Neural Network Models for Object Recognition

Machine learning module assignment

Guilherme Amorim December 2024

Initial setup

In [ ]:
```
# general library import
import os
import numpy as np
import pandas as pd
import itertools
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import pyplot
from datetime import datetime

# random seed generators
from numpy.random import seed
seed(888)
tf.random.set_seed(112)

# load dataset
from keras.datasets import cifar10 # importing the dataset

# modelling functions
from keras.models import Sequential       #to define model/ layers
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormal
from sklearn.metrics import confusion_matrix
from keras.applications.vgg19 import VGG19,preprocess_input

# image viewing
from IPython.display import display
from keras.preprocessing.image import array_to_img

# data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# buzzer when code finishes
from google.colab import output
def buzzer():
  output.eval_js('new Audio("https://ssl.gstatic.com/dictionary/static/pronuncia
```

In [ ]:
```
# transforming dataset into dataframe

(x_train_all, y_train_all), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 13s 0us/step
```

In [ ]:
```
# creating list with labels

LABEL_NAMES = ['airplane', 'automobile','bird','cat', 'deer', 'dog', 'frog', 'hc
```

Dataset exploration

Investigate images

In [ ]:
```
print(x_train_all.shape)

print(x_train_all)
```

```
(50000, 32, 32, 3)
[[[[ 59  62  63]
   [ 43  46  45]
   [ 50  48  43]
   ...
   [158 132 108]
   [152 125 102]
   [148 124 103]]

  [[ 16  20  20]
   [  0   0   0]
   [ 18   8   0]
   ...
   [123  88  55]
   [119  83  50]
   [122  87  57]]

  [[ 25  24  21]
   [ 16   7   0]
   [ 49  27   8]
   ...
   [118  84  50]
   [120  84  50]
   [109  73  42]]

  ...

  [[208 170  96]
   [201 153  34]
   [198 161  26]
   ...
   [160 133  70]
   [ 56  31   7]
   [ 53  34  20]]

  [[180 139  96]
   [173 123  42]
   [186 144  30]
   ...
   [184 148  94]
   [ 97  62  34]
   [ 83  53  34]]

  [[177 144 116]
   [168 129  94]
   [179 142  87]
   ...
   [216 184 140]
   [151 118  84]
   [123  92  72]]]


 [[[154 177 187]
   [126 137 136]
   [105 104  95]
   ...
   [ 91  95  71]
   [ 87  90  71]
   [ 79  81  70]]
```

```
[[140 160 169]
  [145 153 154]
  [125 125 118]
  ...
  [ 96  99  78]
  [ 77  80  62]
  [ 71  73  61]]

 [[140 155 164]
  [139 146 149]
  [115 115 112]
  ...
  [ 79  82  64]
  [ 68  70  55]
  [ 67  69  55]]

 ...

 [[175 167 166]
  [156 154 160]
  [154 160 170]
  ...
  [ 42  34  36]
  [ 61  53  57]
  [ 93  83  91]]

 [[165 154 128]
  [156 152 130]
  [159 161 142]
  ...
  [103  93  96]
  [123 114 120]
  [131 121 131]]

 [[163 148 120]
  [158 148 122]
  [163 156 133]
  ...
  [143 133 139]
  [143 134 142]
  [143 133 144]]]


[[[255 255 255]
  [253 253 253]
  [253 253 253]
  ...
  [253 253 253]
  [253 253 253]
  [253 253 253]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
```

```
  [254 254 254]
  [254 254 254]
  ...
  [254 254 254]
  [254 254 254]
  [254 254 254]]

 ...

 [[113 120 112]
  [111 118 111]
  [105 112 106]
  ...
  [ 72  81  80]
  [ 72  80  79]
  [ 72  80  79]]

 [[111 118 110]
  [104 111 104]
  [ 99 106  98]
  ...
  [ 68  75  73]
  [ 70  76  75]
  [ 78  84  82]]

 [[106 113 105]
  [ 99 106  98]
  [ 95 102  94]
  ...
  [ 78  85  83]
  [ 79  85  83]
  [ 80  86  84]]]


...


[[[ 35 178 235]
  [ 40 176 239]
  [ 42 176 241]
  ...
  [ 99 177 219]
  [ 79 147 197]
  [ 89 148 189]]

 [[ 57 182 234]
  [ 44 184 250]
  [ 50 183 240]
  ...
  [156 182 200]
  [141 177 206]
  [116 149 175]]

 [[ 98 197 237]
  [ 64 189 252]
  [ 69 192 245]
  ...
  [188 195 206]
  [119 135 147]
  [ 61  79  90]]
```

```
  ...

  [[ 73  79  77]
   [ 53  63  68]
   [ 54  68  80]
   ...
   [ 17  40  64]
   [ 21  36  51]
   [ 33  48  49]]

  [[ 61  68  75]
   [ 55  70  86]
   [ 57  79 103]
   ...
   [ 24  48  72]
   [ 17  35  53]
   [  7  23  32]]

  [[ 44  56  73]
   [ 46  66  88]
   [ 49  77 105]
   ...
   [ 27  52  77]
   [ 21  43  66]
   [ 12  31  50]]]


 [[[189 211 240]
   [186 208 236]
   [185 207 235]
   ...
   [175 195 224]
   [172 194 222]
   [169 194 220]]

  [[194 210 239]
   [191 207 236]
   [190 206 235]
   ...
   [173 192 220]
   [171 191 218]
   [167 190 216]]

  [[208 219 244]
   [205 216 240]
   [204 215 239]
   ...
   [175 191 217]
   [172 190 216]
   [169 191 215]]

  ...

  [[207 199 181]
   [203 195 175]
   [203 196 173]
   ...
   [135 132 127]
   [162 158 150]
```

```
  [168 163 151]]

 [[198 190 170]
  [189 181 159]
  [180 172 147]
  ...
  [178 171 160]
  [175 169 156]
  [175 169 154]]

 [[198 189 173]
  [189 181 162]
  [178 170 149]
  ...
  [195 184 169]
  [196 189 171]
  [195 190 171]]]


[[[229 229 239]
  [236 237 247]
  [234 236 247]
  ...
  [217 219 233]
  [221 223 234]
  [222 223 233]]

 [[222 221 229]
  [239 239 249]
  [233 234 246]
  ...
  [223 223 236]
  [227 228 238]
  [210 211 220]]

 [[213 206 211]
  [234 232 239]
  [231 233 244]
  ...
  [220 220 232]
  [220 219 232]
  [202 203 215]]

 ...

 [[150 143 135]
  [140 135 127]
  [132 127 120]
  ...
  [224 222 218]
  [230 228 225]
  [241 241 238]]

 [[137 132 126]
  [130 127 120]
  [125 121 115]
  ...
  [181 180 178]
  [202 201 198]
  [212 211 207]]
```

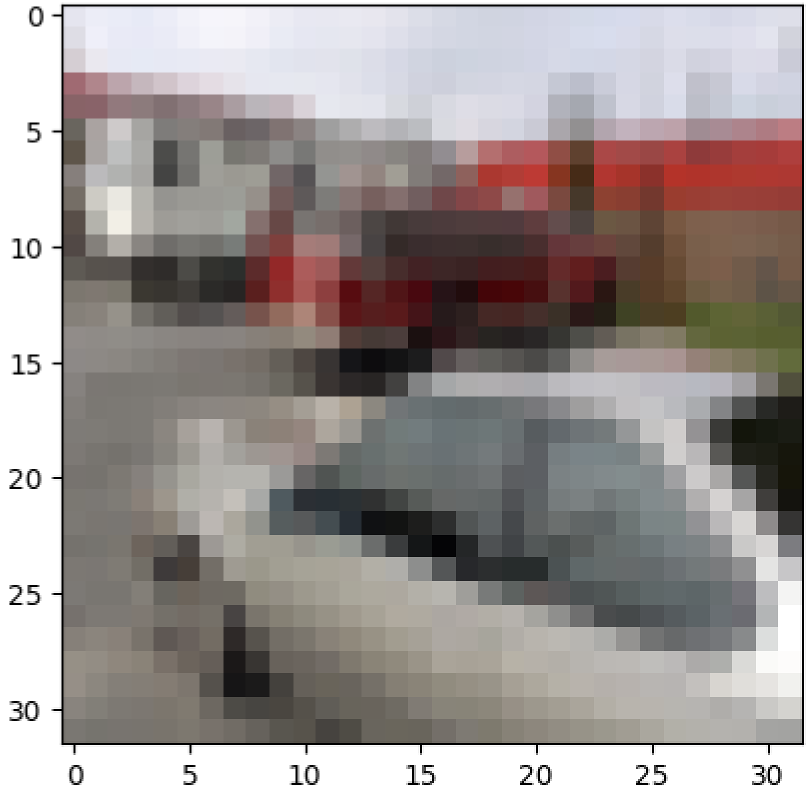
```
[[122 119 114]
 [118 116 110]
 [120 116 111]
 ...
 [179 177 173]
 [164 164 162]
 [163 163 161]]]]
```

50000 elements, 32x32 pixel images, each pixel has 3 values (RGB channels) ranging from 0-255

The structure is: [image [column [row [pixel]]]

In [ ]:
```
# investigate an individual image
plt.imshow(x_train_all[49999])
```

Out[ ]: <matplotlib.image.AxesImage at 0x7886e070afe0>



In [ ]:
```
x_train_all[49999].shape
```

Out[ ]: (32, 32, 3)

In [ ]:
```
# checking test images

number_of_images, x, y, c = x_test.shape
print(f'Number of images (test set) = {number_of_images}\nWidth = {x} pixels\nHe
```

```
Number of images (test set) = 10000
Width = 32 pixels
Height = 32 pixels
Channels = 3
```

Investigate labels

In [ ]:
```
# general shape
y_train_all.shape
```

Out[ ]:
```
(50000, 1)
```

In [ ]:
```
# investigate label for a single image
y_train_all[49999][0]
```

Out[ ]:
```
1
```

In [ ]:
```
# matching with label names
LABEL_NAMES[y_train_all[49999][0]]
```

Out[ ]:
```
'automobile'
```

Preprocessing

Scalling pixel values

In [ ]:
```
x_train_all =x_train_all / 255.0
x_test =  x_test / 255.0
```

In [ ]:
```
x_test
```

```
Out[ ]: array([[[[0.61960784, 0.43921569, 0.19215686],
                 [0.62352941, 0.43529412, 0.18431373],
                 [0.64705882, 0.45490196, 0.2       ],
                 ...,
                 [0.5372549 , 0.37254902, 0.14117647],
                 [0.49411765, 0.35686275, 0.14117647],
                 [0.45490196, 0.33333333, 0.12941176]],

                [[0.59607843, 0.43921569, 0.2       ],
                 [0.59215686, 0.43137255, 0.15686275],
                 [0.62352941, 0.44705882, 0.17647059],
                 ...,
                 [0.53333333, 0.37254902, 0.12156863],
                 [0.49019608, 0.35686275, 0.1254902 ],
                 [0.46666667, 0.34509804, 0.13333333]],

                [[0.59215686, 0.43137255, 0.18431373],
                 [0.59215686, 0.42745098, 0.12941176],
                 [0.61960784, 0.43529412, 0.14117647],
                 ...,
                 [0.54509804, 0.38431373, 0.13333333],
                 [0.50980392, 0.37254902, 0.13333333],
                 [0.47058824, 0.34901961, 0.12941176]],

                ...,

                [[0.26666667, 0.48627451, 0.69411765],
                 [0.16470588, 0.39215686, 0.58039216],
                 [0.12156863, 0.34509804, 0.5372549 ],
                 ...,
                 [0.14901961, 0.38039216, 0.57254902],
                 [0.05098039, 0.25098039, 0.42352941],
                 [0.15686275, 0.33333333, 0.49803922]],

                [[0.23921569, 0.45490196, 0.65882353],
                 [0.19215686, 0.4       , 0.58039216],
                 [0.1372549 , 0.33333333, 0.51764706],
                 ...,
                 [0.10196078, 0.32156863, 0.50980392],
                 [0.11372549, 0.32156863, 0.49411765],
                 [0.07843137, 0.25098039, 0.41960784]],

                [[0.21176471, 0.41960784, 0.62745098],
                 [0.21960784, 0.41176471, 0.58431373],
                 [0.17647059, 0.34901961, 0.51764706],
                 ...,
                 [0.09411765, 0.30196078, 0.48627451],
                 [0.13333333, 0.32941176, 0.50588235],
                 [0.08235294, 0.2627451 , 0.43137255]]],


               [[[0.92156863, 0.92156863, 0.92156863],
                 [0.90588235, 0.90588235, 0.90588235],
                 [0.90980392, 0.90980392, 0.90980392],
                 ...,
                 [0.91372549, 0.91372549, 0.91372549],
                 [0.91372549, 0.91372549, 0.91372549],
                 [0.90980392, 0.90980392, 0.90980392]],

                [[0.93333333, 0.93333333, 0.93333333],
```

```
  [0.92156863, 0.92156863, 0.92156863],
  [0.92156863, 0.92156863, 0.92156863],
  ...,
  [0.9254902 , 0.9254902 , 0.9254902 ],
  [0.9254902 , 0.9254902 , 0.9254902 ],
  [0.92156863, 0.92156863, 0.92156863]],

 [[0.92941176, 0.92941176, 0.92941176],
  [0.91764706, 0.91764706, 0.91764706],
  [0.91764706, 0.91764706, 0.91764706],
  ...,
  [0.92156863, 0.92156863, 0.92156863],
  [0.92156863, 0.92156863, 0.92156863],
  [0.91764706, 0.91764706, 0.91764706]],

 ...,

 [[0.34117647, 0.38823529, 0.34901961],
  [0.16862745, 0.2       , 0.14509804],
  [0.0745098 , 0.09019608, 0.04313725],
  ...,
  [0.6627451 , 0.72156863, 0.70196078],
  [0.71372549, 0.77254902, 0.75686275],
  [0.7372549 , 0.79215686, 0.78823529]],

 [[0.32156863, 0.37647059, 0.32156863],
  [0.18039216, 0.22352941, 0.14117647],
  [0.14117647, 0.17254902, 0.08627451],
  ...,
  [0.68235294, 0.74117647, 0.71764706],
  [0.7254902 , 0.78431373, 0.76862745],
  [0.73333333, 0.79215686, 0.78431373]],

 [[0.33333333, 0.39607843, 0.3254902 ],
  [0.24313725, 0.29411765, 0.18823529],
  [0.22745098, 0.2627451 , 0.14901961],
  ...,
  [0.65882353, 0.71764706, 0.69803922],
  [0.70588235, 0.76470588, 0.74901961],
  [0.72941176, 0.78431373, 0.78039216]]],


[[[0.61960784, 0.74509804, 0.87058824],
  [0.61960784, 0.73333333, 0.85490196],
  [0.54509804, 0.65098039, 0.76078431],
  ...,
  [0.89411765, 0.90588235, 0.91764706],
  [0.92941176, 0.9372549 , 0.95294118],
  [0.93333333, 0.94509804, 0.96470588]],

 [[0.66666667, 0.78431373, 0.89803922],
  [0.6745098 , 0.78039216, 0.88627451],
  [0.59215686, 0.69019608, 0.78823529],
  ...,
  [0.90980392, 0.90980392, 0.9254902 ],
  [0.96470588, 0.96470588, 0.98039216],
  [0.96470588, 0.96862745, 0.98431373]],

 [[0.68235294, 0.78823529, 0.88235294],
  [0.69019608, 0.78431373, 0.87058824],
```

```
  [0.61568627, 0.70196078, 0.78039216],
  ...,
  [0.90196078, 0.89803922, 0.90980392],
  [0.98039216, 0.97647059, 0.98431373],
  [0.96078431, 0.95686275, 0.96862745]],

 ...,

 [[0.12156863, 0.15686275, 0.17647059],
  [0.11764706, 0.15294118, 0.17254902],
  [0.10196078, 0.1372549 , 0.15686275],
  ...,
  [0.14509804, 0.15686275, 0.18039216],
  [0.03529412, 0.05098039, 0.05490196],
  [0.01568627, 0.02745098, 0.01960784]],

 [[0.09019608, 0.13333333, 0.15294118],
  [0.10588235, 0.14901961, 0.16862745],
  [0.09803922, 0.14117647, 0.16078431],
  ...,
  [0.0745098 , 0.07843137, 0.09411765],
  [0.01568627, 0.02352941, 0.01176471],
  [0.01960784, 0.02745098, 0.01176471]],

 [[0.10980392, 0.16078431, 0.18431373],
  [0.11764706, 0.16862745, 0.19607843],
  [0.1254902 , 0.17647059, 0.20392157],
  ...,
  [0.01960784, 0.02352941, 0.03137255],
  [0.01568627, 0.01960784, 0.01176471],
  [0.02745098, 0.03137255, 0.02745098]]],


...,


[[[0.07843137, 0.05882353, 0.04705882],
  [0.0745098 , 0.05490196, 0.04313725],
  [0.05882353, 0.05490196, 0.04313725],
  ...,
  [0.03921569, 0.03529412, 0.02745098],
  [0.04705882, 0.04313725, 0.03529412],
  [0.05098039, 0.04705882, 0.03921569]],

 [[0.08235294, 0.0627451 , 0.05098039],
  [0.07843137, 0.0627451 , 0.05098039],
  [0.07058824, 0.06666667, 0.04705882],
  ...,
  [0.03921569, 0.03529412, 0.02745098],
  [0.03921569, 0.03529412, 0.02745098],
  [0.04705882, 0.04313725, 0.03529412]],

 [[0.08235294, 0.0627451 , 0.05098039],
  [0.08235294, 0.06666667, 0.04705882],
  [0.07843137, 0.07058824, 0.04313725],
  ...,
  [0.04705882, 0.04313725, 0.03529412],
  [0.04705882, 0.04313725, 0.03529412],
  [0.05098039, 0.04705882, 0.03921569]],
```

```
        ...,

        [[0.12941176, 0.09803922, 0.05098039],
         [0.13333333, 0.10196078, 0.05882353],
         [0.13333333, 0.10196078, 0.05882353],
         ...,
         [0.10980392, 0.09803922, 0.20392157],
         [0.11372549, 0.09803922, 0.22745098],
         [0.09019608, 0.07843137, 0.16470588]],

        [[0.12941176, 0.09803922, 0.05490196],
         [0.13333333, 0.10196078, 0.05882353],
         [0.13333333, 0.10196078, 0.05882353],
         ...,
         [0.10588235, 0.09411765, 0.20392157],
         [0.10588235, 0.09411765, 0.21960784],
         [0.09803922, 0.08627451, 0.18431373]],

        [[0.12156863, 0.09019608, 0.04705882],
         [0.1254902 , 0.09411765, 0.05098039],
         [0.12941176, 0.09803922, 0.05490196],
         ...,
         [0.09411765, 0.09019608, 0.19607843],
         [0.10196078, 0.09019608, 0.20784314],
         [0.09803922, 0.07843137, 0.18431373]]],


       [[[0.09803922, 0.15686275, 0.04705882],
         [0.05882353, 0.14117647, 0.01176471],
         [0.09019608, 0.16078431, 0.07058824],
         ...,
         [0.23921569, 0.32156863, 0.30588235],
         [0.36078431, 0.44313725, 0.43921569],
         [0.29411765, 0.34901961, 0.36078431]],

        [[0.04705882, 0.09803922, 0.02352941],
         [0.07843137, 0.14509804, 0.02745098],
         [0.09411765, 0.14117647, 0.05882353],
         ...,
         [0.45098039, 0.5254902 , 0.54117647],
         [0.58431373, 0.65882353, 0.69411765],
         [0.40784314, 0.45882353, 0.51372549]],

        [[0.04705882, 0.09803922, 0.04313725],
         [0.05882353, 0.11372549, 0.02352941],
         [0.13333333, 0.15686275, 0.09411765],
         ...,
         [0.60392157, 0.6745098 , 0.71372549],
         [0.61568627, 0.68627451, 0.75294118],
         [0.45490196, 0.50588235, 0.59215686]],

        ...,

        [[0.39215686, 0.50588235, 0.31764706],
         [0.40392157, 0.51764706, 0.32941176],
         [0.40784314, 0.5254902 , 0.3372549 ],
         ...,
         [0.38039216, 0.50196078, 0.32941176],
         [0.38431373, 0.49411765, 0.32941176],
         [0.35686275, 0.4745098 , 0.30980392]],
```

```
 [[0.40392157, 0.51764706, 0.3254902 ],
  [0.40784314, 0.51372549, 0.3254902 ],
  [0.41960784, 0.52941176, 0.34117647],
  ...,
  [0.39607843, 0.51764706, 0.34117647],
  [0.38823529, 0.49803922, 0.32941176],
  [0.36078431, 0.4745098 , 0.30980392]],

 [[0.37254902, 0.49411765, 0.30588235],
  [0.37254902, 0.48235294, 0.29803922],
  [0.39607843, 0.50196078, 0.31764706],
  ...,
  [0.36470588, 0.48627451, 0.31372549],
  [0.37254902, 0.48235294, 0.31764706],
  [0.36078431, 0.47058824, 0.31372549]]],


[[[0.28627451, 0.30588235, 0.29411765],
  [0.38431373, 0.40392157, 0.44313725],
  [0.38823529, 0.41568627, 0.44705882],
  ...,
  [0.52941176, 0.58823529, 0.59607843],
  [0.52941176, 0.58431373, 0.60392157],
  [0.79607843, 0.84313725, 0.8745098 ]],

 [[0.27058824, 0.28627451, 0.2745098 ],
  [0.32941176, 0.34901961, 0.38039216],
  [0.26666667, 0.29411765, 0.31764706],
  ...,
  [0.33333333, 0.37254902, 0.34901961],
  [0.27843137, 0.32156863, 0.31372549],
  [0.47058824, 0.52156863, 0.52941176]],

 [[0.27058824, 0.28627451, 0.2745098 ],
  [0.35294118, 0.37254902, 0.39215686],
  [0.24313725, 0.27843137, 0.29019608],
  ...,
  [0.29019608, 0.31764706, 0.2745098 ],
  [0.20784314, 0.24313725, 0.21176471],
  [0.24313725, 0.29019608, 0.27058824]],

 ...,

 [[0.48235294, 0.50196078, 0.37647059],
  [0.51764706, 0.51764706, 0.4       ],
  [0.50588235, 0.50196078, 0.39215686],
  ...,
  [0.42352941, 0.41960784, 0.34509804],
  [0.24313725, 0.23529412, 0.21568627],
  [0.10588235, 0.10588235, 0.10980392]],

 [[0.45098039, 0.4745098 , 0.35686275],
  [0.48235294, 0.48627451, 0.37254902],
  [0.50588235, 0.49411765, 0.38823529],
  ...,
  [0.45098039, 0.45490196, 0.36862745],
  [0.25882353, 0.25490196, 0.23137255],
  [0.10588235, 0.10588235, 0.10588235]],
```

```
[[0.45490196, 0.47058824, 0.35294118],
 [0.4745098 , 0.47843137, 0.36862745],
 [0.50588235, 0.50196078, 0.39607843],
 ...,
 [0.45490196, 0.45098039, 0.36862745],
 [0.26666667, 0.25490196, 0.22745098],
 [0.10588235, 0.10196078, 0.10196078]]]])
```

Creating validation set

Original dataset already split 50000:10000 into train and test data.
Validation split could be performed with cross-validation (k-fold, LOOCV) or train-validation split. In this case, given the very large amount of instances in the dataset, we can choose a train-validation split without worrying too much about the impact of the initial split (but can verify this further).

No clear rule as to how to split into train and validation set. Given the large amount of data, an 80/20 split (40000:10000) should be appropriate.

In [ ]:
```
# creating the validation set
VALIDATION_SIZE = 10000

x_val = x_train_all[:VALIDATION_SIZE]
y_val = y_train_all[:VALIDATION_SIZE]
print("Image validation set shape:\n", x_val.shape)
print("Labels validation set shape:\n", y_val.shape)
```

```
Image validation set shape:
 (10000, 32, 32, 3)
Labels validation set shape:
 (10000, 1)
```

In [ ]:
```
# removing validation set instances from training set
x_train = x_train_all[VALIDATION_SIZE:]
y_train= y_train_all[VALIDATION_SIZE:]
print("Image training set shape:\n", x_train.shape)
print("Labels training set shape:\n", y_train.shape)
```

```
Image training set shape:
 (40000, 32, 32, 3)
Labels training set shape:
 (40000, 1)
```

Building model

Defining early stopping rule

In [ ]:
```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

Baseline model (training materials) - SGD optimizer

In [ ]:
```
model_1 = Sequential()

## ************* FIRST SET OF LAYERS ************************

# CONVOLUTIONAL LAYER
model_1.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activa
# POOLING LAYER
model_1.add(MaxPool2D(pool_size=(2, 2)))

## *************** SECOND SET OF LAYERS **********************

# *************CONVOLUTIONAL LAYER
model_1.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activa
# POOLING LAYER
model_1.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 32 x 32 x 3 =3072 BEFORE FINAL LAYER
model_1.add(Flatten())

# 256 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model_1.add(Dense(256, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model_1.add(Dense(10, activation='softmax'))


model_1.compile(loss='sparse_categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.
py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [ ]:
```
model_1.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 29, 29, 32)	
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	
conv2d_1 (Conv2D)	(None, 11, 11, 32)	
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	
flatten_2 (Flatten)	(None, 800)	
dense_6 (Dense)	(None, 256)	
dense_7 (Dense)	(None, 10)	

Total params: 225,610 (881.29 KB)

Trainable params: 225,610 (881.29 KB)

Non-trainable params: 0 (0.00 B)

Fitting model

In [ ]:
```
start = datetime.now()
history_model_1 = model_1.fit(x_train,
                              y_train,
                    epochs=50, # number of epochs
                    batch_size=64, # batch size
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )

# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1583 - loss: 2.2371 - val_
accuracy: 0.2718 - val_loss: 1.9852
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3058 - loss: 1.9283 - val_
accuracy: 0.3528 - val_loss: 1.7943
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3719 - loss: 1.7585 - val_
accuracy: 0.4098 - val_loss: 1.6359
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4167 - loss: 1.6262 - val_
accuracy: 0.4466 - val_loss: 1.5300
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4447 - loss: 1.5432 - val_
accuracy: 0.4726 - val_loss: 1.4671
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4673 - loss: 1.4845 - val_
accuracy: 0.4913 - val_loss: 1.4194
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4869 - loss: 1.4365 - val_
accuracy: 0.5068 - val_loss: 1.3805
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5034 - loss: 1.3943 - val_
accuracy: 0.5210 - val_loss: 1.3466
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5182 - loss: 1.3564 - val_
accuracy: 0.5334 - val_loss: 1.3152
Epoch 10/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5336 - loss: 1.3218 - val_
accuracy: 0.5428 - val_loss: 1.2889
Epoch 11/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5478 - loss: 1.2891 - val_
accuracy: 0.5521 - val_loss: 1.2653
Epoch 12/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5582 - loss: 1.2586 - val_
accuracy: 0.5603 - val_loss: 1.2427
Epoch 13/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5695 - loss: 1.2296 - val_
accuracy: 0.5666 - val_loss: 1.2220
Epoch 14/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5796 - loss: 1.2019 - val_
accuracy: 0.5757 - val_loss: 1.2017
Epoch 15/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.5896 - loss: 1.1749 - val_
accuracy: 0.5814 - val_loss: 1.1868
Epoch 16/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.5990 - loss: 1.1490 - val_
accuracy: 0.5887 - val_loss: 1.1703
Epoch 17/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6072 - loss: 1.1240 - val_
accuracy: 0.5948 - val_loss: 1.1572
Epoch 18/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6166 - loss: 1.1002 - val_
accuracy: 0.5997 - val_loss: 1.1421
Epoch 19/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6251 - loss: 1.0766 - val_
accuracy: 0.6042 - val_loss: 1.1293
Epoch 20/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6331 - loss: 1.0543 - val_
accuracy: 0.6070 - val_loss: 1.1189
```

```
Epoch 21/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6417 - loss: 1.0324 - val_
accuracy: 0.6120 - val_loss: 1.1088
Epoch 22/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6507 - loss: 1.0112 - val_
accuracy: 0.6148 - val_loss: 1.1002
Epoch 23/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6588 - loss: 0.9909 - val_
accuracy: 0.6181 - val_loss: 1.0917
Epoch 24/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6663 - loss: 0.9709 - val_
accuracy: 0.6220 - val_loss: 1.0864
Epoch 25/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6733 - loss: 0.9514 - val_
accuracy: 0.6241 - val_loss: 1.0816
Epoch 26/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6802 - loss: 0.9328 - val_
accuracy: 0.6294 - val_loss: 1.0715
Epoch 27/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6860 - loss: 0.9146 - val_
accuracy: 0.6315 - val_loss: 1.0683
Epoch 28/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6921 - loss: 0.8966 - val_
accuracy: 0.6345 - val_loss: 1.0648
Epoch 29/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6992 - loss: 0.8789 - val_
accuracy: 0.6344 - val_loss: 1.0643
Epoch 30/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7063 - loss: 0.8623 - val_
accuracy: 0.6379 - val_loss: 1.0594
Epoch 31/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7125 - loss: 0.8456 - val_
accuracy: 0.6386 - val_loss: 1.0584
Epoch 32/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7201 - loss: 0.8286 - val_
accuracy: 0.6380 - val_loss: 1.0620
Epoch 33/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7255 - loss: 0.8125 - val_
accuracy: 0.6388 - val_loss: 1.0621
Total elapsed time :  0:00:55.120436
```

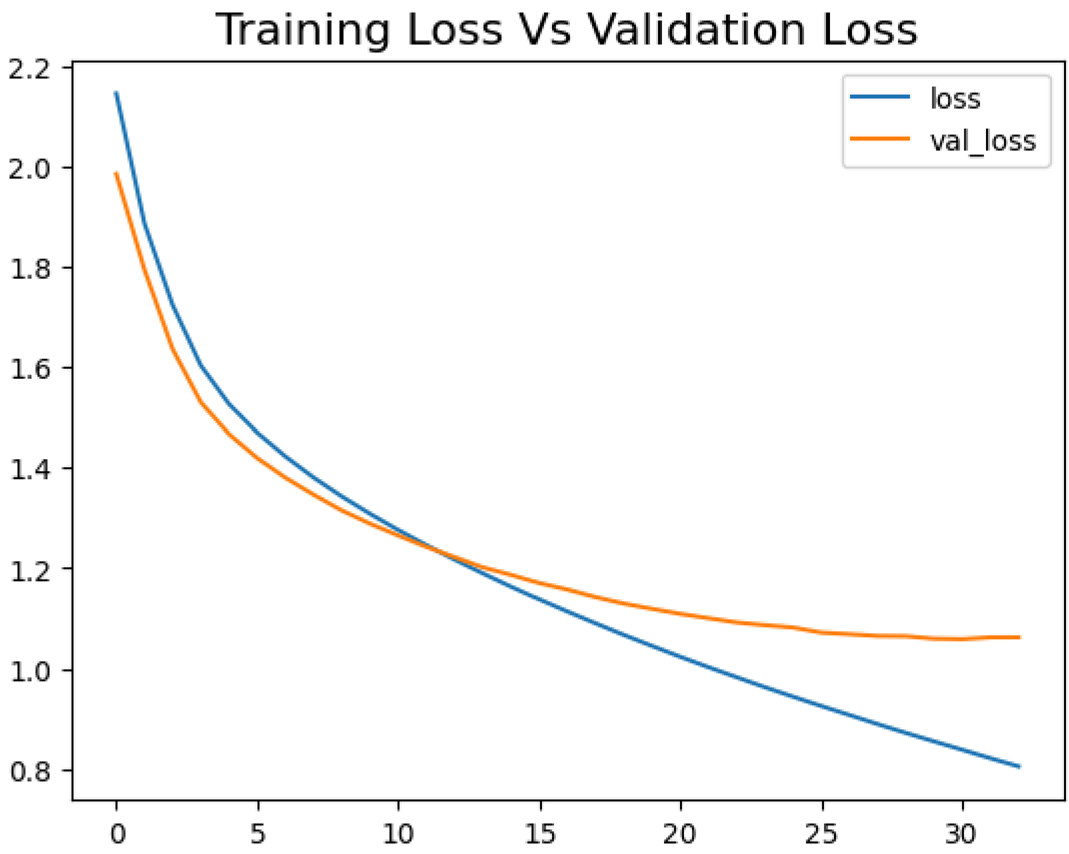
Checking model training metrics

In [ ]:
```
metrics_model_1 = pd.DataFrame(model_1.history.history)
metrics_model_1
```

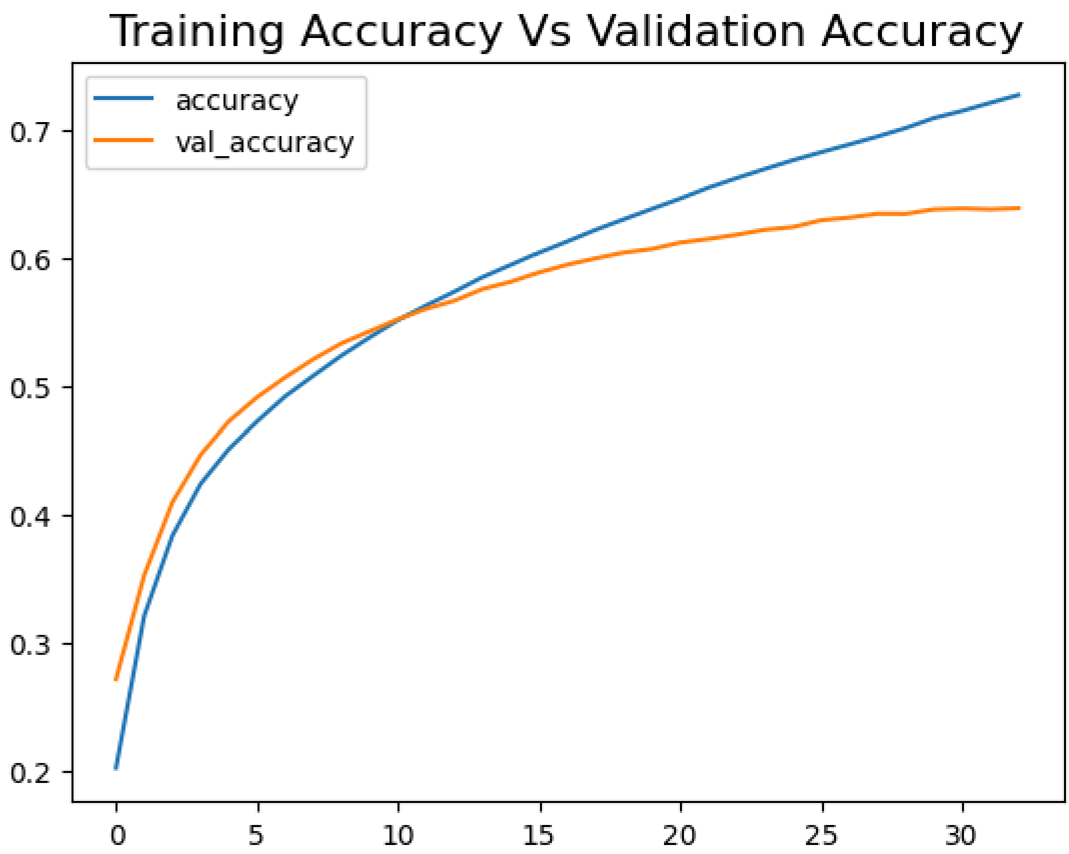
Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.202525	2.145624	0.2718	1.985183
1	0.321150	1.886687	0.3528	1.794304
2	0.383700	1.723804	0.4098	1.635853
3	0.424075	1.603219	0.4466	1.530032
4	0.451100	1.526580	0.4726	1.467087
5	0.472475	1.469995	0.4913	1.419449
6	0.492050	1.422721	0.5068	1.380473
7	0.508075	1.380870	0.5210	1.346616
8	0.523850	1.343125	0.5334	1.315192
9	0.538025	1.308555	0.5428	1.288851
10	0.551475	1.276259	0.5521	1.265348
11	0.562900	1.246051	0.5603	1.242676
12	0.573650	1.217247	0.5666	1.222026
13	0.585000	1.189788	0.5757	1.201673
14	0.594550	1.163250	0.5814	1.186793
15	0.604175	1.137740	0.5887	1.170313
16	0.612925	1.113307	0.5948	1.157204
17	0.621950	1.089764	0.5997	1.142146
18	0.630150	1.066738	0.6042	1.129317
19	0.638275	1.044757	0.6070	1.118892
20	0.646225	1.023162	0.6120	1.108823
21	0.654825	1.002437	0.6148	1.100231
22	0.662300	0.982396	0.6181	1.091750
23	0.669400	0.962601	0.6220	1.086390
24	0.676250	0.943637	0.6241	1.081617
25	0.682475	0.925079	0.6294	1.071511
26	0.688550	0.907107	0.6315	1.068337
27	0.694725	0.888971	0.6345	1.064798
28	0.701350	0.871431	0.6344	1.064288
29	0.709125	0.854522	0.6379	1.059424
30	0.714675	0.837828	0.6386	1.058448
31	0.721000	0.820874	0.6380	1.062004
32	0.727125	0.804672	0.6388	1.062088

In [ ]:
```
metrics_model_1[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_1[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Baseline model (training materials) - Adam optimizer

In [ ]:
```
model_2 = Sequential()

## ************* FIRST SET OF LAYERS *************************

# CONVOLUTIONAL LAYER
model_2.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activa
# POOLING LAYER
model_2.add(MaxPool2D(pool_size=(2, 2)))

## *************** SECOND SET OF LAYERS ***********************

# *************CONVOLUTIONAL LAYER
model_2.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activa
# POOLING LAYER
model_2.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 32 x 32 x 3 =3072 BEFORE FINAL LAYER
model_2.add(Flatten())

# 256 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model_2.add(Dense(256, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model_2.add(Dense(10, activation='softmax'))


model_2.compile(loss='sparse_categorical_crossentropy',
```

```
                optimizer='adam',
                metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [ ]:
```
model_2.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape
conv2d_2 (Conv2D)	(None, 29, 29, 32)
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 32)
conv2d_3 (Conv2D)	(None, 11, 11, 32)
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 32)
flatten_3 (Flatten)	(None, 800)
dense_8 (Dense)	(None, 256)
dense_9 (Dense)	(None, 10)

Total params: 225,610 (881.29 KB)

Trainable params: 225,610 (881.29 KB)

Non-trainable params: 0 (0.00 B)

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
start = datetime.now()
history_model_2 = model_2.fit(x_train,
                                 y_train,
                    epochs=50, # number of epochs
                    batch_size=64, # batch size
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )

# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.3419 - loss: 1.8018 - val_
accuracy: 0.5001 - val_loss: 1.3818
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5380 - loss: 1.3001 - val_
accuracy: 0.5818 - val_loss: 1.1832
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6005 - loss: 1.1355 - val_
accuracy: 0.6186 - val_loss: 1.0845
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6408 - loss: 1.0246 - val_
accuracy: 0.6315 - val_loss: 1.0650
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6726 - loss: 0.9325 - val_
accuracy: 0.6357 - val_loss: 1.0440
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.6995 - loss: 0.8598 - val_
accuracy: 0.6460 - val_loss: 1.0307
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7263 - loss: 0.7907 - val_
accuracy: 0.6564 - val_loss: 1.0179
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7500 - loss: 0.7265 - val_
accuracy: 0.6493 - val_loss: 1.0638
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7700 - loss: 0.6654 - val_
accuracy: 0.6372 - val_loss: 1.1539
Total elapsed time :  0:00:18.752049
```

Checking model training metrics

In [ ]:

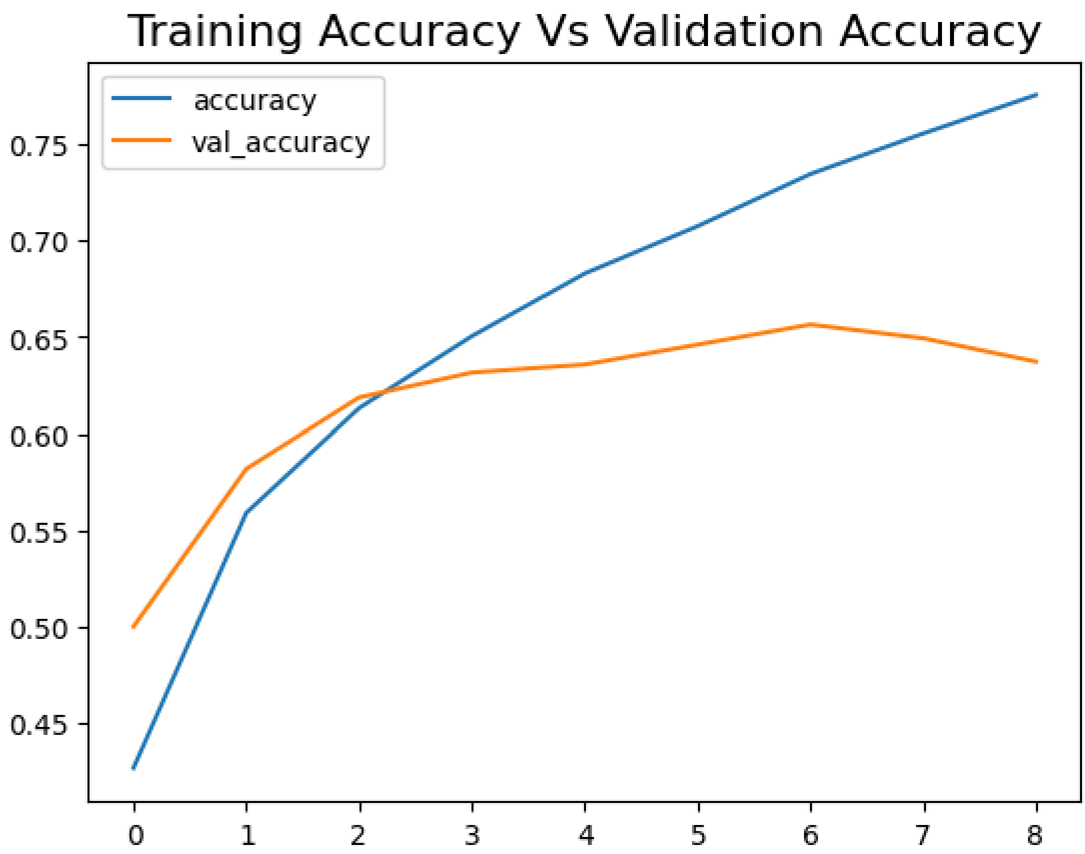
```
metrics_model_2 = pd.DataFrame(model_2.history.history)
metrics_model_2
```

Out[ ]:

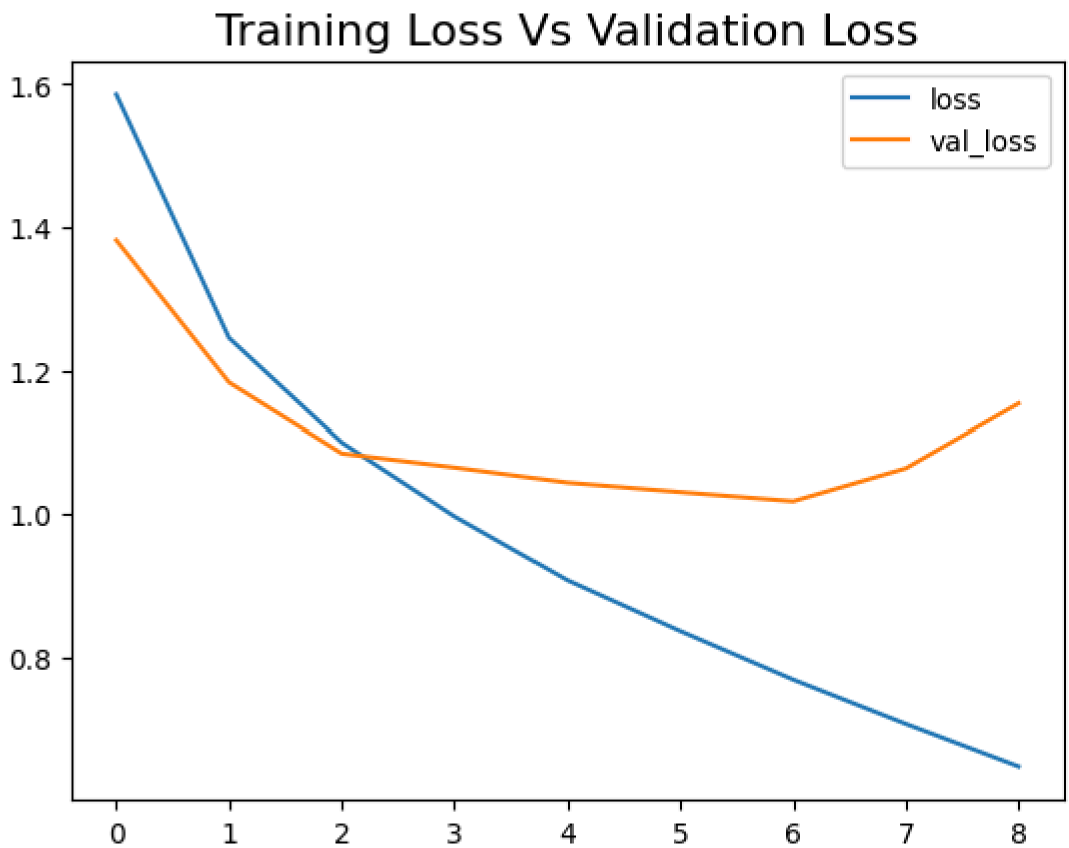
	accuracy	loss	val_accuracy	val_loss
0	0.426775	1.585744	0.5001	1.381804
1	0.559025	1.245304	0.5818	1.183189
2	0.613075	1.099781	0.6186	1.084457
3	0.650350	0.996652	0.6315	1.065022
4	0.682925	0.907494	0.6357	1.043971
5	0.707425	0.836388	0.6460	1.030675
6	0.734450	0.768598	0.6564	1.017902
7	0.755450	0.706603	0.6493	1.063826
8	0.775425	0.647048	0.6372	1.153905

In [ ]:

```
metrics_model_2[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_2[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



VGG-19

Load base model

In [ ]:
```
# loading VGG-19 model

vgg = VGG19( include_top = False,
            input_shape = [32,32,3],
            # weights='imagenet'
            )

vgg.summary()
```

Model: "vgg19"

Layer (type)	Output Shape
input_layer_1 (InputLayer)	(None, 32, 32, 3)
block1_conv1 (Conv2D)	(None, 32, 32, 64)
block1_conv2 (Conv2D)	(None, 32, 32, 64)
block1_pool (MaxPooling2D)	(None, 16, 16, 64)
block2_conv1 (Conv2D)	(None, 16, 16, 128)
block2_conv2 (Conv2D)	(None, 16, 16, 128)
block2_pool (MaxPooling2D)	(None, 8, 8, 128)
block3_conv1 (Conv2D)	(None, 8, 8, 256)
block3_conv2 (Conv2D)	(None, 8, 8, 256)
block3_conv3 (Conv2D)	(None, 8, 8, 256)
block3_conv4 (Conv2D)	(None, 8, 8, 256)
block3_pool (MaxPooling2D)	(None, 4, 4, 256)
block4_conv1 (Conv2D)	(None, 4, 4, 512)
block4_conv2 (Conv2D)	(None, 4, 4, 512)
block4_conv3 (Conv2D)	(None, 4, 4, 512)
block4_conv4 (Conv2D)	(None, 4, 4, 512)
block4_pool (MaxPooling2D)	(None, 2, 2, 512)
block5_conv1 (Conv2D)	(None, 2, 2, 512)
block5_conv2 (Conv2D)	(None, 2, 2, 512)
block5_conv3 (Conv2D)	(None, 2, 2, 512)
block5_conv4 (Conv2D)	(None, 2, 2, 512)
block5_pool (MaxPooling2D)	(None, 1, 1, 512)

Total params: 20,024,384 (76.39 MB)

Trainable params: 20,024,384 (76.39 MB)

Non-trainable params: 0 (0.00 B)

VGG-19 model - SGD optimizer

In [ ]:
```
del(model_3)
```

In [ ]:
```
model_3=Sequential(vgg.layers)

# flatting layer

model_3.add(Flatten())

# ANN layers

# dense layers
model_3.add(Dense(4096,  # number of neurons in dense layer
                activation='relu')) # activation function

model_3.add(Dense(4096,  # number of neurons in dense layer
                activation='relu')) # activation function

# final classifier
model_3.add(Dense(10,  # 10 classes
                activation='softmax')) # activation function

# model configuration (training parameters - loss, optimizer, target metric)
model_3.compile(loss='sparse_categorical_crossentropy', # sparse categorical cro
              optimizer='sgd', # optimizer function
              metrics=['accuracy'])
```

In [ ]:
```
model_3.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	
block1_conv1 (Conv2D)	(None, 32, 32, 64)	
block1_conv2 (Conv2D)	(None, 32, 32, 64)	
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	
block2_conv1 (Conv2D)	(None, 16, 16, 128)	
block2_conv2 (Conv2D)	(None, 16, 16, 128)	
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	
block3_conv1 (Conv2D)	(None, 8, 8, 256)	
block3_conv2 (Conv2D)	(None, 8, 8, 256)	
block3_conv3 (Conv2D)	(None, 8, 8, 256)	
block3_conv4 (Conv2D)	(None, 8, 8, 256)	
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	
block4_conv1 (Conv2D)	(None, 4, 4, 512)	
block4_conv2 (Conv2D)	(None, 4, 4, 512)	
block4_conv3 (Conv2D)	(None, 4, 4, 512)	
block4_conv4 (Conv2D)	(None, 4, 4, 512)	
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	
block5_conv1 (Conv2D)	(None, 2, 2, 512)	
block5_conv2 (Conv2D)	(None, 2, 2, 512)	
block5_conv3 (Conv2D)	(None, 2, 2, 512)	
block5_conv4 (Conv2D)	(None, 2, 2, 512)	
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	
flatten_1 (Flatten)	(None, 512)	
dense_3 (Dense)	(None, 4096)	
dense_4 (Dense)	(None, 4096)	
dense_5 (Dense)	(None, 10)	

Total params: 38,947,914 (148.57 MB)

Trainable params: 38,947,914 (148.57 MB)

Non-trainable params: 0 (0.00 B)

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
start = datetime.now()
history_model_3 = model_3.fit(x_train, y_train,
                    epochs=50, # number of epochs,
                    batch_size=64,
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )

# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.4064 - loss: 1.6823 - va
l_accuracy: 0.7246 - val_loss: 0.7924
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.7215 - loss: 0.8111 - val_
accuracy: 0.7636 - val_loss: 0.6890
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 9ms/step - accuracy: 0.7919 - loss: 0.6110 - val_
accuracy: 0.7929 - val_loss: 0.5987
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.8317 - loss: 0.4971 - val_
accuracy: 0.7846 - val_loss: 0.6447
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.8623 - loss: 0.4099 - val_
accuracy: 0.7533 - val_loss: 0.8358
Total elapsed time :  0:00:32.698796
```

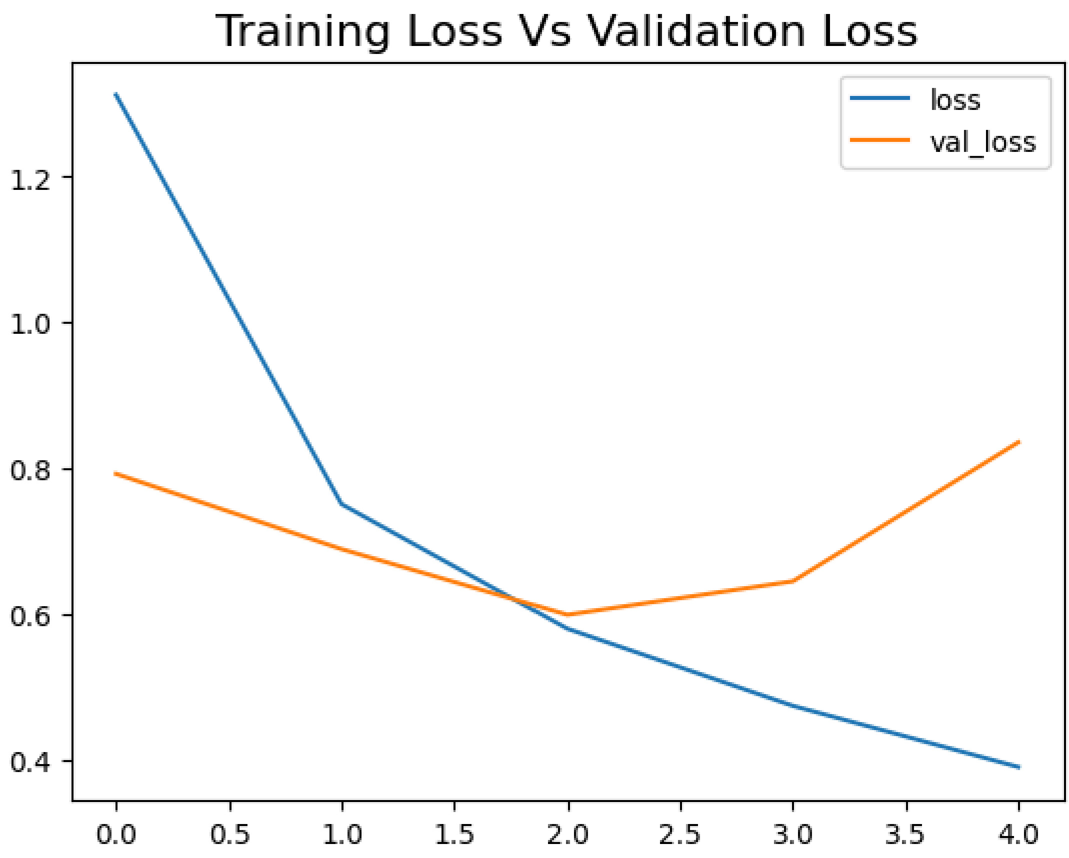
Checking model training metrics

In [ ]:
```
metrics_model_3 = pd.DataFrame(model_3.history.history)
metrics_model_3
```

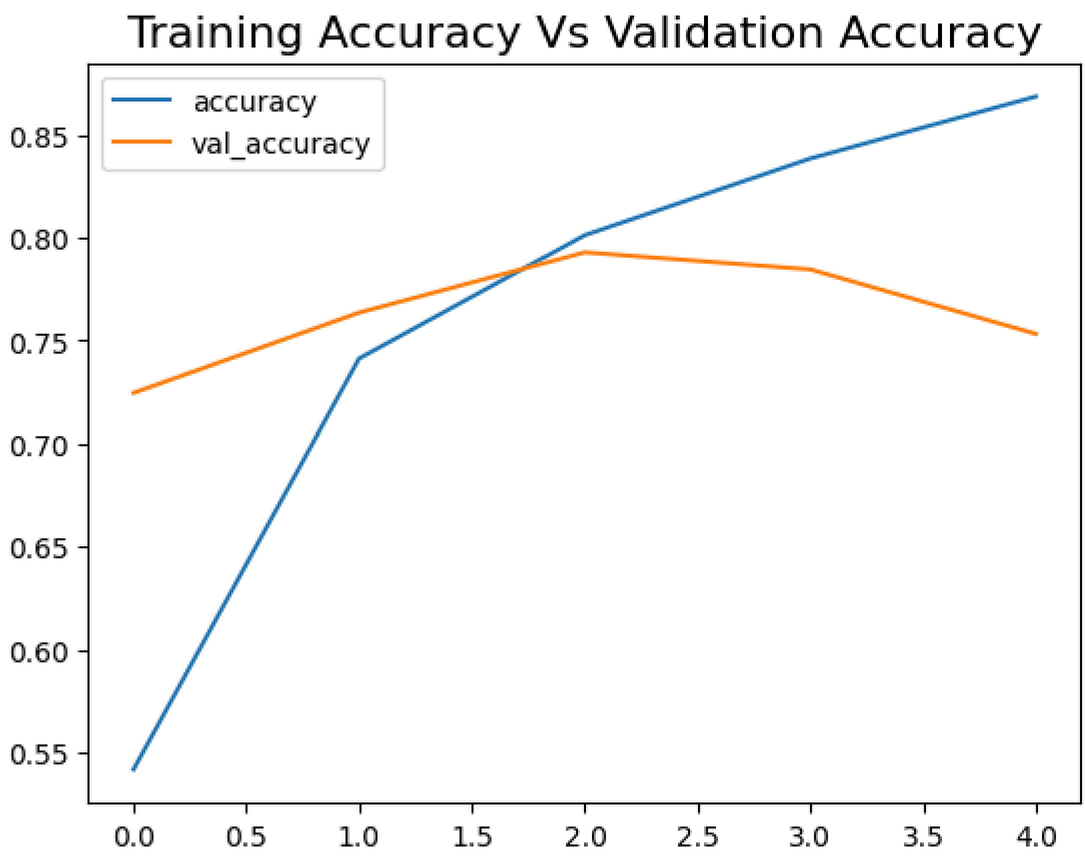
Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.541825	1.312372	0.7246	0.792411
1	0.741275	0.750850	0.7636	0.689007
2	0.801250	0.579588	0.7929	0.598740
3	0.838525	0.473599	0.7846	0.644745
4	0.868650	0.389554	0.7533	0.835764

In [ ]:
```
metrics_model_3[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_3[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



VGG-19 model - Adam optimizer

In [ ]:
```
del(model_4)
```

In [ ]:
```
vgg = VGG19( include_top = False,
            input_shape = [32,32,3],
            # weights='imagenet'
            )

model_4=Sequential(vgg.layers)

# flatting layer

model_4.add(Flatten())

# ANN layers

# dense layers
model_4.add(Dense(4096,  # number of neurons in dense layer
                activation='relu')) # activation function

model_4.add(Dense(4096,  # number of neurons in dense layer
                activation='relu')) # activation function

# final classifier
model_4.add(Dense(10,  # 10 classes
                activation='softmax')) # activation function

# model configuration (training parameters - loss, optimizer, target metric)
model_4.compile(loss='sparse_categorical_crossentropy', # sparse categorical cro
              optimizer='adam', # optimizer function
              metrics=['accuracy'])
```

In [ ]:
```
model_4.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape
block1_conv1 (Conv2D)	(None, 32, 32, 64)
block1_conv2 (Conv2D)	(None, 32, 32, 64)
block1_pool (MaxPooling2D)	(None, 16, 16, 64)
block2_conv1 (Conv2D)	(None, 16, 16, 128)
block2_conv2 (Conv2D)	(None, 16, 16, 128)
block2_pool (MaxPooling2D)	(None, 8, 8, 128)
block3_conv1 (Conv2D)	(None, 8, 8, 256)
block3_conv2 (Conv2D)	(None, 8, 8, 256)
block3_conv3 (Conv2D)	(None, 8, 8, 256)
block3_conv4 (Conv2D)	(None, 8, 8, 256)
block3_pool (MaxPooling2D)	(None, 4, 4, 256)
block4_conv1 (Conv2D)	(None, 4, 4, 512)
block4_conv2 (Conv2D)	(None, 4, 4, 512)
block4_conv3 (Conv2D)	(None, 4, 4, 512)
block4_conv4 (Conv2D)	(None, 4, 4, 512)
block4_pool (MaxPooling2D)	(None, 2, 2, 512)
block5_conv1 (Conv2D)	(None, 2, 2, 512)
block5_conv2 (Conv2D)	(None, 2, 2, 512)
block5_conv3 (Conv2D)	(None, 2, 2, 512)
block5_conv4 (Conv2D)	(None, 2, 2, 512)
block5_pool (MaxPooling2D)	(None, 1, 1, 512)
flatten_6 (Flatten)	(None, 512)
dense_16 (Dense)	(None, 4096)
dense_17 (Dense)	(None, 4096)
dense_18 (Dense)	(None, 10)

Total params: 38,947,914 (148.57 MB)

Trainable params: 38,947,914 (148.57 MB)

Non-trainable params: 0 (0.00 B)

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
start = datetime.now()
history_model_4 = model_4.fit(x_train, y_train,
                    epochs=50, # number of epochs,
                    batch_size=64,
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )

# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - accuracy: 0.1010 - loss: 2.4378 - va
l_accuracy: 0.0999 - val_loss: 2.3027
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.0962 - loss: 2.3030 - val
_accuracy: 0.0999 - val_loss: 2.3027
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.0958 - loss: 2.3028 - val
_accuracy: 0.0999 - val_loss: 2.3027
Total elapsed time :  0:00:30.802084
```

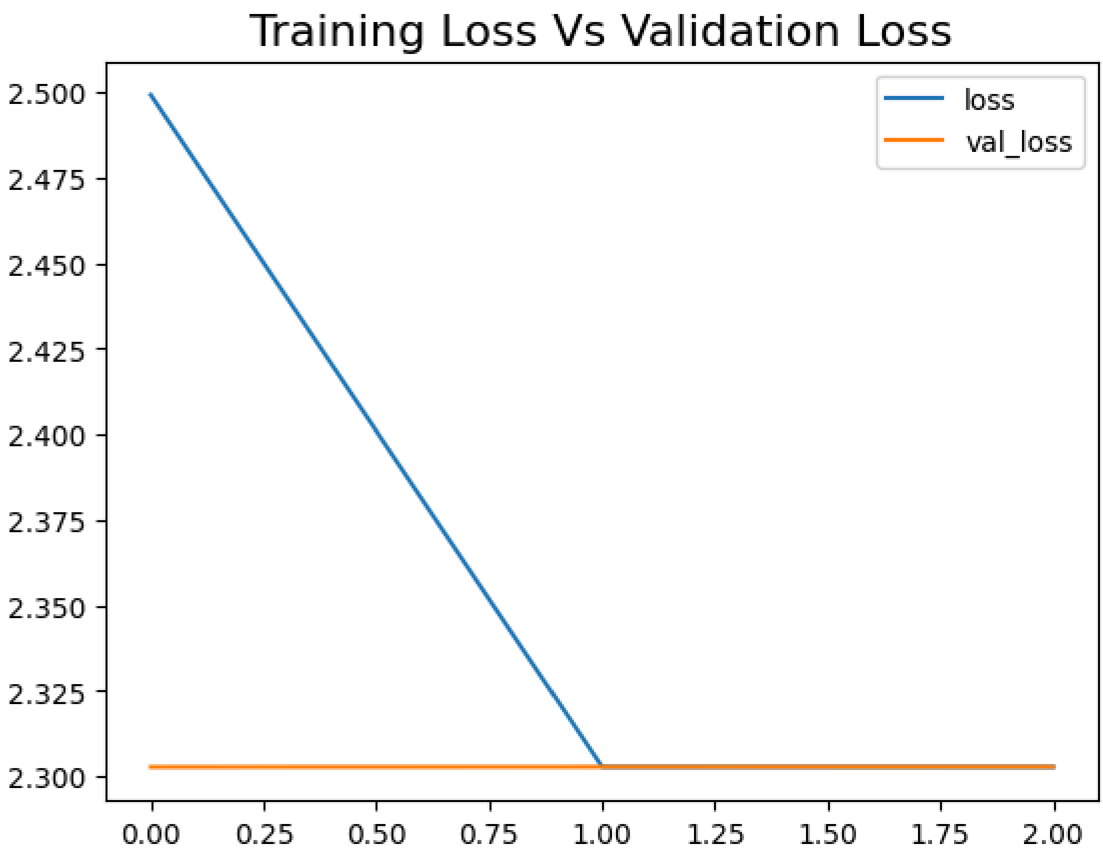
Checking model training metrics

In [ ]:
```
metrics_model_4 = pd.DataFrame(model_4.history.history)
metrics_model_4
```

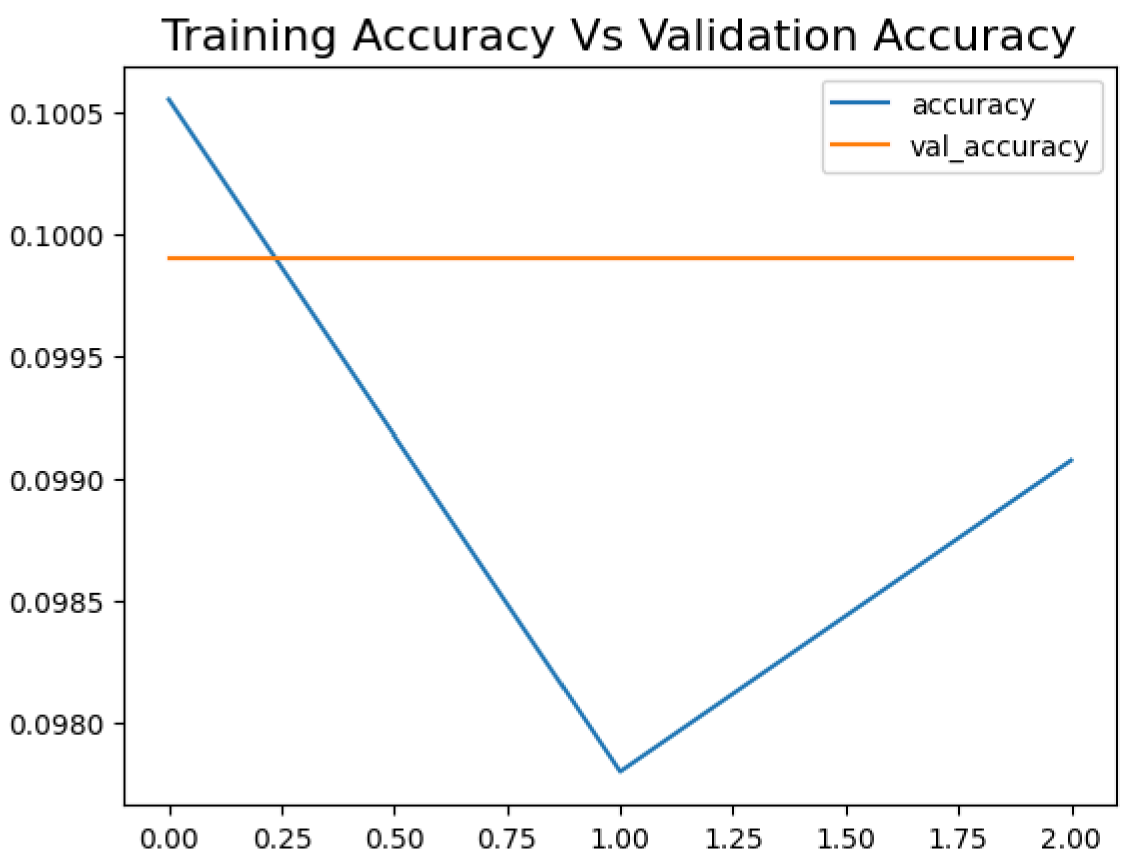
Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.100550	2.499209	0.0999	2.302709
1	0.097800	2.302745	0.0999	2.302745
2	0.099075	2.302723	0.0999	2.302764

In [ ]:
```
metrics_model_4[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_4[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



VGG-19 model - Adam optimizer (changed learning rate)

In [ ]:
```
# Freeze all VGG layers
# for layer in vgg.layers:
#    layer.trainable = False

model_5 = Sequential(vgg.layers)


# Flatten and add  dense layers

model_5.add(Flatten())
model_5.add(Dense(4096,  activation='relu'))
model_5.add(Dense(4096,  activation='relu'))
model_5.add(Dense(10,  activation='softmax'))




# Flatten and add custom dense layers
# model_5.add(Flatten())
# model_5.add(Dense(1024, activation='relu'))
# model_5.add(Dropout(0.5))
# model_5.add(Dense(512, activation='relu'))
# model_5.add(Dropout(0.5))
# model_5.add(Dense(10, activation='softmax'))

# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_5.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_5 = model_5.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - accuracy: 0.9002 - loss: 0.3342 - va
l_accuracy: 0.8285 - val_loss: 0.5695
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.9164 - loss: 0.2557 - val
_accuracy: 0.8287 - val_loss: 0.5998
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9390 - loss: 0.1853 - val
_accuracy: 0.8192 - val_loss: 0.6643
Total elapsed time: 0:00:28.303739
```

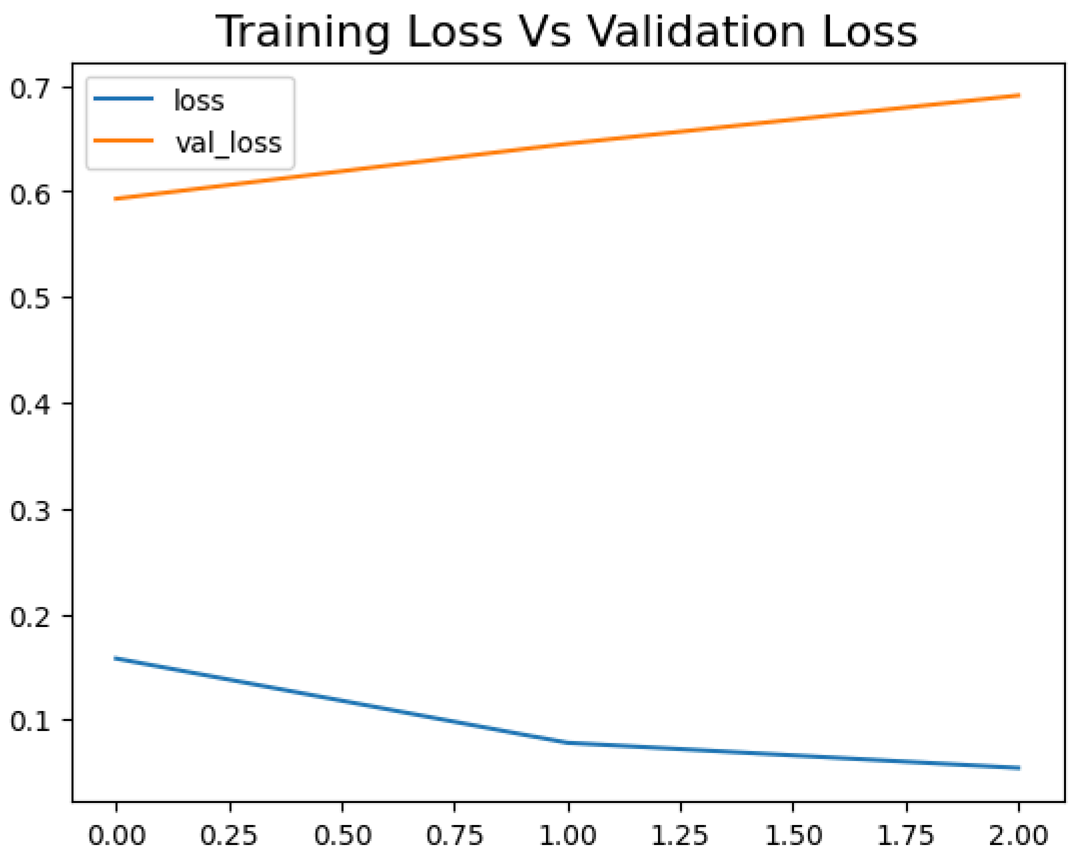
Checking model training metrics

In [ ]:
```
metrics_model_5 = pd.DataFrame(model_5.history.history)
metrics_model_5
```

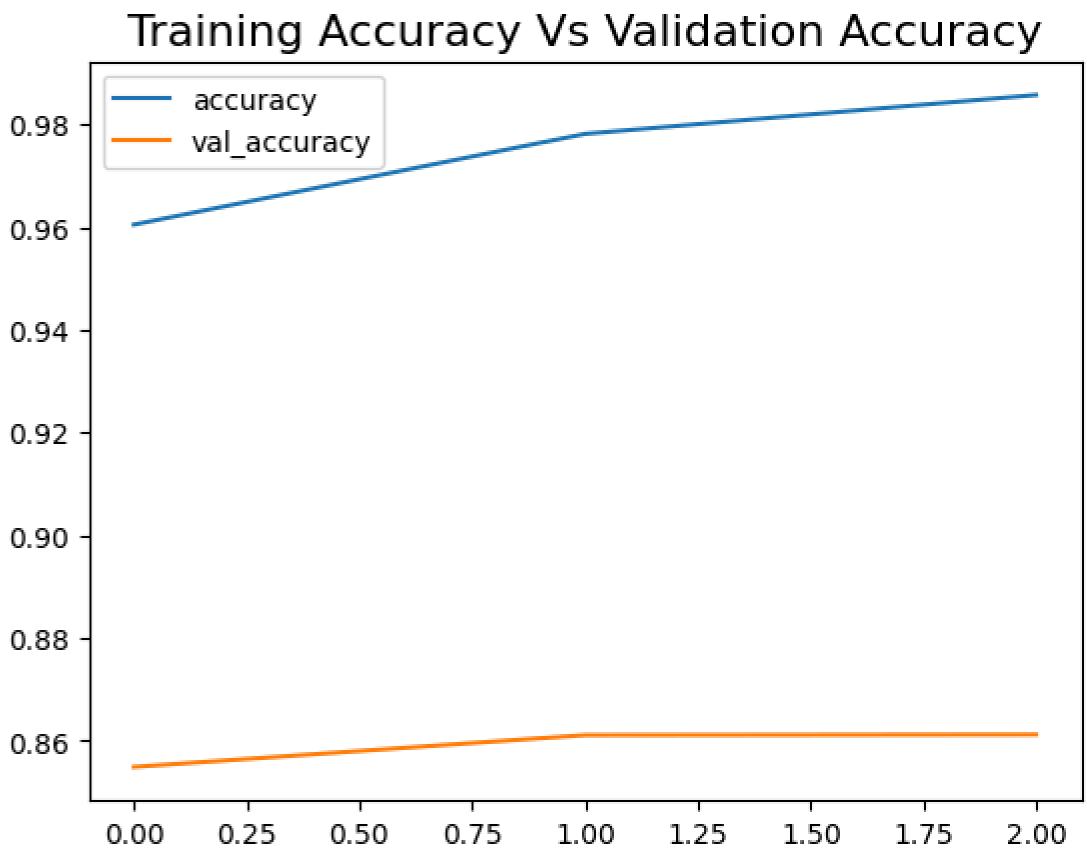
Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.908300	0.291528	0.8285	0.569452
1	0.918575	0.249655	0.8287	0.599762
2	0.937650	0.188845	0.8192	0.664257

In [ ]:
```
metrics_model_5[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_5[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



In [ ]:

VGG-19 model - Adam optimizer (batch normalisation)

In [ ]:
```
# Freeze all VGG Layers
# for layer in vgg.layers:
#    layer.trainable = False

model_6 = Sequential(vgg.layers)


# Flatten and add dense layers + batch normalisation

model_6.add(Flatten())
model_6.add(Dense(4096,  activation='relu'))
model_6.add(BatchNormalization())
model_6.add(Dense(4096,  activation='relu'))
model_6.add(BatchNormalization())
model_6.add(Dense(10,  activation='softmax'))




# Flatten and add custom dense layers
# model_5.add(Flatten())
# model_5.add(Dense(1024, activation='relu'))
# model_5.add(Dropout(0.5))
# model_5.add(Dense(512, activation='relu'))
# model_5.add(Dropout(0.5))
# model_5.add(Dense(10, activation='softmax'))
```

```
# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_6.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_6 = model_6.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 17s 13ms/step - accuracy: 0.9264 - loss: 0.3089 - va
l_accuracy: 0.8020 - val_loss: 0.8578
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9549 - loss: 0.1476 - val
_accuracy: 0.7964 - val_loss: 0.9903
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9611 - loss: 0.1239 - val
_accuracy: 0.8184 - val_loss: 0.8317
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9676 - loss: 0.1032 - val
_accuracy: 0.8169 - val_loss: 0.8983
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9699 - loss: 0.0959 - val
_accuracy: 0.7890 - val_loss: 1.0675
Total elapsed time: 0:00:44.478848
```

Checking model training metrics

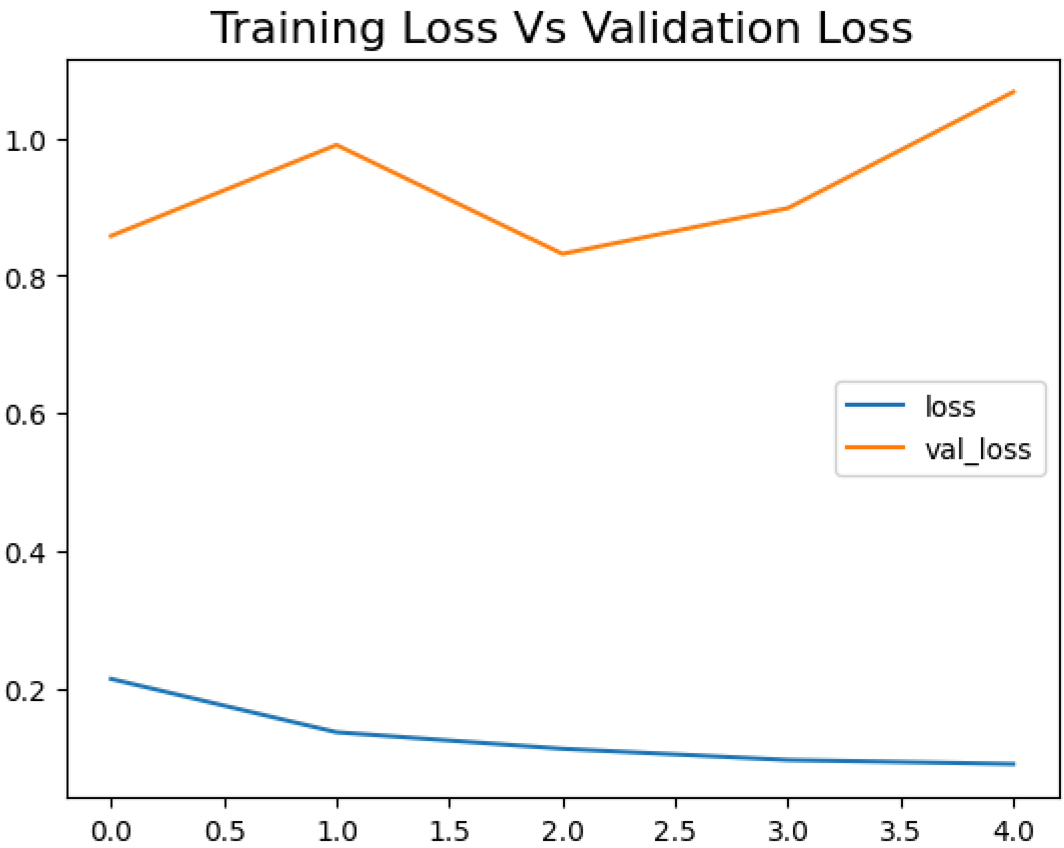
In [ ]:
```
metrics_model_6 = pd.DataFrame(model_6.history.history)
metrics_model_6
```

Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.943725	0.213584	0.8020	0.857794
1	0.957875	0.135923	0.7964	0.990321
2	0.963625	0.111848	0.8184	0.831689
3	0.970025	0.095517	0.8169	0.898287
4	0.971050	0.089650	0.7890	1.067495

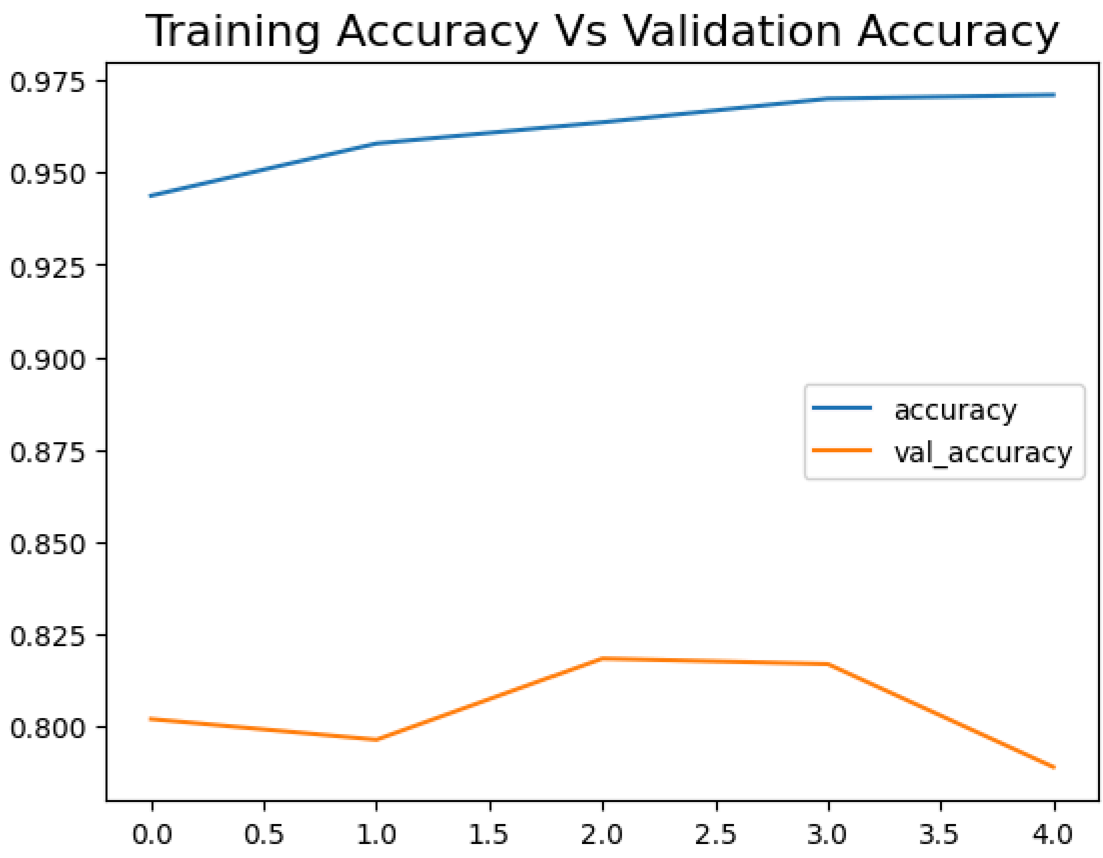
In [ ]:

```
metrics_model_6[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:

```
metrics_model_6[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



VGG-19 model - Adam optimizer (dropout)

In [ ]:
```
# Freeze all VGG layers
# for layer in vgg.layers:
#    layer.trainable = False

model_7 = Sequential(vgg.layers)


# Flatten and add dense layers + batch normalisation + dropout

model_7.add(Flatten())
model_7.add(Dense(4096,  activation='relu'))
model_7.add(BatchNormalization())
model_7.add(Dropout(0.5)) # discard 50% of neurons
model_7.add(Dense(4096,  activation='relu'))
model_7.add(BatchNormalization())
model_7.add(Dropout(0.5)) # discard 50% of neurons
model_7.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_7.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_7 = model_7.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 13ms/step - accuracy: 0.9459 - loss: 0.2071 - va
l_accuracy: 0.8124 - val_loss: 0.9550
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9662 - loss: 0.1106 - val
_accuracy: 0.8249 - val_loss: 0.9273
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9739 - loss: 0.0899 - val
_accuracy: 0.8272 - val_loss: 0.8628
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9778 - loss: 0.0740 - val
_accuracy: 0.8369 - val_loss: 0.8488
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9801 - loss: 0.0671 - val
_accuracy: 0.8247 - val_loss: 0.8369
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9771 - loss: 0.0765 - val
_accuracy: 0.8254 - val_loss: 0.8666
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9815 - loss: 0.0590 - val
_accuracy: 0.8398 - val_loss: 0.7325
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9852 - loss: 0.0495 - val
_accuracy: 0.8244 - val_loss: 0.8506
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9811 - loss: 0.0654 - val
_accuracy: 0.8184 - val_loss: 0.9521
Total elapsed time: 0:01:11.329533
```

Checking model training metrics

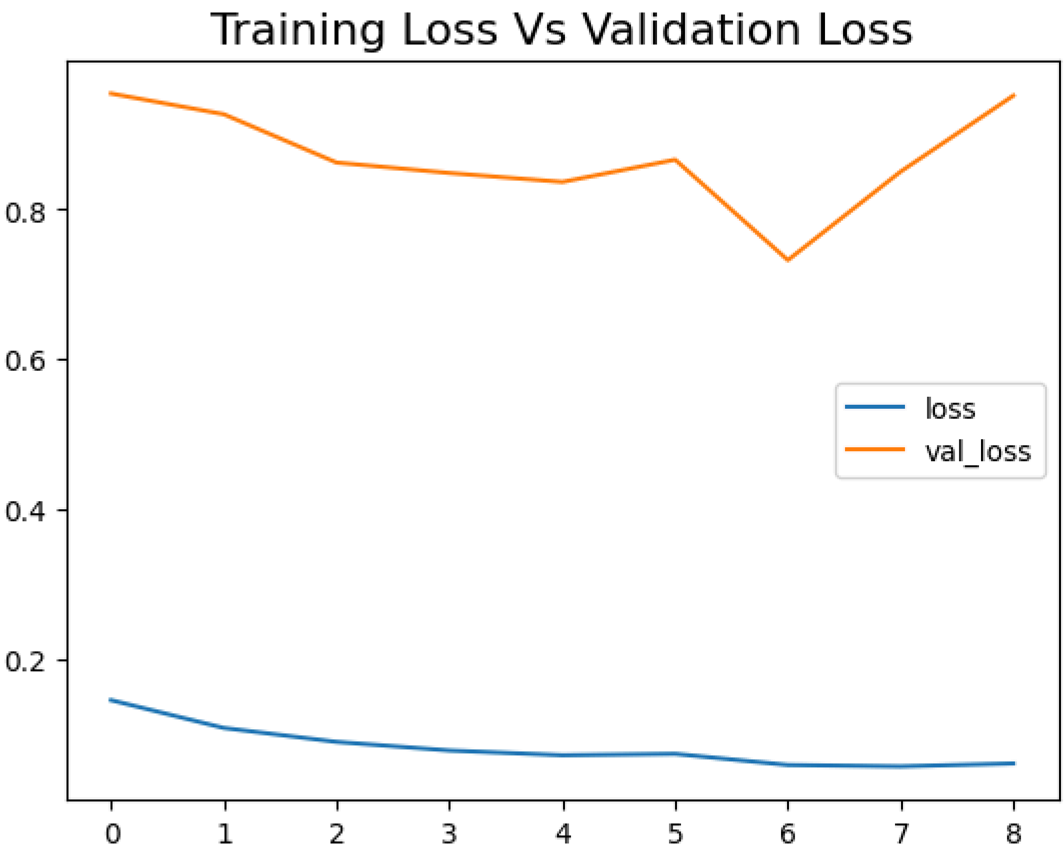
In [ ]:
```
metrics_model_7 = pd.DataFrame(model_7.history.history)
metrics_model_7
```

Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.961050	0.145476	0.8124	0.954991
1	0.967700	0.108327	0.8249	0.927304
2	0.973025	0.089731	0.8272	0.862764
3	0.977275	0.078255	0.8369	0.848847
4	0.978400	0.071917	0.8247	0.836879
5	0.977500	0.073728	0.8254	0.866585
6	0.982100	0.058861	0.8398	0.732463
7	0.983175	0.056988	0.8244	0.850643
8	0.981950	0.060887	0.8184	0.952146

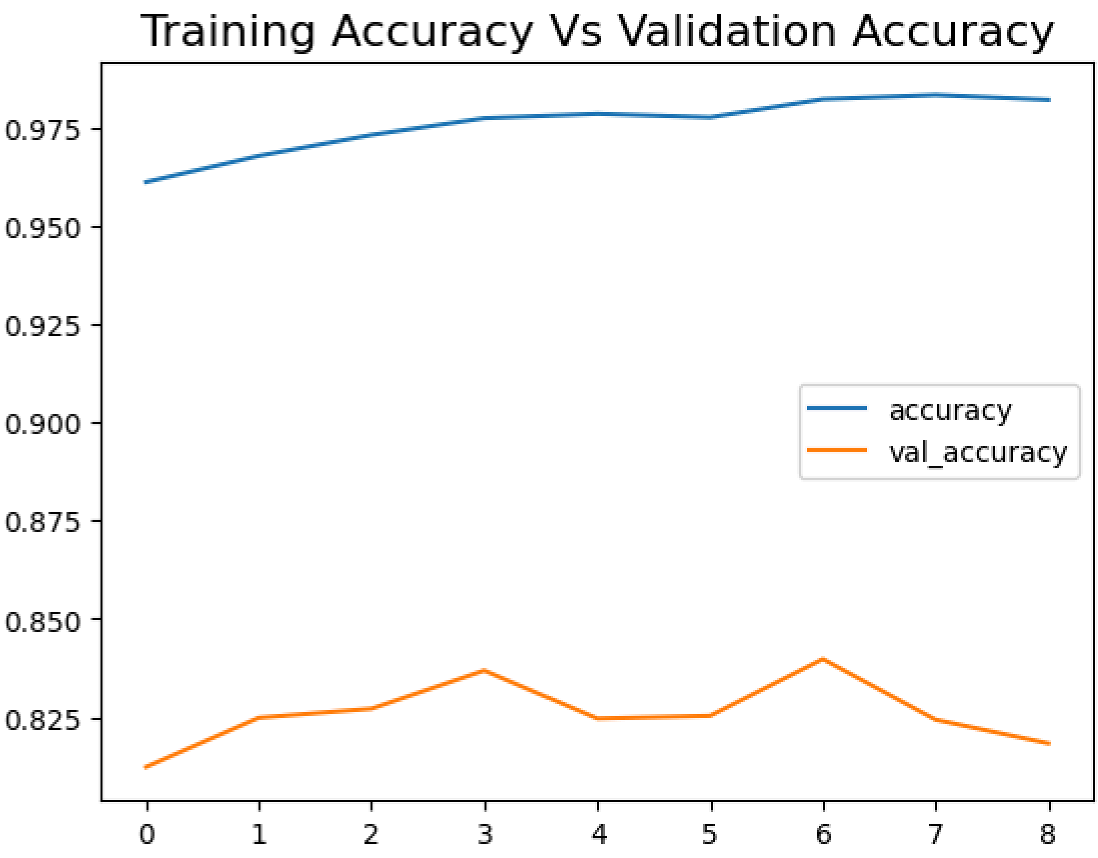
In [ ]:

```
metrics_model_7[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:

```
metrics_model_7[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



VGG-19 model - Adam optimizer (trim dense layers, remove dropout)

In [ ]:
```
del(model_8)
```

In [ ]:
```
# Freeze all VGG Layers
# for layer in vgg.layers:
#    layer.trainable = False

model_8 = Sequential(vgg.layers)


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_8.add(Flatten())
model_8.add(Dense(1024,  activation='relu'))
model_8.add(BatchNormalization())
# model_8.add(Dropout(0.5)) # discard 50% of neurons
model_8.add(Dense(512,  activation='relu'))
model_8.add(BatchNormalization())
# model_8.add(Dropout(0.5)) # discard 50% of neurons
model_8.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_8.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
```

```
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_8 = model_8.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 12ms/step - accuracy: 0.9660 - loss: 0.1127 - va
l_accuracy: 0.8374 - val_loss: 0.8496
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9875 - loss: 0.0431 - val
_accuracy: 0.8155 - val_loss: 0.9429
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9857 - loss: 0.0477 - val
_accuracy: 0.8409 - val_loss: 0.7568
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9889 - loss: 0.0338 - val
_accuracy: 0.8370 - val_loss: 0.8287
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9904 - loss: 0.0328 - val
_accuracy: 0.8357 - val_loss: 0.8737
Total elapsed time: 0:00:41.504467
```

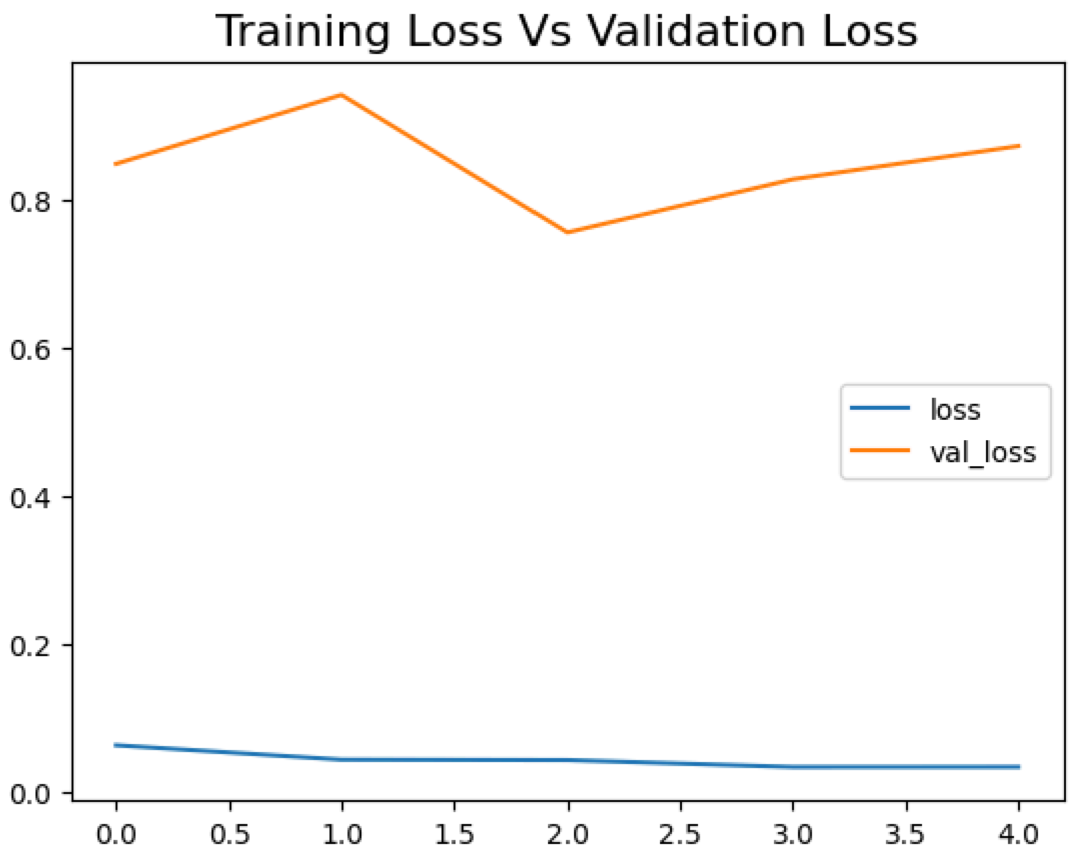
Checking model training metrics

In [ ]:
```
metrics_model_8 = pd.DataFrame(model_8.history.history)
metrics_model_8
```

Out[ ]:

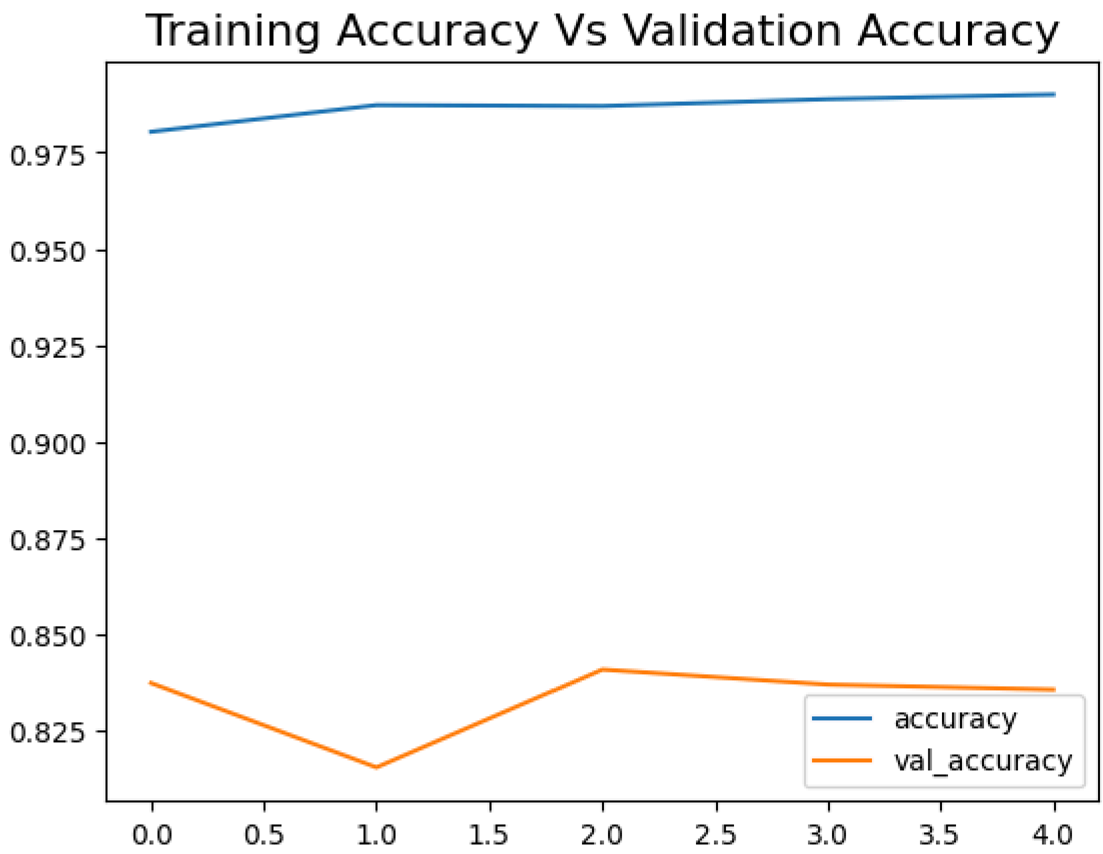
	accuracy	loss	val_accuracy	val_loss
0	0.980375	0.063460	0.8374	0.849573
1	0.987300	0.043969	0.8155	0.942917
2	0.987050	0.043412	0.8409	0.756775
3	0.988850	0.034007	0.8370	0.828700
4	0.990050	0.034072	0.8357	0.873694

In [ ]:
```
metrics_model_8[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:

```
metrics_model_8[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



VGG-19 model - Adam optimizer (trim dense layers, with dropout)

In [ ]:
```
model_9 = Sequential(vgg.layers)


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_9.add(Flatten())
model_9.add(Dense(1024,  activation='relu'))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5)) # discard 50% of neurons
model_9.add(Dense(512,  activation='relu'))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5)) # discard 50% of neurons
model_9.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_9.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_9 = model_9.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 12ms/step - accuracy: 0.9317 - loss: 0.2353 - va
l_accuracy: 0.8434 - val_loss: 0.8764
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9862 - loss: 0.0476 - val
_accuracy: 0.8437 - val_loss: 0.8902
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9890 - loss: 0.0428 - val
_accuracy: 0.8476 - val_loss: 0.8141
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9886 - loss: 0.0398 - val
_accuracy: 0.8402 - val_loss: 0.8865
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9868 - loss: 0.0456 - val
_accuracy: 0.8438 - val_loss: 0.8966
Total elapsed time: 0:00:41.797212
```

Checking model training metrics

In [ ]:

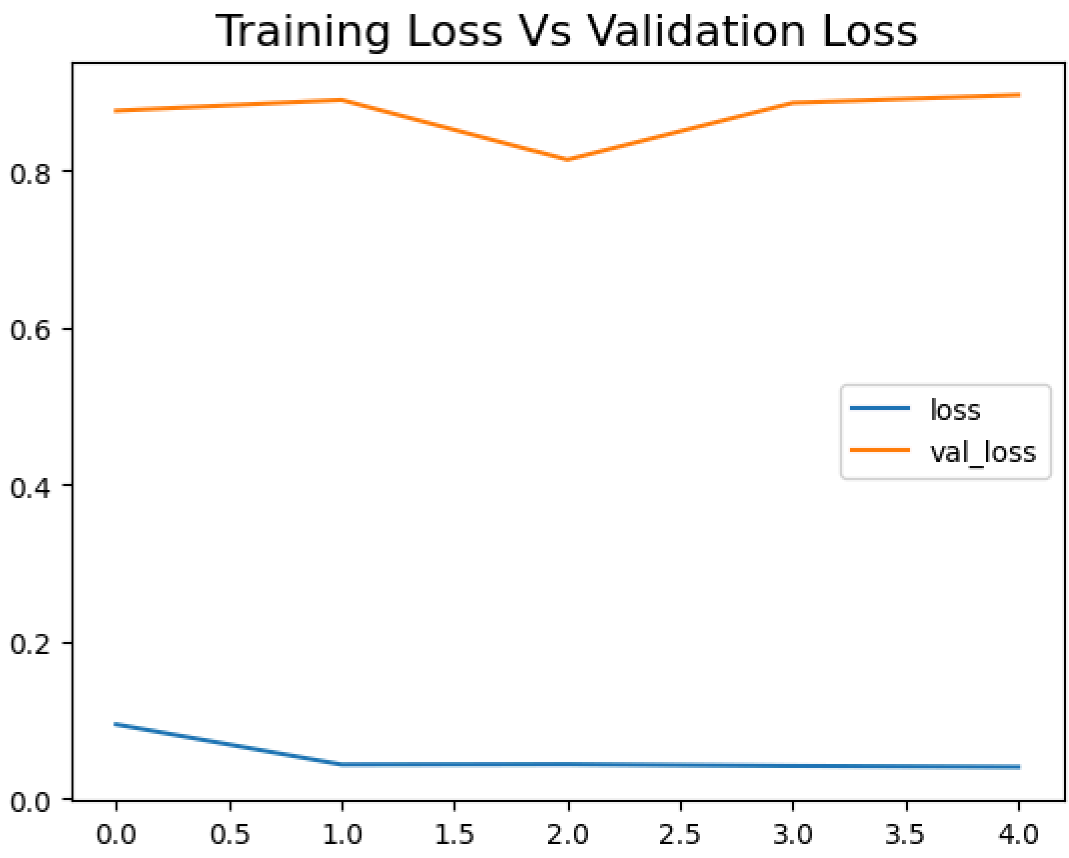
```
metrics_model_9 = pd.DataFrame(model_9.history.history)
metrics_model_9
```

Out[ ]:

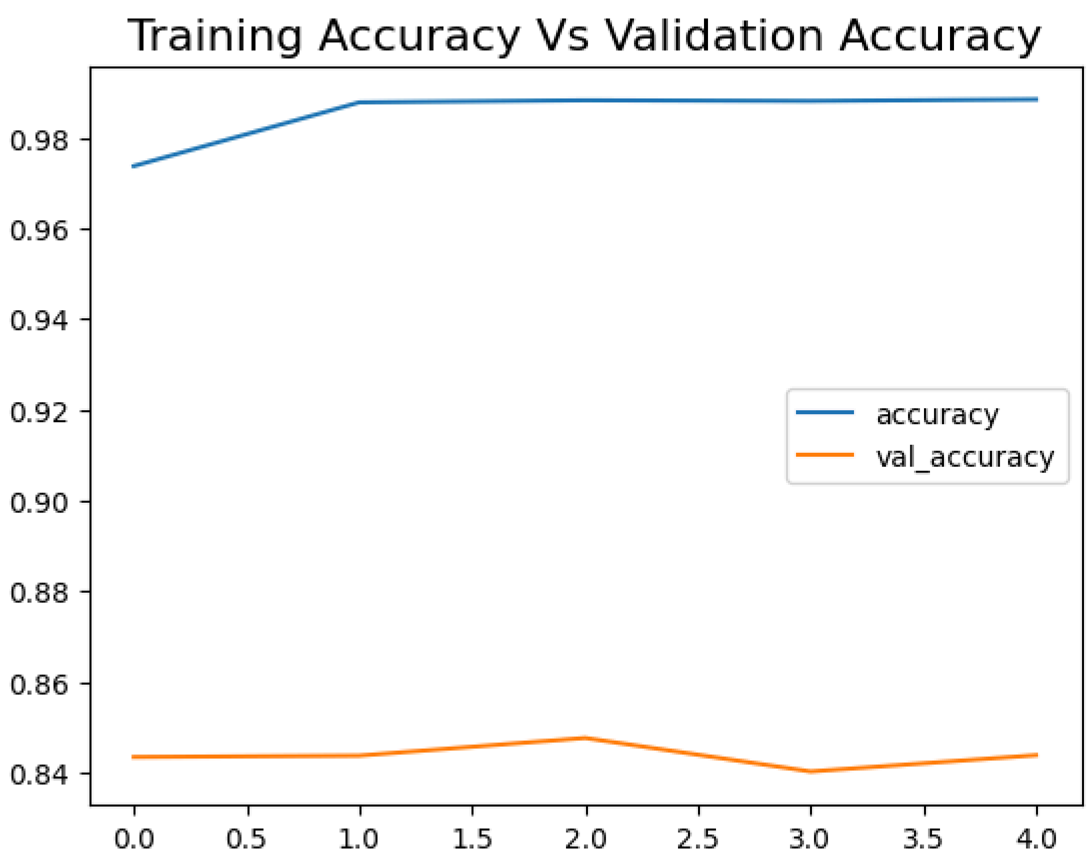
	accuracy	loss	val_accuracy	val_loss
0	0.973750	0.094775	0.8434	0.876441
1	0.987825	0.043589	0.8437	0.890243
2	0.988275	0.043924	0.8476	0.814130
3	0.988125	0.041827	0.8402	0.886450
4	0.988475	0.040523	0.8438	0.896566

In [ ]:

```
metrics_model_9[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_9[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Transfer learning (pre-trained model)

Reload VGG model with weights

In [ ]:
```
vgg_pre_trained = VGG19( include_top = False,
            input_shape = [32,32,3],
            weights='imagenet'
            )
```

In [ ]:
```
# Freeze all VGG layers
for layer in vgg_pre_trained.layers:
   layer.trainable = False

model_10 = Sequential(vgg_pre_trained.layers)


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_10.add(Flatten())
model_10.add(Dense(1024,  activation='relu'))
model_10.add(BatchNormalization())
model_10.add(Dropout(0.5)) # discard 50% of neurons
model_10.add(Dense(512,  activation='relu'))
model_10.add(BatchNormalization())
model_10.add(Dropout(0.5)) # discard 50% of neurons
model_10.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_10.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

In [ ]:
```
model_10.summary()
```

Model: "sequential_15"

Layer (type)	Output Shape	
block1_conv1 (Conv2D)	(None, 32, 32, 64)	
block1_conv2 (Conv2D)	(None, 32, 32, 64)	
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	
block2_conv1 (Conv2D)	(None, 16, 16, 128)	
block2_conv2 (Conv2D)	(None, 16, 16, 128)	
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	
block3_conv1 (Conv2D)	(None, 8, 8, 256)	
block3_conv2 (Conv2D)	(None, 8, 8, 256)	
block3_conv3 (Conv2D)	(None, 8, 8, 256)	
block3_conv4 (Conv2D)	(None, 8, 8, 256)	
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	
block4_conv1 (Conv2D)	(None, 4, 4, 512)	
block4_conv2 (Conv2D)	(None, 4, 4, 512)	
block4_conv3 (Conv2D)	(None, 4, 4, 512)	
block4_conv4 (Conv2D)	(None, 4, 4, 512)	
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	
block5_conv1 (Conv2D)	(None, 2, 2, 512)	
block5_conv2 (Conv2D)	(None, 2, 2, 512)	
block5_conv3 (Conv2D)	(None, 2, 2, 512)	
block5_conv4 (Conv2D)	(None, 2, 2, 512)	
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	
flatten_14 (Flatten)	(None, 512)	
dense_40 (Dense)	(None, 1024)	
batch_normalization_18 (BatchNormalization)	(None, 1024)	
dropout_14 (Dropout)	(None, 1024)	
dense_41 (Dense)	(None, 512)	
batch_normalization_19 (BatchNormalization)	(None, 512)	
dropout_15 (Dropout)	(None, 512)	

```
│ dense_42 (Dense)                │ (None, 10)               │
```

```
Total params: 21,085,770 (80.44 MB)
Trainable params: 1,058,314 (4.04 MB)
Non-trainable params: 20,027,456 (76.40 MB)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
start = datetime.now()
history_model_10 = model_10.fit(
    x_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

duration = datetime.now() - start
print('Total elapsed time:', duration)
buzzer()
```

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.2761 - loss: 2.6108 - val_accuracy: 0.5196 - val_loss: 1.4120
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4174 - loss: 1.9125 - val_accuracy: 0.5527 - val_loss: 1.3182
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4467 - loss: 1.7168 - val_accuracy: 0.5602 - val_loss: 1.2669
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4718 - loss: 1.6022 - val_accuracy: 0.5719 - val_loss: 1.2314
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4943 - loss: 1.4954 - val_accuracy: 0.5773 - val_loss: 1.2100
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5091 - loss: 1.4395 - val_accuracy: 0.5825 - val_loss: 1.1943
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5241 - loss: 1.3867 - val_accuracy: 0.5906 - val_loss: 1.1792
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5306 - loss: 1.3384 - val_accuracy: 0.5939 - val_loss: 1.1658
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5449 - loss: 1.3027 - val_accuracy: 0.5943 - val_loss: 1.1585
Epoch 10/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5580 - loss: 1.2682 - val_accuracy: 0.5999 - val_loss: 1.1520
Epoch 11/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5630 - loss: 1.2408 - val_accuracy: 0.6036 - val_loss: 1.1392
Epoch 12/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5700 - loss: 1.2184 - val_accuracy: 0.6035 - val_loss: 1.1355
Epoch 13/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5778 - loss: 1.1959 - val_accuracy: 0.6055 - val_loss: 1.1297
Epoch 14/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5835 - loss: 1.1760 - val_accuracy: 0.6065 - val_loss: 1.1262
Epoch 15/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5890 - loss: 1.1623 - val_accuracy: 0.6079 - val_loss: 1.1178
Epoch 16/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5983 - loss: 1.1423 - val_accuracy: 0.6130 - val_loss: 1.1132
Epoch 17/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5983 - loss: 1.1297 - val_accuracy: 0.6135 - val_loss: 1.1077
Epoch 18/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6087 - loss: 1.1126 - val_accuracy: 0.6118 - val_loss: 1.1092
Epoch 19/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6115 - loss: 1.0981 - val_accuracy: 0.6147 - val_loss: 1.1066
Epoch 20/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6140 - loss: 1.0955 - val_accuracy: 0.6171 - val_loss: 1.1040
```

```
Epoch 21/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6216 - loss: 1.0711 - val_
accuracy: 0.6150 - val_loss: 1.1039
Epoch 22/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6248 - loss: 1.0644 - val_
accuracy: 0.6178 - val_loss: 1.0954
Epoch 23/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6296 - loss: 1.0463 - val_
accuracy: 0.6181 - val_loss: 1.0998
Epoch 24/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6351 - loss: 1.0360 - val_
accuracy: 0.6188 - val_loss: 1.0928
Epoch 25/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6357 - loss: 1.0233 - val_
accuracy: 0.6228 - val_loss: 1.0877
Epoch 26/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6427 - loss: 1.0128 - val_
accuracy: 0.6228 - val_loss: 1.0904
Epoch 27/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6478 - loss: 0.9974 - val_
accuracy: 0.6214 - val_loss: 1.0886
Total elapsed time: 0:01:17.746000
```

Checking model training metrics

In [ ]:

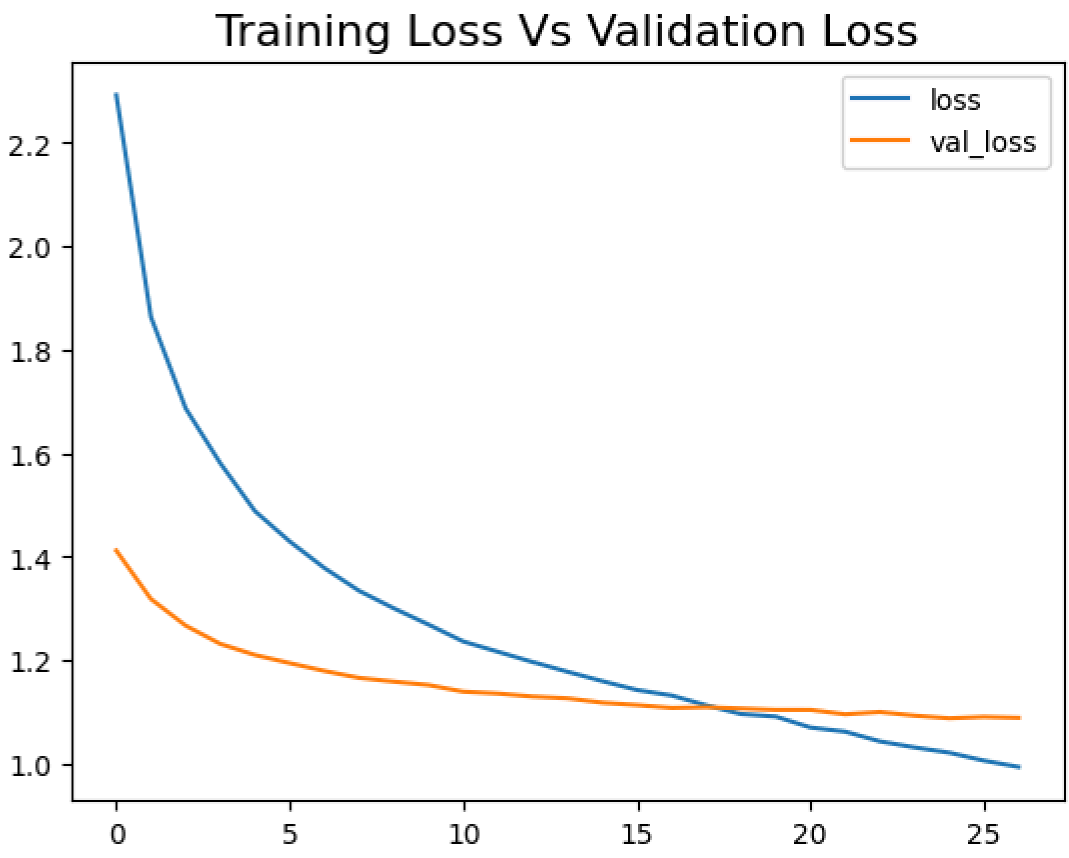
```
metrics_model_10 = pd.DataFrame(model_10.history.history)
metrics_model_10
```

Out[ ]:

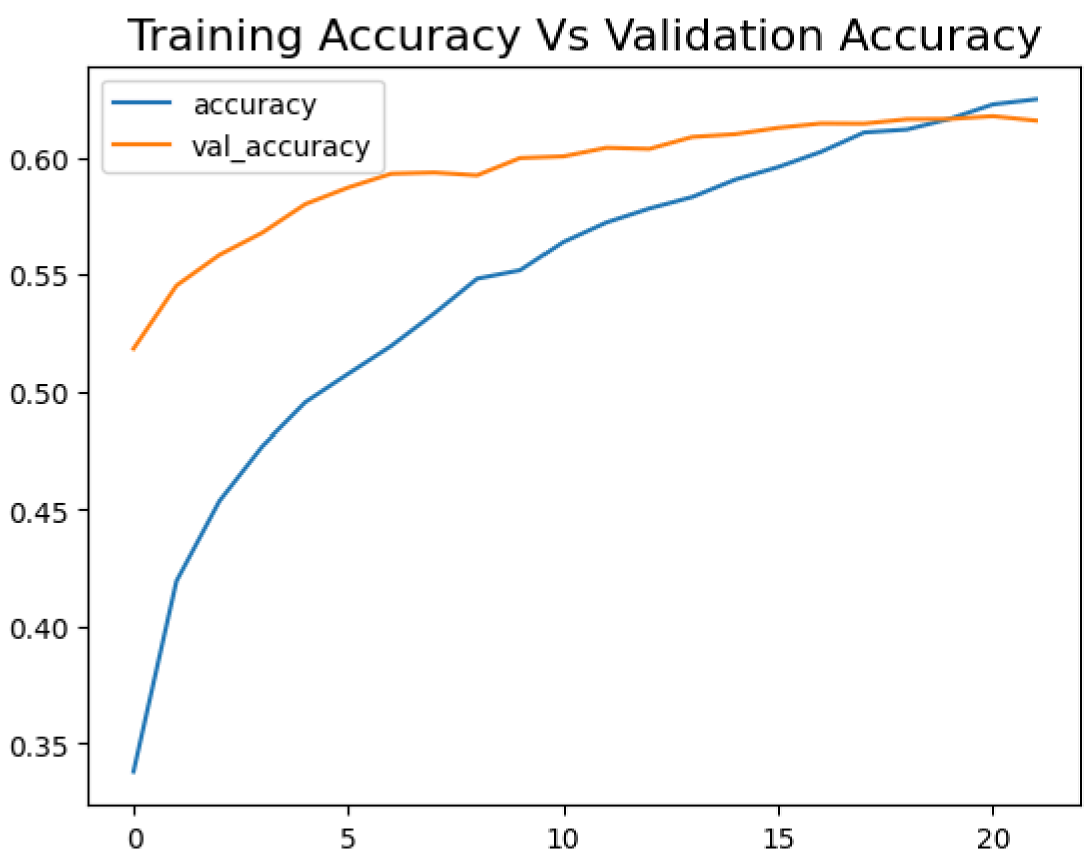
	accuracy	loss	val_accuracy	val_loss
0	0.337775	2.292621	0.5196	1.411963
1	0.421950	1.863356	0.5527	1.318233
2	0.452525	1.686836	0.5602	1.266851
3	0.476300	1.580666	0.5719	1.231443
4	0.496100	1.488023	0.5773	1.210009
5	0.510000	1.429555	0.5825	1.194286
6	0.525700	1.378217	0.5906	1.179150
7	0.532125	1.334097	0.5939	1.165777
8	0.545625	1.300294	0.5943	1.158493
9	0.556825	1.268864	0.5999	1.151955
10	0.565475	1.236239	0.6036	1.139154
11	0.572725	1.216296	0.6035	1.135538
12	0.577175	1.196401	0.6055	1.129682
13	0.583975	1.177494	0.6065	1.126206
14	0.592550	1.159443	0.6079	1.117789
15	0.598850	1.142334	0.6130	1.113209
16	0.598550	1.131674	0.6135	1.107725
17	0.607275	1.112670	0.6118	1.109192
18	0.613375	1.095924	0.6147	1.106581
19	0.614775	1.090944	0.6171	1.104004
20	0.622000	1.069854	0.6150	1.103927
21	0.625300	1.061820	0.6178	1.095364
22	0.632600	1.043063	0.6181	1.099821
23	0.637075	1.031347	0.6188	1.092791
24	0.637050	1.021148	0.6228	1.087670
25	0.646025	1.005807	0.6228	1.090368
26	0.648300	0.993571	0.6214	1.088640

In [ ]:

```
metrics_model_10[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_10[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```

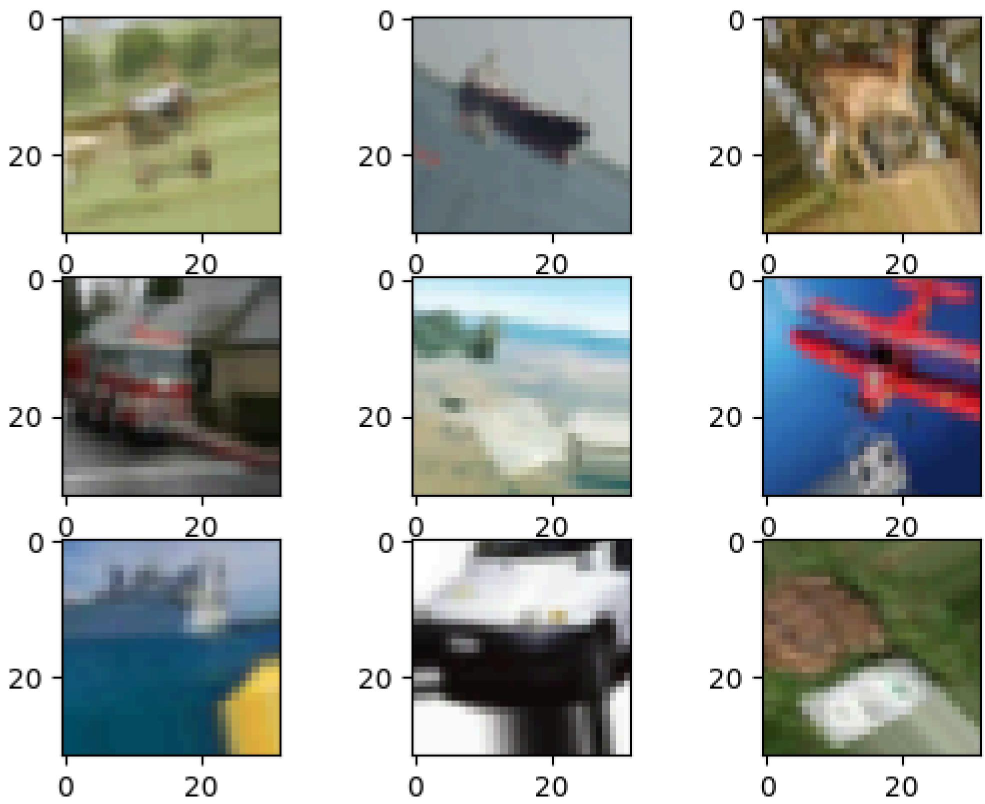


Data augmentation

Generate augmented images

In [ ]:
```
# data augmentation with random transformations (rotation, shifts, zoom, flip, s

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range = 40 ,
    width_shift_range = 0.2 ,
    height_shift_range = 0.2 ,
    shear_range = 0.2 ,
    zoom_range = 0.2 ,
    horizontal_flip = True ,
    fill_mode = 'nearest'
)

datagen.fit(x_train_all)
```

In [ ]:
```
# checking augmented images

for X_batch, y_batch in datagen.flow(x_train_all,
                                     y_train_all,
                                     batch_size=10, # number of images generated
                                     seed=30): # random seed
  for i in range(0,9):
    pyplot.subplot(330 +1 +i)
    pyplot.imshow(X_batch[i])
  pyplot.show()
  break
```



Specify model

In [ ]:
```
del(model_11)
```

In [ ]:
```
model_11 = Sequential(vgg.layers) # same as model 9


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_11.add(Flatten())
model_11.add(Dense(1024,  activation='relu'))
model_11.add(BatchNormalization())
model_11.add(Dropout(0.5)) # discard 50% of neurons
model_11.add(Dense(512,  activation='relu'))
model_11.add(BatchNormalization())
model_11.add(Dropout(0.5)) # discard 50% of neurons
model_11.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_11.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
start = datetime.now()
history_model_11 = model_11.fit(datagen.flow(x_train,
                                 y_train,
                                 batch_size=64),
                    epochs=50, # number of epochs
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )

# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_datas
et_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__ini
t__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multipr
ocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()
```

```
625/625 ━━━━━━━━━━━━━━━━━━━━ 38s 46ms/step - accuracy: 0.1485 - loss: 2.9823 - va
l_accuracy: 0.1196 - val_loss: 3.5630
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.1948 - loss: 2.3621 - va
l_accuracy: 0.2721 - val_loss: 1.8760
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.2234 - loss: 2.1312 - va
l_accuracy: 0.2653 - val_loss: 1.8665
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.2906 - loss: 1.9020 - va
l_accuracy: 0.4069 - val_loss: 1.5698
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.3878 - loss: 1.6256 - va
l_accuracy: 0.4412 - val_loss: 1.5397
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.4320 - loss: 1.5137 - va
l_accuracy: 0.5711 - val_loss: 1.1336
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.4873 - loss: 1.3684 - va
l_accuracy: 0.6059 - val_loss: 1.1101
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.5276 - loss: 1.2853 - va
l_accuracy: 0.5694 - val_loss: 1.2091
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.5689 - loss: 1.1808 - va
l_accuracy: 0.6345 - val_loss: 1.1106
Total elapsed time :  0:04:13.721579
```

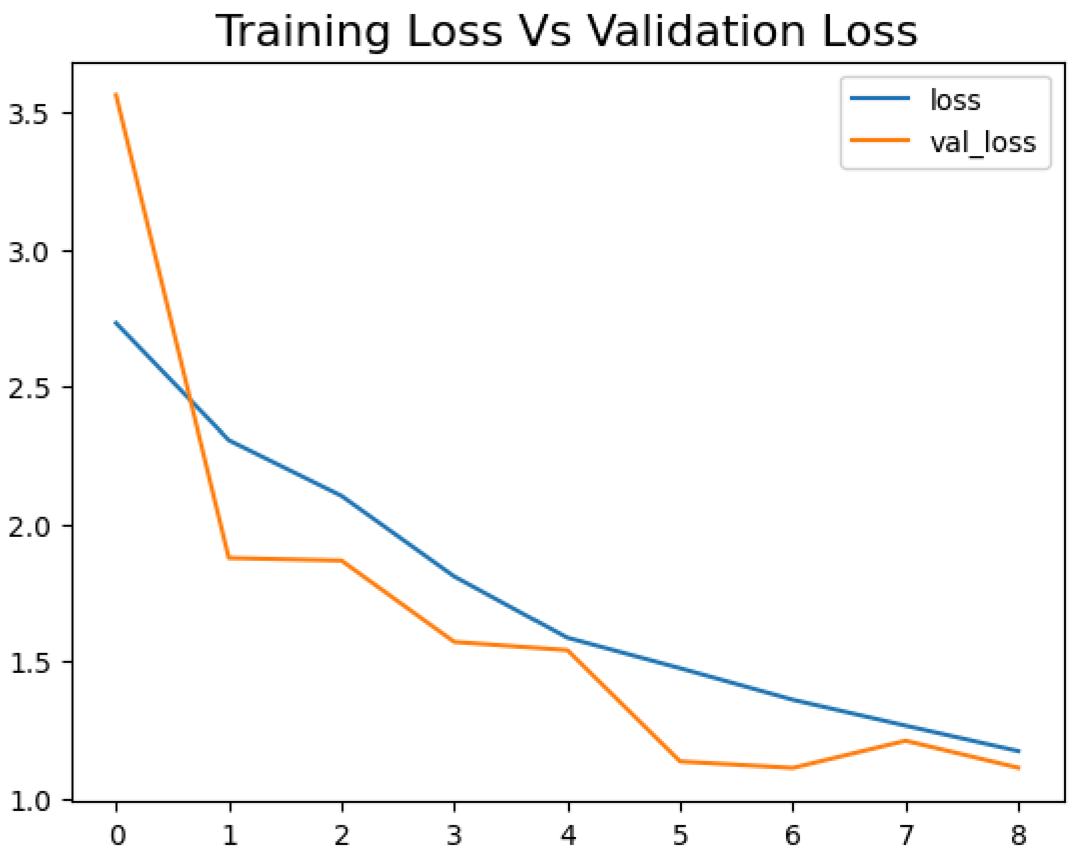
Checking model training metrics

In [ ]:
```
metrics_model_11 = pd.DataFrame(model_11.history.history)
metrics_model_11
```

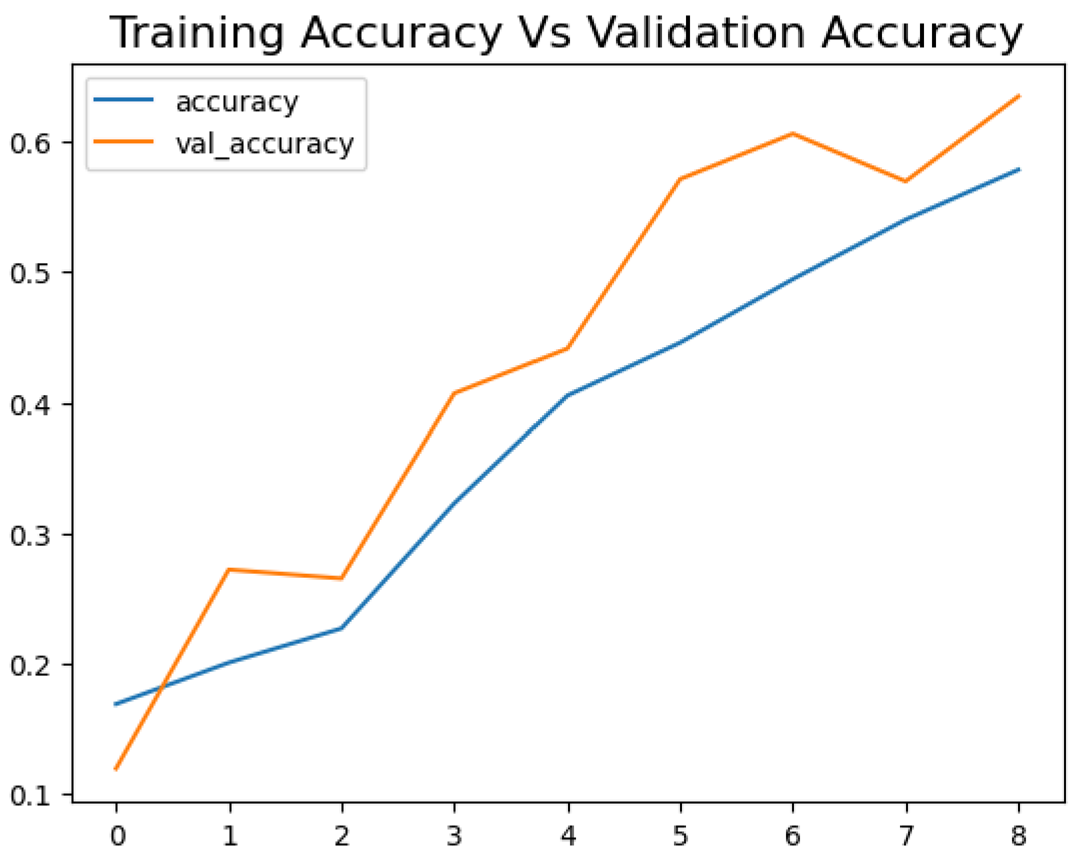
Out[ ]:

	accuracy	loss	val_accuracy	val_loss
0	0.169025	2.731200	0.1196	3.562981
1	0.200750	2.305398	0.2721	1.876022
2	0.227025	2.102570	0.2653	1.866495
3	0.322800	1.808614	0.4069	1.569798
4	0.405175	1.585058	0.4412	1.539678
5	0.445725	1.473634	0.5711	1.133619
6	0.494400	1.358633	0.6059	1.110129
7	0.540025	1.264216	0.5694	1.209114
8	0.578300	1.171503	0.6345	1.110626

In [ ]:
```
metrics_model_11[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_11[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Data augmentation (lower) - model 12

Generate augmented images

In [ ]:
```
# data augmentation with random transformations (rotation, shifts, zoom, flip, s

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,  # Reduce rotation
    width_shift_range=0.1,  # Reduce width shift
    height_shift_range=0.1,  # Reduce height shift
    shear_range=0.1,  # Reduce shear
    zoom_range=0.1,  # Reduce zoom
    horizontal_flip=True,  # Keep horizontal flip
    fill_mode='nearest'
)

datagen.fit(x_train_all)
```

Specify model

In [ ]:
```
model_12 = Sequential(vgg.layers) # same as model 9


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_12.add(Flatten())
model_12.add(Dense(1024,  activation='relu'))
model_12.add(BatchNormalization())
model_12.add(Dropout(0.5)) # discard 50% of neurons
model_12.add(Dense(512,  activation='relu'))
model_12.add(BatchNormalization())
model_12.add(Dropout(0.5)) # discard 50% of neurons
model_12.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_12.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
start = datetime.now()
history_model_12 = model_12.fit(datagen.flow(x_train,
                                 y_train,
                                 batch_size=64),
                    epochs=50, # number of epochs
                    validation_data=(x_val,y_val),
                    callbacks=[early_stop] # early stopping rule
                    )
```

```
# print model training time
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_datas
et_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__ini
t__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multipr
ocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()
625/625 ━━━━━━━━━━━━━━━━━━━━ 36s 45ms/step - accuracy: 0.5327 - loss: 1.4232 - va
l_accuracy: 0.6979 - val_loss: 0.9257
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.6712 - loss: 1.0053 - va
l_accuracy: 0.7069 - val_loss: 0.9355
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.7215 - loss: 0.8763 - va
l_accuracy: 0.7716 - val_loss: 0.6820
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.7645 - loss: 0.7319 - va
l_accuracy: 0.7642 - val_loss: 0.7372
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.7899 - loss: 0.6551 - va
l_accuracy: 0.8035 - val_loss: 0.5997
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8085 - loss: 0.6016 - va
l_accuracy: 0.7973 - val_loss: 0.6232
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8282 - loss: 0.5411 - va
l_accuracy: 0.8000 - val_loss: 0.6431
Total elapsed time :  0:03:16.228897
```

Checking model training metrics

In [ ]:

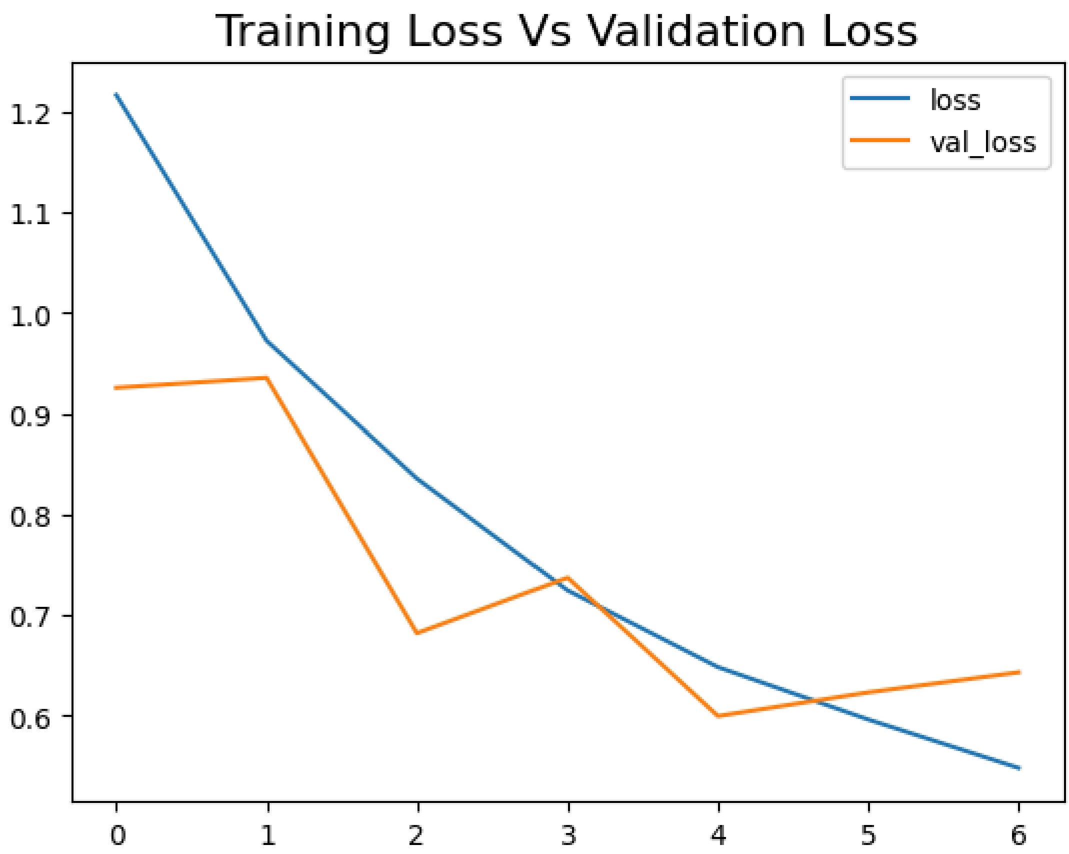
```
metrics_model_12 = pd.DataFrame(model_12.history.history)
metrics_model_12
```

Out[ ]:

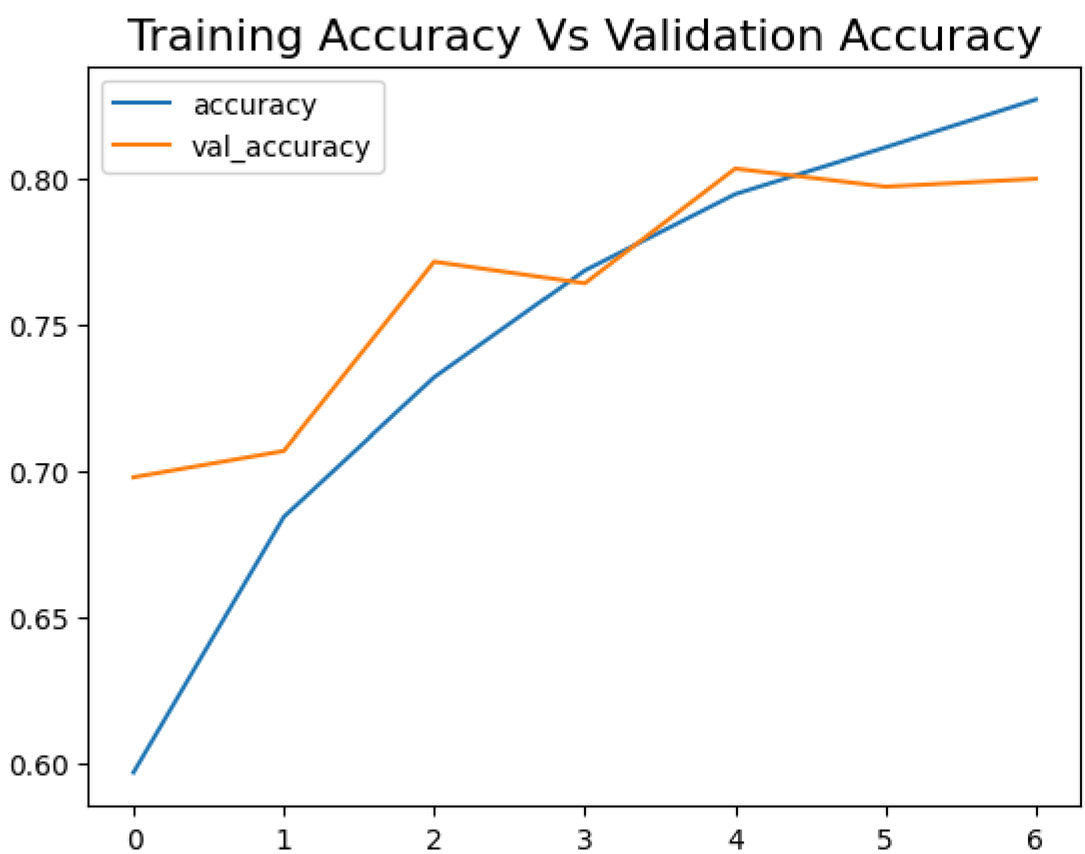
	accuracy	loss	val_accuracy	val_loss
0	0.596750	1.216773	0.6979	0.925662
1	0.684325	0.972460	0.7069	0.935512
2	0.732000	0.835807	0.7716	0.682035
3	0.768600	0.724591	0.7642	0.737224
4	0.794750	0.648303	0.8035	0.599655
5	0.810800	0.596211	0.7973	0.623190
6	0.827175	0.548088	0.8000	0.643121

In [ ]:

```
metrics_model_12[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_12[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Data augmentation (adaptive learning rate) - model 13

Generate augmented images

Specify model

In [ ]:
```
model_13 = Sequential(vgg.layers) # same as model 9


# Flatten and add dense layers + batch normalisation +  trimmed fully connected

model_13.add(Flatten())
model_13.add(Dense(1024,  activation='relu'))
model_13.add(BatchNormalization())
model_13.add(Dropout(0.5)) # discard 50% of neurons
model_13.add(Dense(512,  activation='relu'))
model_13.add(BatchNormalization())
model_13.add(Dropout(0.5)) # discard 50% of neurons
model_13.add(Dense(10,  activation='softmax'))


# Compile with adjusted learning rate
from tensorflow.keras.optimizers import Adam
model_13.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(
        learning_rate=1e-4 # 0.0001; default is 0.001
        ),
    metrics=['accuracy']
)
```

Fitting model

In [ ]:
```
from tensorflow.keras.callbacks import ReduceLROnPlateau

lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=3,
    verbose=1,
    min_lr=1e-6
)

start = datetime.now()
history_model_13 = model_13.fit(datagen.flow(x_train,
                                 y_train,
                                 batch_size=64),
                    epochs=50, # number of epochs
                    validation_data=(x_val,y_val),
                    callbacks=[lr_scheduler] # early stopping rule
                    )

# print model training time
```

```
duration = datetime.now() - start
print('Total elapsed time : ',duration)
buzzer()
```

```
Epoch 1/50
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_datas
et_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__ini
t__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multipr
ocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()
```

```
625/625 ━━━━━━━━━━━━━━━━━━━━ 36s 45ms/step - accuracy: 0.7617 - loss: 0.8013 - va
l_accuracy: 0.8214 - val_loss: 0.5827 - learning_rate: 1.0000e-04
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.8320 - loss: 0.5510 - va
l_accuracy: 0.8485 - val_loss: 0.4946 - learning_rate: 1.0000e-04
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8432 - loss: 0.4963 - va
l_accuracy: 0.8566 - val_loss: 0.4456 - learning_rate: 1.0000e-04
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.8600 - loss: 0.4582 - va
l_accuracy: 0.8369 - val_loss: 0.5417 - learning_rate: 1.0000e-04
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8661 - loss: 0.4302 - va
l_accuracy: 0.8657 - val_loss: 0.4409 - learning_rate: 1.0000e-04
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8710 - loss: 0.4006 - va
l_accuracy: 0.8565 - val_loss: 0.4595 - learning_rate: 1.0000e-04
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8841 - loss: 0.3828 - va
l_accuracy: 0.8586 - val_loss: 0.4755 - learning_rate: 1.0000e-04
Epoch 8/50
624/625 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.8886 - loss: 0.3639
Epoch 8: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.8886 - loss: 0.3639 - va
l_accuracy: 0.8625 - val_loss: 0.4532 - learning_rate: 1.0000e-04
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9069 - loss: 0.3019 - va
l_accuracy: 0.8674 - val_loss: 0.4499 - learning_rate: 5.0000e-05
Epoch 10/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9165 - loss: 0.2687 - va
l_accuracy: 0.8836 - val_loss: 0.3929 - learning_rate: 5.0000e-05
Epoch 11/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9223 - loss: 0.2461 - va
l_accuracy: 0.8768 - val_loss: 0.4246 - learning_rate: 5.0000e-05
Epoch 12/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9265 - loss: 0.2332 - va
l_accuracy: 0.8727 - val_loss: 0.4481 - learning_rate: 5.0000e-05
Epoch 13/50
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9246 - loss: 0.2384
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9246 - loss: 0.2384 - va
l_accuracy: 0.8807 - val_loss: 0.4049 - learning_rate: 5.0000e-05
Epoch 14/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9405 - loss: 0.1921 - va
l_accuracy: 0.8789 - val_loss: 0.4379 - learning_rate: 2.5000e-05
Epoch 15/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9405 - loss: 0.1919 - va
l_accuracy: 0.8900 - val_loss: 0.4004 - learning_rate: 2.5000e-05
Epoch 16/50
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.9476 - loss: 0.1658
Epoch 16: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9476 - loss: 0.1659 - va
l_accuracy: 0.8886 - val_loss: 0.4047 - learning_rate: 2.5000e-05
Epoch 17/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9519 - loss: 0.1551 - va
l_accuracy: 0.8904 - val_loss: 0.3996 - learning_rate: 1.2500e-05
Epoch 18/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9527 - loss: 0.1460 - va
l_accuracy: 0.8872 - val_loss: 0.4225 - learning_rate: 1.2500e-05
Epoch 19/50
```

```
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9554 - loss: 0.1385
Epoch 19: ReduceLROnPlateau reducing learning rate to 6.24999984211172e-06.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9554 - loss: 0.1385 - va
l_accuracy: 0.8879 - val_loss: 0.4247 - learning_rate: 1.2500e-05
Epoch 20/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9608 - loss: 0.1309 - va
l_accuracy: 0.8911 - val_loss: 0.4141 - learning_rate: 6.2500e-06
Epoch 21/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9593 - loss: 0.1328 - va
l_accuracy: 0.8893 - val_loss: 0.4205 - learning_rate: 6.2500e-06
Epoch 22/50
624/625 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9596 - loss: 0.1279
Epoch 22: ReduceLROnPlateau reducing learning rate to 3.12499992105586e-06.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9596 - loss: 0.1279 - va
l_accuracy: 0.8916 - val_loss: 0.4072 - learning_rate: 6.2500e-06
Epoch 23/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9606 - loss: 0.1267 - va
l_accuracy: 0.8938 - val_loss: 0.4050 - learning_rate: 3.1250e-06
Epoch 24/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9622 - loss: 0.1186 - va
l_accuracy: 0.8948 - val_loss: 0.4142 - learning_rate: 3.1250e-06
Epoch 25/50
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9630 - loss: 0.1170
Epoch 25: ReduceLROnPlateau reducing learning rate to 1.56249996052793e-06.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9630 - loss: 0.1170 - va
l_accuracy: 0.8928 - val_loss: 0.4168 - learning_rate: 3.1250e-06
Epoch 26/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9611 - loss: 0.1256 - va
l_accuracy: 0.8940 - val_loss: 0.4101 - learning_rate: 1.5625e-06
Epoch 27/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9647 - loss: 0.1121 - va
l_accuracy: 0.8945 - val_loss: 0.4144 - learning_rate: 1.5625e-06
Epoch 28/50
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.9639 - loss: 0.1156
Epoch 28: ReduceLROnPlateau reducing learning rate to 1e-06.
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9639 - loss: 0.1156 - va
l_accuracy: 0.8923 - val_loss: 0.4212 - learning_rate: 1.5625e-06
Epoch 29/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9652 - loss: 0.1140 - va
l_accuracy: 0.8936 - val_loss: 0.4191 - learning_rate: 1.0000e-06
Epoch 30/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9647 - loss: 0.1144 - va
l_accuracy: 0.8937 - val_loss: 0.4193 - learning_rate: 1.0000e-06
Epoch 31/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9647 - loss: 0.1144 - va
l_accuracy: 0.8938 - val_loss: 0.4197 - learning_rate: 1.0000e-06
Epoch 32/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9621 - loss: 0.1168 - va
l_accuracy: 0.8928 - val_loss: 0.4222 - learning_rate: 1.0000e-06
Epoch 33/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9631 - loss: 0.1196 - va
l_accuracy: 0.8932 - val_loss: 0.4183 - learning_rate: 1.0000e-06
Epoch 34/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9644 - loss: 0.1144 - va
l_accuracy: 0.8948 - val_loss: 0.4186 - learning_rate: 1.0000e-06
Epoch 35/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9648 - loss: 0.1122 - va
l_accuracy: 0.8935 - val_loss: 0.4239 - learning_rate: 1.0000e-06
Epoch 36/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9625 - loss: 0.1156 - va
```

```
l_accuracy: 0.8949 - val_loss: 0.4191 - learning_rate: 1.0000e-06
Epoch 37/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9654 - loss: 0.1117 - va
l_accuracy: 0.8939 - val_loss: 0.4249 - learning_rate: 1.0000e-06
Epoch 38/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9630 - loss: 0.1093 - va
l_accuracy: 0.8939 - val_loss: 0.4232 - learning_rate: 1.0000e-06
Epoch 39/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9653 - loss: 0.1105 - va
l_accuracy: 0.8930 - val_loss: 0.4233 - learning_rate: 1.0000e-06
Epoch 40/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9647 - loss: 0.1096 - va
l_accuracy: 0.8944 - val_loss: 0.4192 - learning_rate: 1.0000e-06
Epoch 41/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9641 - loss: 0.1152 - va
l_accuracy: 0.8936 - val_loss: 0.4228 - learning_rate: 1.0000e-06
Epoch 42/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9623 - loss: 0.1179 - va
l_accuracy: 0.8939 - val_loss: 0.4239 - learning_rate: 1.0000e-06
Epoch 43/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9650 - loss: 0.1107 - va
l_accuracy: 0.8940 - val_loss: 0.4218 - learning_rate: 1.0000e-06
Epoch 44/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9662 - loss: 0.1073 - va
l_accuracy: 0.8944 - val_loss: 0.4210 - learning_rate: 1.0000e-06
Epoch 45/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9654 - loss: 0.1076 - va
l_accuracy: 0.8936 - val_loss: 0.4282 - learning_rate: 1.0000e-06
Epoch 46/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9643 - loss: 0.1103 - va
l_accuracy: 0.8950 - val_loss: 0.4226 - learning_rate: 1.0000e-06
Epoch 47/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9641 - loss: 0.1102 - va
l_accuracy: 0.8941 - val_loss: 0.4274 - learning_rate: 1.0000e-06
Epoch 48/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9661 - loss: 0.1108 - va
l_accuracy: 0.8941 - val_loss: 0.4277 - learning_rate: 1.0000e-06
Epoch 49/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9660 - loss: 0.1090 - va
l_accuracy: 0.8944 - val_loss: 0.4252 - learning_rate: 1.0000e-06
Epoch 50/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 42ms/step - accuracy: 0.9686 - loss: 0.1059 - va
l_accuracy: 0.8928 - val_loss: 0.4290 - learning_rate: 1.0000e-06
Total elapsed time :  0:22:30.500634
```

Checking model training metrics

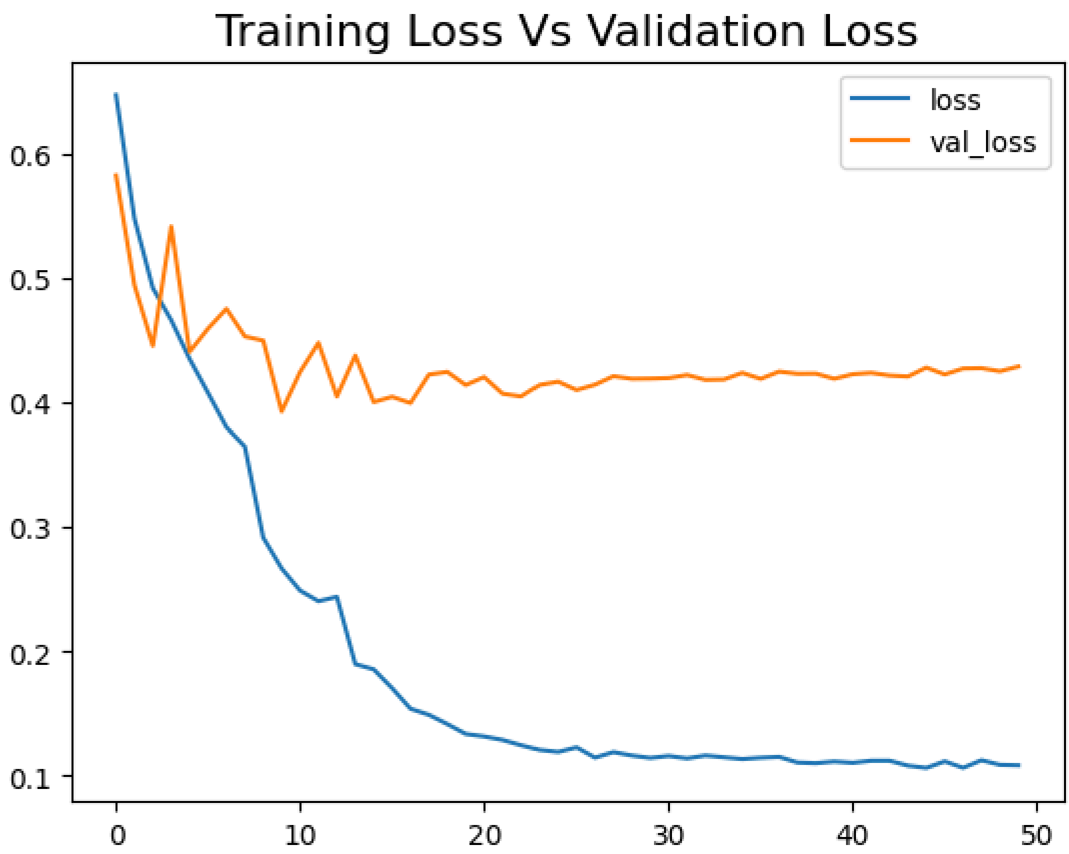
In [ ]:
```
metrics_model_13 = pd.DataFrame(model_13.history.history)
metrics_model_13
```

Out[ ]:

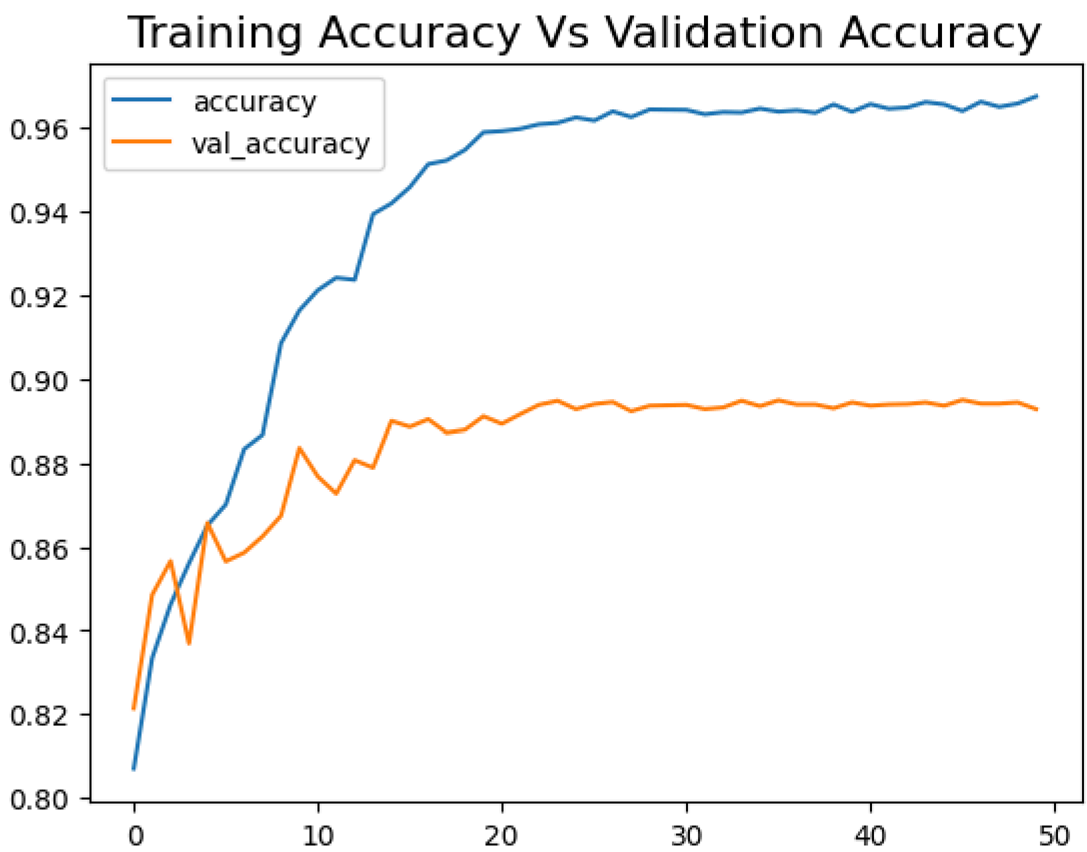
	accuracy	loss	val_accuracy	val_loss	learning_rate
0	0.807000	0.647473	0.8214	0.582718	1.000000e-04
1	0.833400	0.548178	0.8485	0.494560	1.000000e-04
2	0.846200	0.492141	0.8566	0.445649	1.000000e-04
3	0.856050	0.466268	0.8369	0.541673	1.000000e-04
4	0.865125	0.435057	0.8657	0.440912	1.000000e-04
5	0.870100	0.407805	0.8565	0.459536	1.000000e-04
6	0.883350	0.380275	0.8586	0.475468	1.000000e-04
7	0.886750	0.364836	0.8625	0.453232	1.000000e-04
8	0.908550	0.291687	0.8674	0.449856	5.000000e-05
9	0.916400	0.266891	0.8836	0.392896	5.000000e-05
10	0.921225	0.249131	0.8768	0.424623	5.000000e-05
11	0.924200	0.240544	0.8727	0.448095	5.000000e-05
12	0.923700	0.244065	0.8807	0.404878	5.000000e-05
13	0.939400	0.189939	0.8789	0.437879	2.500000e-05
14	0.941975	0.185730	0.8900	0.400438	2.500000e-05
15	0.945850	0.170599	0.8886	0.404657	2.500000e-05
16	0.951350	0.154037	0.8904	0.399633	1.250000e-05
17	0.952250	0.149112	0.8872	0.422543	1.250000e-05
18	0.954775	0.141677	0.8879	0.424689	1.250000e-05
19	0.958975	0.133739	0.8911	0.414119	6.250000e-06
20	0.959250	0.131737	0.8893	0.420538	6.250000e-06
21	0.959800	0.128914	0.8916	0.407177	6.250000e-06
22	0.960875	0.124693	0.8938	0.404986	3.125000e-06
23	0.961225	0.120876	0.8948	0.414205	3.125000e-06
24	0.962525	0.119264	0.8928	0.416817	3.125000e-06
25	0.961775	0.122943	0.8940	0.410122	1.562500e-06
26	0.963975	0.114663	0.8945	0.414419	1.562500e-06
27	0.962625	0.119130	0.8923	0.421229	1.562500e-06
28	0.964425	0.116357	0.8936	0.419102	1.000000e-06
29	0.964375	0.114435	0.8937	0.419285	1.000000e-06
30	0.964325	0.116028	0.8938	0.419722	1.000000e-06
31	0.963275	0.114059	0.8928	0.422208	1.000000e-06
32	0.963800	0.116370	0.8932	0.418283	1.000000e-06

	accuracy	loss	val_accuracy	val_loss	learning_rate
33	0.963700	0.114981	0.8948	0.418579	1.000000e-06
34	0.964650	0.113522	0.8935	0.423912	1.000000e-06
35	0.963900	0.114598	0.8949	0.419085	1.000000e-06
36	0.964225	0.115294	0.8939	0.424936	1.000000e-06
37	0.963625	0.110750	0.8939	0.423183	1.000000e-06
38	0.965625	0.110212	0.8930	0.423293	1.000000e-06
39	0.963875	0.111553	0.8944	0.419162	1.000000e-06
40	0.965700	0.110498	0.8936	0.422827	1.000000e-06
41	0.964600	0.112209	0.8939	0.423914	1.000000e-06
42	0.964900	0.112217	0.8940	0.421808	1.000000e-06
43	0.966200	0.108230	0.8944	0.420951	1.000000e-06
44	0.965650	0.106419	0.8936	0.428169	1.000000e-06
45	0.964050	0.111678	0.8950	0.422644	1.000000e-06
46	0.966275	0.106492	0.8941	0.427404	1.000000e-06
47	0.965050	0.112597	0.8941	0.427674	1.000000e-06
48	0.965850	0.108906	0.8944	0.425198	1.000000e-06
49	0.967550	0.108512	0.8928	0.428991	1.000000e-06

In [ ]:
```
metrics_model_13[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



In [ ]:
```
metrics_model_13[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Assessing model performance on test data

Fitting model on test set

In [ ]:
```
model = model_13

model.evaluate(x_test,y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8837 - loss: 0.4736
```

Out[ ]:
```
[0.4791160523891449, 0.8835999965667725]
```

Classification report and confusion matrix

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

predictions = np.argmax(model.predict(x_test), axis=-1)

print(classification_report(y_test,predictions))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1000
           1       0.93      0.95      0.94      1000
           2       0.88      0.84      0.86      1000
           3       0.78      0.73      0.76      1000
           4       0.89      0.86      0.87      1000
           5       0.85      0.77      0.80      1000
           6       0.86      0.95      0.90      1000
           7       0.88      0.93      0.90      1000
           8       0.95      0.94      0.95      1000
           9       0.91      0.94      0.93      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

cm=confusion_matrix(y_test,predictions)

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm,
            annot=True,
            fmt='g',
            cmap='Reds',
            xticklabels=np.arange(10),
            yticklabels=np.arange(10))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```
plt.title('Confusion Matrix Heatmap')
plt.show()
```