Type *Markdown* and LaTeX: α^2

Ninjacart : Image classification using CNN and transfer learning

In [ ]:

```
Link - https://colab.research.google.com/drive/1A6jRmhFRFHIFiUYou7HDzlwT6F6MC8bk?usp=sharing

Summary and Insights :

1.  CNN from Scratch: 14,21,148 params
Validation accuracy - 93.38%
Test Accuracy - 88.03

2. VGG : 3,16,71,740 params
Validation accuracy - 96.12%
Test Accuracy - 87.18 %

3. Resnet : 7,51,21,020
Validation accuracy - 98.00%%
Test Accuracy - 90.88%

4. Mobilenet : 1,27,55,900
Validation accuracy - 98.25%
Test Accuracy - 92.31%

The best model is Mobilenet as it has the least number of parameters and the best testing accur

To train a CNN we will need a lot of images. Tensorboard showed in the last page of the noteboo
```

In [ ]:

```
!gdown 1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT
# Downloading the Data
```

```
Downloading...
From: https://drive.google.com/uc?id=1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT (https://d
rive.google.com/uc?id=1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT)
To: /content/ninjacart_data.zip
100% 275M/275M [00:08<00:00, 31.4MB/s]
```

In [ ]:

```
import shutil
shutil.rmtree('ninjacart_data')
```

In [ ]:

```
!unzip /content/ninjacart_data.zip
# Unziped the data
```

```
Archive:  /content/ninjacart_data.zip
   creating: ninjacart_data/test/
   creating: ninjacart_data/test/indian market/
  inflating: ninjacart_data/test/indian market/bhl.jpeg
  inflating: ninjacart_data/test/indian market/bhv.jpeg
  inflating: ninjacart_data/test/indian market/bn.jpeg
  inflating: ninjacart_data/test/indian market/hjx.jpeg
  inflating: ninjacart_data/test/indian market/igis.jpeg
  inflating: ninjacart_data/test/indian market/in.jpeg
  inflating: ninjacart_data/test/indian market/india-4898453__340.jpg
  inflating: ninjacart_data/test/indian market/indianmarket10.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket12.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket13.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket14.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket15.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket18.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket19.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket20.jpeg
  inflating: ninjacart_data/test/indian market/indianmarket21.jpeg
```

In [ ]:

In [ ]:

```
# Import common libraries
import os
import glob
import random
import numpy as np
import pandas as pd
import sklearn.metrics as metrics
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array, random_shift
from tensorflow.keras.preprocessing.image import array_to_img


import tensorflow as tf
from tensorflow import keras # this allows <keras.> instead of <tf.keras.>
from tensorflow.keras import layers # this allows <layers.> instead of <tf.keras.layers.>
tf.keras.utils.set_random_seed(111) # set random seed

import os
import numpy as np
import tensorflow as tf
tf.keras.utils.set_random_seed(111)
from tensorflow import keras
from tensorflow.keras import layers, regularizers
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.metrics as metrics

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers import BatchNormalization
import numpy as np
from tensorflow.keras.callbacks import TensorBoard

# To supress any warnings during the flow
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
class_dirs = os.listdir("ninjacart_data/train") # List all directories inside "train" folder
image_dict = {} # dict to store image array(key) for every class(value)
count_dict = {} # dict to store count of files(key) for every class(value)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'ninjacart_data/train/{cls}/*')
    # count number of files in each class and add it to count_dict
    count_dict[cls] = len(file_paths)
    # select random item from list of image paths
    image_path = random.choice(file_paths)
    # load image using keras utility function and save it in image_dict
    image_dict[cls] = tf.keras.utils.load_img(image_path)
```

In [ ]:

```python
plt.figure(figsize=(15, 12))
# iterate over dictionary items (class label, image array)
for i, (cls,img) in enumerate(image_dict.items()):
    # create a subplot axis
    ax = plt.subplot(3, 4, i + 1)
    # plot each image
    plt.imshow(img)
    # set "class name" along with "image size" as title
    plt.title(f'{cls}, {img.size}')
    plt.axis("off")
# Plotting a few images from each class for viewing
# We can conclude that image sizes are not constant
```

In [ ]:

```
## Let's now Plot the Data Distribution of Training Data across Classes
df_count_train = pd.DataFrame({
    "class": count_dict.keys(),     # keys of count_dict are class labels
    "count": count_dict.values(),   # value of count_dict contain counts of each class
})
print("Count of training samples per class:\n", df_count_train)

# draw a bar plot using pandas in-built plotting function
df_count_train.plot.bar(x='class', y='count', title="Training Data Count per class")

# Below table shows the number of samples avaialble in each class
# Data is imbalanced
# There are a total of 4 classes
```

```
Count of training samples per class:
             class  count
0           onion    849
1          tomato    789
2  indian market    599
3          potato    898
```

Out[7]:

```
<Axes: title={'center': 'Training Data Count per class'}, xlabel='class'>
```

Data augmentation to make sample size as 1000 for each class

1. Random Crop
2. Image rotation
3. Random flip

In [ ]:

```
class_dirs = os.listdir("ninjacart_data/train") # list all directories inside "train" folder
image_dict = {} # dict to store image array(key) for every class(value)
count_dict = {} # dict to store count of files(key) for every class(value)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'ninjacart_data/train/{cls}/*')
    last=len(file_paths)
    list_path=random.choices(file_paths,k=1000-last)
    print(len(list_path))
    for i in range(0,len(list_path)-2,3):
      #print(i,cls)

      img = load_img(list_path[i],target_size=(300,300))
      x = img_to_array(img)
      x_crop = tf.image.random_crop(value=x, size=(224, 224, 3))
      array_to_img(x_crop)
      out_filename = f'{str(i)}.jpg'
      img.save(f'ninjacart_data/train/{cls}/' + out_filename)

      img = load_img(list_path[i+1],target_size=(300,300))
      x = img_to_array(img)
      x_rot=tf.keras.preprocessing.image.random_rotation(x, rg=125,channel_axis = 2, row_axis=0
      array_to_img(x_rot)
      out_filename = f'{str(i+1)}.jpg'
      img.save(f'ninjacart_data/train/{cls}/'+ out_filename)

      img = load_img(list_path[i+2],target_size=(300,300))
      x = img_to_array(img)
      x_flip = tf.image.random_flip_left_right(x, 10)
      array_to_img(x_flip)
      out_filename = f'{str(i+2)}.jpg'
      img.save(f'ninjacart_data/train/{cls}/' + out_filename)
```

```
151
211
401
102
```

In [ ]:

```
class_dirs = os.listdir("ninjacart_data/train") # list all directories inside "train" folder
image_dict = {} # dict to store image array(key) for every class(value)
count_dict = {} # dict to store count of files(key) for every class(value)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'ninjacart_data/train/{cls}/*')
    # count number of files in each class and add it to count_dict
    count_dict[cls] = len(file_paths)
    # select random item from list of image paths
    image_path = random.choice(file_paths)
    # load image using keras utility function and save it in image_dict
    image_dict[cls] = tf.keras.utils.load_img(image_path)
```

In [ ]:

```
plt.figure(figsize=(15, 12))
# iterate over dictionary items (class label, image array)
for i, (cls,img) in enumerate(image_dict.items()):
    # create a subplot axis
    ax = plt.subplot(3, 4, i + 1)
    # plot each image
    plt.imshow(img)
    # set "class name" along with "image size" as title
    plt.title(f'{cls}, {img.size}')
    plt.axis("off")

# Checking image in each class
```

In [ ]:

```
## Let's now Plot the Data Distribution of Training Data across Classes
df_count_train = pd.DataFrame({
    "class": count_dict.keys(),     # keys of count_dict are class labels
    "count": count_dict.values(),   # value of count_dict contain counts of each class
})
print("Count of training samples per class:\n", df_count_train)

# draw a bar plot using pandas in-built plotting function
df_count_train.plot.bar(x='class', y='count', title="Training Data Count per class")

# All classes have image count close to 1000
# Now it is a balanced dataset
```

```
Count of training samples per class:
            class  count
0          onion    999
1         tomato    999
2  indian market    998
3         potato   1000
```

Out[11]:

```
<Axes: title={'center': 'Training Data Count per class'}, xlabel='class'>
```

Creating Validation Data Set of size 200 each

In [ ]:

```
import os
import random
from shutil import move

train_dir = 'ninjacart_data/train'
validation_dir = 'ninjacart_data/validation'

# Create the validation directory if it does not exist
if not os.path.exists(validation_dir):
    os.mkdir(validation_dir)

# Get a list of all the files in the train directory
files = os.listdir(train_dir)
for i in files:
  os.mkdir(f'ninjacart_data/validation/{i}')

for i in files:
  files = os.listdir(f'ninjacart_data/train/{i}')
  #print(len(files))
  random_files = random.sample(files, 200)
  #print(len(files))

  # Move the selected files to the validation directory
  for file in random_files:
      move(f'ninjacart_data/train/{i}/' + file, f'ninjacart_data/validation/{i}/' + file)
```

In [ ]:

```
class_dirs = os.listdir("ninjacart_data/train") # list all directories inside "train" folder
image_dict = {} # dict to store image array(key) for every class(value)
count_dict = {} # dict to store count of files(key) for every class(value)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'ninjacart_data/train/{cls}/*')
    # count number of files in each class and add it to count_dict
    count_dict[cls] = len(file_paths)
```

In [ ]:

```
count_dict
```

Out[14]:

```
{'onion': 799, 'tomato': 799, 'indian market': 798, 'potato': 800}
```

In [ ]:

```
class_dirs = os.listdir("ninjacart_data/validation") # list all directories inside "train" folde
image_dict = {} # dict to store image array(key) for every class(value)
count_dict = {} # dict to store count of files(key) for every class(value)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'ninjacart_data/validation/{cls}/*')
    # count number of files in each class and add it to count_dict
    count_dict[cls] = len(file_paths)
```

In [ ]:

```
count_dict
# We have 200 images in validation dataset (20% from training data)
```

Out[16]:

```
{'onion': 200, 'tomato': 200, 'indian market': 200, 'potato': 200}
```

Loading Data

In [ ]:

```
def load_data(base_dir="ninjacart_data"):

    print('\nLoading Data...')
    train_data = tf.keras.utils.image_dataset_from_directory(
        f"{base_dir}/train", shuffle=True, label_mode='categorical'
    )
    val_data = tf.keras.utils.image_dataset_from_directory(
        f"{base_dir}/validation", shuffle=False, label_mode='categorical'
    )
    test_data = tf.keras.utils.image_dataset_from_directory(
        f"{base_dir}/test", shuffle=False, label_mode='categorical'
    )
    return train_data, val_data, test_data, train_data.class_names
```

In [ ]:

```
train_data, val_data, test_data, class_names = load_data()
```

```
Loading Data...
Found 3196 files belonging to 4 classes.
Found 800 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
```

Setting image dimension and rescaling images

In [ ]:

```
def preprocess(train_data, val_data, test_data, target_height=256, target_width=256):

    # Data Processing Stage with resizing and rescaling operations
    data_preprocess = keras.Sequential(
        name="data_preprocess",
        layers=[
            layers.Resizing(target_height, target_width),
            layers.Rescaling(1.0/255),
        ]
    )

    # Perform Data Processing on the train, val, test dataset
    train_ds = train_data.map(lambda x, y: (data_preprocess(x), y), num_parallel_calls=tf.data.
    val_ds = val_data.map(lambda x, y: (data_preprocess(x), y), num_parallel_calls=tf.data.AUTO
    test_ds = test_data.map(lambda x, y: (data_preprocess(x), y), num_parallel_calls=tf.data.AU

    return train_ds, val_ds, test_ds
```

