ASSIGNMENT 1

Full Name	UBIT Name	UB Number
Kusha Kumar Dharavath	**kushakum**	**50532663**
Dharma. Acha	**dharmaac**	**50511275**

Part I: CNN Classification

Summary of the VGG Model:

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1          [-1, 64, 64, 64]           1,792
            ReLU-2          [-1, 64, 64, 64]               0
          Conv2d-3          [-1, 64, 64, 64]          36,928
            ReLU-4          [-1, 64, 64, 64]               0
       MaxPool2d-5          [-1, 64, 32, 32]               0
          Conv2d-6         [-1, 128, 32, 32]          73,856
            ReLU-7         [-1, 128, 32, 32]               0
          Conv2d-8         [-1, 128, 32, 32]         147,584
            ReLU-9         [-1, 128, 32, 32]               0
      MaxPool2d-10         [-1, 128, 16, 16]               0
         Conv2d-11         [-1, 256, 16, 16]         295,168
           ReLU-12         [-1, 256, 16, 16]               0
         Conv2d-13         [-1, 256, 16, 16]         590,080
           ReLU-14         [-1, 256, 16, 16]               0
      MaxPool2d-15          [-1, 256, 8, 8]               0
         Conv2d-16          [-1, 512, 8, 8]       1,180,160
           ReLU-17          [-1, 512, 8, 8]               0
         Conv2d-18          [-1, 512, 8, 8]       2,359,808
           ReLU-19          [-1, 512, 8, 8]               0
      MaxPool2d-20          [-1, 512, 4, 4]               0
         Conv2d-21          [-1, 512, 4, 4]       2,359,808
           ReLU-22          [-1, 512, 4, 4]               0
         Conv2d-23          [-1, 512, 4, 4]       2,359,808
           ReLU-24          [-1, 512, 4, 4]               0
      MaxPool2d-25          [-1, 512, 2, 2]               0
         Linear-26             [-1, 4096]       8,392,704
           ReLU-27             [-1, 4096]               0
        Dropout-28             [-1, 4096]               0
         Linear-29             [-1, 4096]      16,781,312
           ReLU-30             [-1, 4096]               0
```

