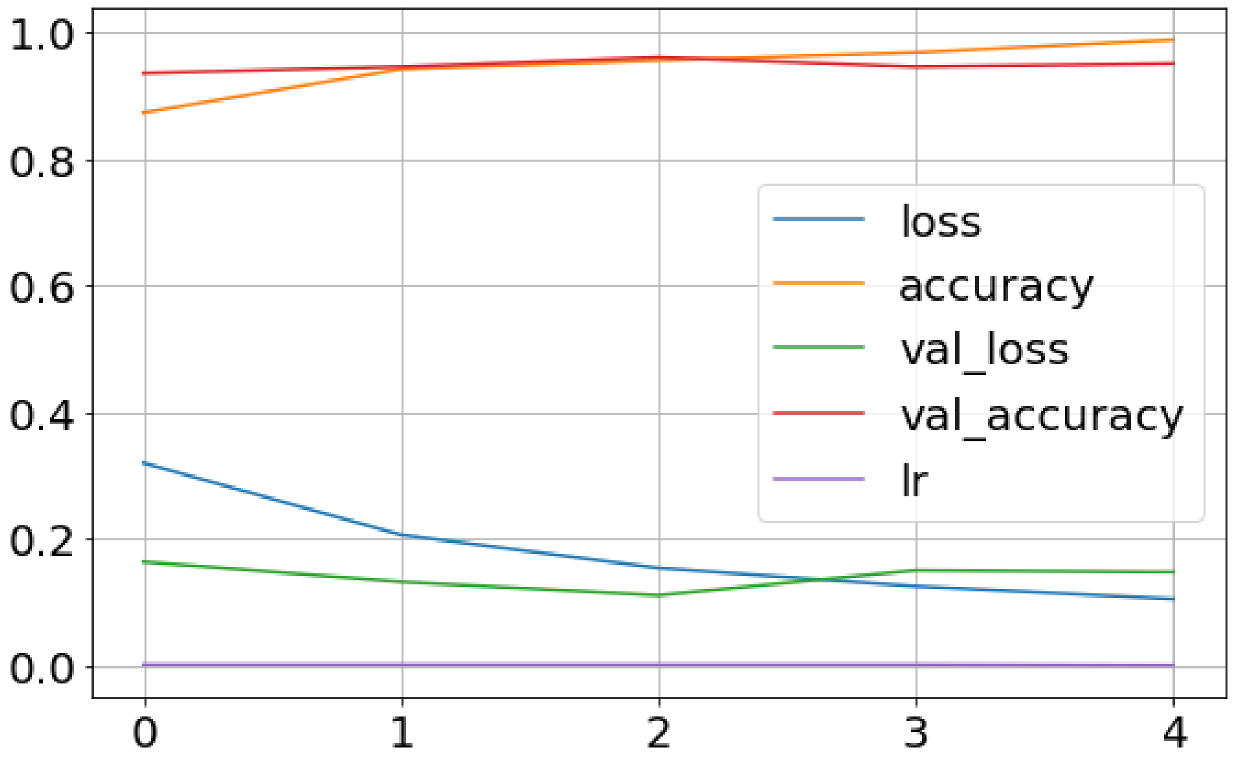
Freeze Layers – epoch – 5, learning rate = 0.01, Batch size = 32

```
Epoch 1/5
26/26 [==============================] - 165s 6s/step - loss: 0.8534 - accuracy: 0.7154 - val_loss: 1.1567 - val_accuracy: 0.6800 - lr: 0.0100
Epoch 2/5
26/26 [==============================] - 175s 6s/step - loss: 0.4137 - accuracy: 0.8489 - val_loss: 1.0685 - val_accuracy: 0.7100 - lr: 0.0100
Epoch 3/5
26/26 [==============================] - 156s 6s/step - loss: 0.2235 - accuracy: 0.9164 - val_loss: 5.5781 - val_accuracy: 0.3550 - lr: 0.0100
Epoch 4/5
26/26 [==============================] - 159s 6s/step - loss: 1.1284 - accuracy: 0.7341 - val_loss: 0.1887 - val_accuracy: 0.9250 - lr: 1.0000e-03
Epoch 5/5
26/26 [==============================] - 158s 6s/step - loss: 0.2311 - accuracy: 0.8976 - val_loss: 0.1554 - val_accuracy: 0.9550 - lr: 1.0000e-03
```



Unfreeze Layers – epoch – 5, learning rate = 0.001, Batch size = 32

```
Epoch 1/5
26/26 [==============================] - 565s 21s/step - loss: 0.3193 - accuracy: 0.8727 - val_loss: 0.1632 - val_accuracy: 0.9350 - lr: 0.0010
Epoch 2/5
26/26 [==============================] - 561s 21s/step - loss: 0.2059 - accuracy: 0.9413 - val_loss: 0.1316 - val_accuracy: 0.9450 - lr: 0.0010
Epoch 3/5
26/26 [==============================] - 561s 21s/step - loss: 0.1535 - accuracy: 0.9551 - val_loss: 0.1103 - val_accuracy: 0.9600 - lr: 0.0010
Epoch 4/5
26/26 [==============================] - 557s 21s/step - loss: 0.1246 - accuracy: 0.9675 - val_loss: 0.1497 - val_accuracy: 0.9450 - lr: 0.0010
Epoch 5/5
26/26 [==============================] - 572s 22s/step - loss: 0.1047 - accuracy: 0.9875 - val_loss: 0.1470 - val_accuracy: 0.9500 - lr: 1.0000e-04
```



Conclusion

As mentioned earlier, Object Detection is widely used by organizations interested in automated vehicle systems, surveillance, security and in some areas of insurance as well. Using CNN allows us the ability to build networks (models) that correctly identify and classify any given object.

The ResNet50 model was able to identify "Car" with a very good accuracy of ~ **87%** using the SGD Optimizer

Future Work:

- There are many other advanced algorithms used for object detection include convolutional neural networks (R-CNN, Region-Based CNN), Fast R-CNN, and YOLO (You Only Look Once). Also, with faster computers and bigger GPU's we can also try many more combination of learning rate and epochs.
- We can also combine this project with "highway lane detection" to create a mini-automated vehicle cruise system.

Final Term Project – Machine Learning

Appendix

GitHub Link - https://github.com/IshanKuchroo/Car_Object_Detection

Deep Learning Utilities –
https://github.com/yuxiaohuang/teaching/blob/master/gwu/machine_learning_I/spring_2022/code/utilities/p3_deep_learning/pmlm_utilities_shallow.ipynb