In [ ]:

```

```

In [ ]:

```
train_ds, val_ds, test_ds = preprocess(train_data, val_data, test_data)
```

In [ ]:

```
L2Reg = tf.keras.regularizers.L2(l2=1e-6)
# Regularizers
```

In [ ]:

```
def scheduler(epoch, lr):
  if epoch < 20:
    return lr
  if epoch < 35:
    return 0.0001
  if epoch < 50:
    return 0.00001
  if epoch < 80:
    return 0.000001
  else:
    return 0.0000001

# Learning rate scheduler
```

Function for fitting

1. Saving weights

2. Learning rate scheduler

In [ ]:

```
def compile_train_v1(model, train_ds, val_ds, tb_callback,epochs=100, ckpt_path="/tmp/checkpoin
    model.compile(optimizer=tf.keras.optimizers.Adam(beta_1=0.9,beta_2=0.999),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model_fit = model.fit(train_ds, validation_data=val_ds, epochs=epochs, batch_size=230,callb
        keras.callbacks.ModelCheckpoint(ckpt_path, save_weights_only=True, monitor='val_accurac
    tf.keras.callbacks.LearningRateScheduler(scheduler),tb_callback,tf.keras.callbacks.EarlySto
    return model_fit
```

Function to plot accuracy and confusion matrix

In [ ]:

```
def print_accuracy_stats(model, ds, class_names,ckpt_path):
    model.load_weights(ckpt_path)
    true_onehot = tf.concat([y for x, y in ds], axis=0)
    true_categories = tf.argmax(true_onehot, axis=1)
    y_pred = model.predict(ds)
    predicted_categories = tf.argmax(y_pred, axis=1)

    test_acc = metrics.accuracy_score(true_categories, predicted_categories) * 100
    print(f'\n Accuracy: {test_acc:.2f}%\n')

# Note: This doesn't work with shuffled datasets
def plot_confusion_matrix(model, ds, class_names,ckpt_path):
    model.load_weights(ckpt_path)
    true_onehot = tf.concat([y for x, y in ds], axis=0)
    true_categories = tf.argmax(true_onehot, axis=1)
    y_pred = model.predict(ds)
    predicted_categories = tf.argmax(y_pred, axis=1)
    cm = metrics.confusion_matrix(true_categories,predicted_categories) # last batch
    sns.heatmap(cm, annot=True, xticklabels=class_names, yticklabels=class_names, cmap="YlGnBu"
    plt.show()
```

In [ ]:

```

```

CNN Classifier model from scratch

Tensorboard

In [ ]:

```
%load_ext tensorboard
log_folder='logs/1'
%reload_ext tensorboard
!rm -rf logs
tb_callback=TensorBoard(log_dir=log_folder,histogram_freq=1)
```

In [ ]:

```
model1 = keras.Sequential(
        name="model_cnn",
        layers=[
            layers.Conv2D(filters=16, kernel_size=(3,3), padding="same", input_shape=(256,256, 
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),


            layers.Conv2D(filters=32, kernel_size=(5,5), padding="same",
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),


            layers.Conv2D(filters=64, kernel_size=(6,6), padding="same",
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),


            layers.Conv2D(filters=128, kernel_size=(7,7), padding="same",
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),


            layers.Conv2D(filters=256, kernel_size=(8,8), padding="same",
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),

            layers.Conv2D(filters=512, kernel_size=(8,8), padding="same",
                            kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.MaxPooling2D(),


            layers.GlobalAveragePooling2D(),
            layers.Flatten(),

            layers.Dense(units=512, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),


            layers.Dense(units=256, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=128, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),
```

```
            layers.Dense(units=64, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=32, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=16, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=8, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.1),


            layers.Dense(units=4, activation='softmax')
        ]
    )
```

In [ ]:

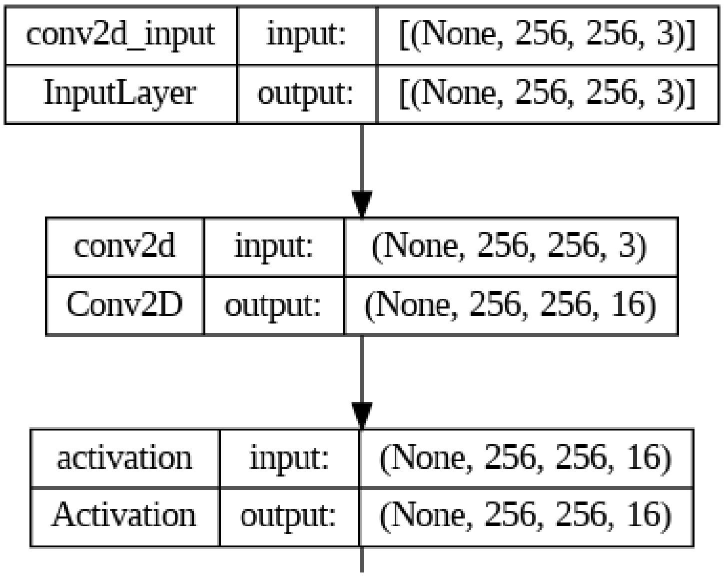
```
model1.summary()
```

```
Model: "model_cnn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      448

 activation (Activation)     (None, 256, 256, 16)      0

 batch_normalization (BatchN  (None, 256, 256, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      12832

 activation_1 (Activation)   (None, 128, 128, 32)      0

 batch_normalization_1 (Batc  (None, 128, 128, 32)     128
```

In [ ]:

```
tf.keras.utils.plot_model(model1, show_shapes=True)
```

Out[28]:



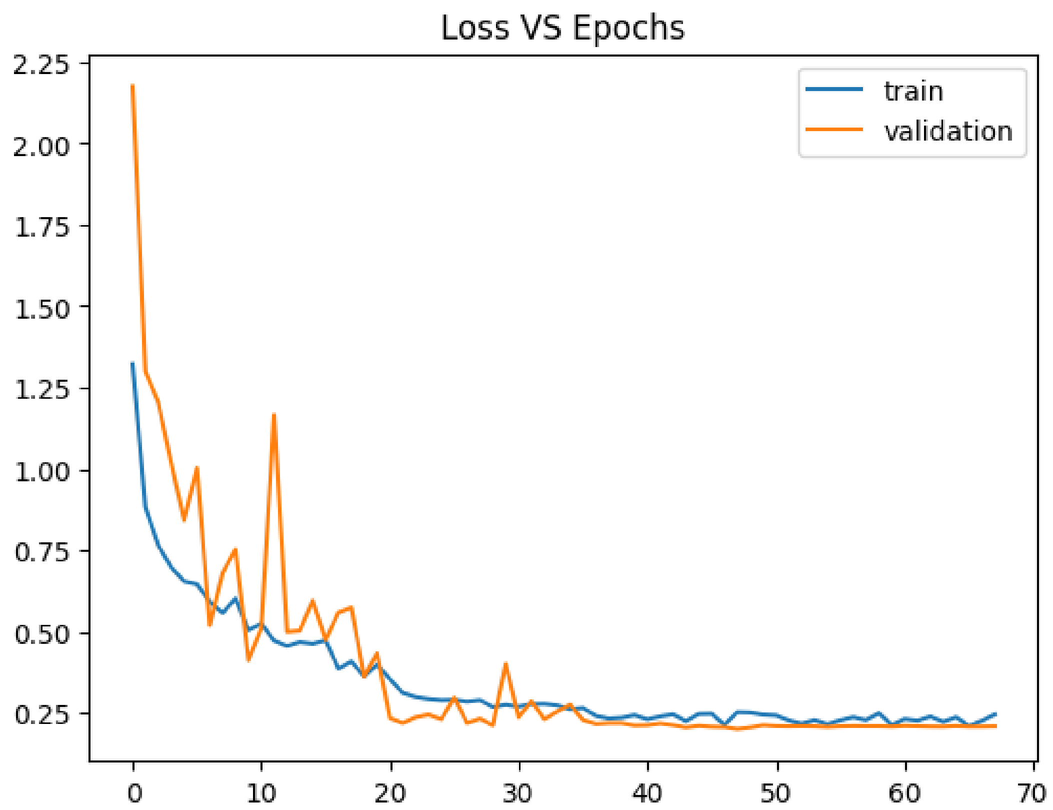
In [ ]:

```
model_fit1 = compile_train_v1(model1, train_ds, val_ds,tb_callback,ckpt_path="/tmp/checkpoint1"
```

```
Epoch 1/100
100/100 [==============================] - 47s 222ms/step - loss: 1.3223 - acc
uracy: 0.4305 - val_loss: 2.1755 - val_accuracy: 0.2912 - lr: 0.0010
Epoch 2/100
100/100 [==============================] - 19s 182ms/step - loss: 0.8822 - acc
uracy: 0.6399 - val_loss: 1.2996 - val_accuracy: 0.4787 - lr: 0.0010
Epoch 3/100
100/100 [==============================] - 19s 180ms/step - loss: 0.7648 - acc
uracy: 0.7181 - val_loss: 1.2037 - val_accuracy: 0.4625 - lr: 0.0010
Epoch 4/100
100/100 [==============================] - 18s 175ms/step - loss: 0.6984 - acc
uracy: 0.7597 - val_loss: 1.0190 - val_accuracy: 0.6212 - lr: 0.0010
Epoch 5/100
100/100 [==============================] - 19s 180ms/step - loss: 0.6549 - acc
uracy: 0.7813 - val_loss: 0.8417 - val_accuracy: 0.6363 - lr: 0.0010
Epoch 6/100
100/100 [==============================] - 19s 184ms/step - loss: 0.6462 - acc
uracy: 0.7735 - val_loss: 1.0029 - val_accuracy: 0.6413 - lr: 0.0010
Epoch 7/100
```