```
        Dropout-31             [-1, 4096]               0
         Linear-32                [-1, 3]          12,291
================================================================
Total params: 34,591,299
Trainable params: 34,591,299
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 16.39
Params size (MB): 131.96
Estimated Total Size (MB): 148.39
----------------------------------------------------------------
```

Influence of techniques on base model:

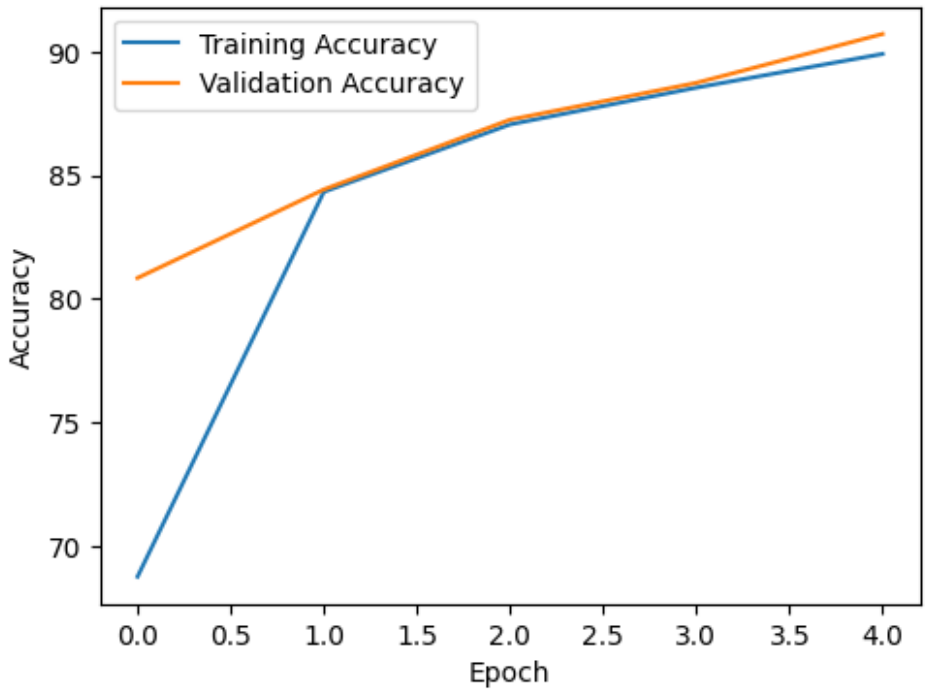
The test accuracy of the base model is 90 % and test loss is 0.250 when we apply the overfitting prevention techniques the test accuracy increased by 3 % i.e 93% and test loss is 0.185

Base VGG model:

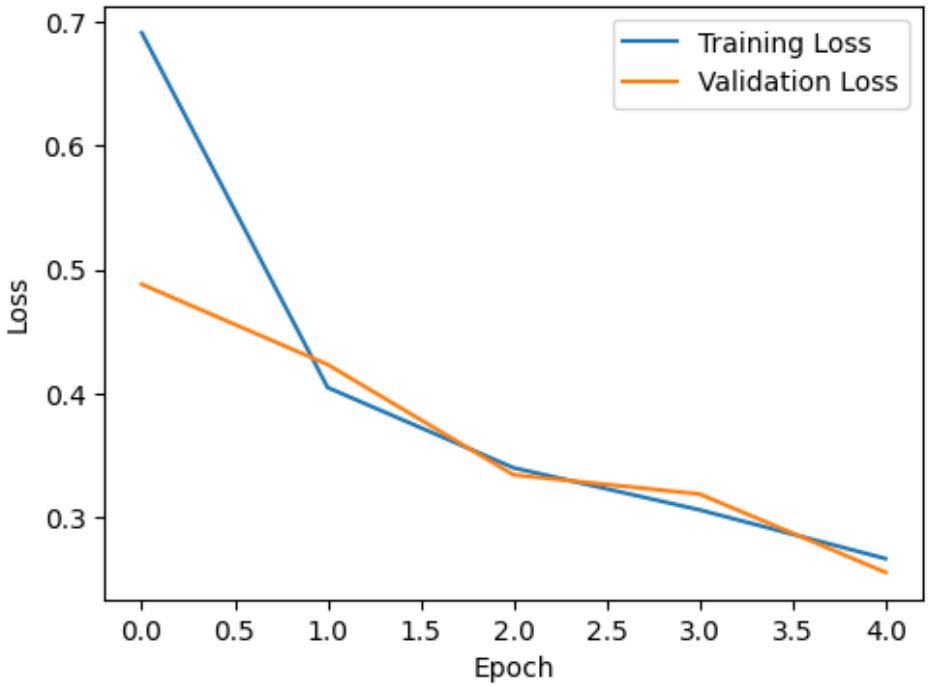
Train accuracy: 89.90%
Train loss: 0.26
Validation accuracy: 90.71%
Validation loss: 0.25
Test accuracy: 90%
Test Loss: 0.25

From the below graphs we can infer that as we increase the number of epochs it is clear that the accuracy of the training and validation increased. The training accuracy rocketed from approximately 68 % to 89 %. On the other hand training loss and validation loss decreased from 0.69 and 0.48 to 0.266 and 0.255 respectively.

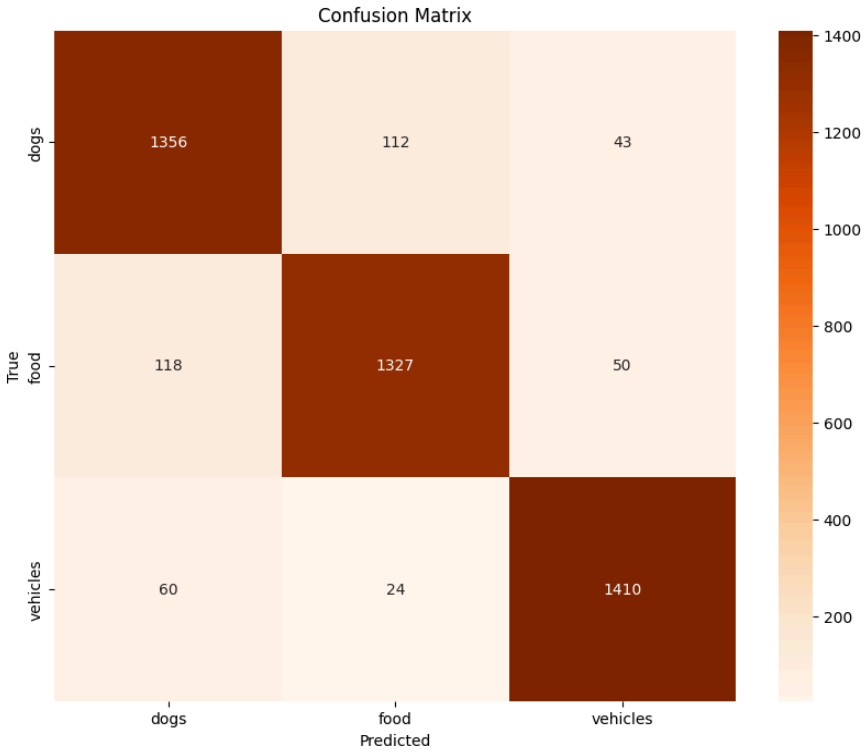
Training and Validation Accuracy:



Training and Validation Loss:



Confusion matrix:



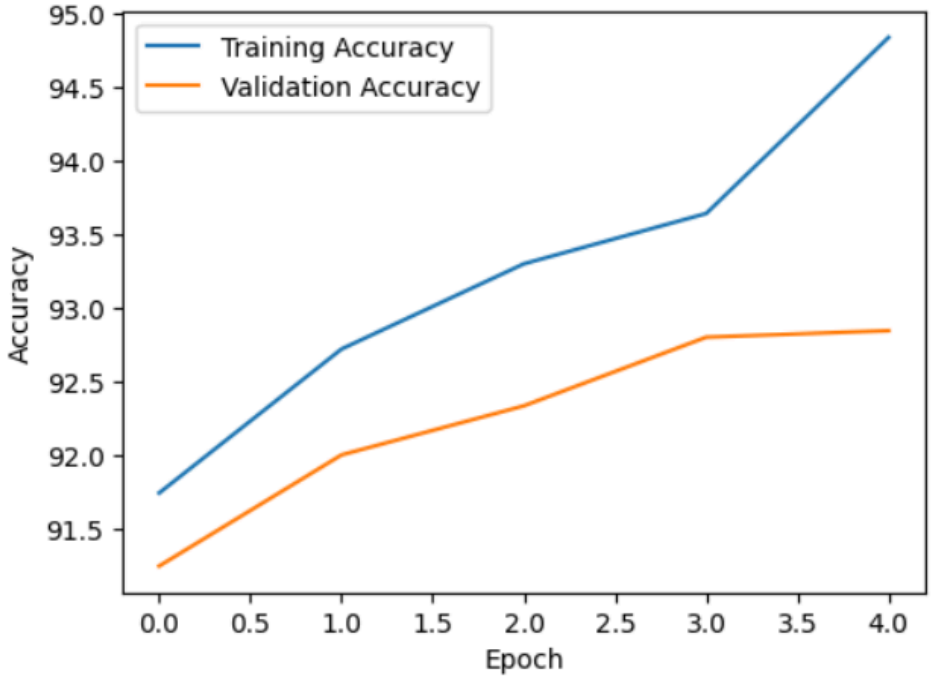
Evaluation metrics:

Precision: 0.91
Recall: 0.91
F1 Score: 0.91

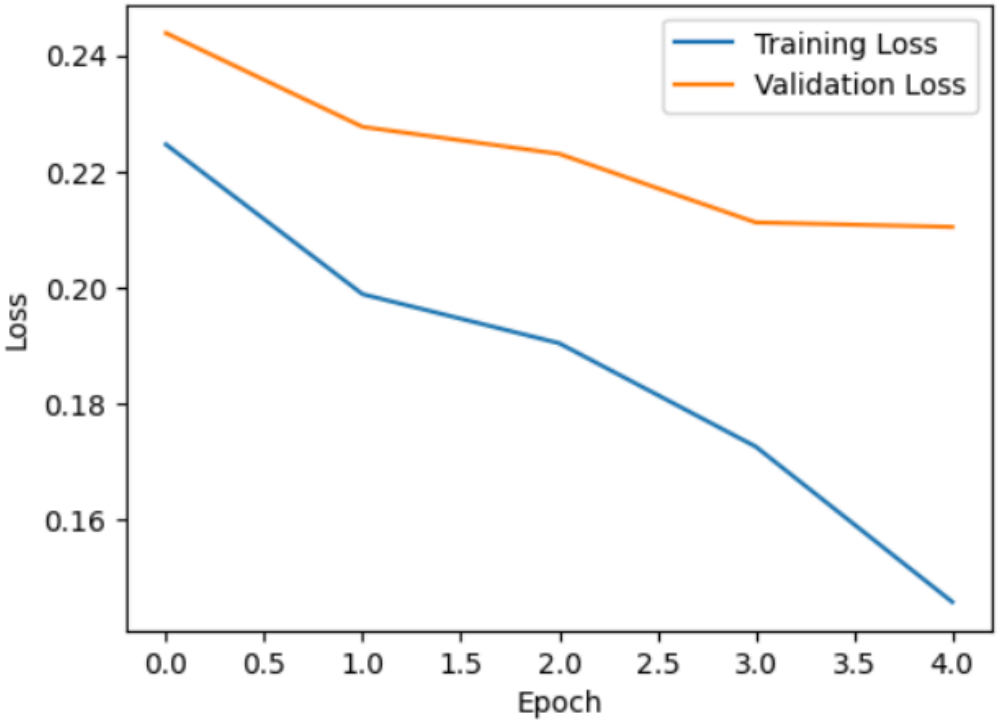
Base + L2 Regularization:
Train accuracy: 94.84%
Train loss: 0.14
Validation accuracy: 92.84%
Validation loss: 0.21
Test accuracy: 92%
Test Loss: 0.21

From the below graphs we can say that as we go on increasing the number of epochs the training accuracy and validation accuracy increased to 94 % and 92 % respectively whereas training loss and validation loss decreased at every epoch.

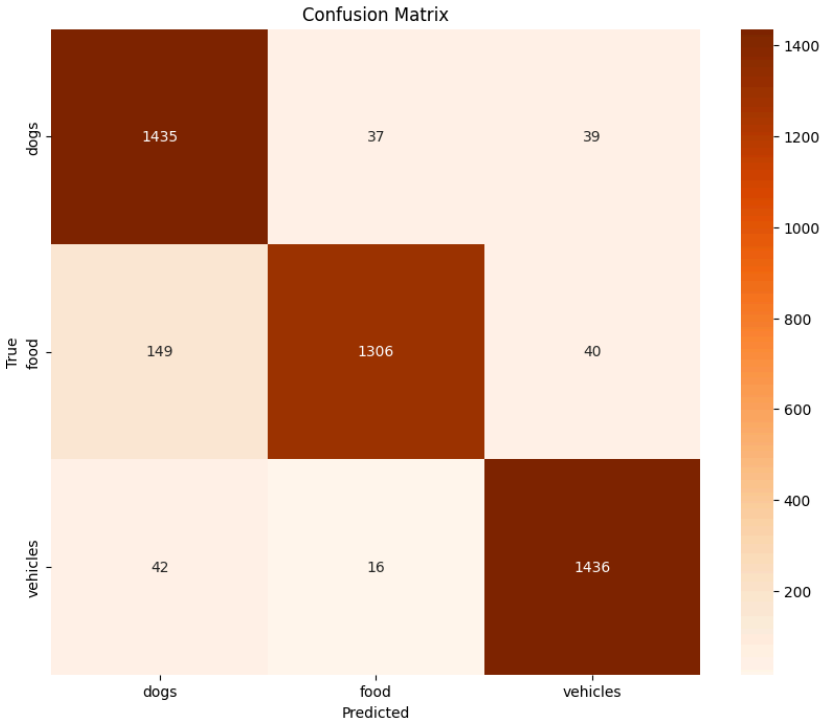
Training and Validation Accuracy:



Training and Validation Loss:



Confusion Matrix:



Evaluation Metrics:
Precision: 0.93
Recall: 0.93
F1 Score: 0.93

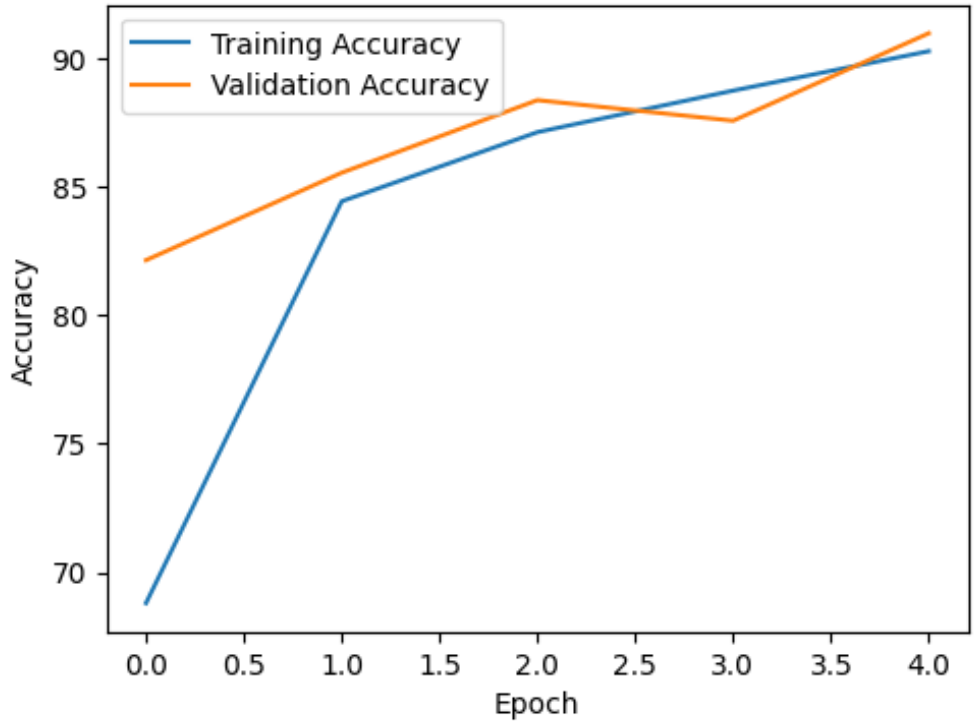
Base + L2 Regularization + DropOut

Train accuracy: 90.26%
Train loss: 0. 26
Validation accuracy: 90.96%
Validation loss: 0.25
Test accuracy: 90%
Test Loss: 0.24

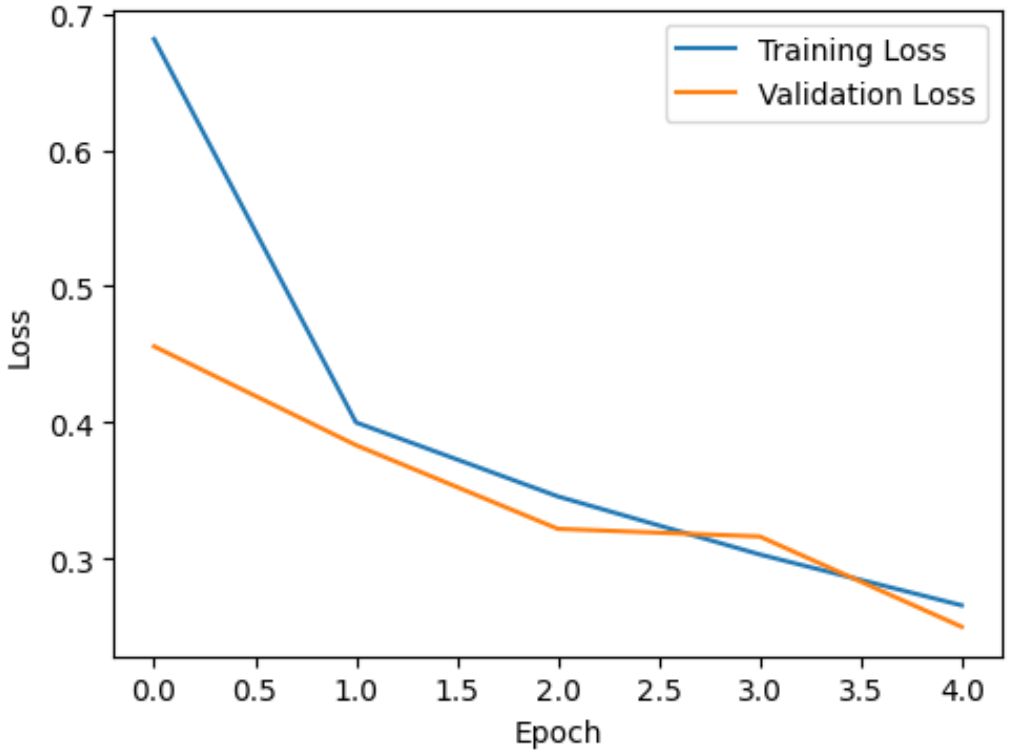
At the 1st Epoch the training accuracy is very low i.e 68% and validation accuracy is 82% after running 5 epochs training accuracy and validation accuracy settled at 90% and 90.96% respectively. See the below graphs for better understanding

When we look into the validation and Training loss the losses decreased.

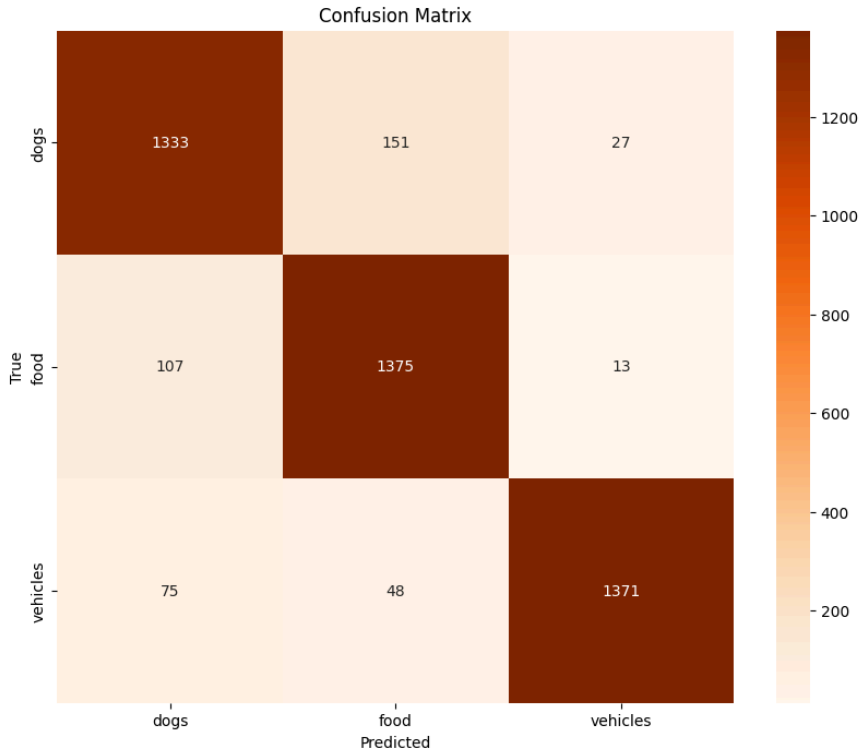
Training and Validation Accuracy:



Training and Validation Loss:



Confusion matrix:

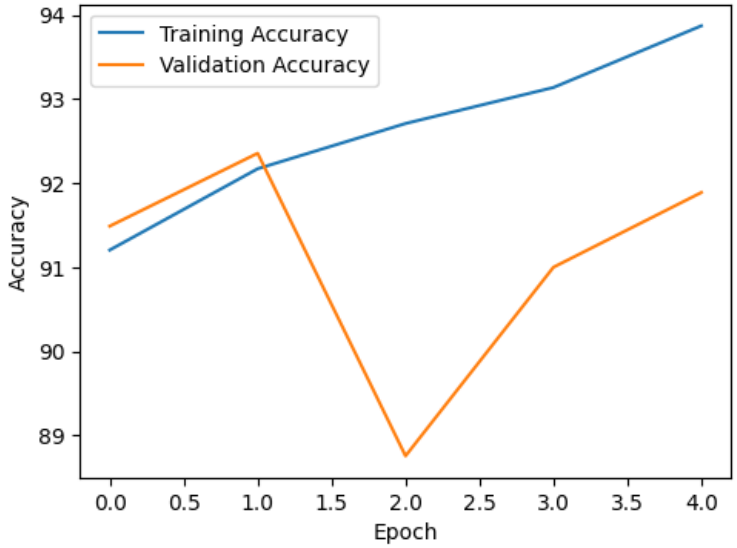


Evaluation Metrics:
Precision: 0.91
Recall: 0.91
F1 Score: 0.91

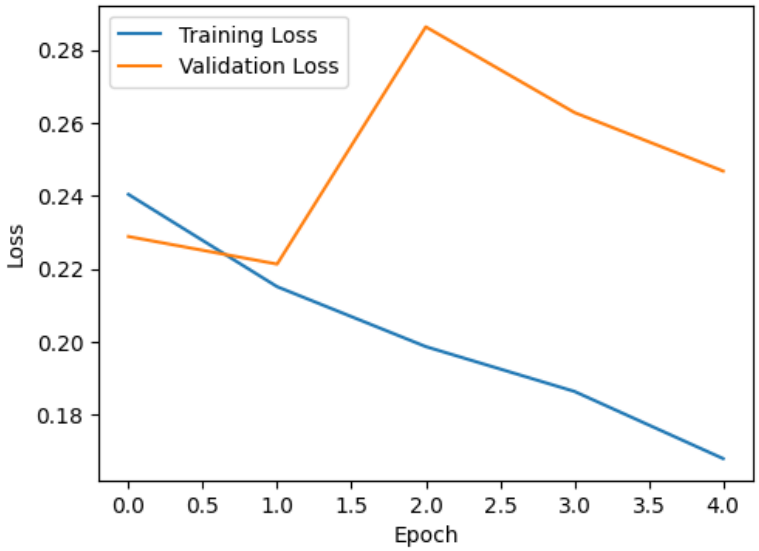
Base + L2 Regularization + DropOut + Early Stopping:
Train accuracy: 93.87%
Train loss: 0. 16
Validation accuracy: 91.89%
Validation loss: 0.24
Test accuracy: 91%
Test Loss: 0.24

The graph depicts the training accuracy increased to 93.87% and validation accuracy is 91.89% and respective losses also decreased as number of epochs increased.

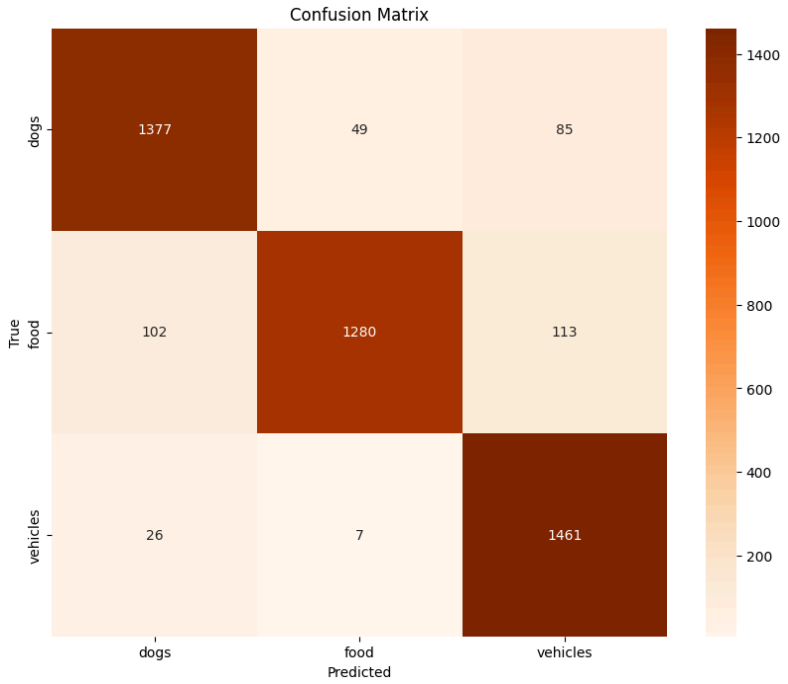
Training and Validation Accuracy:



Training and Validation Loss:



Confusion matrix:



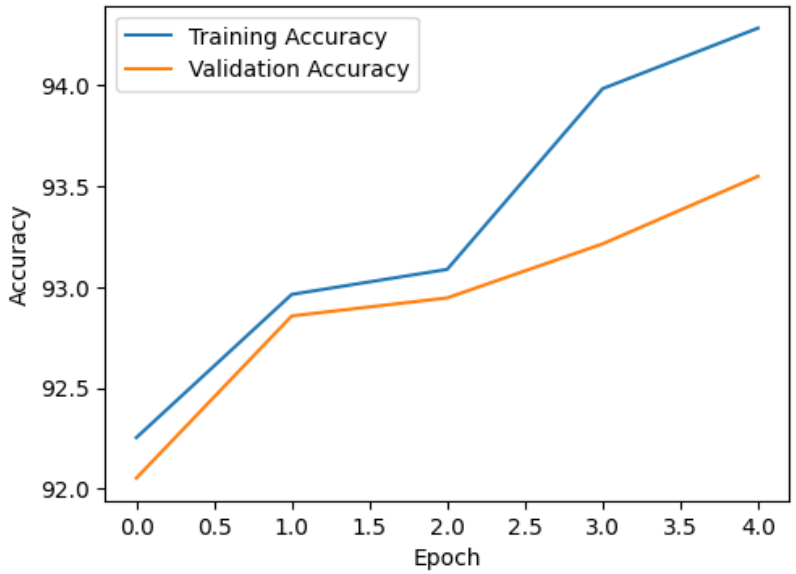
Evaluation Metrics:
Precision: 0.92
Recall: 0.92
F1 Score: 0.91

Base + L2 Regularization + DropOut +Early Stopping + Image Augmentation:

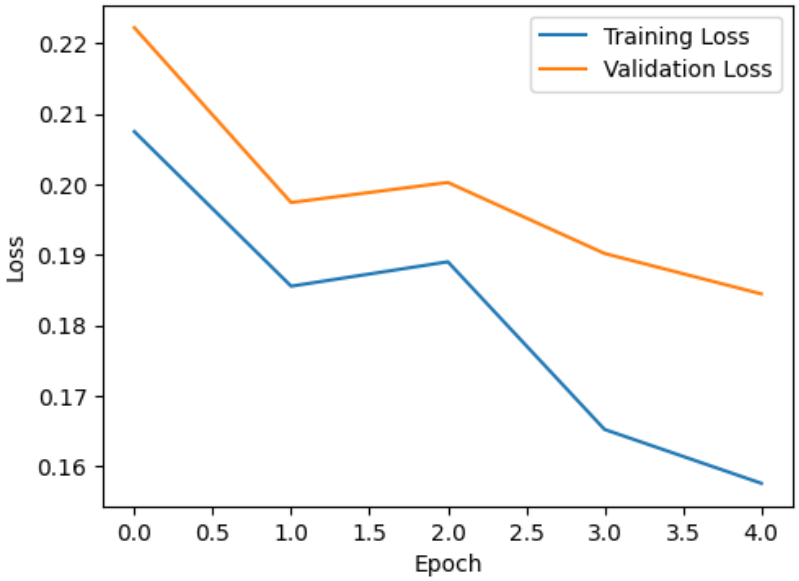
Train accuracy: 94.28%
Train loss: 0. 15
Validation accuracy: 93.55%
Validation loss: 0.18
Test accuracy: 93%
Test Loss: 0.18

From the below graphs we can infer that as we increase the number of epochs it is clear that the accuracy of the training and validation increased. On the other hand training loss and validation loss decreases.

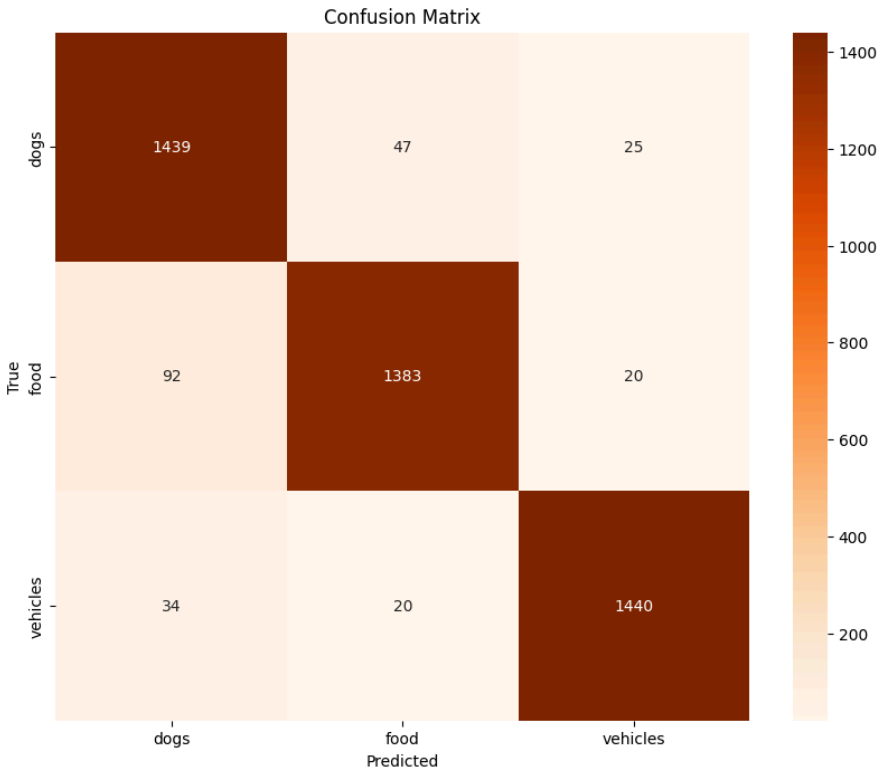
Training and Validation Accuracy:



Training and Validation Loss:



Confusion Matrix:



Evaluation Metrics:
Precision: 0.95
Recall: 0.95
F1 Score: 0.95

Part II: Implementing ResNet Architecture

Resnet

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	9,408
BatchNorm2d-2	[-1, 64, 32, 32]	128
ReLU-3	[-1, 64, 32, 32]	0
MaxPool2d-4	[-1, 64, 16, 16]	0
Conv2d-5	[-1, 64, 16, 16]	36,864
BatchNorm2d-6	[-1, 64, 16, 16]	128
ReLU-7	[-1, 64, 16, 16]	0
Conv2d-8	[-1, 64, 16, 16]	36,864
BatchNorm2d-9	[-1, 64, 16, 16]	128
ReLU-10	[-1, 64, 16, 16]	0
BasicBlock-11	[-1, 64, 16, 16]	0
Conv2d-12	[-1, 64, 16, 16]	36,864
BatchNorm2d-13	[-1, 64, 16, 16]	128

ReLU-14	[-1, 64, 16, 16]	0
Conv2d-15	[-1, 64, 16, 16]	36,864
BatchNorm2d-16	[-1, 64, 16, 16]	128
ReLU-17	[-1, 64, 16, 16]	0
BasicBlock-18	[-1, 64, 16, 16]	0
Conv2d-19	[-1, 128, 8, 8]	73,728
BatchNorm2d-20	[-1, 128, 8, 8]	256
ReLU-21	[-1, 128, 8, 8]	0
Conv2d-22	[-1, 128, 8, 8]	147,456
BatchNorm2d-23	[-1, 128, 8, 8]	256
Conv2d-24	[-1, 128, 8, 8]	8,192
BatchNorm2d-25	[-1, 128, 8, 8]	256
ReLU-26	[-1, 128, 8, 8]	0
BasicBlock-27	[-1, 128, 8, 8]	0
Conv2d-28	[-1, 128, 8, 8]	147,456
BatchNorm2d-29	[-1, 128, 8, 8]	256
ReLU-30	[-1, 128, 8, 8]	0
Conv2d-31	[-1, 128, 8, 8]	147,456
BatchNorm2d-32	[-1, 128, 8, 8]	256
ReLU-33	[-1, 128, 8, 8]	0
BasicBlock-34	[-1, 128, 8, 8]	0
Conv2d-35	[-1, 256, 4, 4]	294,912
BatchNorm2d-36	[-1, 256, 4, 4]	512
ReLU-37	[-1, 256, 4, 4]	0
Conv2d-38	[-1, 256, 4, 4]	589,824
BatchNorm2d-39	[-1, 256, 4, 4]	512
Conv2d-40	[-1, 256, 4, 4]	32,768
BatchNorm2d-41	[-1, 256, 4, 4]	512
ReLU-42	[-1, 256, 4, 4]	0
BasicBlock-43	[-1, 256, 4, 4]	0
Conv2d-44	[-1, 256, 4, 4]	589,824
BatchNorm2d-45	[-1, 256, 4, 4]	512
ReLU-46	[-1, 256, 4, 4]	0
Conv2d-47	[-1, 256, 4, 4]	589,824
BatchNorm2d-48	[-1, 256, 4, 4]	512
ReLU-49	[-1, 256, 4, 4]	0
BasicBlock-50	[-1, 256, 4, 4]	0
Conv2d-51	[-1, 512, 2, 2]	1,179,648
BatchNorm2d-52	[-1, 512, 2, 2]	1,024
ReLU-53	[-1, 512, 2, 2]	0
Conv2d-54	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-55	[-1, 512, 2, 2]	1,024
Conv2d-56	[-1, 512, 2, 2]	131,072
BatchNorm2d-57	[-1, 512, 2, 2]	1,024