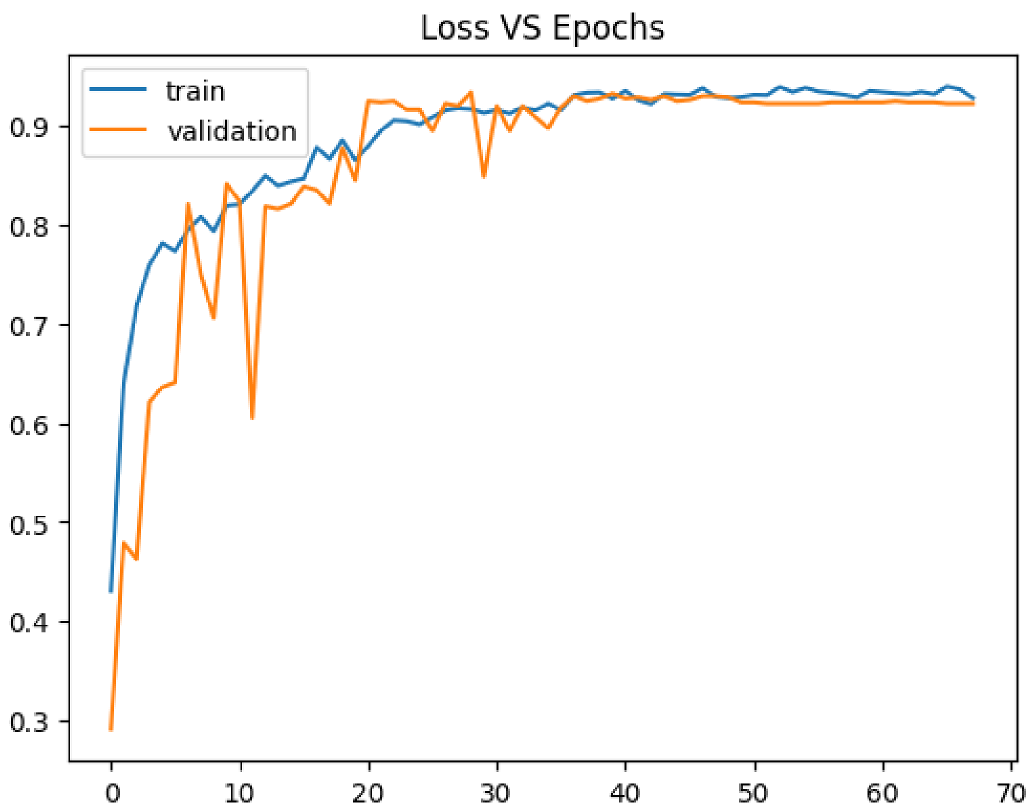
In [ ]:

```
epochs = model_fit1.epoch
loss = model_fit1.history["loss"]
val_loss = model_fit1.history["val_loss"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

```
epochs = model_fit1.epoch
loss = model_fit1.history["accuracy"]
val_loss = model_fit1.history["val_accuracy"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

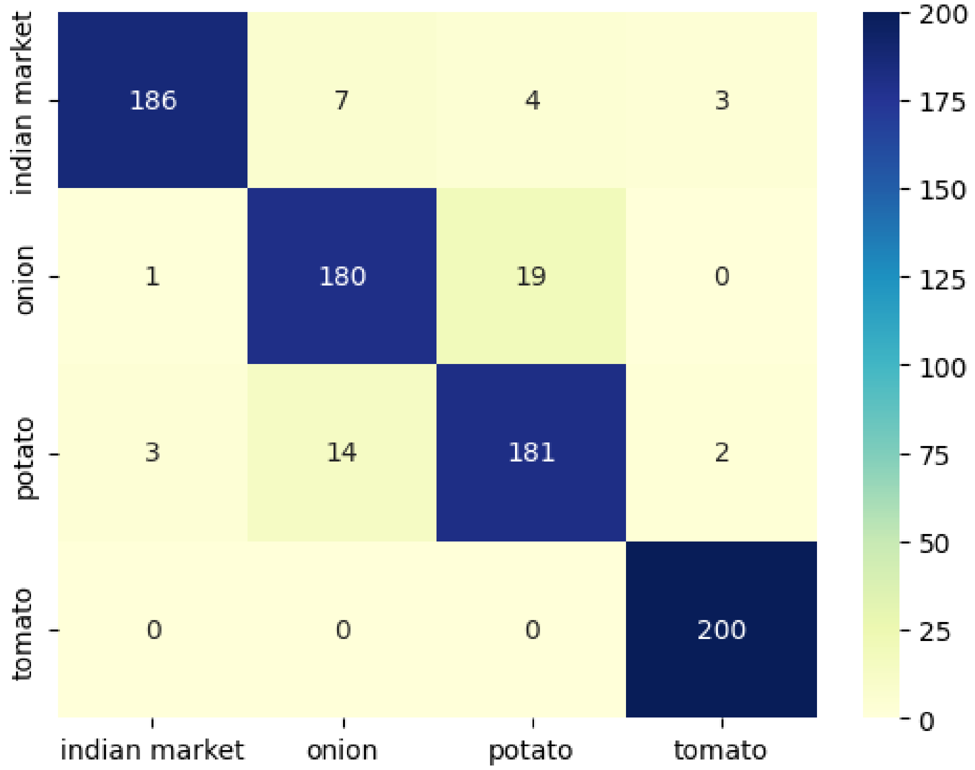
```
print_accuracy_stats(model1, test_ds, class_names,"/tmp/checkpoint1")
# Test Accuracy for CNN from scratch is 91.74%
```

```
11/11 [==============================] - 3s 225ms/step

 Accuracy: 88.03%
```

In [ ]:

```
plot_confusion_matrix(model1, val_ds, class_names,"/tmp/checkpoint1")
# Confusion matrix for validation data
```

```
25/25 [==============================] - 2s 86ms/step
```



In [ ]:

```
print_accuracy_stats(model1, val_ds, class_names,"/tmp/checkpoint1")
# TValidation Accuracy for CNN from scratch is 91.74%
```

```
25/25 [==============================] - 3s 119ms/step

 Accuracy: 93.38%
```

In [ ]:

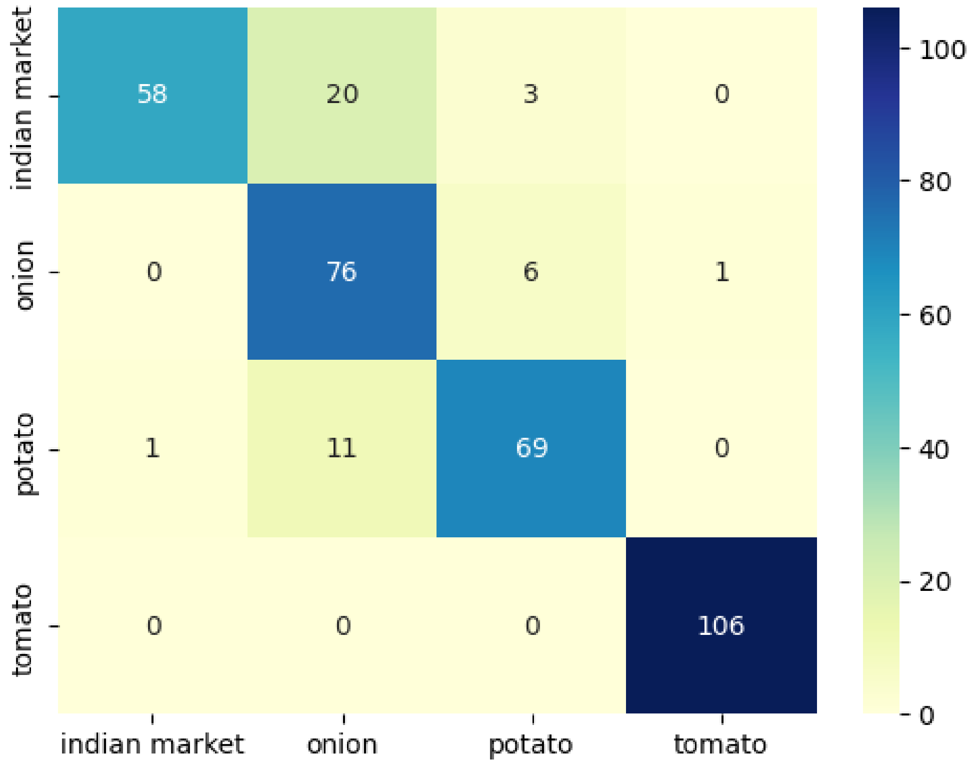
```
print_accuracy_stats(model1, test_ds, class_names,"/tmp/checkpoint1")
```

```
11/11 [==============================] - 1s 107ms/step

 Accuracy: 88.03%
```

In [ ]:

```
plot_confusion_matrix(model1, test_ds, class_names,"/tmp/checkpoint1")
```

```
11/11 [==============================] - 1s 108ms/step
```



In [ ]:

```
#%tensorboard --logdir logs
```

VGG

In [ ]:

```
#%load_ext tensorboard
log_folder='logs/2'
#%reload_ext tensorboard
#!rm -rf logs
tb_callback=TensorBoard(log_dir=log_folder,histogram_freq=1)
```

In [ ]:

```
pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_sha
pretrained_model.trainable=False
vgg16_model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    #tf.keras.layers.Dense(10, activation='softmax')



            layers.Dense(units=512, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=256, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=128, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=64, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=32, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=16, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=8, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.1),

            layers.Dense(units=4, activation='softmax')
])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.google
apis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_ker
nels_notop.h5)
58889256/58889256 [==============================] - 0s 0us/step
```

In [ ]:

```
vgg16_model.summary()
```

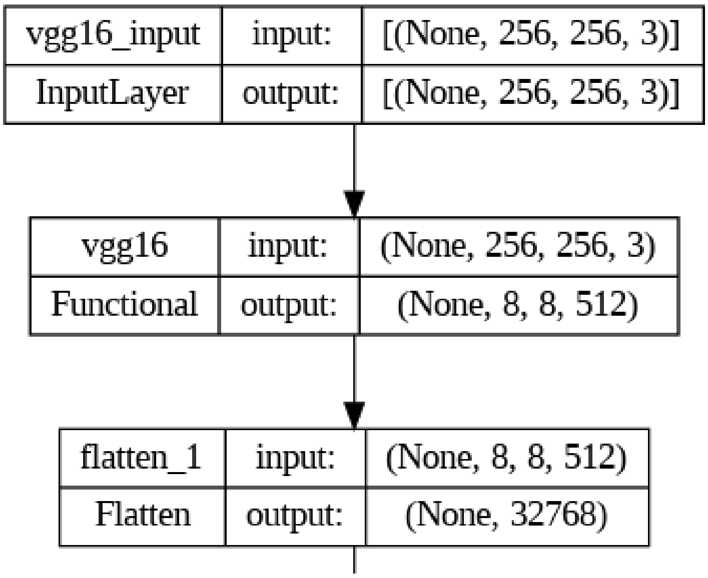
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688  
                                                                 
 flatten_1 (Flatten)         (None, 32768)             0         
                                                                 
 dense_8 (Dense)             (None, 512)               16777728  
                                                                 
 activation_13 (Activation)  (None, 512)               0         
                                                                 
 batch_normalization_13 (Bat  (None, 512)              2048      
 chNormalization)                                                
                                                                 
 dropout_7 (Dropout)         (None, 512)               0         
                                                                 
 dense_9 (Dense)             (None, 256)               131328    
                                                                 
 activation_14 (Activation)  (None, 256)               0         
                                                                 
 batch_normalization_14 (Bat  (None, 256)              1024      
 chNormalization)                                                
                                                                 
 dropout_8 (Dropout)         (None, 256)               0         
                                                                 
 dense_10 (Dense)            (None, 128)               32896     
                                                                 
 activation_15 (Activation)  (None, 128)               0         
                                                                 
 batch_normalization_15 (Bat  (None, 128)              512       
 chNormalization)                                                
                                                                 
 dropout_9 (Dropout)         (None, 128)               0         
                                                                 
 dense_11 (Dense)            (None, 64)                8256      
                                                                 
 activation_16 (Activation)  (None, 64)                0         
                                                                 
 batch_normalization_16 (Bat  (None, 64)               256       
 chNormalization)                                                
                                                                 
 dropout_10 (Dropout)        (None, 64)                0         
                                                                 
 dense_12 (Dense)            (None, 32)                2080      
                                                                 
 activation_17 (Activation)  (None, 32)                0         
                                                                 
 batch_normalization_17 (Bat  (None, 32)               128       
 chNormalization)                                                
                                                                 
 dropout_11 (Dropout)        (None, 32)                0         
                                                                 
 dense_13 (Dense)            (None, 16)                528       
                                                                 
 activation_18 (Activation)  (None, 16)                0         
                                                                 
 batch_normalization_18 (Bat  (None, 16)               64        
 chNormalization)                                                
                                                                 
 dropout_12 (Dropout)        (None, 16)                0         
                                                                 
 dense_14 (Dense)            (None, 8)                 136       
```

```
 activation_19 (Activation)  (None, 8)                 0

 batch_normalization_19 (Bat  (None, 8)                32
 chNormalization)

 dropout_13 (Dropout)        (None, 8)                 0

 dense_15 (Dense)            (None, 4)                 36

=================================================================
Total params: 31,671,740
Trainable params: 16,955,020
Non-trainable params: 14,716,720
_________________________________________________________________
```

In [ ]:

```
tf.keras.utils.plot_model(vgg16_model, show_shapes=True)
```

Out[41]:



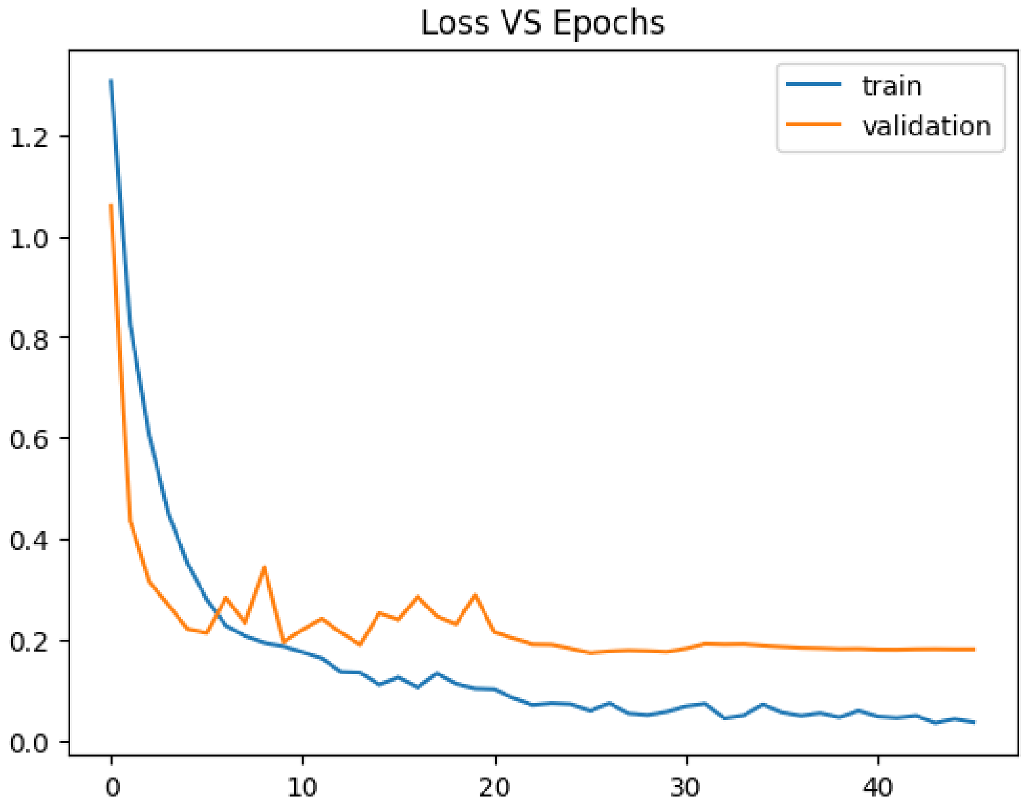
In [ ]:

```
model_fit2 = compile_train_v1(vgg16_model, train_ds, val_ds,tb_callback,ckpt_path="/tmp/checkpo
```

```
Epoch 1/100
100/100 [==============================] - 50s 373ms/step - loss: 1.3063 - acc
uracy: 0.4399 - val_loss: 1.0592 - val_accuracy: 0.5537 - lr: 0.0010
Epoch 2/100
100/100 [==============================] - 32s 315ms/step - loss: 0.8256 - acc
uracy: 0.7021 - val_loss: 0.4367 - val_accuracy: 0.8512 - lr: 0.0010
Epoch 3/100
100/100 [==============================] - 31s 310ms/step - loss: 0.6044 - acc
uracy: 0.7972 - val_loss: 0.3156 - val_accuracy: 0.8975 - lr: 0.0010
Epoch 4/100
100/100 [==============================] - 32s 320ms/step - loss: 0.4501 - acc
uracy: 0.8570 - val_loss: 0.2691 - val_accuracy: 0.9112 - lr: 0.0010
Epoch 5/100
100/100 [==============================] - 31s 307ms/step - loss: 0.3513 - acc
uracy: 0.8936 - val_loss: 0.2220 - val_accuracy: 0.9275 - lr: 0.0010
Epoch 6/100
100/100 [==============================] - 32s 315ms/step - loss: 0.2802 - acc
uracy: 0.9221 - val_loss: 0.2142 - val_accuracy: 0.9300 - lr: 0.0010
Epoch 7/100
```

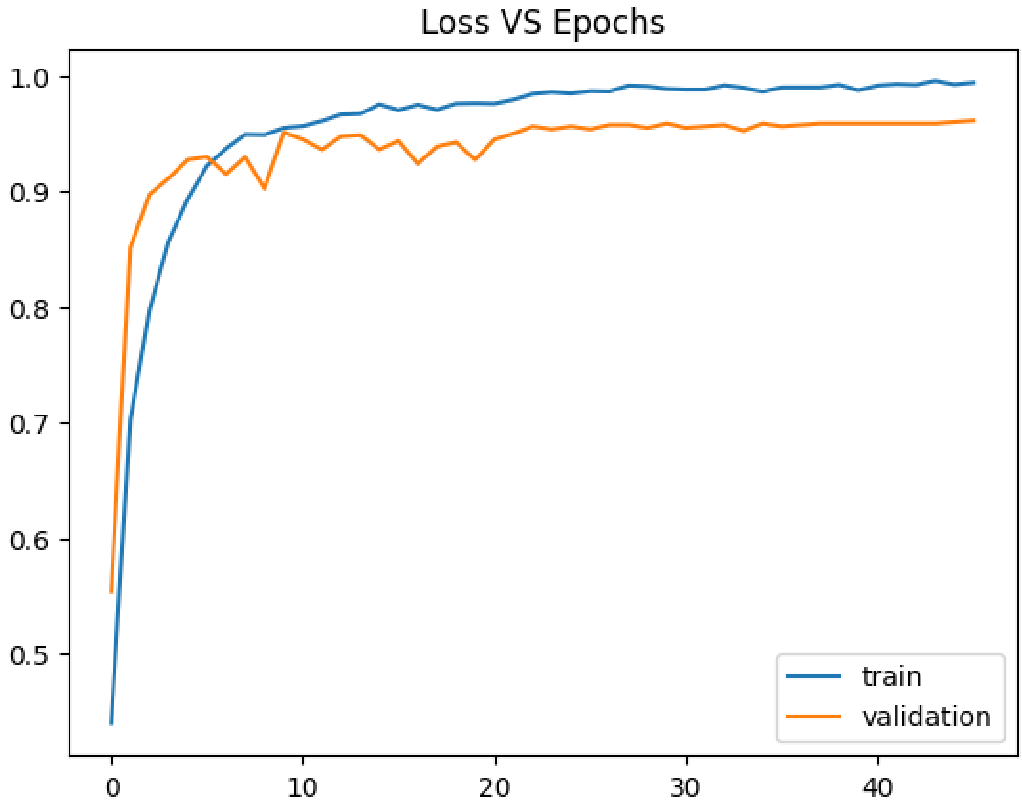
In [ ]:

```
epochs = model_fit2.epoch
loss = model_fit2.history["loss"]
val_loss = model_fit2.history["val_loss"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

```
epochs = model_fit2.epoch
loss = model_fit2.history["accuracy"]
val_loss = model_fit2.history["val_accuracy"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

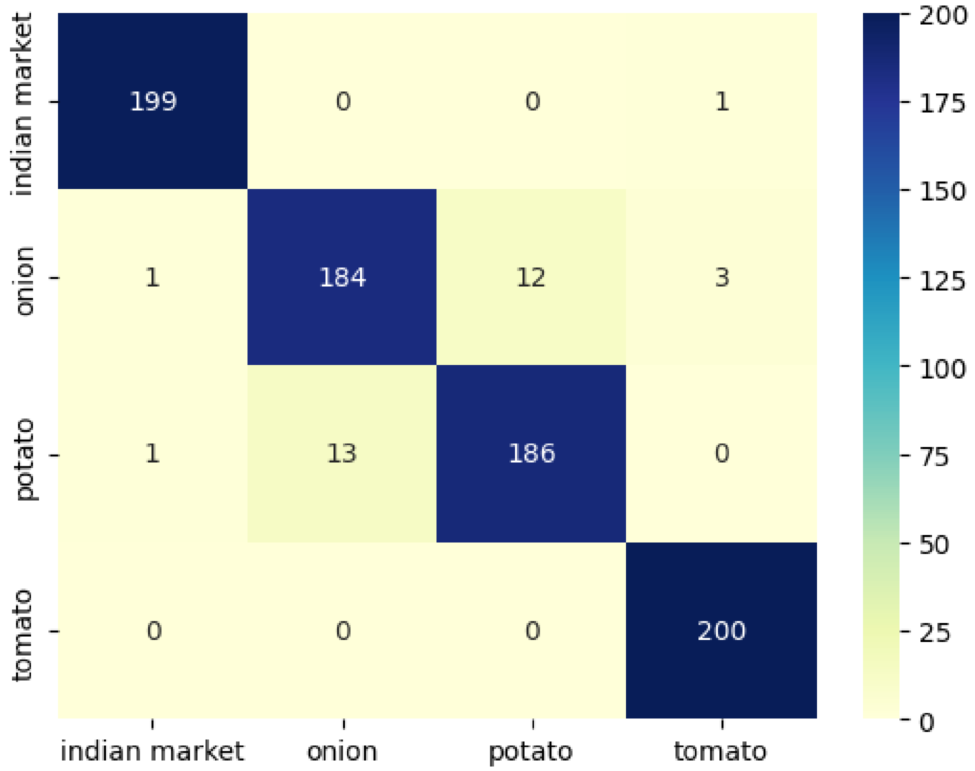
```
print_accuracy_stats(vgg16_model, test_ds, class_names,"/tmp/checkpoint2")
```

```
11/11 [==============================] - 7s 708ms/step

 Accuracy: 87.18%
```

In [ ]:

```
plot_confusion_matrix(vgg16_model, val_ds, class_names,"/tmp/checkpoint2")
```

```
25/25 [==============================] - 4s 181ms/step
```



In [ ]:

```
print_accuracy_stats(vgg16_model, val_ds, class_names,"/tmp/checkpoint2")
```