```
              ReLU-58          [-1, 512, 2, 2]               0
        BasicBlock-59          [-1, 512, 2, 2]               0
          Conv2d-60          [-1, 512, 2, 2]       2,359,296
     BatchNorm2d-61          [-1, 512, 2, 2]           1,024
              ReLU-62          [-1, 512, 2, 2]               0
          Conv2d-63          [-1, 512, 2, 2]       2,359,296
     BatchNorm2d-64          [-1, 512, 2, 2]           1,024
              ReLU-65          [-1, 512, 2, 2]               0
        BasicBlock-66          [-1, 512, 2, 2]               0
AdaptiveAvgPool2d-67          [-1, 512, 1, 1]               0
          Linear-68                [-1, 3]           1,539
================================================================
Total params: 11,178,051
Trainable params: 11,178,051
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 5.13
Params size (MB): 42.64
Estimated Total Size (MB): 47.82
----------------------------------------------------------------
```

Influence of techniques on base model:

The test accuracy of the base model is 88% and test loss is 0.38. when we apply the overfitting prevention techniques the test accuracy increased by 1% i.e 89% and test loss is 0.43

Base Model:

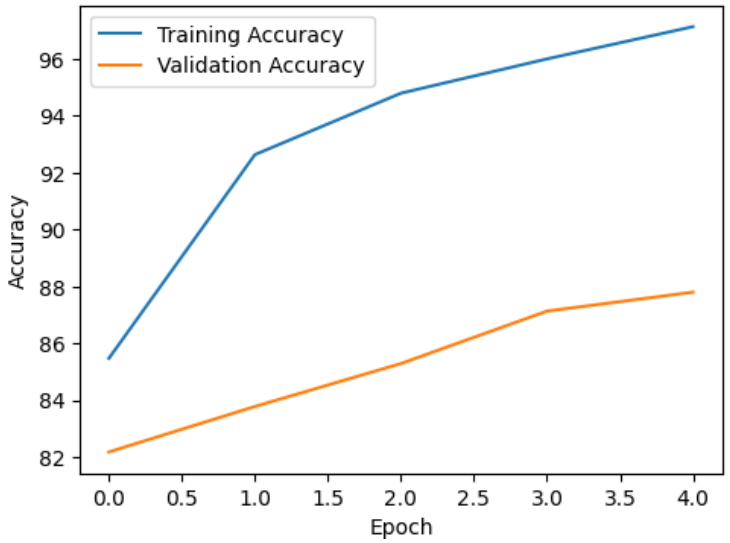
Train accuracy: 97.14%
Train loss: 0. 07
Validation accuracy: 87.8%
Validation loss: 0.44
Test accuracy: 88%
Test Loss: 0.42

From the below graphs, we can see that there's a consistent improvement in training accuracy, from 85.47% in the first epoch to 97.14% by the fifth epoch. This indicates that the model is effectively learning from the training data over time.

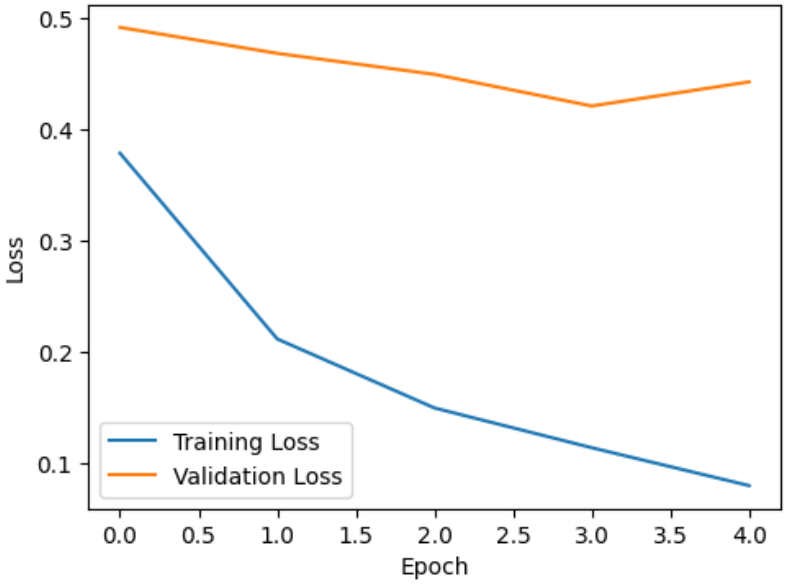
Validation accuracy started at 82.18% in the first epoch and experienced minor fluctuations, eventually improved to 87.8% by the fifth epoch

Test accuracy reported 88% and test loss is 0.42.

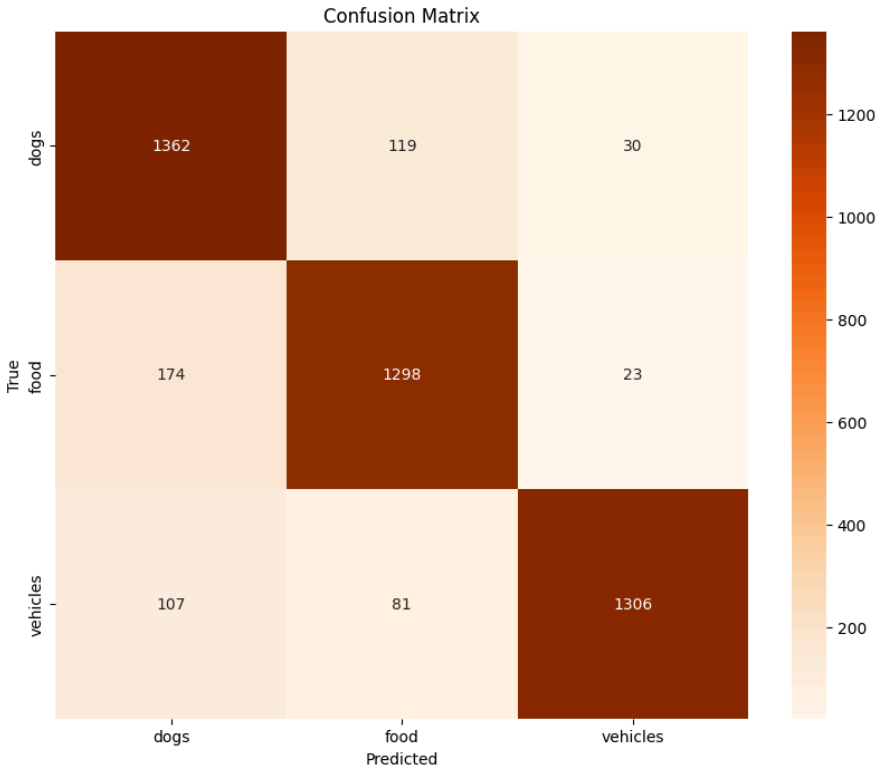
Training and Validation Accuracy:



Training and Validation Loss:



Confusion Matrix:



Evaluation Metrics:

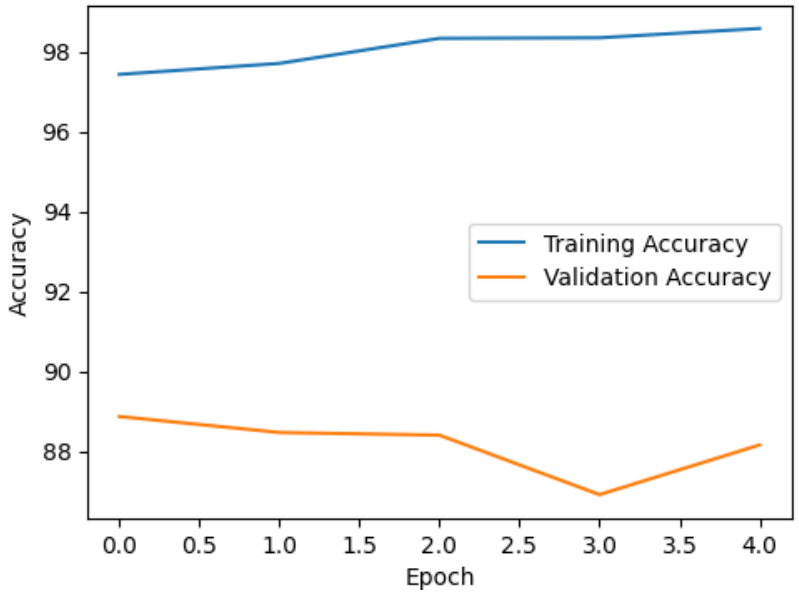
Precision: 0.89
Recall: 0.88
F1 Score: 0.88

Base+L2 Regularization:

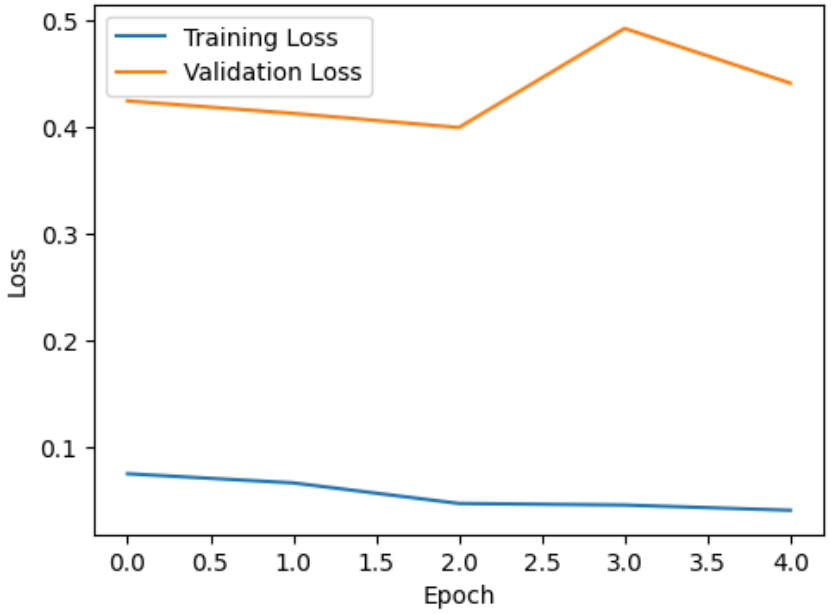
Train accuracy: 98.58%
Train loss: 0. 04
Validation accuracy: 88.16%
Validation loss: 0.44
Test accuracy: 88%
Test Loss: 0.42

From the below graphs we can see that the discrepancy between training accuracy and validation accuracy suggests that the model may be overfitting the training data.
Training loss is decreasing while the validation loss is not decreasing consistently which could be an indication of overfitting.

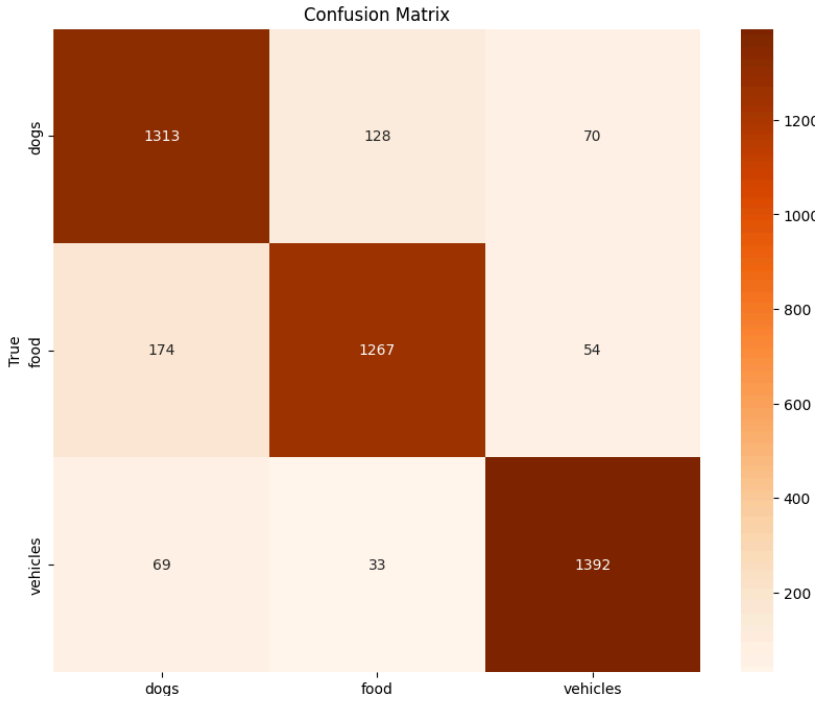
Training and Validation Accuracy:



Training and Validation Loss:



Confusion Matrix:

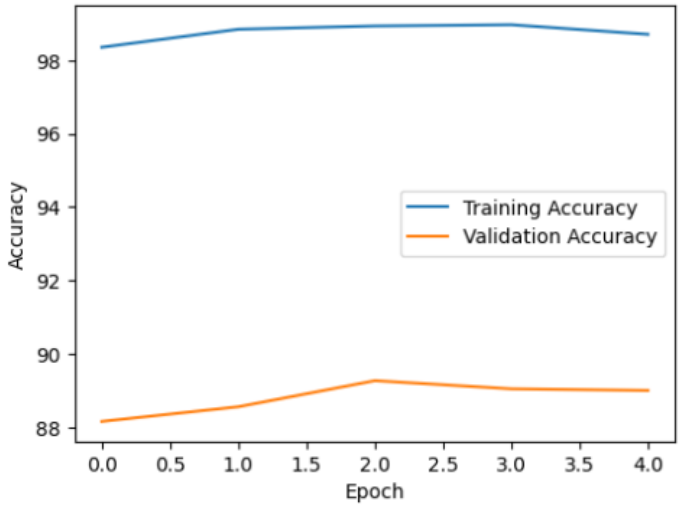


Evaluation Metrics:
Precision: 0.88
Recall: 0.88
F1 Score: 0.88

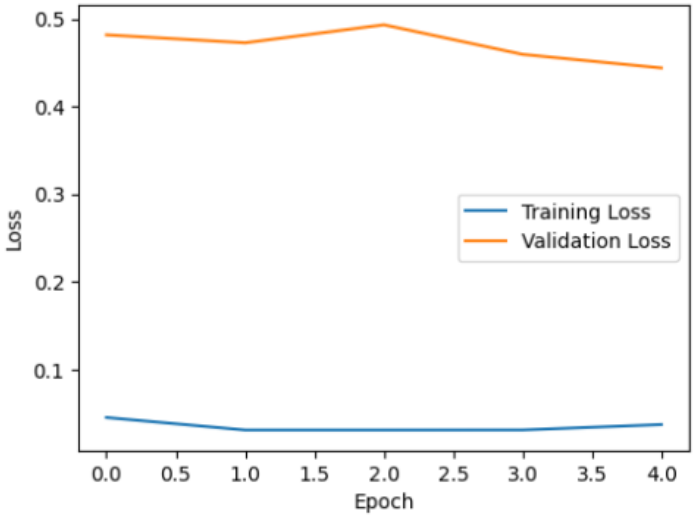
Base+L2 Regularization+Early Stopping:
Train accuracy: 98.71%
Train loss: 0. 03
Validation accuracy: 89.0%
Validation loss: 0.44
Test accuracy: 89%
Test Loss: 0.43

Even after applying Early stopping, train accuracy is above 95% and validation accuracy is below 90%. Again, indicates model overfitting.

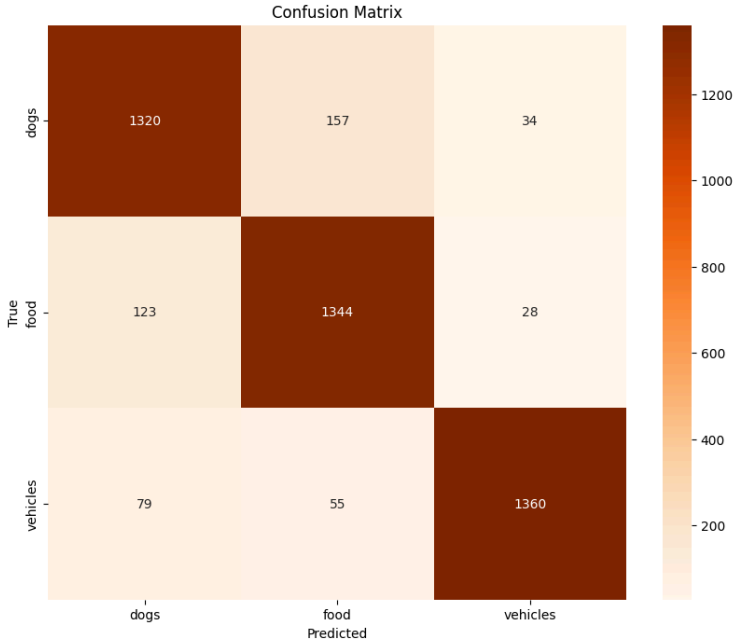
Training and Validation Accuracy:



Training and Validation Loss:



Confusion Matrix:



Evaluation Metrics:

Precision: 0.90
Recall: 0.89
F1 Score: 0.89

Part III

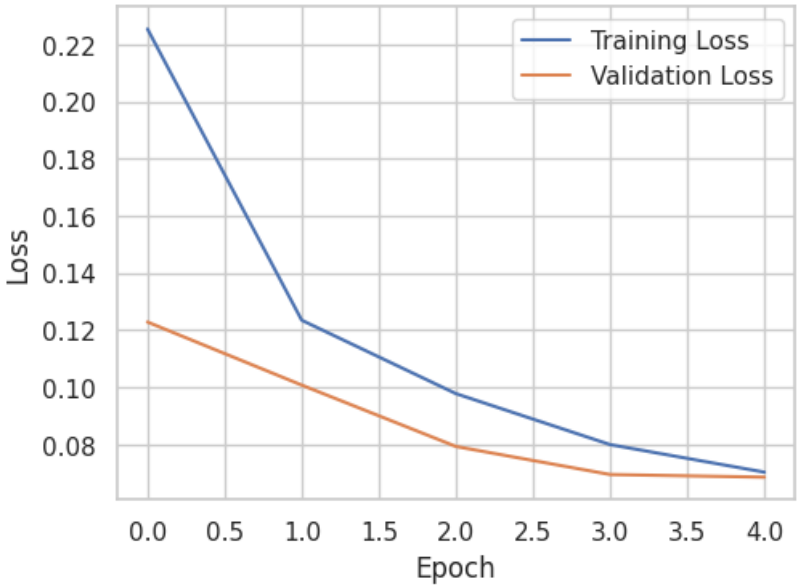
For this task, we chose Retail Sales Dataset. It consists of three csv files: features dataset.csv, sales dataset.csv and store dataset.csv.

Combination of these files provides a consolidated dataset that can be used for sales forecasting and retail analysis. The features and sales data can be joined on store and date fields to provide a temporal analysis of sales performance against various economic indicators.

Features dataset.csv has 8190 entries with 12 variables
Sales dataset.csv has 421570 entries with 5 variables
Stores dataset.csv has 45 entries with 3 variables.

RNN Architecture:
In our RNN model, we dynamically accept the input size based on the training dataset's feature count. The architecture consists of a three-layer RNN with a hidden size of 50 units. Each RNN layer passes its output to the subsequent layer, creating a deep network capable of capturing complex patterns in the sequential data. The final output from the last RNN layer's last time step is then fed into a fully connected linear layer to produce a single continuous output, suitable for regression tasks.

Training and Validation Loss:



The training and validation loss lines are converging, which suggests that the model is generalising well and not overfitting the training data.

RMSE: 0.967523422647
R2: -0.00023381550278323004

Part IV

Details of the Dataset
Twitter US Airline Sentiment

Nature/type of the dataset:

The dataset is a collection of tweets related to airlines, designed for sentiment analysis tasks.

This dataset consists of **14640** entries
And Number features are **15**

The total number of columns and their type are:

tweet_id int64
airline_sentiment object
airline_sentiment_confidence float64
negativereason object
negativereason_confidence float64
airline object
airline_sentiment_gold object
name object
negativereason_gold object
retweet_count int64
text object
tweet_coord object
tweet_created object
tweet_location object
user_timezone object
dtype: object

Defining Base LSTM Model

Summary of the architecture

base_LSTM(
  (embedding): Embedding(17065, 128)
  (lstm1): LSTM(128, 256, batch_first=True, dropout=0.1)
  (lstm2): LSTM(256, 256, batch_first=True, dropout=0.1)
  (lstm3): LSTM(256, 256, batch_first=True, dropout=0.1)

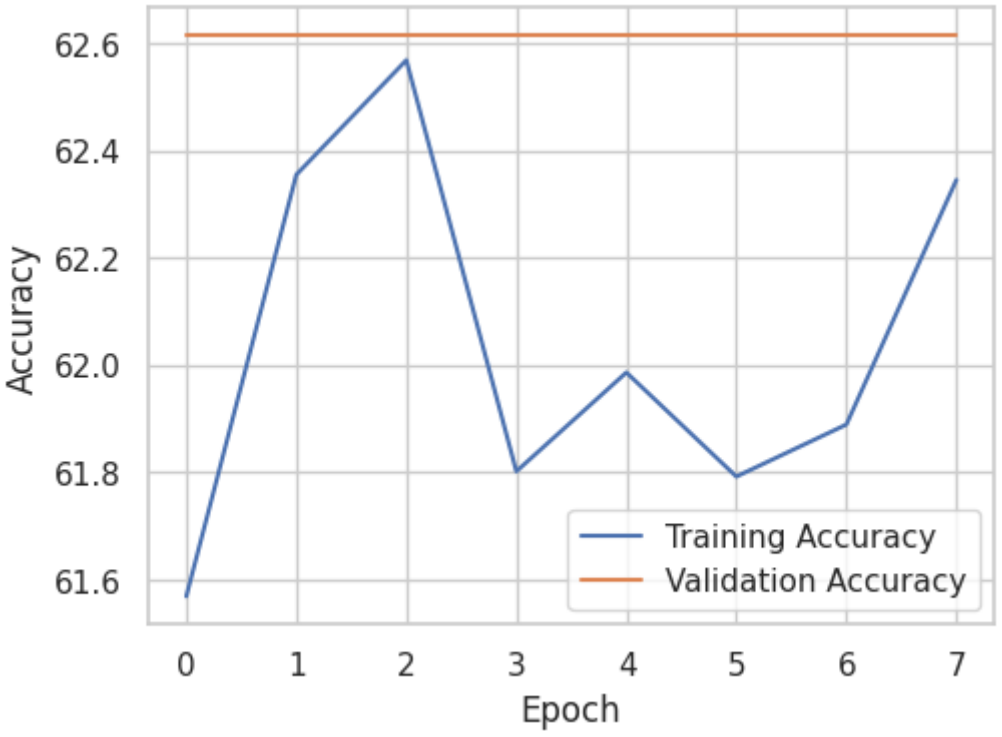
```
  (fc): Linear(in_features=256, out_features=3, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