```
25/25 [==============================] - 4s 178ms/step

 Accuracy: 96.12%
```

In [ ]:

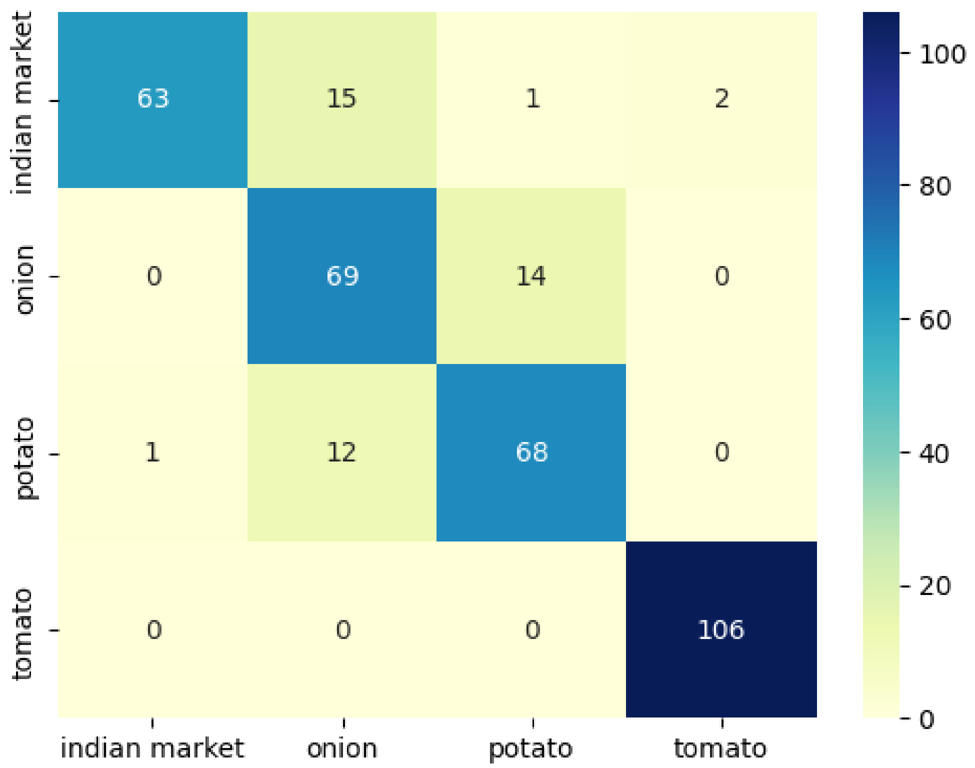
```
print_accuracy_stats(vgg16_model, test_ds, class_names,"/tmp/checkpoint2")
```

```
11/11 [==============================] - 2s 171ms/step

 Accuracy: 87.18%
```

In [ ]:

```
plot_confusion_matrix(vgg16_model, test_ds, class_names,"/tmp/checkpoint2")
```

```
11/11 [==============================] - 2s 233ms/step
```



In [ ]:

```
#%tensorboard --logdir logs
```

ResNet

In [ ]:

```
#%load_ext tensorboard
log_folder='logs/3'
#%reload_ext tensorboard
#!rm -rf logs
tb_callback=TensorBoard(log_dir=log_folder,histogram_freq=1)
```

In [ ]:

```
pretrained_model = tf.keras.applications.resnet_v2.ResNet152V2(weights='imagenet', include_top=
pretrained_model.trainable=False
resnet = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    #tf.keras.layers.Dense(10, activation='softmax')





            layers.Dense(units=128, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=64, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=32, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=16, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=8, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.1),

            layers.Dense(units=4, activation='softmax')
])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storag
e.googleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_
ordering_tf_kernels_notop.h5)
234545216/234545216 [==============================] - 1s 0us/step
```

In [ ]:

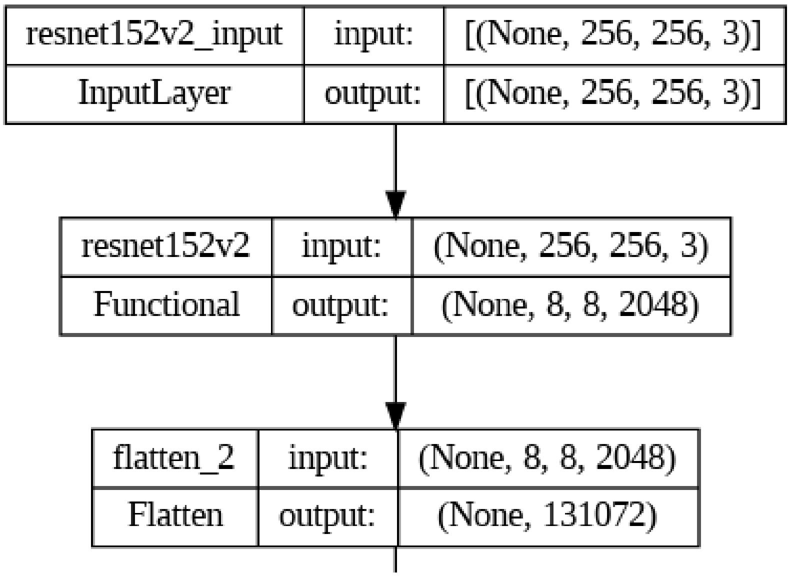
```
resnet.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet152v2 (Functional)    (None, 8, 8, 2048)        58331648  
                                                                 
 flatten_2 (Flatten)         (None, 131072)            0         
                                                                 
 dense_16 (Dense)            (None, 128)               16777344  
                                                                 
 activation_20 (Activation)  (None, 128)               0         
                                                                 
 batch_normalization_20 (Bat  (None, 128)              512       
 chNormalization)                                                
                                                                 
 dropout_14 (Dropout)        (None, 128)               0         
                                                                 
 dense_17 (Dense)            (None, 64)                8256      
                                                                 
 activation_21 (Activation)  (None, 64)                0         
                                                                 
 batch_normalization_21 (Bat  (None, 64)               256       
 chNormalization)                                                
                                                                 
 dropout_15 (Dropout)        (None, 64)                0         
                                                                 
 dense_18 (Dense)            (None, 32)                2080      
                                                                 
 activation_22 (Activation)  (None, 32)                0         
                                                                 
 batch_normalization_22 (Bat  (None, 32)               128       
 chNormalization)                                                
                                                                 
 dropout_16 (Dropout)        (None, 32)                0         
                                                                 
 dense_19 (Dense)            (None, 16)                528       
                                                                 
 activation_23 (Activation)  (None, 16)                0         
                                                                 
 batch_normalization_23 (Bat  (None, 16)               64        
 chNormalization)                                                
                                                                 
 dropout_17 (Dropout)        (None, 16)                0         
                                                                 
 dense_20 (Dense)            (None, 8)                 136       
                                                                 
 activation_24 (Activation)  (None, 8)                 0         
                                                                 
 batch_normalization_24 (Bat  (None, 8)                32        
 chNormalization)                                                
                                                                 
 dropout_18 (Dropout)        (None, 8)                 0         
                                                                 
 dense_21 (Dense)            (None, 4)                 36        
                                                                 
=================================================================
Total params: 75,121,020
Trainable params: 16,788,876
Non-trainable params: 58,332,144
_________________________________________________________________
```

In [ ]:

```
tf.keras.utils.plot_model(resnet, show_shapes=True)
```

Out[54]:



In [ ]:

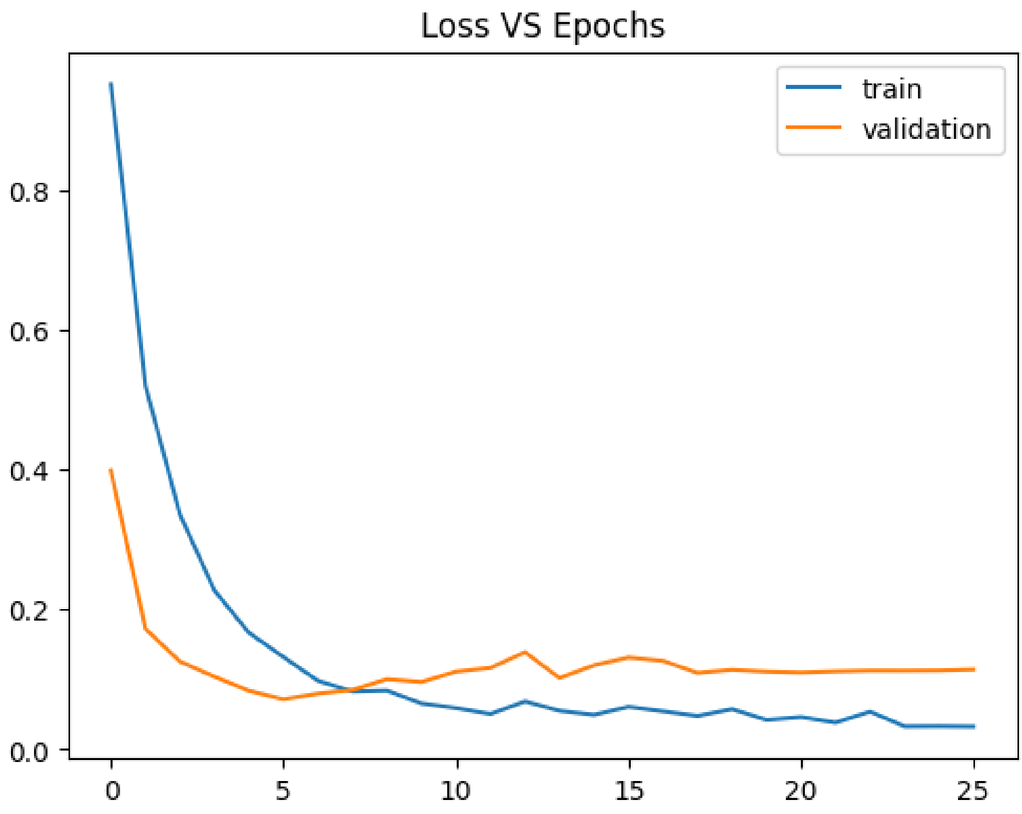
```
model_fit3 = compile_train_v1(resnet, train_ds, val_ds,tb_callback,ckpt_path="/tmp/checkpoint3"
```

```
Epoch 1/100
100/100 [==============================] - 78s 615ms/step - loss: 0.9522 - accura
cy: 0.6089 - val_loss: 0.3988 - val_accuracy: 0.9038 - lr: 0.0010
Epoch 2/100
100/100 [==============================] - 60s 598ms/step - loss: 0.5206 - accura
cy: 0.8326 - val_loss: 0.1724 - val_accuracy: 0.9550 - lr: 0.0010
Epoch 3/100
100/100 [==============================] - 58s 571ms/step - loss: 0.3361 - accura
cy: 0.9127 - val_loss: 0.1256 - val_accuracy: 0.9675 - lr: 0.0010
Epoch 4/100
100/100 [==============================] - 62s 615ms/step - loss: 0.2272 - accura
cy: 0.9452 - val_loss: 0.1038 - val_accuracy: 0.9725 - lr: 0.0010
Epoch 5/100
100/100 [==============================] - 59s 586ms/step - loss: 0.1667 - accura
cy: 0.9643 - val_loss: 0.0837 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 6/100
100/100 [==============================] - 56s 557ms/step - loss: 0.1320 - accura
cy: 0.9712 - val_loss: 0.0715 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 7/100
100/100 [==============================] - 55s 547ms/step - loss: 0.0979 - accura
cy: 0.9778 - val_loss: 0.0797 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 8/100
100/100 [==============================] - 57s 567ms/step - loss: 0.0827 - accura
cy: 0.9853 - val_loss: 0.0850 - val_accuracy: 0.9787 - lr: 0.0010
Epoch 9/100
100/100 [==============================] - 54s 535ms/step - loss: 0.0839 - accura
cy: 0.9812 - val_loss: 0.1002 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 10/100
100/100 [==============================] - 58s 575ms/step - loss: 0.0655 - accura
cy: 0.9872 - val_loss: 0.0962 - val_accuracy: 0.9737 - lr: 0.0010
Epoch 11/100
100/100 [==============================] - 58s 573ms/step - loss: 0.0589 - accura
cy: 0.9897 - val_loss: 0.1111 - val_accuracy: 0.9725 - lr: 0.0010
Epoch 12/100
100/100 [==============================] - 54s 539ms/step - loss: 0.0504 - accura
cy: 0.9909 - val_loss: 0.1165 - val_accuracy: 0.9725 - lr: 0.0010
Epoch 13/100
100/100 [==============================] - 54s 538ms/step - loss: 0.0683 - accura
cy: 0.9844 - val_loss: 0.1389 - val_accuracy: 0.9700 - lr: 0.0010
Epoch 14/100
100/100 [==============================] - 54s 535ms/step - loss: 0.0550 - accura
cy: 0.9900 - val_loss: 0.1019 - val_accuracy: 0.9775 - lr: 0.0010
Epoch 15/100
100/100 [==============================] - 58s 575ms/step - loss: 0.0492 - accura
cy: 0.9869 - val_loss: 0.1202 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 16/100
100/100 [==============================] - 58s 573ms/step - loss: 0.0607 - accura
cy: 0.9875 - val_loss: 0.1313 - val_accuracy: 0.9737 - lr: 0.0010
Epoch 17/100
100/100 [==============================] - 58s 577ms/step - loss: 0.0542 - accura
cy: 0.9878 - val_loss: 0.1264 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 18/100
100/100 [==============================] - 57s 562ms/step - loss: 0.0476 - accura
cy: 0.9903 - val_loss: 0.1093 - val_accuracy: 0.9775 - lr: 0.0010
Epoch 19/100
100/100 [==============================] - 55s 551ms/step - loss: 0.0573 - accura
cy: 0.9865 - val_loss: 0.1137 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 20/100
100/100 [==============================] - 54s 539ms/step - loss: 0.0420 - accura
cy: 0.9922 - val_loss: 0.1111 - val_accuracy: 0.9787 - lr: 0.0010
Epoch 21/100
100/100 [==============================] - 55s 542ms/step - loss: 0.0460 - accura
cy: 0.9903 - val_loss: 0.1098 - val_accuracy: 0.9787 - lr: 1.0000e-04
Epoch 22/100
```

```
100/100 [==============================] - 54s 539ms/step - loss: 0.0383 - accura
cy: 0.9916 - val_loss: 0.1115 - val_accuracy: 0.9787 - lr: 1.0000e-04
Epoch 23/100
100/100 [==============================] - 61s 603ms/step - loss: 0.0536 - accura
cy: 0.9903 - val_loss: 0.1126 - val_accuracy: 0.9800 - lr: 1.0000e-04
Epoch 24/100
100/100 [==============================] - 54s 538ms/step - loss: 0.0329 - accura
cy: 0.9944 - val_loss: 0.1125 - val_accuracy: 0.9800 - lr: 1.0000e-04
Epoch 25/100
100/100 [==============================] - 58s 578ms/step - loss: 0.0331 - accura
cy: 0.9931 - val_loss: 0.1128 - val_accuracy: 0.9800 - lr: 1.0000e-04
Epoch 26/100
100/100 [==============================] - 54s 536ms/step - loss: 0.0326 - accura
cy: 0.9947 - val_loss: 0.1141 - val_accuracy: 0.9800 - lr: 1.0000e-04
```

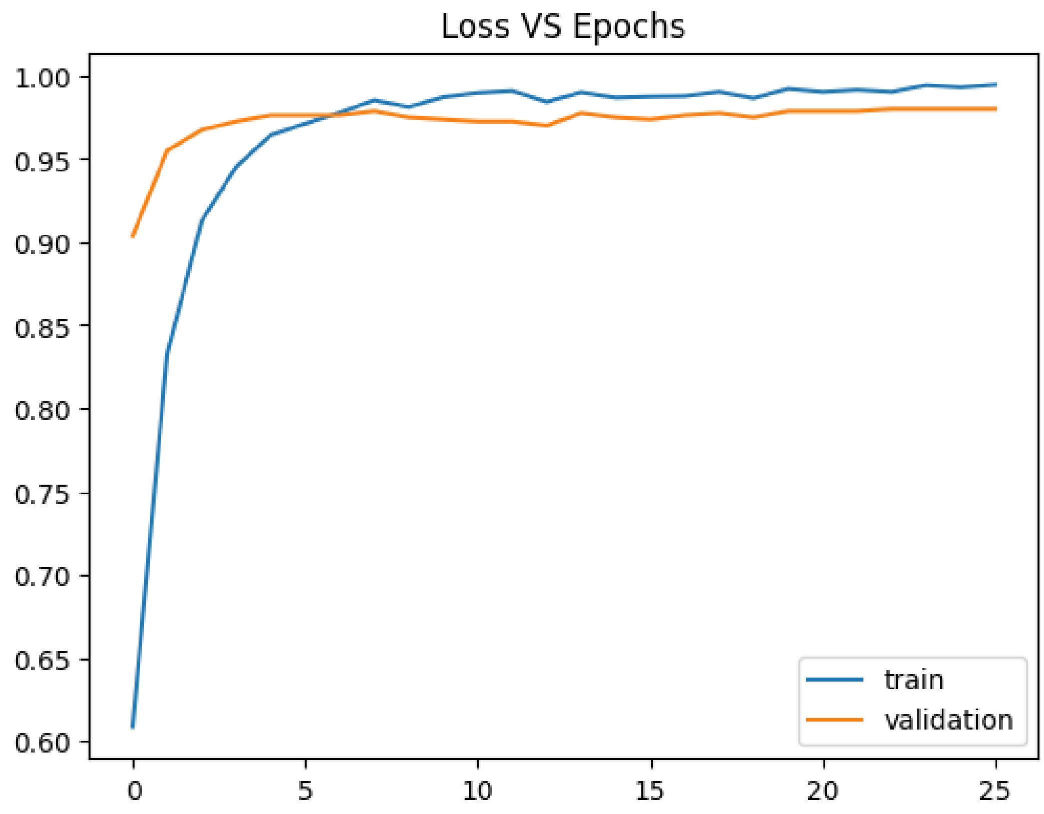
In [ ]:

```
epochs = model_fit3.epoch
loss = model_fit3.history["loss"]
val_loss = model_fit3.history["val_loss"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

```
epochs = model_fit3.epoch
loss = model_fit3.history["accuracy"]
val_loss = model_fit3.history["val_accuracy"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

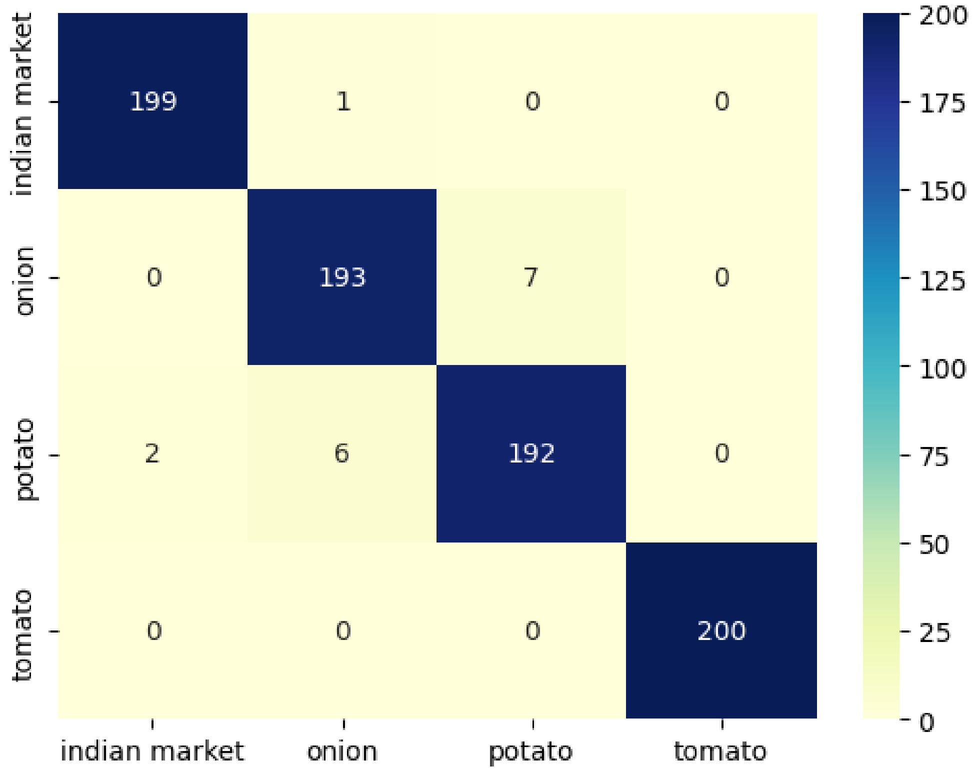
```
print_accuracy_stats(resnet, test_ds, class_names,"/tmp/checkpoint3")
```

```
11/11 [==============================] - 7s 446ms/step

 Accuracy: 90.88%
```

In [ ]:

```
plot_confusion_matrix(resnet, val_ds, class_names,"/tmp/checkpoint3")
```

```
25/25 [==============================] - 7s 281ms/step
```



In [ ]:

```
print_accuracy_stats(resnet, val_ds, class_names,"/tmp/checkpoint3")
```

```
25/25 [==============================] - 7s 275ms/step
 Accuracy: 98.00%
```

In [ ]:

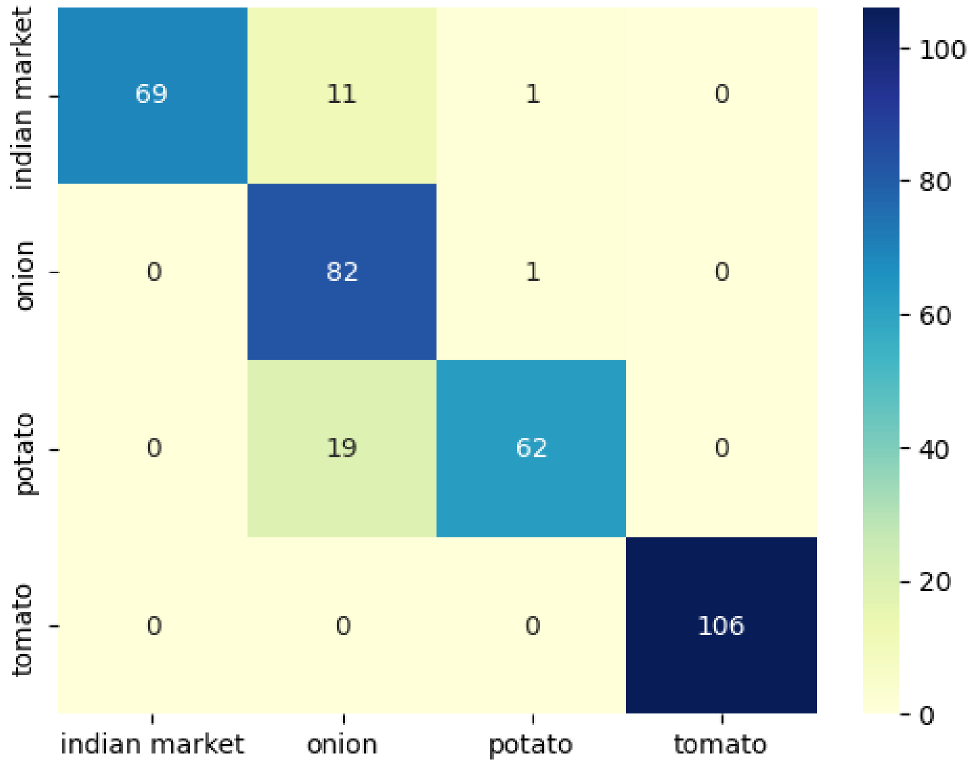
```
print_accuracy_stats(resnet, test_ds, class_names,"/tmp/checkpoint3")
```