After running 8 epochs the results we obtained are:

Train Accuracy: 62.34
Train Loss: 0.921
Validation Accuracy: 62.61
Validation Loss: 0.91
Test Loss: 0.879
Test Accuracy: 65%

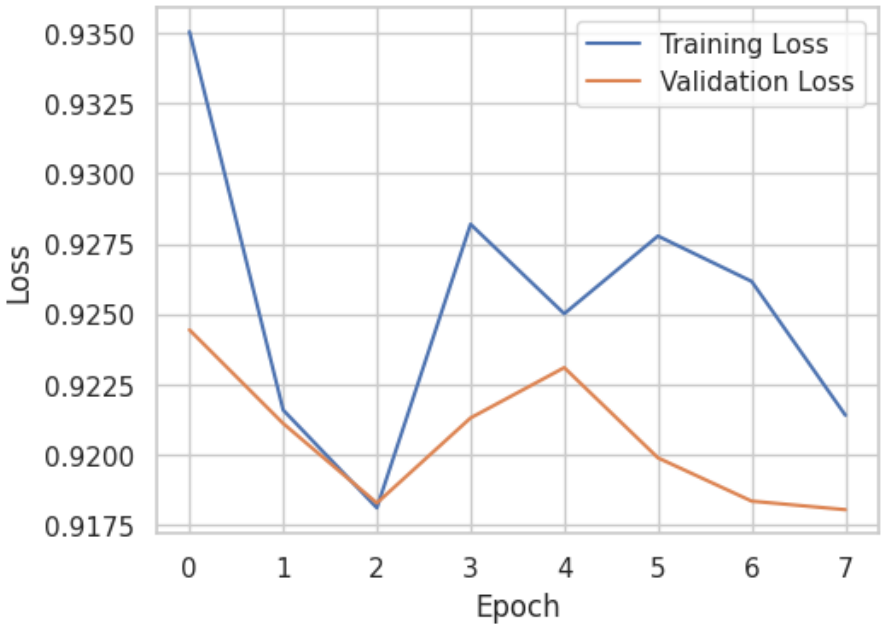
Training and Validation accuracy:

The training accuracy is consistently slightly lower than validation accuracy. The gap between the training accuracy and validation accuracy appears to be narrowing as the number of epochs are increasing. Therefore we can conclude the model is starting to generalise better to unseen data.

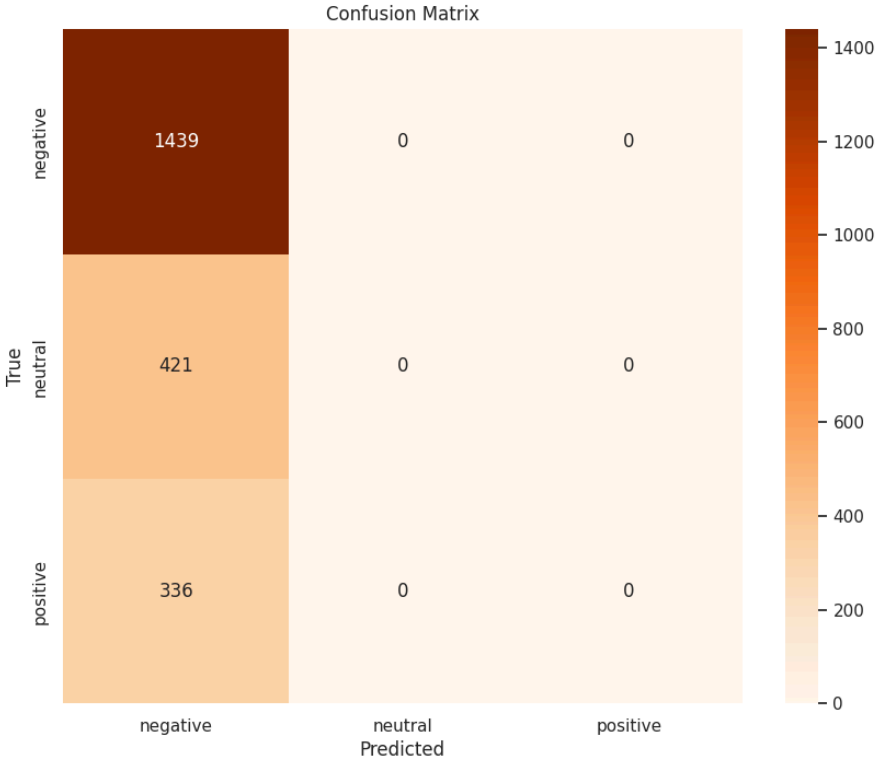


Training loss and Validation Loss:

The training loss starts at higher value than the validation loss and then decreases over the number of epochs. On the other hand the validation loss also trends downwards overtime and training loss is consistently higher than the validation loss throughout all the epochs as shown in the graph.



Confusion Matrix



Evaluation Metrics:

Precision: 0.43
Recall: 0.66
F1 Score: 0.5

Improved LSTM (Bidirectional LSTM):

Summary of the architecture:

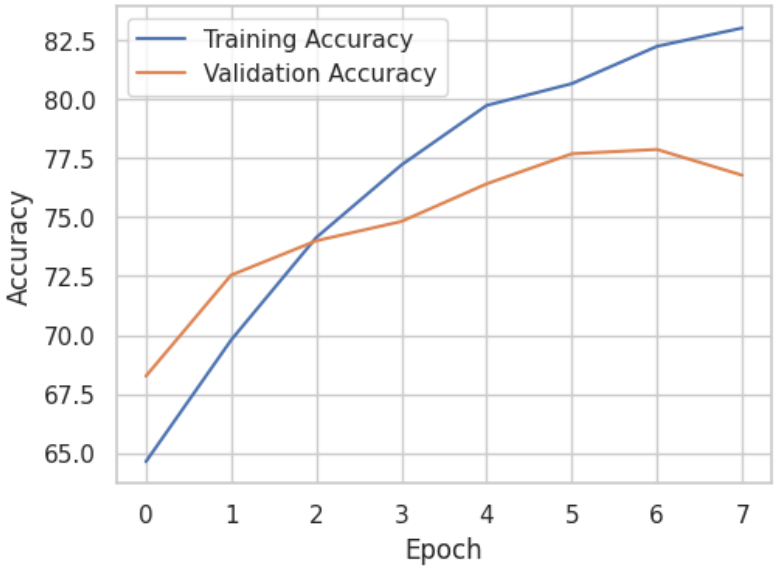
```
base_BiLSTM(
  (embedding): Embedding(17065, 128)
  (bilstm1): LSTM(128, 256, batch_first=True, dropout=0.1, bidirectional=True)
  (bilstm2): LSTM(512, 256, batch_first=True, dropout=0.1, bidirectional=True)
  (bilstm3): LSTM(512, 256, batch_first=True, dropout=0.1, bidirectional=True)
  (fc): Linear(in_features=512, out_features=3, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

The improved model is trained for 8 epochs and we obtained following results:

Training Accuracy: 83.02%
Train Loss: 0.43
Validation Accuracy: 76.78%
Validation loss: 0.57
Test accuracy: 79%
Test Loss: 0.53

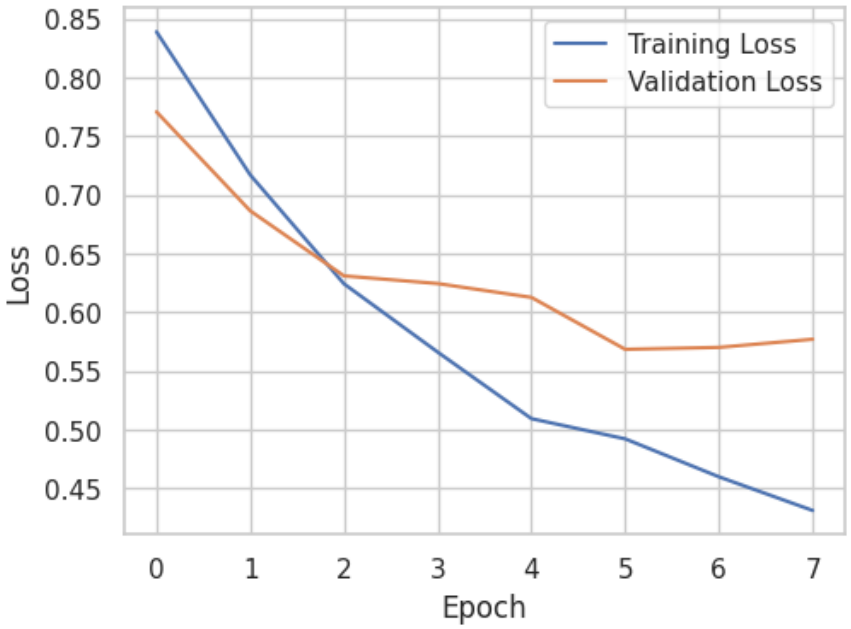
Training and validation accuracy:

The train accuracy starts at 65% and goes up to 82.5%. Whereas validation accuracy settled at approximately 77.5% which started at ~ 68%. The train accuracy is higher than validation accuracy. This implies the model is performing well on training data.



Training and validation loss:

The Training loss starts at a higher value than the validation loss and then decreases over the training. However the validation loss trends downward over time. From the graph we can infer that training loss is higher than validation loss for some epochs.The trends in the graph represents the model performing better over time.



Comparing the results of the LSTM and Bidirectional LSTM

Observation:

The comparison of the LSTM and Bi-LSTM models on the same dataset represents that Bi-LSTM outperforms the standard LSTM. The test accuracy of base LSTM increased by

14% by implementing bidirectional layers in the base LSTM model. The Bi-LSTM model is more stable and consistent learning with a smooth decrease in validation and training loss. On the other hand we can observe a steady increase in both training and validation accuracy.

Strengths and limitations of using RNN models for sentiment analysis.

Strengths:

1. RNNs are implemented to handle sequential data, which are best suited to sentiment analysis tasks
2. RNNs can handle variable lengths of input sequences, which makes them adapt to different lengths of text data.
3. RNNs consider the temporal aspect of text data, this is the crucial part of sentiment analysis because this sentiment depends on time period for instance product reviews or social media posts.

Limitations:

1. RNNs suffer from vanishing gradient problems, this leads to difficulties in capturing long-range dependencies in the text data.
2. They have limitations in capturing long term dependencies for example in vanilla RNN's
3. The RNNs are computationally intensive and time-consuming especially when dealing with large datasets.

Part V.I CNN

1. What is the output size after the first layer?
Given:
The input images are 32x28 RGB - so, W=32, H=28

P=0
F=5
D=10
S=1

Output Width = (W-F+2P)/S +1 = (32-5+0)/1 +1 = 28
Output Height= (H-F+2P)/S +1 = (28-5+0)/1 +1 = 24

Output Size = WxHxD = 28x24x10

2. How many parameters are there in the first layer?
Given:
Filter size = 5x5

No.of filters in the layer = 10
No.of input channels = 3(RGB)
No.of parameters = (5x5x3+1)x10 = 760

3. What would be the output size if a padding of 1 was used instead of zero Padding?

If P=1,
Output Width = (W-F+2P)/S +1 = (32-5+2x1)/1 +1 = 30
Output Height= (H-F+2P)/S +1 = (28-5+2x1)/1 +1 = 26

Output Size = WxHxD = 30x26x10

4. What would be the number of parameters if the input images were grayscale instead of RGB?

If input images were grayscale, then no.of input channels = 1
Filter size = 5x5
No.of filters in the layer = 10
No.of input channels = 1
No.of parameters = (5x5x1+1)x10 = 260

5. Considering the above specific task and the requirement of probabilistic outputs, which activation function would be most suitable for the output layer in this scenario?

Softmax Function

6. Prove that the activation function used here is invariant to constant shifts in the input values, meaning that adding a constant value to all the input values will not change the resulting probabilities.

6. The activation function used is Softmax.

Let's prove that Softmax activation function is invariant to constant shifts in the input values

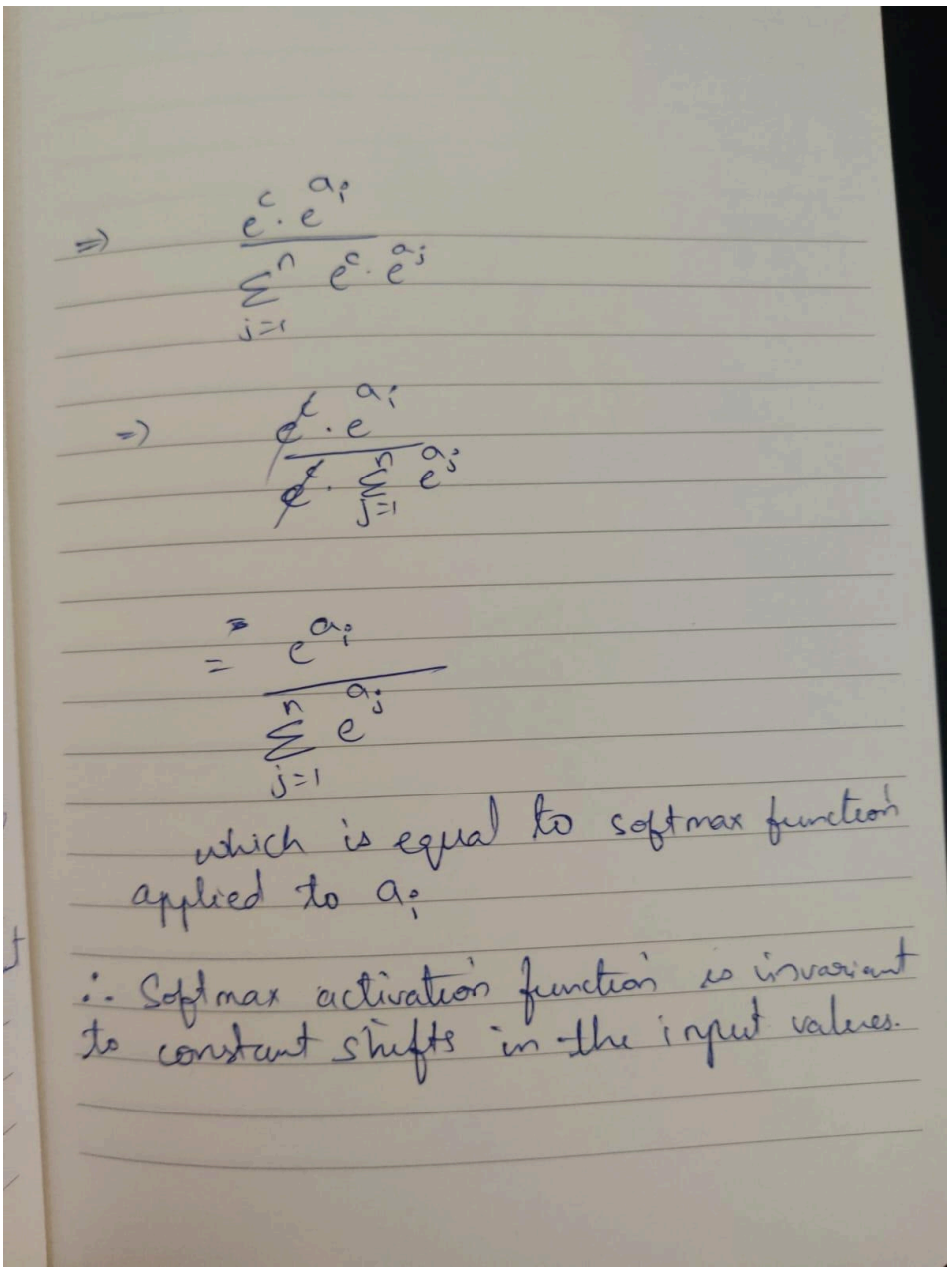
Softmax function for neural network A, with components $a_1, a_2, \ldots a_n$ for a classification problem with n classes, for i^{th} component is

$$\text{Softmax}(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}$$

Let's say a constant c is added to each input value, then new input vector is $A' = A + c$ (where $A' = a_1+c, a_2+c, \ldots + a_n+c$)

Softmax function applied to an element $a_i + c$ of A' is

$$\text{Softmax}(a_i+c) = \frac{e^{a_i+c}}{\sum_{j=1}^{n} e^{a_j+c}}$$



Part V.II: LSTM Derivation

Part V.II

Given:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

$$h_t = o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot g_t)$$

Derivative w.r.t c_t

$$\frac{\partial L}{\partial i_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial i_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial}{\partial i_t}\left(f_t \odot c_{t-1} + i_t \odot g_t\right) = \frac{\partial L}{\partial c_t} \cdot g_t$$

$$\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial}{\partial f_t}\left(f_t \odot c_{t-1} + i_t \odot g_t\right) = \frac{\partial L}{\partial c_t} \cdot (c_{t-1})$$

$$\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial o_t}\left(o_t \odot \tanh(c_t)\right)$$
$$= \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial o_t} \cdot \left(o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot g_t)\right)$$

$$\frac{\partial L}{\partial g_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial g_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial}{\partial g_t}\left(f_t \odot c_{t-1} + i_t \odot g_t\right) = \frac{\partial L}{\partial c_t} \cdot i_t$$

$$\frac{\partial L}{\partial c_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial c_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial c_t}\left(o_t \odot \tanh(c_t)\right)$$
$$= \frac{\partial L}{\partial h_t} \cdot o_t \left(1 - \tan^2 h(t)\right)$$

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_t} + \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial h_t} + \frac{\partial L}{\partial f_t} \cdot \frac{\partial f_t}{\partial h_t} + \frac{\partial L}{\partial i_t} \cdot \frac{\partial i_t}{\partial h_t}$$

For h_t

$$\frac{\partial L}{\partial i_t} = \frac{\partial h_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial i_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial i_t}\left(o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot g_t)\right)$$

$$\frac{\partial L}{\partial g_t} = \frac{\partial h_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial g_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial g_t}\left(o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot g_t)\right)$$

$$\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial f_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial}{\partial f_t}\left(o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot g_t)\right)$$

$$i_t = \sigma(w_1 \cdot [h_{t-1}, x_t])$$
$$f_t = \sigma(w_2 \cdot [h_{t-1}, x_t])$$
$$o_t = \sigma(w_3 \cdot [h_{t-1}, x_t])$$
$$g_t = \tanh(w_4 \cdot [h_{t-1}, x_t])$$

$$\therefore c_t = \sigma(w_2[h_{t-1}, x_t]) \odot c_{t-1} + (\sigma(w_1[h_{t-1}, x_t] \odot \tanh(w_4[h_{t-1}, x_t]))$$

$$h_t = \sigma(w_3[h_{t-1}, x_t]) \odot \tanh(c_t)$$

References:

1. https://arxiv.org/abs/1409.1556
2. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html
3. https://en.wikipedia.org/wiki/Early_stopping
4. https://pytorch.org/vision/stable/transforms.html
5. CSE 574 Machine Learning Assignment 2 submission by Dharma. Acha
6. https://pandas.pydata.org/
7. https://pytorch.org/vision/stable/models.html