```
11/11 [==============================] - 3s 271ms/step
 Accuracy: 90.88%
```

In [ ]:

```
plot_confusion_matrix(resnet, test_ds, class_names,"/tmp/checkpoint3")
```

```
11/11 [==============================] - 3s 276ms/step
```



In [ ]:

```
#%tensorboard --logdir logs
```

In [ ]:

```

```

Mobilenet

In [ ]:

```
#%load_ext tensorboard
log_folder='logs/4'
#%reload_ext tensorboard
#!rm -rf logs
tb_callback=TensorBoard(log_dir=log_folder,histogram_freq=1)
```

In [ ]:

```
pretrained_model = tf.keras.applications.mobilenet_v2.MobileNetV2(weights='imagenet', include_t
pretrained_model.trainable=False
mobilenet = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    #tf.keras.layers.Dense(10, activation='softmax')



            layers.Dense(units=128, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=64, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(units=32, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=16, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.2),

            layers.Dense(units=8, kernel_regularizer = L2Reg),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.1),

            layers.Dense(units=4, activation='softmax')
])
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in
[96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as th
e default.

Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
(https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobile
net_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5)
9406464/9406464 [==============================] - 0s 0us/step
```

In [ ]:

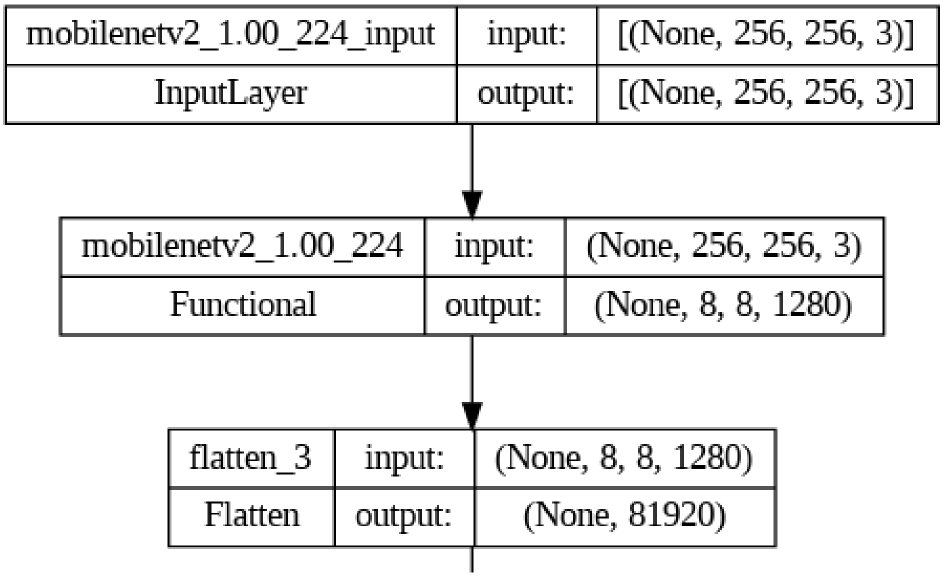
```
mobilenet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984   
 ional)                                                          
                                                                 
 flatten_3 (Flatten)         (None, 81920)             0         
                                                                 
 dense_22 (Dense)            (None, 128)               10485888  
                                                                 
 activation_25 (Activation)  (None, 128)               0         
                                                                 
 batch_normalization_25 (Bat  (None, 128)              512       
 chNormalization)                                                
                                                                 
 dropout_19 (Dropout)        (None, 128)               0         
                                                                 
 dense_23 (Dense)            (None, 64)                8256      
                                                                 
 activation_26 (Activation)  (None, 64)                0         
                                                                 
 batch_normalization_26 (Bat  (None, 64)               256       
 chNormalization)                                                
                                                                 
 dropout_20 (Dropout)        (None, 64)                0         
                                                                 
 dense_24 (Dense)            (None, 32)                2080      
                                                                 
 activation_27 (Activation)  (None, 32)                0         
                                                                 
 batch_normalization_27 (Bat  (None, 32)               128       
 chNormalization)                                                
                                                                 
 dropout_21 (Dropout)        (None, 32)                0         
                                                                 
 dense_25 (Dense)            (None, 16)                528       
                                                                 
 activation_28 (Activation)  (None, 16)                0         
                                                                 
 batch_normalization_28 (Bat  (None, 16)               64        
 chNormalization)                                                
                                                                 
 dropout_22 (Dropout)        (None, 16)                0         
                                                                 
 dense_26 (Dense)            (None, 8)                 136       
                                                                 
 activation_29 (Activation)  (None, 8)                 0         
                                                                 
 batch_normalization_29 (Bat  (None, 8)                32        
 chNormalization)                                                
                                                                 
 dropout_23 (Dropout)        (None, 8)                 0         
                                                                 
 dense_27 (Dense)            (None, 4)                 36        
                                                                 
=================================================================
Total params: 12,755,900
Trainable params: 10,497,420
Non-trainable params: 2,258,480
_________________________________________________________________
```

In [ ]:

```
tf.keras.utils.plot_model(mobilenet, show_shapes=True)
```

Out[67]:



In [ ]:

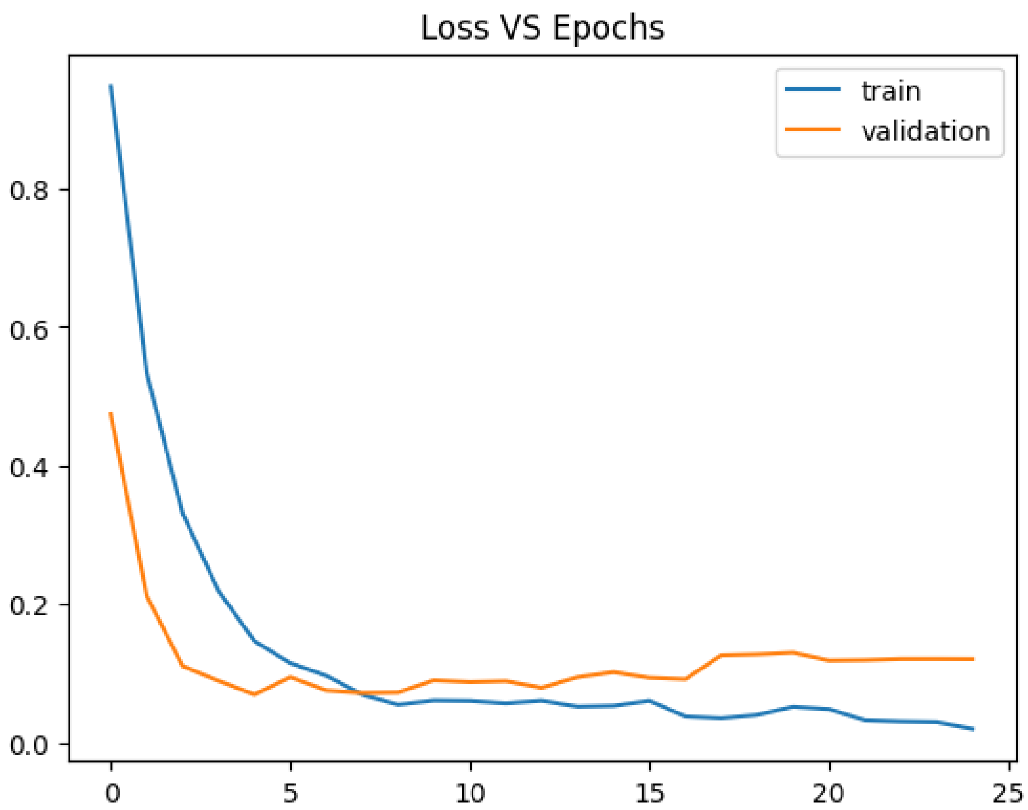
```
model_fit4 = compile_train_v1(mobilenet, train_ds, val_ds,tb_callback,ckpt_path="/tmp/checkpoin
```

```
Epoch 1/100
100/100 [==============================] - 27s 193ms/step - loss: 0.9483 - accura
cy: 0.6133 - val_loss: 0.4741 - val_accuracy: 0.8925 - lr: 0.0010
Epoch 2/100
100/100 [==============================] - 18s 172ms/step - loss: 0.5342 - accura
cy: 0.8376 - val_loss: 0.2118 - val_accuracy: 0.9563 - lr: 0.0010
Epoch 3/100
100/100 [==============================] - 18s 179ms/step - loss: 0.3310 - accura
cy: 0.9124 - val_loss: 0.1104 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 4/100
100/100 [==============================] - 18s 171ms/step - loss: 0.2188 - accura
cy: 0.9462 - val_loss: 0.0891 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 5/100
100/100 [==============================] - 18s 175ms/step - loss: 0.1464 - accura
cy: 0.9725 - val_loss: 0.0692 - val_accuracy: 0.9825 - lr: 0.0010
Epoch 6/100
100/100 [==============================] - 17s 165ms/step - loss: 0.1146 - accura
cy: 0.9759 - val_loss: 0.0944 - val_accuracy: 0.9725 - lr: 0.0010
Epoch 7/100
100/100 [==============================] - 18s 168ms/step - loss: 0.0966 - accura
cy: 0.9812 - val_loss: 0.0752 - val_accuracy: 0.9775 - lr: 0.0010
Epoch 8/100
100/100 [==============================] - 18s 171ms/step - loss: 0.0692 - accura
cy: 0.9890 - val_loss: 0.0716 - val_accuracy: 0.9812 - lr: 0.0010
Epoch 9/100
100/100 [==============================] - 17s 165ms/step - loss: 0.0546 - accura
cy: 0.9890 - val_loss: 0.0723 - val_accuracy: 0.9825 - lr: 0.0010
Epoch 10/100
100/100 [==============================] - 18s 174ms/step - loss: 0.0608 - accura
cy: 0.9850 - val_loss: 0.0900 - val_accuracy: 0.9800 - lr: 0.0010
Epoch 11/100
100/100 [==============================] - 17s 169ms/step - loss: 0.0602 - accura
cy: 0.9862 - val_loss: 0.0874 - val_accuracy: 0.9800 - lr: 0.0010
Epoch 12/100
100/100 [==============================] - 18s 167ms/step - loss: 0.0566 - accura
cy: 0.9865 - val_loss: 0.0887 - val_accuracy: 0.9825 - lr: 0.0010
Epoch 13/100
100/100 [==============================] - 17s 166ms/step - loss: 0.0605 - accura
cy: 0.9847 - val_loss: 0.0790 - val_accuracy: 0.9812 - lr: 0.0010
Epoch 14/100
100/100 [==============================] - 18s 175ms/step - loss: 0.0518 - accura
cy: 0.9919 - val_loss: 0.0946 - val_accuracy: 0.9787 - lr: 0.0010
Epoch 15/100
100/100 [==============================] - 17s 165ms/step - loss: 0.0529 - accura
cy: 0.9875 - val_loss: 0.1018 - val_accuracy: 0.9775 - lr: 0.0010
Epoch 16/100
100/100 [==============================] - 18s 169ms/step - loss: 0.0603 - accura
cy: 0.9894 - val_loss: 0.0937 - val_accuracy: 0.9812 - lr: 0.0010
Epoch 17/100
100/100 [==============================] - 17s 169ms/step - loss: 0.0376 - accura
cy: 0.9931 - val_loss: 0.0915 - val_accuracy: 0.9787 - lr: 0.0010
Epoch 18/100
100/100 [==============================] - 19s 184ms/step - loss: 0.0350 - accura
cy: 0.9950 - val_loss: 0.1259 - val_accuracy: 0.9775 - lr: 0.0010
Epoch 19/100
100/100 [==============================] - 17s 167ms/step - loss: 0.0397 - accura
cy: 0.9916 - val_loss: 0.1273 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 20/100
100/100 [==============================] - 17s 165ms/step - loss: 0.0516 - accura
cy: 0.9887 - val_loss: 0.1298 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 21/100
100/100 [==============================] - 18s 170ms/step - loss: 0.0481 - accura
cy: 0.9900 - val_loss: 0.1184 - val_accuracy: 0.9787 - lr: 1.0000e-04
Epoch 22/100
```

```
100/100 [==============================] - 17s 169ms/step - loss: 0.0321 - accura
cy: 0.9937 - val_loss: 0.1191 - val_accuracy: 0.9775 - lr: 1.0000e-04
Epoch 23/100
100/100 [==============================] - 17s 163ms/step - loss: 0.0303 - accura
cy: 0.9937 - val_loss: 0.1208 - val_accuracy: 0.9775 - lr: 1.0000e-04
Epoch 24/100
100/100 [==============================] - 18s 175ms/step - loss: 0.0295 - accura
cy: 0.9947 - val_loss: 0.1208 - val_accuracy: 0.9762 - lr: 1.0000e-04
Epoch 25/100
100/100 [==============================] - 18s 171ms/step - loss: 0.0196 - accura
cy: 0.9975 - val_loss: 0.1206 - val_accuracy: 0.9762 - lr: 1.0000e-04
```

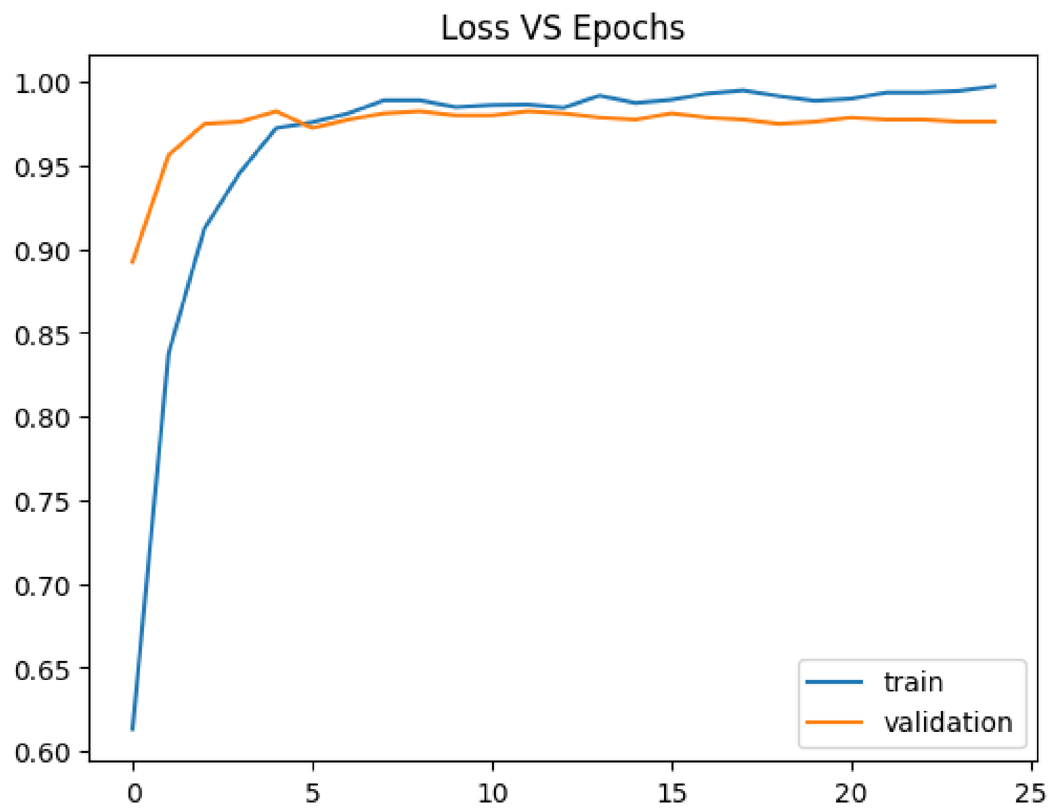
In [ ]:

```
epochs = model_fit4.epoch
loss = model_fit4.history["loss"]
val_loss = model_fit4.history["val_loss"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

```
epochs = model_fit4.epoch
loss = model_fit4.history["accuracy"]
val_loss = model_fit4.history["val_accuracy"]
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```



In [ ]:

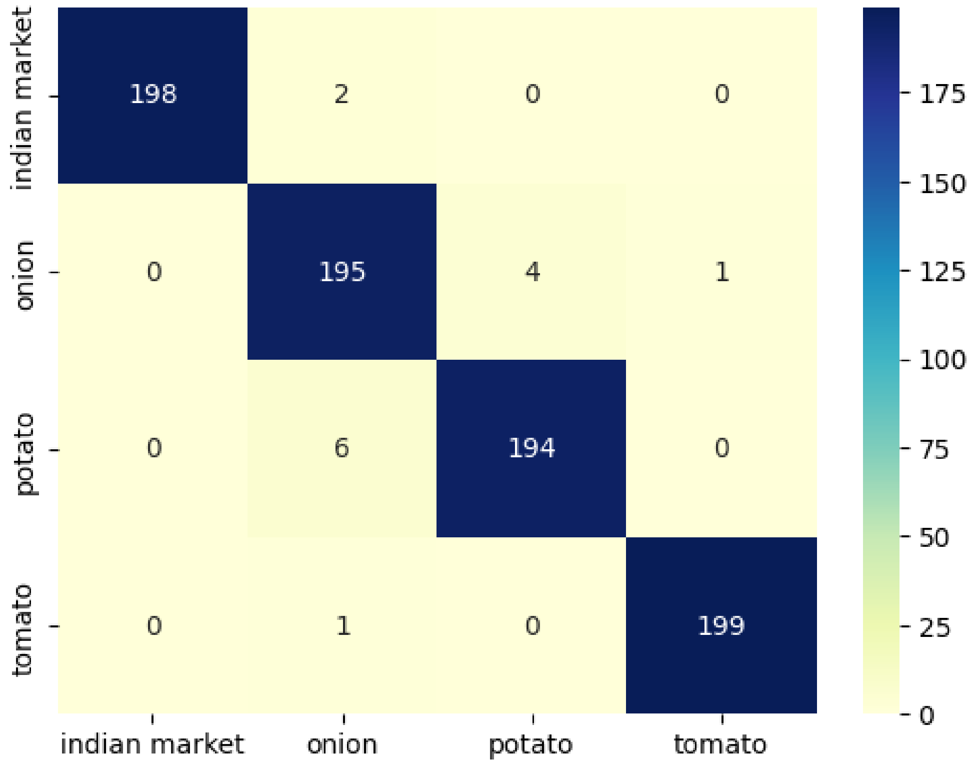
```
print_accuracy_stats(mobilenet, test_ds, class_names,"/tmp/checkpoint4")
```

```
11/11 [==============================] - 4s 251ms/step

 Accuracy: 92.31%
```

In [ ]:

```
plot_confusion_matrix(mobilenet, val_ds, class_names,"/tmp/checkpoint4")
```

```
25/25 [==============================] - 2s 94ms/step
```



In [ ]:

```
print_accuracy_stats(mobilenet, val_ds, class_names,"/tmp/checkpoint4")
```

```
25/25 [==============================] - 3s 115ms/step

 Accuracy: 98.25%
```

In [ ]:

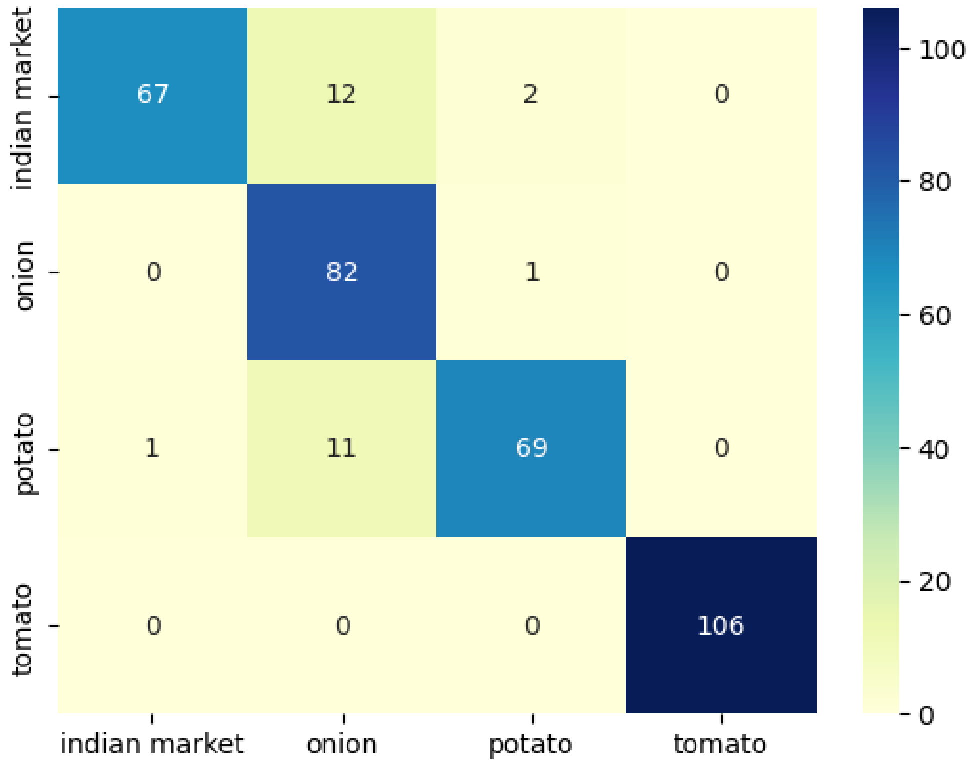
```
print_accuracy_stats(mobilenet, test_ds, class_names,"/tmp/checkpoint4")
```

```
11/11 [==============================] - 1s 111ms/step

 Accuracy: 92.31%
```

In [ ]:

```
plot_confusion_matrix(mobilenet, test_ds, class_names,"/tmp/checkpoint4")
```

```
11/11 [==============================] - 1s 114ms/step
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
%tensorboard --logdir logs
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

In [ ]:

In [ ]: