Ninjacart_cv_classification

July 12, 2023

1 Ninjacart: CV Classification

1.1 Problem Statement

Ninjacart is India's largest fresh produce supply chain company. They are pioneers in solving one of the toughest supply chain problems of the world by leveraging innovative technology. They source fresh produce from farmers and deliver them to businesses within 12 hours. An integral component of their automation process is the development of robust classifiers which can distinguish between images of different types of vegetables, while also correctly labeling images that do not contain any one type of vegetable as noise.

As a starting point, ninjacart has provided us with a dataset scraped from the web which contains train and test folders, each having 4 sub-folders with images of onions, potatoes, tomatoes and some market scenes. We have been tasked with preparing a multiclass classifier for identifying these vegetables. The dataset provided has all the required images to achieve the task.

DataSet

The dataset contains images of the following food items: noise-Indian market and images of vegetables- onion, potato and tomato.

Data Collection

The images in this dataset were scraped from Google.

Content

This dataset contains a folder train, which has a total of 3135 images, split into four folders as follows:

- Tomato : 789
- Potato : 898
- Onion : 849
- Indian market : 599

This dataset contains another folder test which has a total of 351 images, split into four folders

- Tomato : 106
- potato : 83
- onion : 81
- Indian market : 81

Inspiration

The objective is to develop a program that can recognize the vegetable item(s) in a photo and identify them for the user.

Concepts Tested:

- Dataset Preparation & Visualization
- CNN models
- Implementing Callbacks
- Deal with Overfitting
- Transfer Learning

1.2 Additional views

We will begin our solution with visual analysis of image counts and distribition of sizes (height, width, and aspect ratio) across different classes. This analysis will help us decide choose appropriate resize method (basic resize vs crop-and-resize). We will then define helper methods to load/save tensorflow models on google drive (to avoid GPU training whenever model is already saved), compile and train models, manage model checkpoint callbacks, plot confusion matrix and report various overall and classwise classification metrics. After that, we will begin creating a base CNN model from scratch based on Alexnet architecture. We choose Alexnet architecture for its simplicity, ease of training on limited hardware resources, and limited availibility of train data. Post that we will use pretrained VGG19, EffiecientnetV2B0, and resnet50 models for transfer learning. We will use techniques such as early stopping, batch normalizarion, regularization, drop outs to address overfitting. We will also use ModelCheckpoints to store intermediate checkpoints. In the last section, we will also use tensorboard to visalize various plots related to training/validaton loss and accuracy. Finally, we report test scores for different models and summarize the results.

1.3 Solution

1.3.1 Data Exploration

Load Data

[1]:
```
try:
  assert(firstload)
  firstload = False
except:
  firstload = True
```

[2]:
```
# Import dataset
if(firstload):
  !gdown https://drive.google.com/uc?id=1Y4fIRJdR47e8HNfQpnlUmVamEsRYoZ8q
```

```
Downloading…
From: https://drive.google.com/uc?id=1Y4fIRJdR47e8HNfQpnlUmVamEsRYoZ8q
To: /content/ninjacart_data.zip
100% 275M/275M [00:03<00:00, 90.4MB/s]
```

[3]:
```
# unzip dataset
import os
```

```
if(not os.path.isdir('/content/ninjacart_data')):
  !unzip -qq /content/ninjacart_data.zip
else:
  print('directory already exists')
```

[4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[5]:
```
# 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
sns.set_theme()

# Import tensorflow and its modules
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(47) # set random seed

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

[6]:
```
# global constants
NJ_DATA_PATH = '/content/ninjacart_data'
NJ_TRAIN_DATA_PATH = '/content/ninjacart_data/train'
NJ_TEST_DATA_PATH = '/content/ninjacart_data/test'

CHECKPOINT_PATH = './checkpoints_test'
GDRIVE_SAVEDMODEL_PATH = '/content/drive/MyDrive/Ninjacart_case_Study/
 ↪SavedModels'
```

[7]:
```
gpu_available = not not tf.config.list_physical_devices('GPU')
print(f'GPU available: {gpu_available}')
```

```
GPU available: True
```

[8]:
```
from google.colab import runtime

#function to disconnect this runtime. call at the end of the notebook.
def disconnect_runtime():
  runtime.unassign()
```

[9]:
```
from IPython.core.magic import register_cell_magic

@register_cell_magic('disconnect_on_error')
def disconnect_on_error(line, cell):
    try:
        exec(cell)
    except Exception as e:
        print('disconnecting the runtime..')
        disconnect_runtime()
```

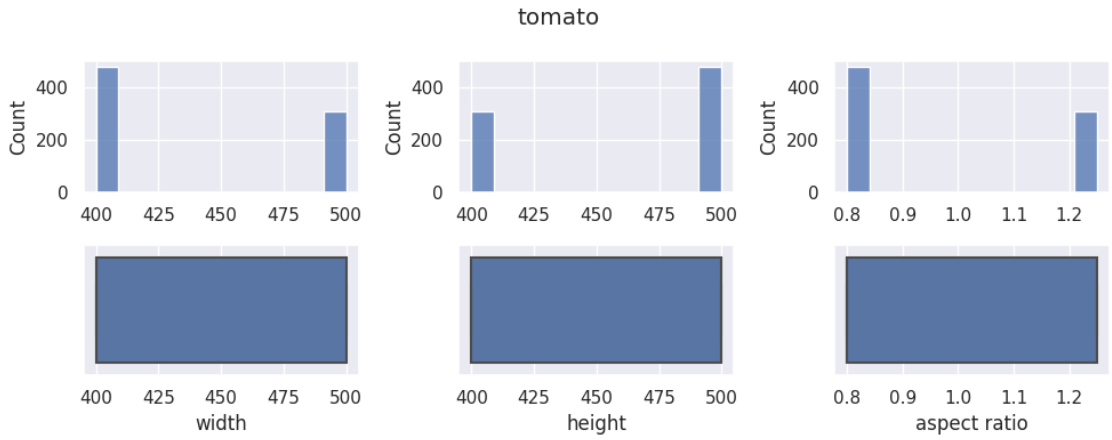
Visualize image size distribution

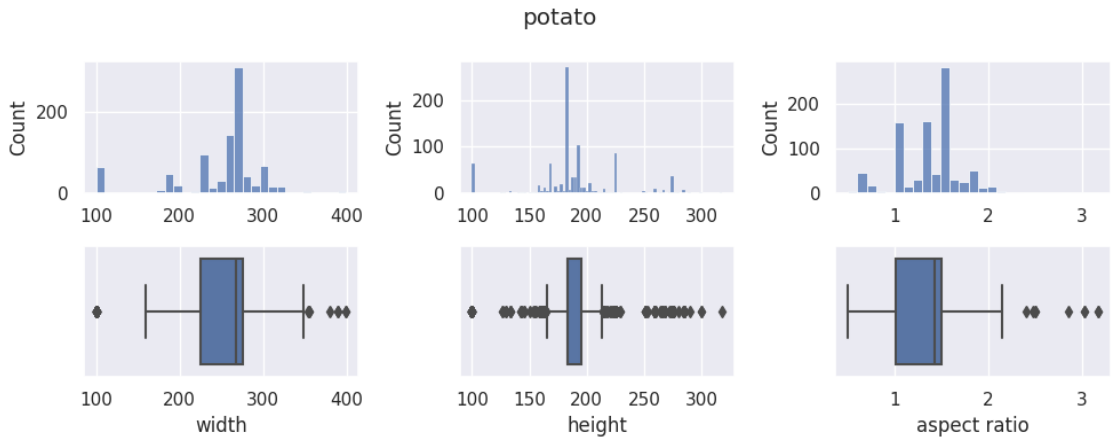
[66]:
```
# Check and plot distribution of image sizes under each class

import imagesize

data_info = {}
folders = ['train', 'test']

for folder in folders:

  data_info[folder] = {}
  class_dirs = os.listdir(f'{NJ_DATA_PATH}/{folder}')

  for cls in class_dirs:

      file_paths = glob.glob(f'{NJ_DATA_PATH}/{folder}/{cls}/*')

      data_info[folder][cls] = {
          'image_sizes': np.array([imagesize.get(path) for path in file_paths]),
          'sample_image': tf.keras.utils.load_img(random.choice(file_paths))
      }
```

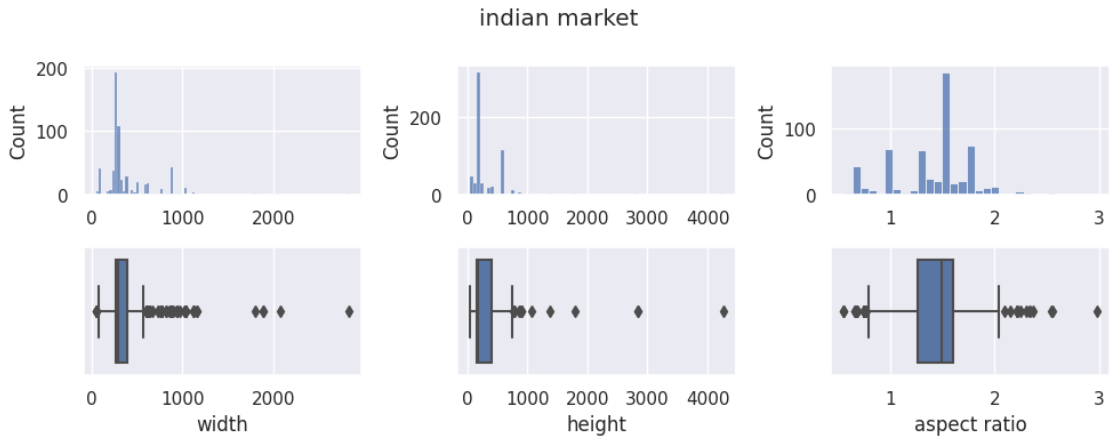
[67]:
```
#check count of images

ddf2 = []
for cls in data_info['train'].keys():
  ddf2.append([cls, len(data_info['train'][cls]['image_sizes']),␣
 ↪len(data_info['test'][cls]['image_sizes'])])

df = pd.DataFrame(ddf2, columns=['class', 'training_count', 'test_count'])
```

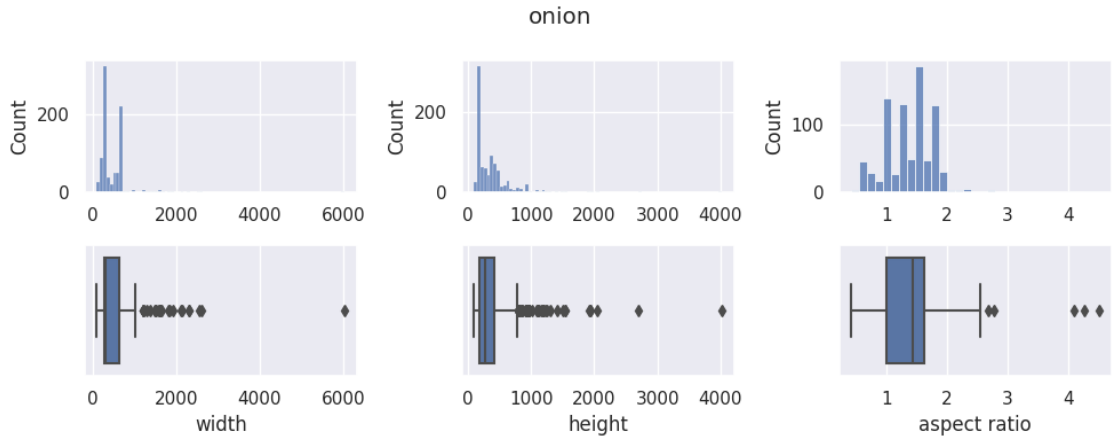
[68]:
```
# Visualize distribution of train image widths, heights, and aspect ratio


from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight:
 ↪5000})'''))

ddf = []
for cls in data_info['train'].keys():

  fig, axes = plt.subplots(2, 3, figsize=(10, 4))
  fig.suptitle(cls)

  widths = data_info['train'][cls]['image_sizes'][:, 0]
  heights = data_info['train'][cls]['image_sizes'][:, 1]
  aspect_ratio = np.round(widths / (heights + 0.01), 2)

  ddf.append([cls, 'width', np.min(widths), np.max(widths), np.mean(widths), np.
 ↪std(widths)])
  ddf.append([cls, 'height', np.min(heights), np.max(heights), np.
 ↪mean(heights), np.std(heights)])
  ddf.append([cls, 'aspect_ratio', np.min(aspect_ratio), np.max(aspect_ratio),
 ↪np.mean(aspect_ratio), np.std(aspect_ratio)])

  sns.histplot(x = widths, ax=axes[0][0])
  sns.boxplot(x = widths, ax=axes[1][0])
  axes[1][0].set_xlabel(f'width')

  sns.histplot(x = heights, ax=axes[0][1])
  sns.boxplot(x = heights, ax=axes[1][1])
  axes[1][1].set_xlabel(f'height')

  sns.histplot(x = aspect_ratio, ax=axes[0][2])
  sns.boxplot(x = aspect_ratio, ax=axes[1][2])
  axes[1][2].set_xlabel(f'aspect ratio')

  plt.tight_layout()
  plt.show()
```

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


tomato
Count
400
200
0
400 425 450 475 500
width
height
0.8 0.9 1.0 1.1 1.2
aspect ratio


potato
Count
200
100
0
100 200 300 400
100 150 200 250 300
1 2 3
width
height
aspect ratio


indian market
Count
200
100
0
0 1000 2000
0 1000 2000 3000 4000
1 2 3
width
height
aspect ratio



[69]:
```
# print image size stats
df = pd.DataFrame(ddf, columns=['class', 'attribute', 'min', 'max', 'mean',␣
 ↪'std'])
df.set_index(['class', 'attribute'])
```

[69]:

class	attribute	min	max	mean	std
tomato	width	400.00	500.00	439.290241	48.839544
	height	400.00	500.00	460.709759	48.839544
	aspect_ratio	0.80	1.25	0.976806	0.219778
potato	width	100.00	399.00	250.178174	52.945716
	height	100.00	318.00	188.674833	37.492197
	aspect_ratio	0.50	3.17	1.358541	0.344736
indian market	width	48.00	2832.00	391.983306	278.758839
	height	48.00	4256.00	304.696160	298.182185
	aspect_ratio	0.56	2.98	1.408347	0.368655
onion	width	100.00	6030.00	460.209658	346.999358
	height	100.00	4020.00	360.182568	293.242995
	aspect_ratio	0.42	4.51	1.383004	0.406394

Observations

1. Average resolution of the tomato images is highest, onion images is second highest, followed by indian market images. Average resolution of Potato images is smallest.
2. Overall variability in width and height of onion and indian market images is high, wherease, that in tomato and potato is low.
3. Average aspect ratio of potato, onion, and indian market images range between 1.35 to 1.4, which means, these images are generally wider (width > height). Average aspect ratio of tomato images, on the other hand, is 0.97, which means that tomato images are in general are closer to being square shaped.
4. In this case study, we will be using VGG19, EfficientNetV2B0, and resnet50 architectures for

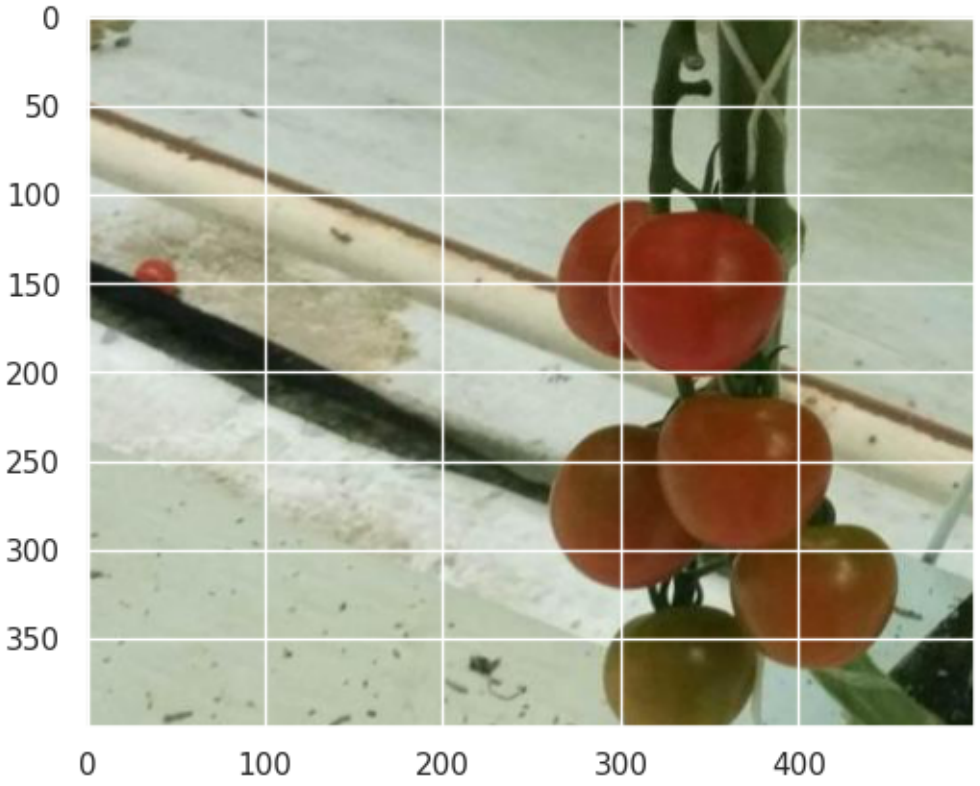
transfer learning. All these architectures take images in 224 x 224x 3 dimensions. So we need to convert input images to 224 x 224 x 3 dimension. There are two high-level approaches. First, we can use resizing layer, which simply resizes the input image to the target dimension. However it doesn't preserve aspect ratio. The second approach is to first crop the largest square window in the center of the image, and then resize it to required target dimension. Since our images vary considerably, we will use the second resize approach.

Plotting sample images

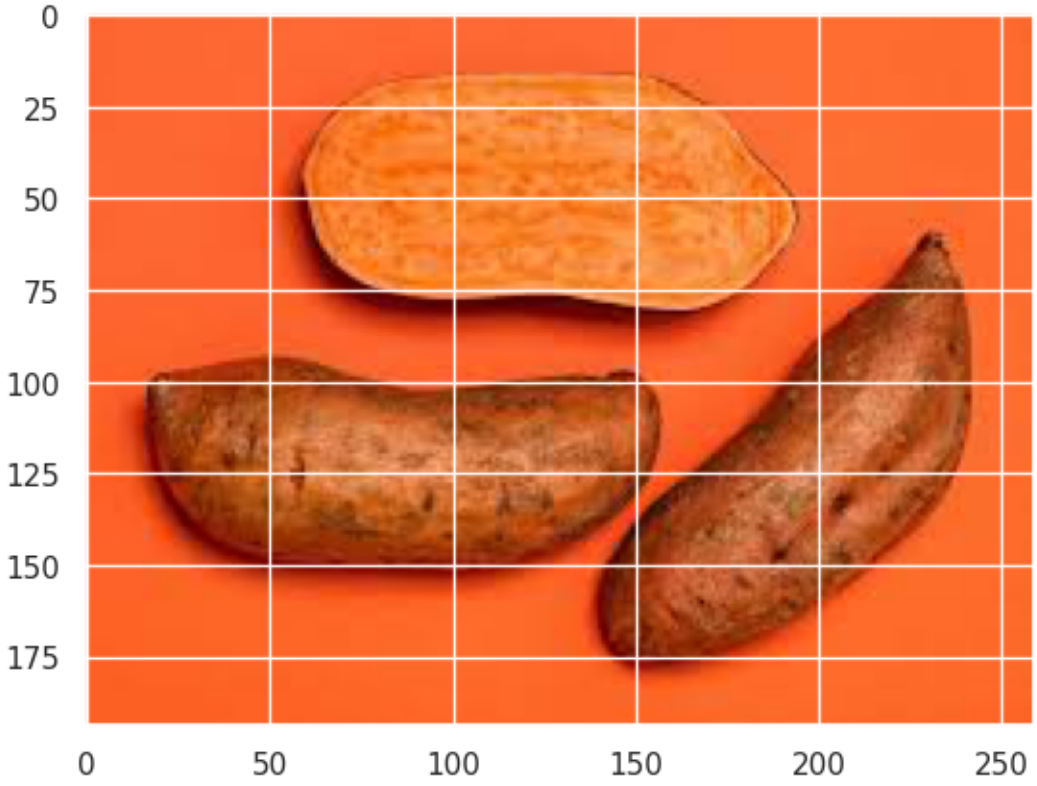
[14]:
```
# plot sample images for each class
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight:␣
 ↪5000})'''))

for cls in data_info['train'].keys():
  print(cls)
  plt.imshow(data_info['train'][cls]['sample_image'])
  plt.show()
```

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

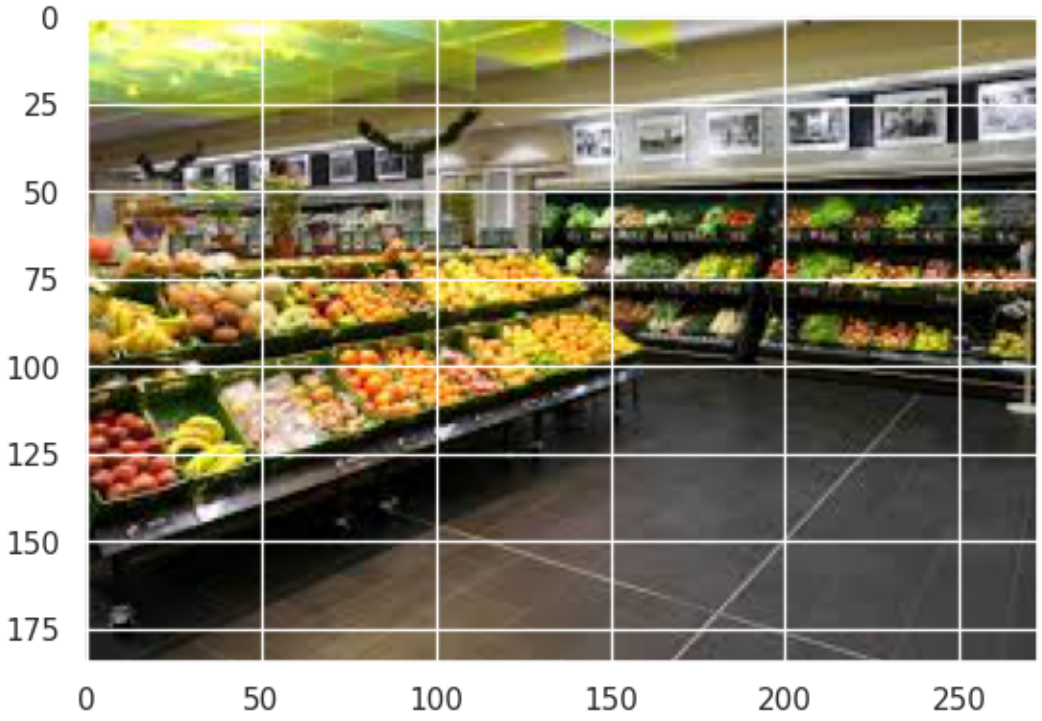
```
tomato
```



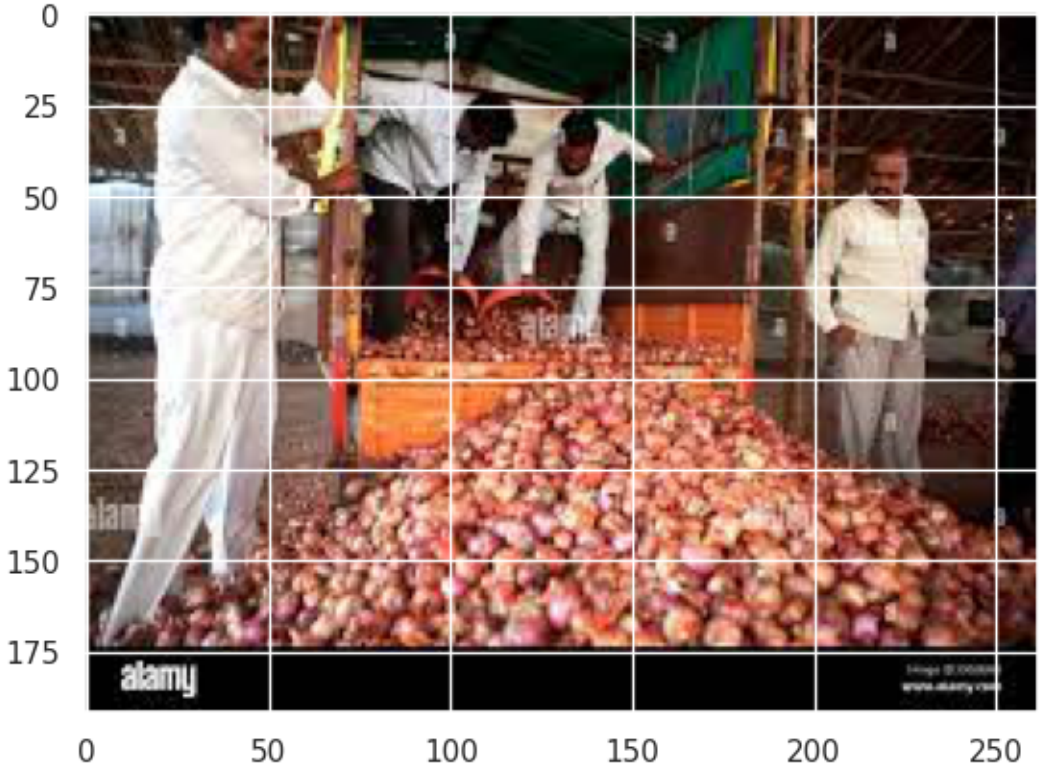
potato



indian market


0
25
50
75
100
125
150
175
0
50
100
150
200
250

```
onion
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250
alamy

1.3.2 Create train, validation, test split

[12]:
```
def load_data(val_split = 0.2, image_size = (300, 300), batch_size=32):

    dev_data = tf.keras.utils.image_dataset_from_directory(
        f"{NJ_DATA_PATH}/train", shuffle=True, label_mode='categorical',␣
 ↪image_size = image_size, batch_size = batch_size, crop_to_aspect_ratio=True
    )
    test_data = tf.keras.utils.image_dataset_from_directory(
        f"{NJ_DATA_PATH}/test", shuffle=False, label_mode='categorical',␣
 ↪image_size = image_size, batch_size = batch_size, crop_to_aspect_ratio=True
    )

    train_split = 1 - val_split
    ds_size = len(dev_data)
    train_size = int(train_split * ds_size)
    val_size = ds_size - train_size

    train_data = dev_data.take(train_size)
    val_data = dev_data.skip(train_size).take(val_size)

    return train_data, val_data, test_data, dev_data.class_names
```

[13]:
```
batch_size = 32
print(f'Creating train, validation, and test sets with batchsize: {batch_size}')
train_data, val_data, test_data, classes = load_data(batch_size=batch_size)
print(f'Number of batches: train - {len(train_data)}, validation -␣
 ↪{len(val_data)}, test - {len(test_data)}')
```

```
Creating train, validation, and test sets with batchsize: 32
Found 3135 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Number of batches: train - 78, validation - 20, test - 11
```

1.4 Data pre-processing pipelines

[14]:
```
# Helper class to manage different data pre-processing pipelines
class DataPipelines:

  """
  th and tw are target image height and width respectively.
  input_shape - shape of input images in the dataset
  """
  def __init__(self, th, tw, input_shape=(300, 300, 3)):
```

```
  self.th = th
  self.tw = tw
  self.input_shape = input_shape

  #self.bbh = 1.2 * th if (bbh is None) else bbh
  #self.bbw = 1.2 * tw if (bbw is None) else bbw
  self.create_pipelines_models(th, tw, input_shape)
  self.create_pipelines()

"""
creates diffenent preprocessing models for both training and inference
"""
def create_pipelines_models(self, th, tw, input_shape):

  self.models = {

    'dp1': keras.Sequential(
        name="data_preprocess_basic_resize",
        layers=[
            layers.Resizing(th, tw, input_shape=input_shape),
            layers.Rescaling(1.0/255),
        ]
    ),
    'da1': keras.Sequential(
      name="data_augmentation_1",
      layers=[
          layers.Resizing(th, tw, input_shape=input_shape),
          layers.Rescaling(1.0/255),
      ]
    ),
    'da2': keras.Sequential(
      name="data_augmentation_2",
      layers=[
          layers.RandomCrop(th, tw, input_shape=input_shape),
          layers.Rescaling(1.0/255)
      ]
    ),
    'da3': keras.Sequential(
      name="data_augmentation_3",
      layers=[
          layers.RandomTranslation(height_factor = (-0.2, 0.3), width_factor␣
↪= (-0.2, 0.3)),
          layers.RandomRotation(factor = (-0.2, 0.3)),
          layers.RandomFlip(),
          layers.RandomCrop(th, tw, input_shape=input_shape),
          layers.Rescaling(1.0/255)
```

```
        ]
      )
    }

  """
  creates named pair of pipelines corresponding to train and inference operations
  """
  def create_pipelines(self):
    self.pipelines = {
        'v1' : (self.models['da1'], self.models['dp1']),
        'v2' : (self.models['da2'], self.models['dp1']),
        'v3' : (self.models['da3'], self.models['dp1']),
    }

  """
  generator to yield all pipeline architectures
  """
  def get_all_pipelines(self):
    for pl_key in self.pipelines.keys():
      yield self.pipelines[pl_key]

  """
  returns a pipeline specificied by input name
  """
  def get_pipeline(self, name='v1'):
    return self.pipelines[name]
```

[15]:
```
#create an instance of data pipelines class to manage pre-processing pipelines
data_pipeline_helper = DataPipelines(224, 224, input_shape=(300, 300, 3))
```

[16]:
```
def get_preprocessed_datasets(pl_name):

  # fetch preprocessing pipelines
  train_pl, test_pl = data_pipeline_helper.get_pipeline(pl_name)

  # apply appropriate pipelines to train, test, and validation sets

  train_ds = train_data.map(
    lambda x, y: (train_pl(x), y), num_parallel_calls=tf.data.AUTOTUNE
  ).prefetch(tf.data.AUTOTUNE)

  val_ds = val_data.map(
    lambda x, y: (test_pl(x), y), num_parallel_calls=tf.data.AUTOTUNE
  ).prefetch(tf.data.AUTOTUNE)

  test_ds = test_data.map(
```

```
    lambda x, y: (test_pl(x), y), num_parallel_calls=tf.data.AUTOTUNE
  ).prefetch(tf.data.AUTOTUNE)

  return train_ds, val_ds, test_ds

train_ds1, val_ds1, test_ds1 = get_preprocessed_datasets('v1')
train_ds2, val_ds2, test_ds2 = get_preprocessed_datasets('v2')
train_ds3, val_ds3, test_ds3 = get_preprocessed_datasets('v3')
```

1.5 Checkpoint and Model State helpers

[17]:
```
import shutil
import datetime as dt

class CheckpointHelper:

  def __init__(self, root_path):
    self.root_path = root_path
    self.checkpoints = {}

  def getCallback(self, model_name):
    if model_name not in self.checkpoints:

      cp_callback = tf.keras.callbacks.ModelCheckpoint(
          filepath=f'{self.root_path}/{model_name}',
          save_weights_only=True,
          verbose=1)

      self.checkpoints[model_name] = cp_callback

    return self.checkpoints[model_name]

  def getPath(self, model_name):
    return self.checkpoints[model_name]

checkpoint_helper = CheckpointHelper(CHECKPOINT_PATH)
```

[18]:
```
import pickle

#class to store relevant training history details
class ModelHistory(object):
  def __init__(self, history, epoch, params):
    self.history = history
    self.epoch = epoch
    self.params = params
```

```
def get_modelhistory_from_history(history):
  return ModelHistory(history.history, history.epoch, history.params)

# Helper class to save/load models
class ModelStateHelper:

  def __init__(self, remote_path):
    self.remote_path = remote_path

  # saves model history
  def save_model_history(self, model, save_path):

    with open(f'{save_path}/history', 'wb') as file:
      model_history= ModelHistory(model.history.history, model.history.epoch,␣
 ↪model.history.params)
      pickle.dump(model_history, file, pickle.HIGHEST_PROTOCOL)

  # loads model history
  def load_model_history(self, save_path):

    with open(f'{save_path}/history', 'rb') as file:
      history=pickle.load(file)

    return history

  # saves model including history
  def save_model(self, model, model_name):

    save_path = f'{self.remote_path}/{model_name}'

    # if already exists, then rename it by appending its creation time at the␣
 ↪end
    if(os.path.isdir(save_path)):
      ts = str(dt.datetime.fromtimestamp(os.path.getctime(save_path))) #read␣
 ↪creation time
      os.rename(save_path, f'{save_path}_{ts}')

    # create a new empty directory
    os.makedirs(save_path)

    #save model
    model.save(save_path)

    #save history
    self.save_model_history(model, save_path)

  # loads model and its history details
```

```
  def load_model(self, model_name):

    save_path = f'{self.remote_path}/{model_name}'

    if(os.path.isdir(save_path)):

      model = tf.keras.models.load_model(save_path)
      history = self.load_model_history(save_path)

      return model, history

    return None, None

  def canloadsavedmodel(self, model_name):

    save_path = f'{self.remote_path}/{model_name}'
    return os.path.isdir(save_path)
    #return False


model_state_helper = ModelStateHelper(GDRIVE_SAVEDMODEL_PATH)
```

1.6 Common Utility functions

[19]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
 ↪recall_score

# helper function to annotate maximum values in the plots
def annot_max(x,y, xytext=(0.94,0.96), ax=None, only_y=True):
    xmax = x[np.argmax(y)]
    ymax = max(y)
    if only_y:
        text = "{:.2f}%".format(ymax)
    else:
        text= "x={:.2f}, y={:.2f}%".format(xmax, ymax)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="->",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=xytext, **kw)

# accuracy graph

def plot_accuracy(model_fit):
```

```
    x = range(0,len(model_fit.history['accuracy']))
    y_train = [acc * 100 for acc in model_fit.history['accuracy']]
    y_val = [acc * 100 for acc in model_fit.history['val_accuracy']]

    plt.plot(x, y_train, label='Train', color='b')
    annot_max(x, y_train, xytext=(0.7,0.9))
    plt.plot(x, y_val, label='Val', color='r')
    annot_max(x, y_val, xytext=(0.8,0.7))
    plt.ylabel('Accuracy', fontsize=15)
    plt.xlabel('epoch', fontsize=15)
    plt.legend()
    plt.show()

def plot_confusion_matrix(ylabels, ypred, class_names):
    cm = metrics.confusion_matrix(ylabels, ypred)
    sns.heatmap(cm, annot=True, xticklabels=class_names,␣
 ↪yticklabels=class_names, cmap="YlGnBu", fmt='g')
    plt.show()

def get_ytrue_and_ypred(preds_ohe, dataset):

  # extract y_labels
  ytrue_ohe = np.concatenate([np.array(batch[1]) for batch in iter(dataset)])
  ytrue = tf.argmax(ytrue_ohe, axis=1)

  # extract y_pred
  ypred = tf.argmax(preds_ohe, axis=1)

  return ytrue, ypred

def predict_and_plot_confusion_matrix(model, dataset, class_names):

  preds = model.predict(dataset, verbose=2, use_multiprocessing=True)
  ytrue, ypred = get_ytrue_and_ypred(preds, dataset)
  plot_confusion_matrix(ytrue, ypred, class_names)

  acc = np.round(accuracy_score(ytrue, ypred), 3)
  precision = np.round(precision_score(ytrue, ypred, average='macro'), 2)
  recall = np.round(recall_score(ytrue, ypred, average='macro'), 2)

  classwise_scores = []

  print('')

  for ci in range(len(class_names)):
    ytrue_bin = list(map(lambda x: 1 if (x == ci) else 0, ytrue))
    ypred_bin = list(map(lambda x: 1 if (x == ci) else 0, ypred))
```

```
    classwise_scores.append((
        class_names[ci],
        np.round(precision_score(ytrue_bin, ypred_bin, average='binary'), 2),
        np.round(recall_score(ytrue_bin, ypred_bin, average='binary'), 2)
    ))

  print(pd.DataFrame(classwise_scores, columns=['class', 'precision',␣
 ↪'recall']))
  print(f'Overall accuracy: {np.round(acc * 100,2)}%, macro precision:␣
 ↪{precision}, macro recall: {recall}')

  #return acc, precision, recall, ytrue, ypred

def training_plot(metrics, history):
  f, ax = plt.subplots(1, len(metrics), figsize=(5*len(metrics), 5))
  for idx, metric in enumerate(metrics):
    ax[idx].plot(history.history[metric], ls='dashed')
    ax[idx].set_xlabel("Epochs")
    ax[idx].set_ylabel(metric)
    ax[idx].plot(history.history['val_' + metric]);
    ax[idx].legend([metric, 'val_' + metric])

  plt.show()
```

[20]:
```
# Helper function to save/load, train, compile, and train model
def get_trained_model(model_name, create_model_fn, compile_model_fn,␣
 ↪train_model_fn, train_set, val_set, force_train = False):

  if(not force_train and model_state_helper.canloadsavedmodel(model_name)):

    print('saved model found. loading it..')

    # load saved model and history
    model_pl, model_history = model_state_helper.load_model(model_name)

    # compile it
    compile_model_fn(model_pl)

  else:
    # create model
    model_pl = create_model_fn(model_name)

    # compile
    compile_model_fn(model_pl)

    # train
```

```
    history = train_model_fn(model_pl, train_set, val_set)

    model_history = get_modelhistory_from_history(history)

    #save model for future use
    print('saving model and history')
    model_state_helper.save_model(model_pl, model_name)

  return model_pl, model_history
```

[21]:
```
def get_compile_fn(model, optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy']):
  return model.compile(
      optimizer=optimizer,
      loss = loss,
      metrics=metrics
    )

def model_fit_fn(model, train_data, val_data, epochs=5, callbacks=[]):
  return model.fit(
      train_data,
      epochs=epochs,
      validation_data=val_data,
      callbacks=callbacks
      )
```

1.7 CNN model from scratch

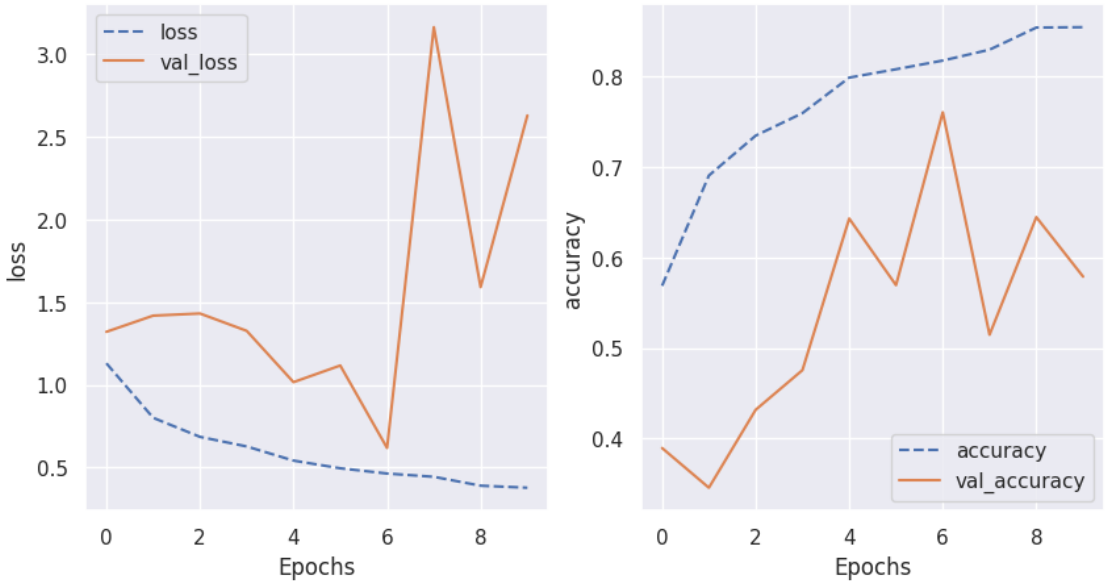
Since the amount of training data is relatively small, to build a CNN classifier from scratch, we will choose a smaller architecture like Alexnet. The code below creates a model based on Alexnet architecture, trains it on the training dataset, and measures validation/test loss, as well as classification metrics such as accuracy, precision, and recall. We will use this model for benchmarking purpose.

[27]:
```
model_name = 'base_model_alexnet_v1'

def get_alexnet_model():
  return keras.models.Sequential([
    keras.layers.Resizing(64, 64),
    keras.layers.Conv2D(filters=128, kernel_size=(11,11), strides=(4,4),␣
 ↪activation='relu', input_shape=(64,64,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),␣
 ↪activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3)),
```

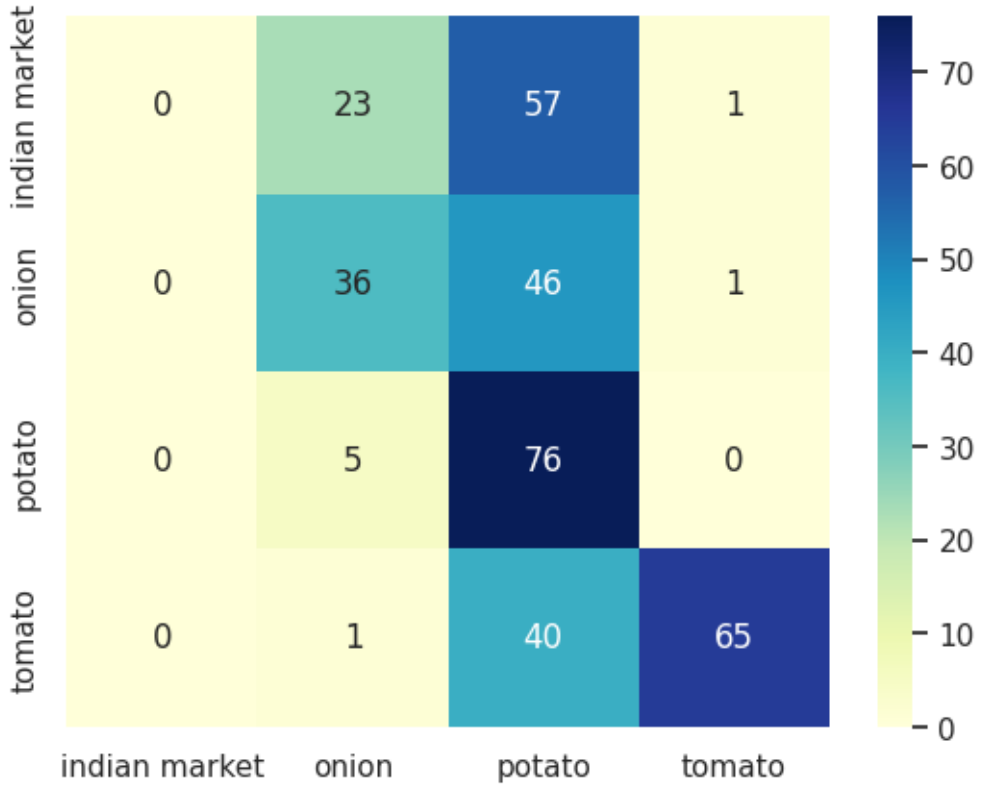
```
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),␣
 ↪activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),␣
 ↪activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),␣
 ↪activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(1024,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1024,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4,activation='softmax')
  ])

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

base_model_v1, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: get_alexnet_model(),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10),

    #training data
    train_ds1,

    #validation data
    val_ds1,

    )
```

```
saved model found. loading it..
```

[28]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(base_model_v1, test_ds1, classes)
```




```
11/11 - 8s - 8s/epoch - 700ms/step
```



```
           class  precision  recall
0  indian market       0.00    0.00
1          onion       0.55    0.43
2         potato       0.35    0.94
3         tomato       0.97    0.61
Overall accuracy: 50.4%, macro precision: 0.47, macro recall: 0.5
```

Observation:

Our base CNN model produces ~58% accuracy on validation data and ~50% accuracy on test data. The model is extremely biased against 'Indian market' class with zero TP. From the training plot we can observe that validation loss fluctuated a lot. We can try to address this by adding 'ReduceLROnPlateau' LR scheduler and earlystopping. We will also increase batch_size to 64 to see if that can help reduce fluctuations.

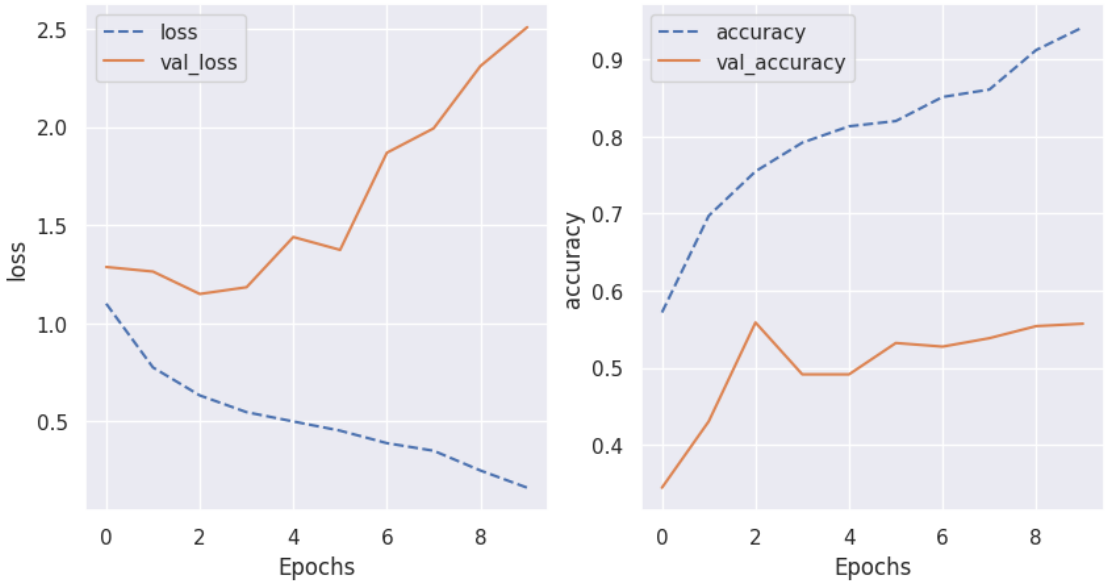
[33]:
```
model_name = 'base_model_alexnet_v2'

callbacks = [
  keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.3, patience=5, min_lr=0.00005
  ),
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, min_delta=0.001, mode='min'
  )
]

optimizer = keras.optimizers.Adam(learning_rate=0.001)

base_model_v2, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: get_alexnet_model(),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10, callbacks=callbacks),

    #training data
    train_ds1.unbatch().batch(64),

    #validation data
    val_ds1.unbatch().batch(64),
```

```
    force_train = True

    )
```

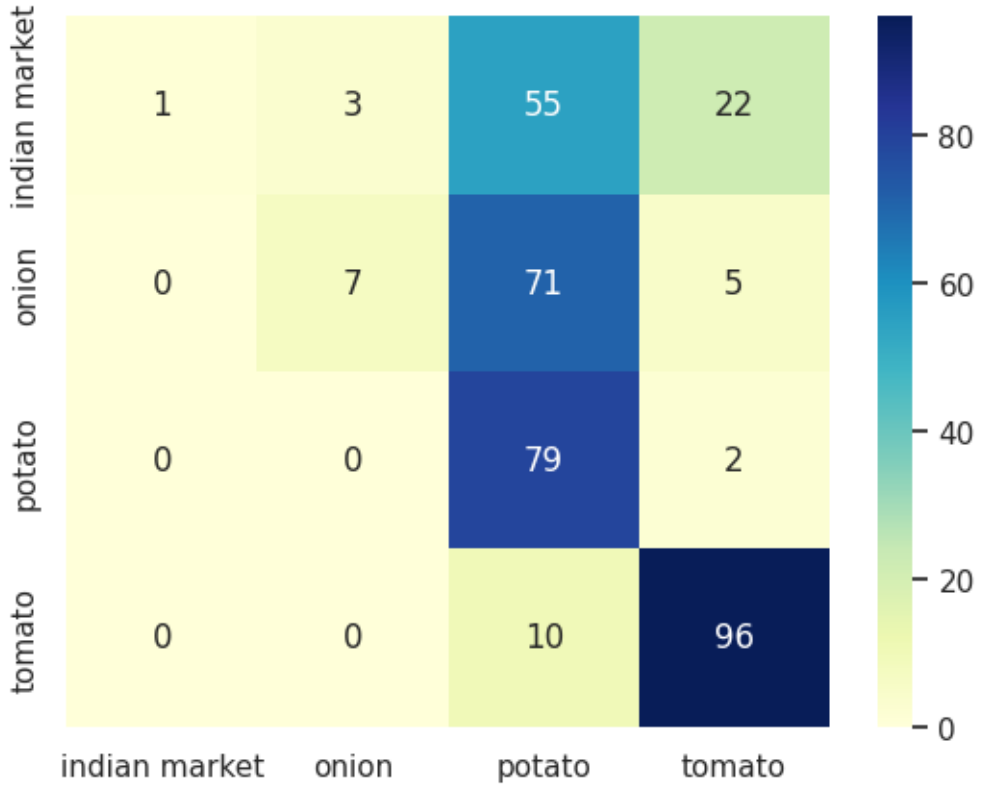
```
Epoch 1/10
39/39 [==============================] - 41s 610ms/step - loss: 1.1002 -
accuracy: 0.5717 - val_loss: 1.2870 - val_accuracy: 0.3443 - lr: 5.0000e-04
Epoch 2/10
39/39 [==============================] - 23s 587ms/step - loss: 0.7751 -
accuracy: 0.6971 - val_loss: 1.2638 - val_accuracy: 0.4304 - lr: 5.0000e-04
Epoch 3/10
39/39 [==============================] - 33s 829ms/step - loss: 0.6327 -
accuracy: 0.7548 - val_loss: 1.1496 - val_accuracy: 0.5587 - lr: 5.0000e-04
Epoch 4/10
39/39 [==============================] - 24s 619ms/step - loss: 0.5474 -
accuracy: 0.7921 - val_loss: 1.1839 - val_accuracy: 0.4914 - lr: 5.0000e-04
Epoch 5/10
39/39 [==============================] - 33s 838ms/step - loss: 0.5000 -
accuracy: 0.8133 - val_loss: 1.4397 - val_accuracy: 0.4914 - lr: 5.0000e-04
Epoch 6/10
39/39 [==============================] - 33s 847ms/step - loss: 0.4532 -
accuracy: 0.8201 - val_loss: 1.3741 - val_accuracy: 0.5321 - lr: 5.0000e-04
Epoch 7/10
39/39 [==============================] - 23s 558ms/step - loss: 0.3895 -
accuracy: 0.8514 - val_loss: 1.8690 - val_accuracy: 0.5274 - lr: 5.0000e-04
Epoch 8/10
39/39 [==============================] - 32s 784ms/step - loss: 0.3508 -
accuracy: 0.8610 - val_loss: 1.9935 - val_accuracy: 0.5383 - lr: 5.0000e-04
Epoch 9/10
39/39 [==============================] - 32s 806ms/step - loss: 0.2501 -
accuracy: 0.9123 - val_loss: 2.3109 - val_accuracy: 0.5540 - lr: 1.5000e-04
Epoch 10/10
39/39 [==============================] - 33s 847ms/step - loss: 0.1624 -
accuracy: 0.9419 - val_loss: 2.5088 - val_accuracy: 0.5571 - lr: 1.5000e-04
saving model and history
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 6). These functions will not be directly callable after loading.
```

[34]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(base_model_v2, test_ds1, classes)
```




```
11/11 - 2s - 2s/epoch - 137ms/step
```



```
           class  precision  recall
0  indian market       1.00    0.01
1          onion       0.70    0.08
2         potato       0.37    0.98
3         tomato       0.77    0.91
Overall accuracy: 52.1%, macro precision: 0.71, macro recall: 0.49
```

Observation

We see improvement in our base model performance after applying earlystopping and LR scheduler. The validation accuracy reached ~87% while the test accuracy reached 75.5%, with precision and recall values being 0.76 and 0.74. However, we still see fluctuations in validation loss. We can try introducing regularization and see its impact.

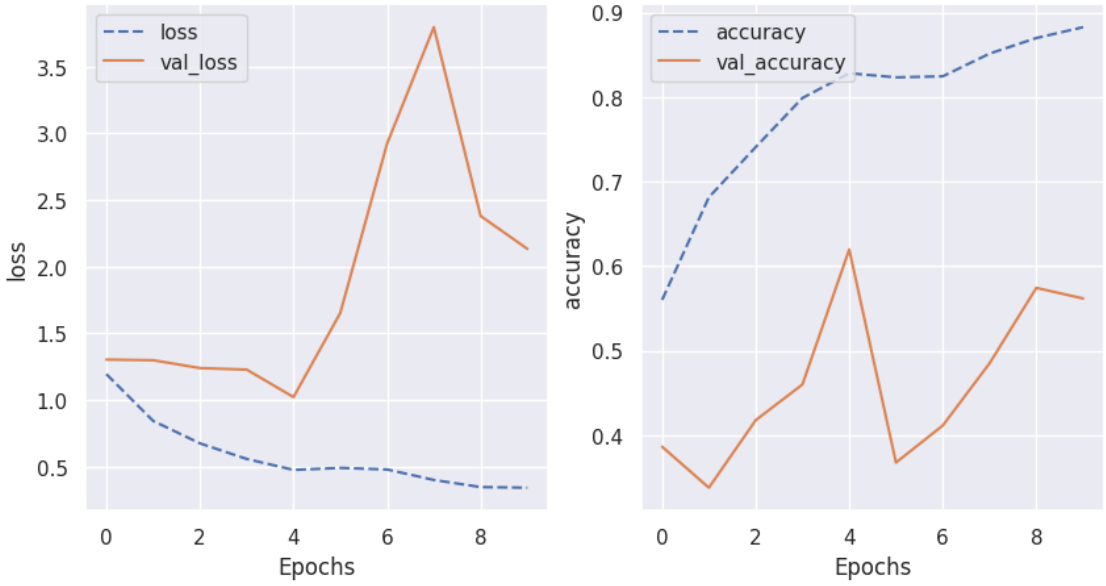
[35]:
```
#adding L2 regularization to CNN
from tensorflow.keras import regularizers

def get_alexnet_model_v3():
  penalty_fact = 1e-5
  return keras.models.Sequential([
    keras.layers.Resizing(64, 64),
    keras.layers.Conv2D(filters=128, kernel_size=(11,11), strides=(4,4),␣
 ↪activation='relu', input_shape=(64,64,3), kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),␣
 ↪activation='relu', padding="same", kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3)),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),␣
 ↪activation='relu', padding="same", kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),␣
 ↪activation='relu', padding="same", kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),␣
 ↪activation='relu', padding="same", kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(1024,activation='relu', kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
```

```
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1024,activation='relu', kernel_regularizer=regularizers.
 ↪l2(penalty_fact)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4,activation='softmax')
  ])

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

callbacks = [
  keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.3, patience=5, min_lr=0.000005
  ),
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, min_delta=0.001, mode='min',␣
 ↪restore_best_weights=True
  )
]

model_name = 'base_model_alexnet_v3'

base_model_v3, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: get_alexnet_model_v3(),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=20, callbacks=callbacks),

    #training data
    train_ds1.unbatch().batch(64),

    #validation data
    val_ds1.unbatch().batch(64),

    )
```

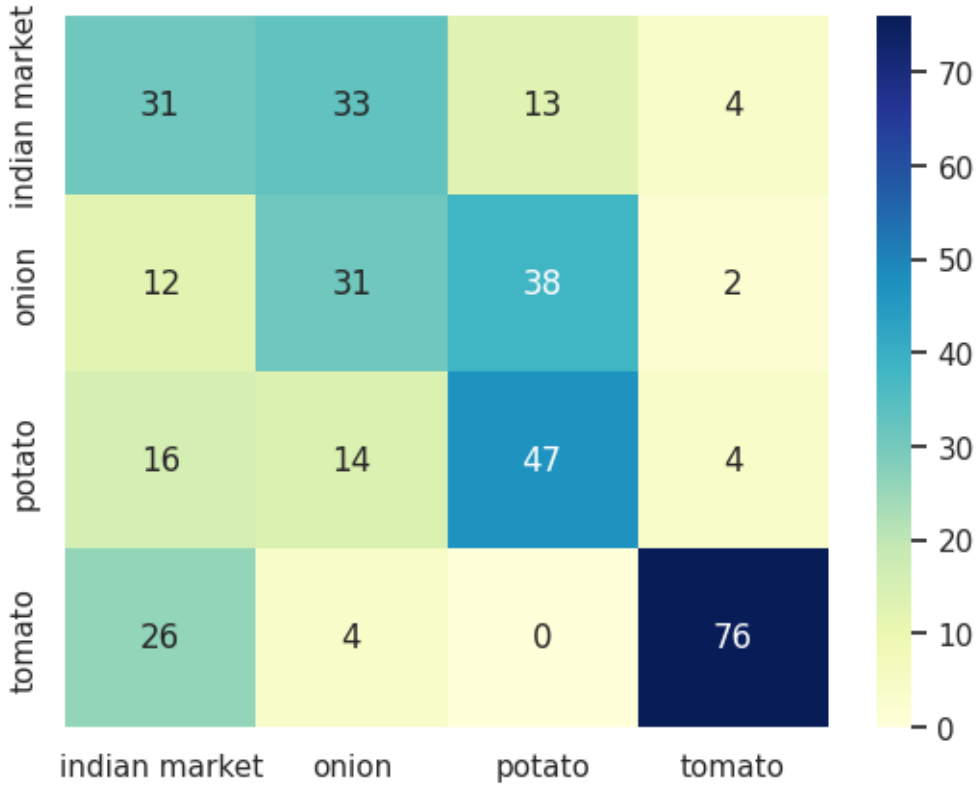
```
Epoch 1/20
39/39 [==============================] - 28s 545ms/step - loss: 1.1945 -
accuracy: 0.5601 - val_loss: 1.3035 - val_accuracy: 0.3865 - lr: 5.0000e-04
Epoch 2/20
```

```
39/39 [==============================] - 33s 845ms/step - loss: 0.8438 -
accuracy: 0.6815 - val_loss: 1.2986 - val_accuracy: 0.3380 - lr: 5.0000e-04
Epoch 3/20
39/39 [==============================] - 22s 559ms/step - loss: 0.6754 -
accuracy: 0.7408 - val_loss: 1.2399 - val_accuracy: 0.4178 - lr: 5.0000e-04
Epoch 4/20
39/39 [==============================] - 32s 826ms/step - loss: 0.5584 -
accuracy: 0.7989 - val_loss: 1.2282 - val_accuracy: 0.4601 - lr: 5.0000e-04
Epoch 5/20
39/39 [==============================] - 26s 650ms/step - loss: 0.4763 -
accuracy: 0.8281 - val_loss: 1.0225 - val_accuracy: 0.6197 - lr: 5.0000e-04
Epoch 6/20
39/39 [==============================] - 24s 569ms/step - loss: 0.4911 -
accuracy: 0.8233 - val_loss: 1.6525 - val_accuracy: 0.3678 - lr: 5.0000e-04
Epoch 7/20
39/39 [==============================] - 32s 817ms/step - loss: 0.4795 -
accuracy: 0.8245 - val_loss: 2.9218 - val_accuracy: 0.4116 - lr: 5.0000e-04
Epoch 8/20
39/39 [==============================] - 25s 637ms/step - loss: 0.4009 -
accuracy: 0.8514 - val_loss: 3.7960 - val_accuracy: 0.4851 - lr: 5.0000e-04
Epoch 9/20
39/39 [==============================] - 33s 848ms/step - loss: 0.3477 -
accuracy: 0.8698 - val_loss: 2.3805 - val_accuracy: 0.5743 - lr: 5.0000e-04
Epoch 10/20
39/39 [==============================] - 24s 584ms/step - loss: 0.3427 -
accuracy: 0.8826 - val_loss: 2.1338 - val_accuracy: 0.5618 - lr: 5.0000e-04
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 6). These functions will not be directly callable after loading.
```

[36]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(base_model_v3, test_ds1, classes)
```




```
11/11 - 3s - 3s/epoch - 234ms/step
```



```
           class  precision  recall
0  indian market       0.36    0.38
1          onion       0.38    0.37
2         potato       0.48    0.58
3         tomato       0.88    0.72
Overall accuracy: 52.7%, macro precision: 0.53, macro recall: 0.51
```

[36]:

Observation:

After adding regularization, we infact see a drop in validation accuracy and test accuracy. Let's revert to the previous model and apply data augmentation to see if that can help increase accuracy.

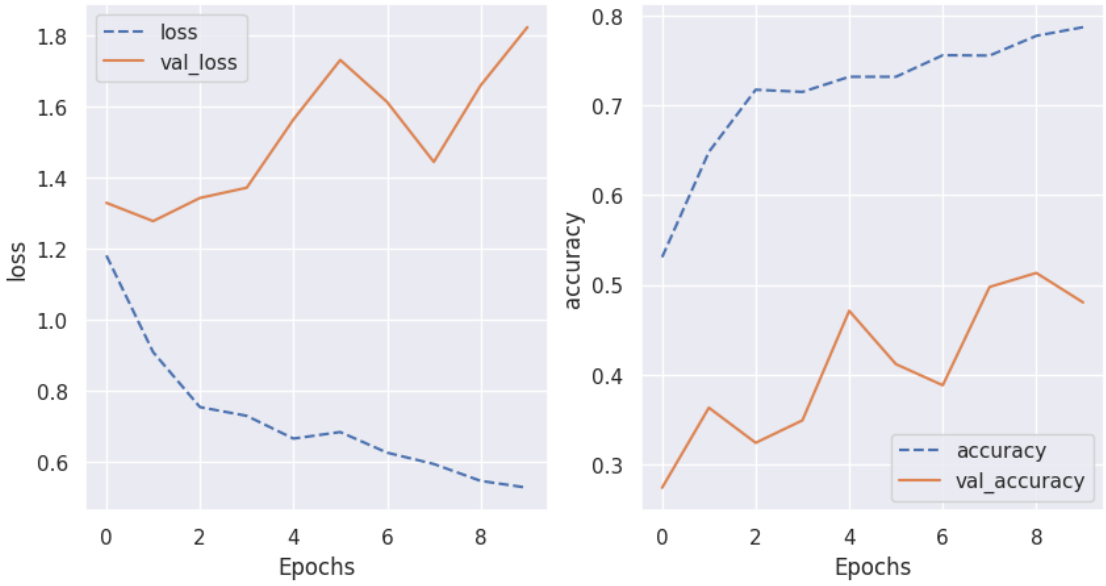
[37]:
```
model_name = 'base_model_alexnet_v2.1'

callbacks = [
  keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.3, patience=5, min_lr=0.0001
  ),
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, min_delta=0.001, mode='min'
  )
]

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

base_model_v21, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: get_alexnet_model(),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10, callbacks=callbacks),

    #training data
    train_ds3.unbatch().batch(64),

    #validation data
    val_ds3.unbatch().batch(64),
```

```
    )
```

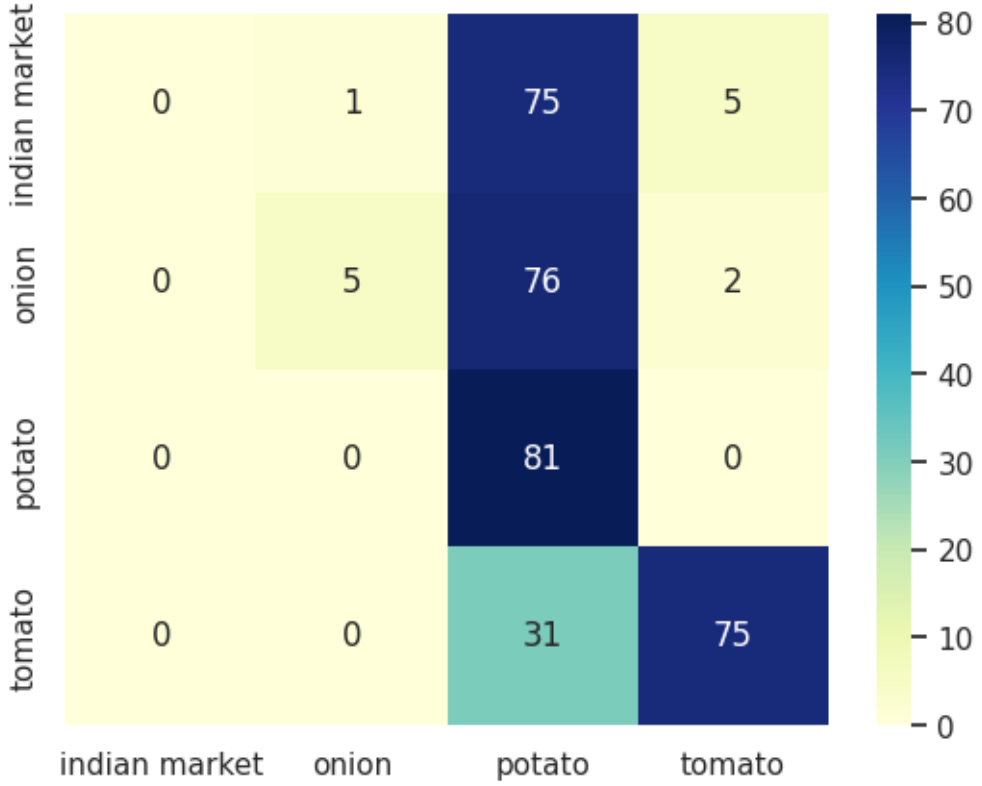
```
Epoch 1/10
39/39 [==============================] - 72s 2s/step - loss: 1.1813 - accuracy:
0.5308 - val_loss: 1.3296 - val_accuracy: 0.2739 - lr: 5.0000e-04
Epoch 2/10
39/39 [==============================] - 64s 2s/step - loss: 0.9087 - accuracy:
0.6482 - val_loss: 1.2781 - val_accuracy: 0.3631 - lr: 5.0000e-04
Epoch 3/10
39/39 [==============================] - 66s 2s/step - loss: 0.7539 - accuracy:
0.7175 - val_loss: 1.3433 - val_accuracy: 0.3239 - lr: 5.0000e-04
Epoch 4/10
39/39 [==============================] - 66s 2s/step - loss: 0.7293 - accuracy:
0.7151 - val_loss: 1.3721 - val_accuracy: 0.3490 - lr: 5.0000e-04
Epoch 5/10
39/39 [==============================] - 66s 2s/step - loss: 0.6652 - accuracy:
0.7320 - val_loss: 1.5646 - val_accuracy: 0.4710 - lr: 5.0000e-04
Epoch 6/10
39/39 [==============================] - 75s 2s/step - loss: 0.6837 - accuracy:
0.7320 - val_loss: 1.7321 - val_accuracy: 0.4116 - lr: 5.0000e-04
Epoch 7/10
39/39 [==============================] - 67s 2s/step - loss: 0.6256 - accuracy:
0.7560 - val_loss: 1.6138 - val_accuracy: 0.3881 - lr: 5.0000e-04
Epoch 8/10
39/39 [==============================] - 66s 2s/step - loss: 0.5936 - accuracy:
0.7556 - val_loss: 1.4449 - val_accuracy: 0.4977 - lr: 1.5000e-04
Epoch 9/10
39/39 [==============================] - 65s 2s/step - loss: 0.5460 - accuracy:
0.7776 - val_loss: 1.6603 - val_accuracy: 0.5133 - lr: 1.5000e-04
Epoch 10/10
39/39 [==============================] - 64s 2s/step - loss: 0.5267 - accuracy:
0.7873 - val_loss: 1.8245 - val_accuracy: 0.4804 - lr: 1.5000e-04
saving model and history
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 6). These functions will not be directly callable after loading.
```

[38]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(base_model_v21, test_ds3, classes)
```




```
11/11 - 2s - 2s/epoch - 148ms/step
```



```
           class  precision  recall
0  indian market       0.00    0.00
1          onion       0.83    0.06
2         potato       0.31    1.00
3         tomato       0.91    0.71
Overall accuracy: 45.9%, macro precision: 0.51, macro recall: 0.44
```

1.8 Extending VGGNet-19

Load vgg19 pretrained model

[39]:
```
pretrained_vgg19 = tf.keras.applications.vgg19.VGG19(weights='imagenet',␣
↪include_top=False, input_shape=[224,224,3])
pretrained_vgg19.trainable=False

pretrained_vgg19.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
```

```
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

1.8.1 Baseline model - V1

We will first create a baseline model which simply flattens the output of pre-trained vgg19 model and connects it with output layer with 4 classes. We will also apply LR scheduler, early stopping, and model checkpoint callbacks.

[40]:
```
model_name = 'vgg19_v1'

callbacks = [
  keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.3, patience=5, min_lr=0.000005
  ),
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, min_delta=0.001, mode='min',␣
  ↪restore_best_weights=True
  ),
  checkpoint_helper.getCallback(model_name)
]

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

vgg19_model_v1, model_history = get_trained_model(
```

```
  model_name,

  # create_model_fn
  lambda model_name: tf.keras.Sequential(
    name=model_name,
    layers=[
        pretrained_vgg19,
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(4, activation='softmax')]
    ),

  # compile_model_fn
  lambda model: get_compile_fn(model, optimizer=optimizer),

  # train_model_fn
  lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
↪val_data, epochs=10, callbacks=callbacks),

  #training data
  train_ds1,

  #validation data
  val_ds1
  )
```

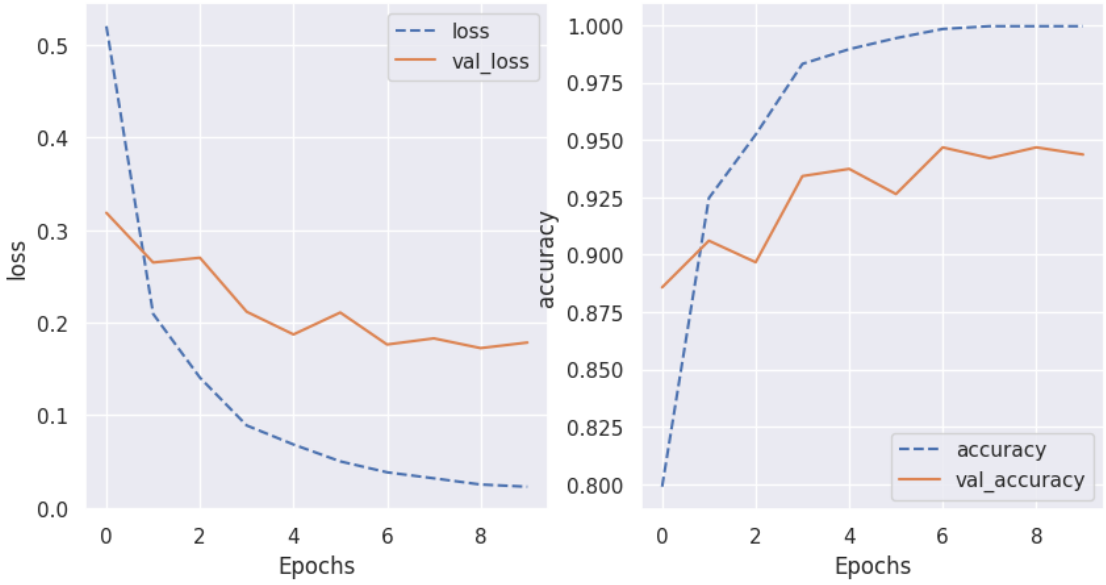
```
Epoch 1/10
78/78 [==============================] - ETA: 0s - loss: 0.5205 - accuracy:
0.7989
Epoch 1: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 42s 450ms/step - loss: 0.5205 -
accuracy: 0.7989 - val_loss: 0.3188 - val_accuracy: 0.8858 - lr: 5.0000e-04
Epoch 2/10
78/78 [==============================] - ETA: 0s - loss: 0.2096 - accuracy:
0.9247
Epoch 2: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 26s 328ms/step - loss: 0.2096 -
accuracy: 0.9247 - val_loss: 0.2651 - val_accuracy: 0.9061 - lr: 5.0000e-04
Epoch 3/10
78/78 [==============================] - ETA: 0s - loss: 0.1407 - accuracy:
0.9523
Epoch 3: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 35s 435ms/step - loss: 0.1407 -
accuracy: 0.9523 - val_loss: 0.2703 - val_accuracy: 0.8967 - lr: 5.0000e-04
Epoch 4/10
78/78 [==============================] - ETA: 0s - loss: 0.0890 - accuracy:
0.9832
```

```
Epoch 4: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 35s 436ms/step - loss: 0.0890 -
accuracy: 0.9832 - val_loss: 0.2119 - val_accuracy: 0.9343 - lr: 5.0000e-04
Epoch 5/10
78/78 [==============================] - ETA: 0s - loss: 0.0685 - accuracy:
0.9896
Epoch 5: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 28s 351ms/step - loss: 0.0685 -
accuracy: 0.9896 - val_loss: 0.1874 - val_accuracy: 0.9374 - lr: 5.0000e-04
Epoch 6/10
78/78 [==============================] - ETA: 0s - loss: 0.0502 - accuracy:
0.9944
Epoch 6: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 31s 392ms/step - loss: 0.0502 -
accuracy: 0.9944 - val_loss: 0.2111 - val_accuracy: 0.9264 - lr: 5.0000e-04
Epoch 7/10
78/78 [==============================] - ETA: 0s - loss: 0.0384 - accuracy:
0.9984
Epoch 7: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 35s 437ms/step - loss: 0.0384 -
accuracy: 0.9984 - val_loss: 0.1765 - val_accuracy: 0.9468 - lr: 5.0000e-04
Epoch 8/10
78/78 [==============================] - ETA: 0s - loss: 0.0319 - accuracy:
0.9996
Epoch 8: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 35s 435ms/step - loss: 0.0319 -
accuracy: 0.9996 - val_loss: 0.1831 - val_accuracy: 0.9421 - lr: 5.0000e-04
Epoch 9/10
78/78 [==============================] - ETA: 0s - loss: 0.0252 - accuracy:
0.9996
Epoch 9: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 35s 444ms/step - loss: 0.0252 -
accuracy: 0.9996 - val_loss: 0.1726 - val_accuracy: 0.9468 - lr: 5.0000e-04
Epoch 10/10
78/78 [==============================] - ETA: 0s - loss: 0.0228 - accuracy:
0.9996
Epoch 10: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 27s 338ms/step - loss: 0.0228 -
accuracy: 0.9996 - val_loss: 0.1786 - val_accuracy: 0.9437 - lr: 5.0000e-04
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 16). These functions will not be directly callable after loading.
```

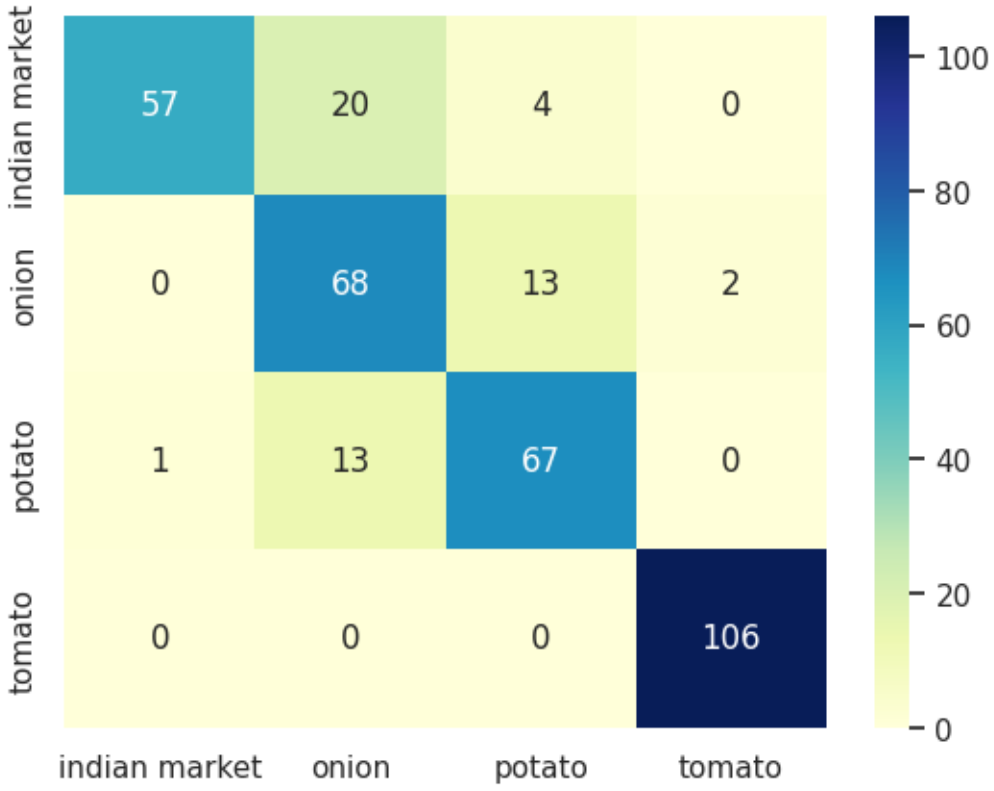
[41]:
```
vgg19_model_v1.summary()
```

```
Model: "vgg19_v1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 flatten_6 (Flatten)         (None, 25088)             0

 dense_18 (Dense)            (None, 4)                 100356

=================================================================
Total params: 20,124,740
Trainable params: 100,356
Non-trainable params: 20,024,384
_________________________________________________________________
```

[42]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(vgg19_model_v1, test_ds2, classes)
```




```
11/11 - 2s - 2s/epoch - 188ms/step
```




```
           class  precision  recall
0  indian market       0.98    0.70
1          onion       0.67    0.82
2         potato       0.80    0.83
3         tomato       0.98    1.00
Overall accuracy: 84.9%, macro precision: 0.86, macro recall: 0.84
```

Observation

Out model reports validation accuracy of 94% and test accuracy of ~85%. It also seems much more balanced in its prediction compared to the base model.

The model has 100,356 trainable parameters. Let's try to reduce the number of trainable parameters by adding global average pooling at the end of vgg model and check its performance.

1.8.2 VGG19 v2 - global average pooling

The output of pretrained VGG19 model is 7x7x512. Before flattening it out, we apply global max pooling to reduce it to 1x1x512. This substantially reduces trainable parameters from 100,356 to 2052.

[52]:
```
model_name = 'vgg19_v2'
```

```
optimizer = keras.optimizers.Adam(learning_rate=0.0005)

vgg19_model_v2, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_vgg19,
          tf.keras.layers.GlobalAveragePooling2D(),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10, callbacks=callbacks),

    #training data
    train_ds1,

    #validation data
    val_ds1
    )
```

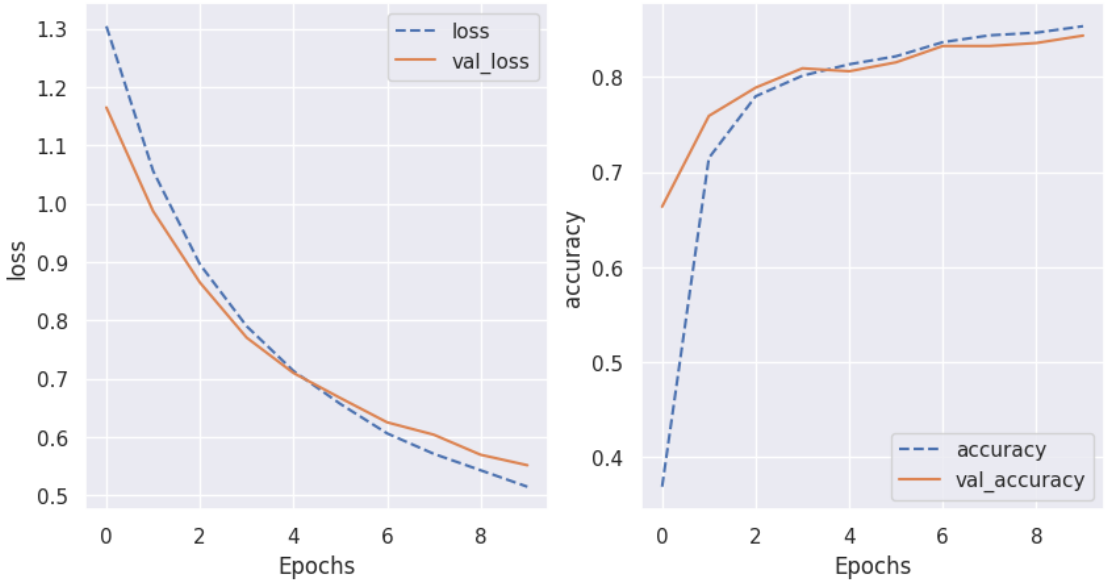
```
saved model found. loading it..
```

[44]:
```
vgg19_model_v2.summary()
```

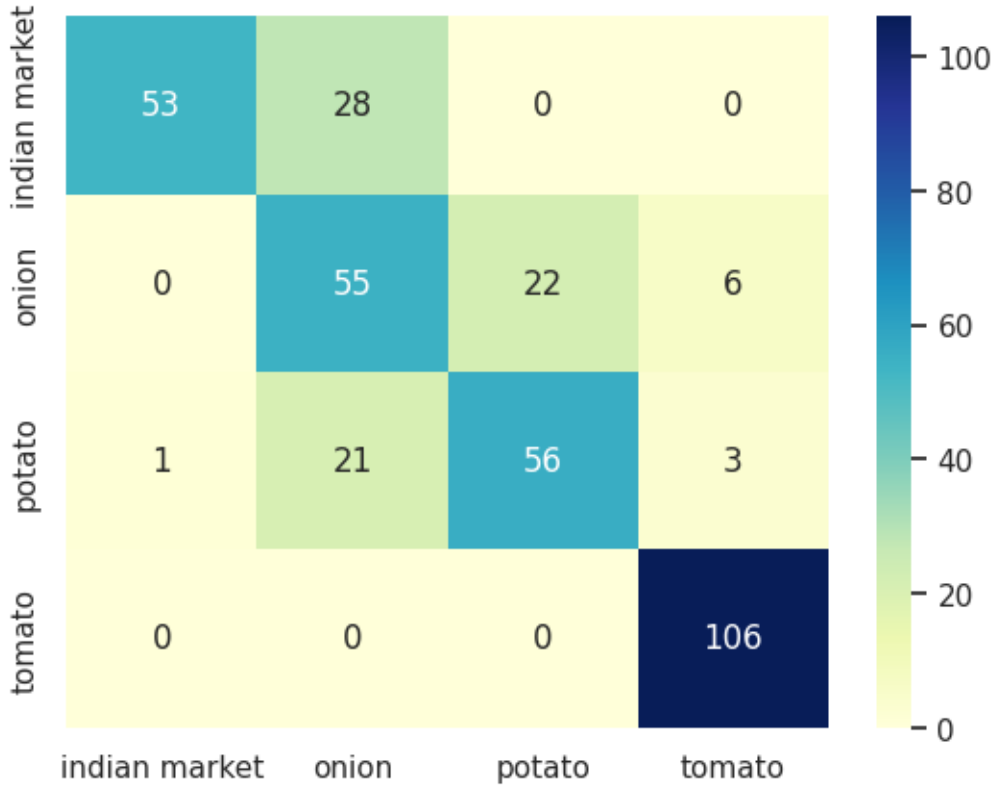
```
Model: "vgg19_v2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 flatten_7 (Flatten)         (None, 512)               0

 dense_19 (Dense)            (None, 4)                 2052
```

```
=================================================================
Total params: 20,026,436
Trainable params: 2,052
Non-trainable params: 20,024,384
_________________________________________________________________
```

[45]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(vgg19_model_v2, test_ds1, classes)
```




```
11/11 - 2s - 2s/epoch - 183ms/step
```




```
           class  precision  recall
0  indian market       0.98    0.65
1          onion       0.53    0.66
2         potato       0.72    0.69
3         tomato       0.92    1.00
Overall accuracy: 76.9%, macro precision: 0.79, macro recall: 0.75
```

Observation:

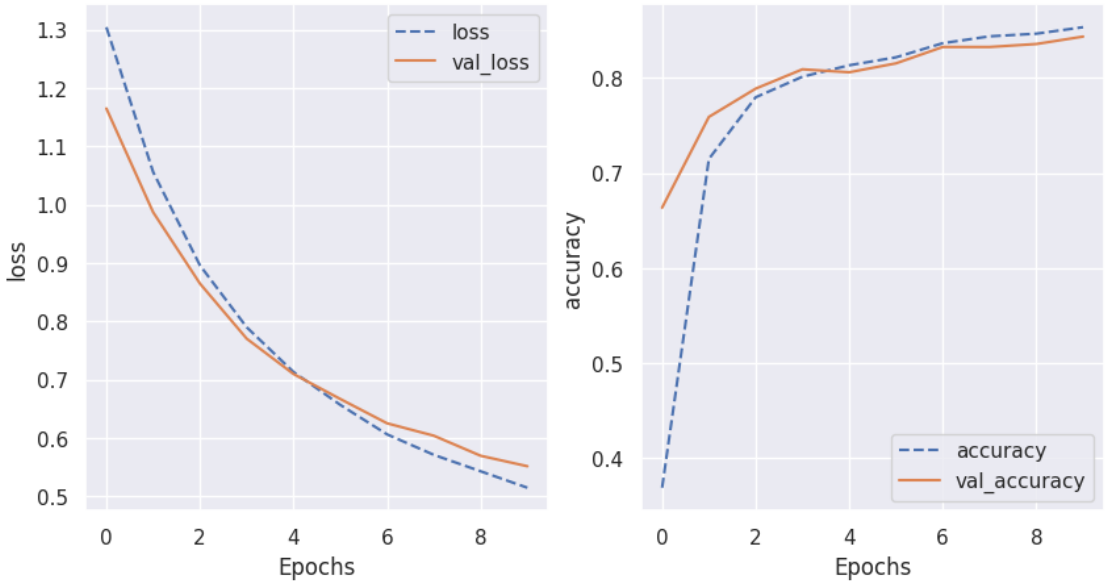
From the training plots we can see that both training and validation losses had smoother, gradual decrease, and accuracy curves increased gradually as well. We try running the same model for a few more epochs to check if the performance improves further.

[55]:
```
model_history2 = vgg19_model_v2.fit(
      train_ds1,
      epochs=10,
      validation_data=val_ds1,
      callbacks=callbacks
      )
```

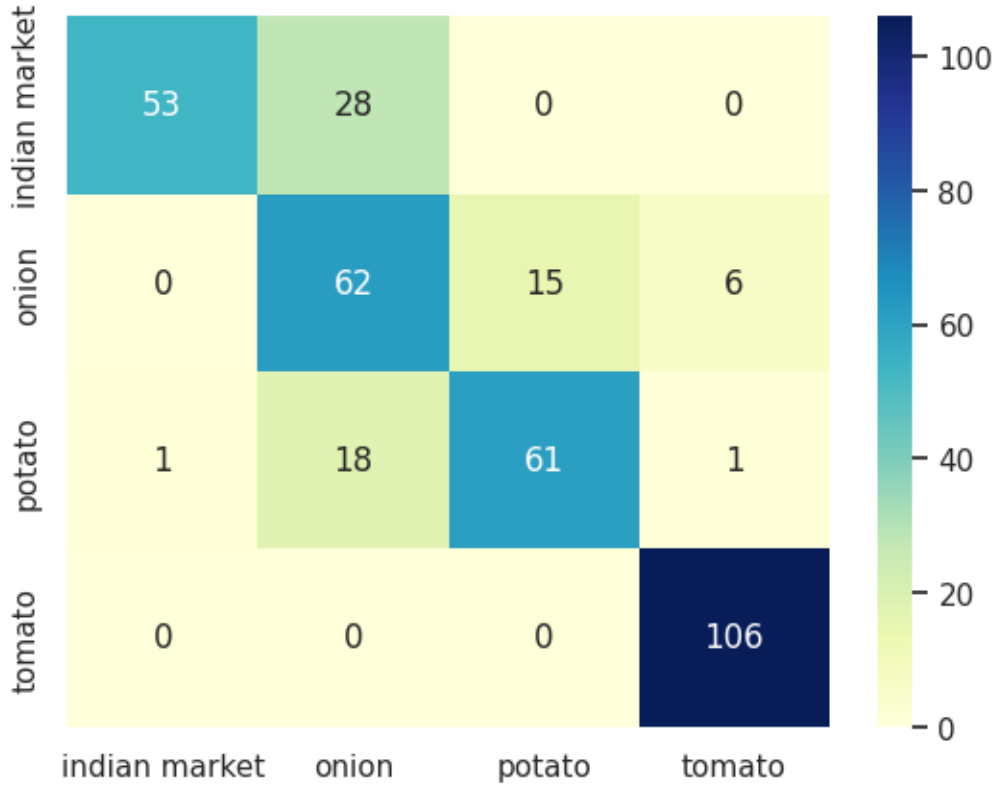
```
Epoch 1/5
78/78 [==============================] - ETA: 0s - loss: 0.3945 - accuracy:
```

```
0.8718
Epoch 1: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 35s 433ms/step - loss: 0.3945 -
accuracy: 0.8718 - val_loss: 0.4442 - val_accuracy: 0.8685
Epoch 2/5
78/78 [==============================] - ETA: 0s - loss: 0.3884 - accuracy:
0.8794
Epoch 2: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 27s 341ms/step - loss: 0.3884 -
accuracy: 0.8794 - val_loss: 0.4213 - val_accuracy: 0.8717
Epoch 3/5
78/78 [==============================] - ETA: 0s - loss: 0.3779 - accuracy:
0.8786
Epoch 3: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 35s 439ms/step - loss: 0.3779 -
accuracy: 0.8786 - val_loss: 0.4404 - val_accuracy: 0.8576
Epoch 4/5
78/78 [==============================] - ETA: 0s - loss: 0.3703 - accuracy:
0.8786
Epoch 4: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 28s 348ms/step - loss: 0.3703 -
accuracy: 0.8786 - val_loss: 0.4173 - val_accuracy: 0.8701
Epoch 5/5
78/78 [==============================] - ETA: 0s - loss: 0.3589 - accuracy:
0.8818
Epoch 5: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 28s 356ms/step - loss: 0.3589 -
accuracy: 0.8818 - val_loss: 0.4039 - val_accuracy: 0.8670
```

[56]:
```
# Evaluate the model training plot
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(vgg19_model_v2, test_ds1, classes)
```




```
11/11 - 2s - 2s/epoch - 159ms/step
```



```
           class  precision  recall
0  indian market       0.98    0.65
1          onion       0.57    0.75
2         potato       0.80    0.75
3         tomato       0.94    1.00
Overall accuracy: 80.3%, macro precision: 0.82, macro recall: 0.79
```

Observations

We see further improvement in test and validation accuracy after running the same model for 10 more epochs. The test accuracy is at 80% and validation accuracy at 87.7%. We also see training and validation loss curves plateauing. Next, we will try data augmentation by applying random crop, flip, translation operations.

[46]:
```
#train on augmented dataset

model_name = 'vgg19_v2.1'

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

vgg19_model_v2_1, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_vgg19,
          tf.keras.layers.GlobalAveragePooling2D(),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=20, callbacks=callbacks),

    #training data
    train_ds3,

    #validation data
    val_ds3,
```

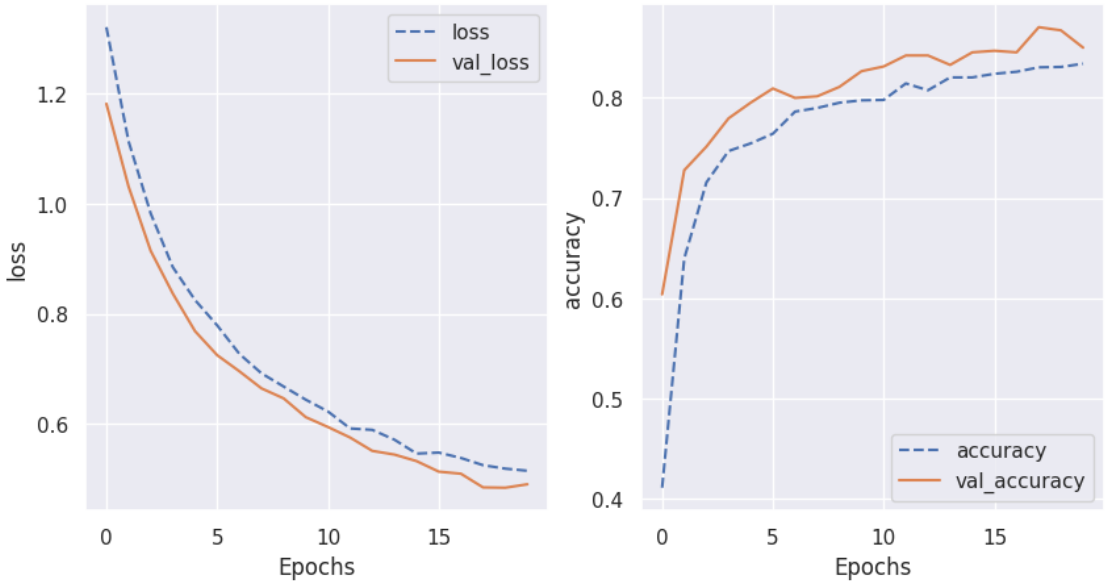
```
    )
```

```
Epoch 1/20
78/78 [==============================] - ETA: 0s - loss: 1.3209 - accuracy:
0.4115
Epoch 1: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 77s 946ms/step - loss: 1.3209 -
accuracy: 0.4115 - val_loss: 1.1817 - val_accuracy: 0.6041 - lr: 5.0000e-04
Epoch 2/20
78/78 [==============================] - ETA: 0s - loss: 1.1136 - accuracy:
0.6398
Epoch 2: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 66s 838ms/step - loss: 1.1136 -
accuracy: 0.6398 - val_loss: 1.0313 - val_accuracy: 0.7277 - lr: 5.0000e-04
Epoch 3/20
78/78 [==============================] - ETA: 0s - loss: 0.9823 - accuracy:
0.7155
Epoch 3: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 68s 858ms/step - loss: 0.9823 -
accuracy: 0.7155 - val_loss: 0.9143 - val_accuracy: 0.7512 - lr: 5.0000e-04
Epoch 4/20
78/78 [==============================] - ETA: 0s - loss: 0.8846 - accuracy:
0.7468
Epoch 4: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 66s 830ms/step - loss: 0.8846 -
accuracy: 0.7468 - val_loss: 0.8365 - val_accuracy: 0.7793 - lr: 5.0000e-04
Epoch 5/20
78/78 [==============================] - ETA: 0s - loss: 0.8250 - accuracy:
0.7544
Epoch 5: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 75s 939ms/step - loss: 0.8250 -
accuracy: 0.7544 - val_loss: 0.7684 - val_accuracy: 0.7950 - lr: 5.0000e-04
Epoch 6/20
78/78 [==============================] - ETA: 0s - loss: 0.7789 - accuracy:
0.7640
Epoch 6: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 76s 940ms/step - loss: 0.7789 -
accuracy: 0.7640 - val_loss: 0.7247 - val_accuracy: 0.8091 - lr: 5.0000e-04
Epoch 7/20
78/78 [==============================] - ETA: 0s - loss: 0.7272 - accuracy:
0.7861
Epoch 7: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 69s 856ms/step - loss: 0.7272 -
accuracy: 0.7861 - val_loss: 0.6955 - val_accuracy: 0.7997 - lr: 5.0000e-04
Epoch 8/20
78/78 [==============================] - ETA: 0s - loss: 0.6917 - accuracy:
0.7897
```

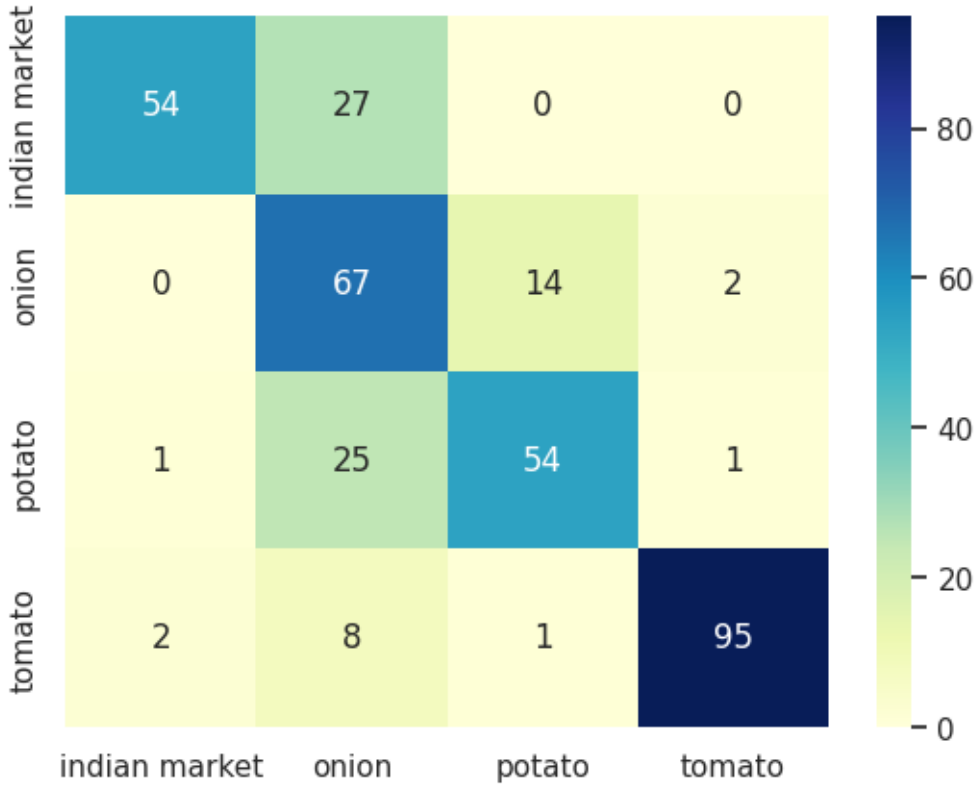
```
Epoch 8: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 68s 859ms/step - loss: 0.6917 -
accuracy: 0.7897 - val_loss: 0.6643 - val_accuracy: 0.8013 - lr: 5.0000e-04
Epoch 9/20
78/78 [==============================] - ETA: 0s - loss: 0.6677 - accuracy:
0.7949
Epoch 9: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 68s 859ms/step - loss: 0.6677 -
accuracy: 0.7949 - val_loss: 0.6462 - val_accuracy: 0.8106 - lr: 5.0000e-04
Epoch 10/20
78/78 [==============================] - ETA: 0s - loss: 0.6437 - accuracy:
0.7973
Epoch 10: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 77s 969ms/step - loss: 0.6437 -
accuracy: 0.7973 - val_loss: 0.6120 - val_accuracy: 0.8263 - lr: 5.0000e-04
Epoch 11/20
78/78 [==============================] - ETA: 0s - loss: 0.6224 - accuracy:
0.7977
Epoch 11: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 68s 835ms/step - loss: 0.6224 -
accuracy: 0.7977 - val_loss: 0.5942 - val_accuracy: 0.8310 - lr: 5.0000e-04
Epoch 12/20
78/78 [==============================] - ETA: 0s - loss: 0.5915 - accuracy:
0.8141
Epoch 12: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 72s 891ms/step - loss: 0.5915 -
accuracy: 0.8141 - val_loss: 0.5750 - val_accuracy: 0.8419 - lr: 5.0000e-04
Epoch 13/20
78/78 [==============================] - ETA: 0s - loss: 0.5890 - accuracy:
0.8073
Epoch 13: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 66s 825ms/step - loss: 0.5890 -
accuracy: 0.8073 - val_loss: 0.5507 - val_accuracy: 0.8419 - lr: 5.0000e-04
Epoch 14/20
78/78 [==============================] - ETA: 0s - loss: 0.5711 - accuracy:
0.8201
Epoch 14: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 65s 820ms/step - loss: 0.5711 -
accuracy: 0.8201 - val_loss: 0.5440 - val_accuracy: 0.8326 - lr: 5.0000e-04
Epoch 15/20
78/78 [==============================] - ETA: 0s - loss: 0.5458 - accuracy:
0.8201
Epoch 15: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 73s 921ms/step - loss: 0.5458 -
accuracy: 0.8201 - val_loss: 0.5321 - val_accuracy: 0.8451 - lr: 5.0000e-04
Epoch 16/20
78/78 [==============================] - ETA: 0s - loss: 0.5476 - accuracy:
0.8237
```

```
Epoch 16: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 76s 945ms/step - loss: 0.5476 -
accuracy: 0.8237 - val_loss: 0.5130 - val_accuracy: 0.8466 - lr: 5.0000e-04
Epoch 17/20
78/78 [==============================] - ETA: 0s - loss: 0.5379 - accuracy:
0.8257
Epoch 17: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 65s 815ms/step - loss: 0.5379 -
accuracy: 0.8257 - val_loss: 0.5092 - val_accuracy: 0.8451 - lr: 5.0000e-04
Epoch 18/20
78/78 [==============================] - ETA: 0s - loss: 0.5247 - accuracy:
0.8301
Epoch 18: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 65s 822ms/step - loss: 0.5247 -
accuracy: 0.8301 - val_loss: 0.4844 - val_accuracy: 0.8701 - lr: 5.0000e-04
Epoch 19/20
78/78 [==============================] - ETA: 0s - loss: 0.5185 - accuracy:
0.8305
Epoch 19: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 66s 828ms/step - loss: 0.5185 -
accuracy: 0.8305 - val_loss: 0.4838 - val_accuracy: 0.8670 - lr: 5.0000e-04
Epoch 20/20
78/78 [==============================] - ETA: 0s - loss: 0.5146 - accuracy:
0.8337
Epoch 20: saving model to ./checkpoints_test/vgg19_v1
78/78 [==============================] - 62s 787ms/step - loss: 0.5146 -
accuracy: 0.8337 - val_loss: 0.4900 - val_accuracy: 0.8498 - lr: 5.0000e-04
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 16). These functions will not be directly callable after loading.
```

[47]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(vgg19_model_v2_1, test_ds3, classes)
```




```
11/11 - 3s - 3s/epoch - 279ms/step
```



```
           class  precision  recall
0  indian market       0.95    0.67
1          onion       0.53    0.81
2         potato       0.78    0.67
3         tomato       0.97    0.90
Overall accuracy: 76.9%, macro precision: 0.81, macro recall: 0.76
```

Observations:

We observe that data augmentation didn't improve the overall accuracy Infact, it has degraded the accuracy by 3 percent points. Next, we will try adding an additional dense layer in the custom model, and also adding dropouts after each dense layer.

[57]:
```
#add a fully connected layer in classifier head.

model_name = 'vgg19_v3'

optimizer = keras.optimizers.Adam(learning_rate=0.0005)

vgg19_model_v3, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_vgg19,
          tf.keras.layers.GlobalAveragePooling2D(),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dropout(0.2),
          tf.keras.layers.Dense(1024, activation='relu'),
          tf.keras.layers.Dropout(0.2),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10, callbacks=callbacks),

    #training data
    train_ds1,

    #validation data
```

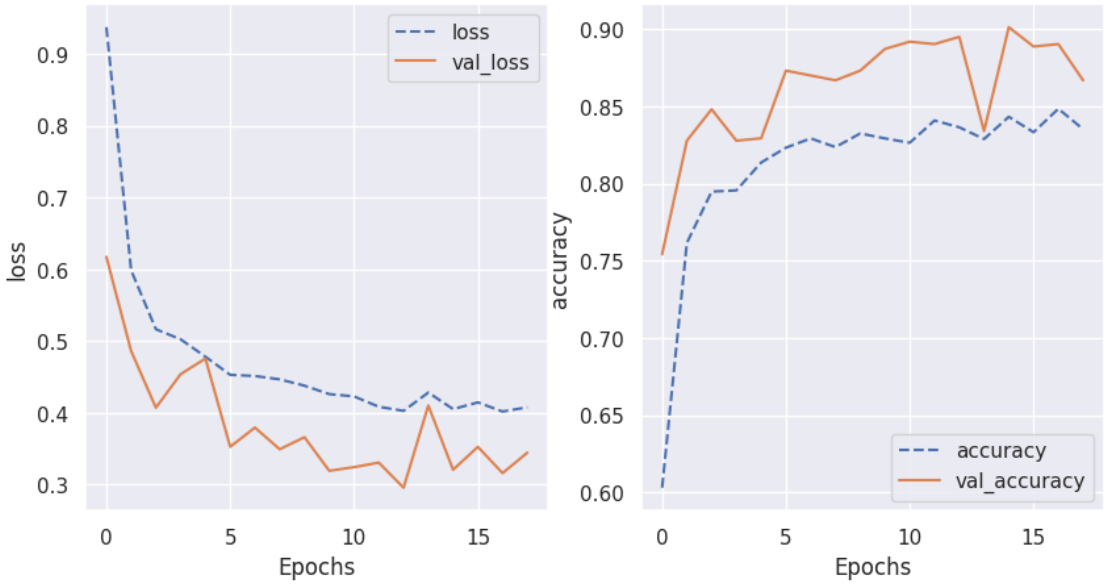
```
    val_ds1,

    )
```

```
saved model found. loading it..
```

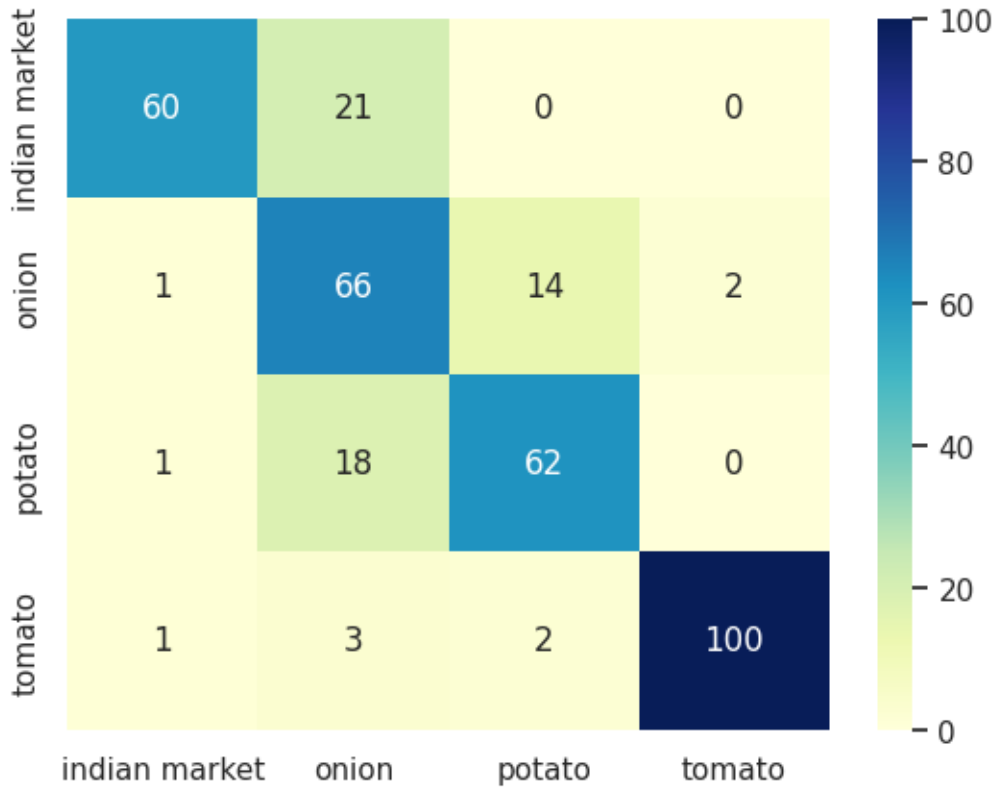
[24]:
```
vgg19_model_v3.summary()
```

```
Model: "vgg19_v3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d_2   (None, 512)              0
 (GlobalAveragePooling2D)

 flatten_9 (Flatten)         (None, 512)               0

 dropout_12 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 1024)              525312

 dropout_13 (Dropout)        (None, 1024)              0

 dense_22 (Dense)            (None, 4)                 4100

=================================================================
Total params: 20,553,796
Trainable params: 529,412
Non-trainable params: 20,024,384
_________________________________________________________________
```

[23]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(vgg19_model_v3, test_ds3, classes)
```




```
11/11 - 24s - 24s/epoch - 2s/step
```



```
           class  precision  recall
0  indian market       0.95    0.74
1          onion       0.61    0.80
2         potato       0.79    0.77
3         tomato       0.98    0.94
Overall accuracy: 82.1%, macro precision: 0.83, macro recall: 0.81
```

Observation:

We see that test accuracy is 82% while the validation accuracy is 0.87. While this is better compared to the V2 model, overall, V1 model provides highest test accuracy at ~85%

1.9 Extending Efficientnet

In this section, we will extend pretrained efficientnet V2 B0 architecture.

Load EfficientNet pretrained model

[37]:
```
#!pip install -U efficientnet
from tensorflow.keras.applications import EfficientNetV2B0

#from tensorflow.keras.applications.EfficientNetB0 import EfficientNetB0 as efn

pretrained_efn = EfficientNetV2B0(
    input_shape = (224, 224, 3),
    include_top = False,
    weights = 'imagenet')

pretrained_efn.trainable=False

pretrained_efn.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/efficientnet_v2/efficientnetv2-b0_notop.h5
24274472/24274472 [==============================] - 2s 0us/step
Model: "efficientnetv2-b0"
__________________________________________________________________________________
_________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
=================
 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 rescaling_6 (Rescaling)        (None, 224, 224, 3)  0
['input_2[0][0]']

 normalization_1 (Normalization  (None, 224, 224, 3)  0
['rescaling_6[0][0]']
```

```
 )

 stem_conv (Conv2D)             (None, 112, 112, 32  864
['normalization_1[0][0]']
                                )

 stem_bn (BatchNormalization)   (None, 112, 112, 32  128
['stem_conv[0][0]']
                                )

 stem_activation (Activation)   (None, 112, 112, 32  0
['stem_bn[0][0]']
                                )

 block1a_project_conv (Conv2D)  (None, 112, 112, 16  4608
['stem_activation[0][0]']
                                )

 block1a_project_bn (BatchNorma  (None, 112, 112, 16  64
['block1a_project_conv[0][0]']
 lization)                      )

 block1a_project_activation (Ac  (None, 112, 112, 16  0
['block1a_project_bn[0][0]']
 tivation)                      )

 block2a_expand_conv (Conv2D)   (None, 56, 56, 64)   9216
['block1a_project_activation[0][0
                                                                  ]']

 block2a_expand_bn (BatchNormal  (None, 56, 56, 64)  256
['block2a_expand_conv[0][0]']
 ization)

 block2a_expand_activation (Act  (None, 56, 56, 64)  0
['block2a_expand_bn[0][0]']
 ivation)

 block2a_project_conv (Conv2D)  (None, 56, 56, 32)   2048
['block2a_expand_activation[0][0]
                                                                  ']

 block2a_project_bn (BatchNorma  (None, 56, 56, 32)  128
['block2a_project_conv[0][0]']
 lization)

 block2b_expand_conv (Conv2D)   (None, 56, 56, 128)  36864
['block2a_project_bn[0][0]']
```

```
 block2b_expand_bn (BatchNormal  (None, 56, 56, 128)  512
['block2b_expand_conv[0][0]']
 ization)

 block2b_expand_activation (Act  (None, 56, 56, 128)  0
['block2b_expand_bn[0][0]']
 ivation)

 block2b_project_conv (Conv2D)  (None, 56, 56, 32)   4096
['block2b_expand_activation[0][0]
                                                                 ']

 block2b_project_bn (BatchNorma  (None, 56, 56, 32)  128
['block2b_project_conv[0][0]']
 lization)

 block2b_drop (Dropout)         (None, 56, 56, 32)   0
['block2b_project_bn[0][0]']

 block2b_add (Add)              (None, 56, 56, 32)   0
['block2b_drop[0][0]',
'block2a_project_bn[0][0]']

 block3a_expand_conv (Conv2D)   (None, 28, 28, 128)  36864
['block2b_add[0][0]']

 block3a_expand_bn (BatchNormal  (None, 28, 28, 128)  512
['block3a_expand_conv[0][0]']
 ization)

 block3a_expand_activation (Act  (None, 28, 28, 128)  0
['block3a_expand_bn[0][0]']
 ivation)

 block3a_project_conv (Conv2D)  (None, 28, 28, 48)   6144
['block3a_expand_activation[0][0]
                                                                 ']

 block3a_project_bn (BatchNorma  (None, 28, 28, 48)  192
['block3a_project_conv[0][0]']
 lization)

 block3b_expand_conv (Conv2D)   (None, 28, 28, 192)  82944
['block3a_project_bn[0][0]']

 block3b_expand_bn (BatchNormal  (None, 28, 28, 192)  768
['block3b_expand_conv[0][0]']
```

```
 ization)

 block3b_expand_activation (Act  (None, 28, 28, 192)  0
['block3b_expand_bn[0][0]']
 ivation)

 block3b_project_conv (Conv2D)  (None, 28, 28, 48)   9216
['block3b_expand_activation[0][0]
                                                                 ']

 block3b_project_bn (BatchNorma  (None, 28, 28, 48)  192
['block3b_project_conv[0][0]']
 lization)

 block3b_drop (Dropout)         (None, 28, 28, 48)   0
['block3b_project_bn[0][0]']

 block3b_add (Add)              (None, 28, 28, 48)   0
['block3b_drop[0][0]',
'block3a_project_bn[0][0]']

 block4a_expand_conv (Conv2D)   (None, 28, 28, 192)  9216
['block3b_add[0][0]']

 block4a_expand_bn (BatchNormal  (None, 28, 28, 192)  768
['block4a_expand_conv[0][0]']
 ization)

 block4a_expand_activation (Act  (None, 28, 28, 192)  0
['block4a_expand_bn[0][0]']
 ivation)

 block4a_dwconv2 (DepthwiseConv  (None, 14, 14, 192)  1728
['block4a_expand_activation[0][0]
 2D)                                                             ']

 block4a_bn (BatchNormalization  (None, 14, 14, 192)  768
['block4a_dwconv2[0][0]']
 )

 block4a_activation (Activation  (None, 14, 14, 192)  0
['block4a_bn[0][0]']
 )

 block4a_se_squeeze (GlobalAver  (None, 192)         0
['block4a_activation[0][0]']
 agePooling2D)
```

```
 block4a_se_reshape (Reshape)   (None, 1, 1, 192)    0
['block4a_se_squeeze[0][0]']

 block4a_se_reduce (Conv2D)     (None, 1, 1, 12)     2316
['block4a_se_reshape[0][0]']

 block4a_se_expand (Conv2D)     (None, 1, 1, 192)    2496
['block4a_se_reduce[0][0]']

 block4a_se_excite (Multiply)   (None, 14, 14, 192)  0
['block4a_activation[0][0]',
'block4a_se_expand[0][0]']

 block4a_project_conv (Conv2D)  (None, 14, 14, 96)   18432
['block4a_se_excite[0][0]']

 block4a_project_bn (BatchNorma  (None, 14, 14, 96)  384
['block4a_project_conv[0][0]']
 lization)

 block4b_expand_conv (Conv2D)   (None, 14, 14, 384)  36864
['block4a_project_bn[0][0]']

 block4b_expand_bn (BatchNormal  (None, 14, 14, 384)  1536
['block4b_expand_conv[0][0]']
 ization)

 block4b_expand_activation (Act  (None, 14, 14, 384)  0
['block4b_expand_bn[0][0]']
 ivation)

 block4b_dwconv2 (DepthwiseConv  (None, 14, 14, 384)  3456
['block4b_expand_activation[0][0]
 2D)                                                              ']

 block4b_bn (BatchNormalization  (None, 14, 14, 384)  1536
['block4b_dwconv2[0][0]']
 )

 block4b_activation (Activation  (None, 14, 14, 384)  0
['block4b_bn[0][0]']
 )

 block4b_se_squeeze (GlobalAver  (None, 384)         0
['block4b_activation[0][0]']
 agePooling2D)

 block4b_se_reshape (Reshape)   (None, 1, 1, 384)    0
```

```
['block4b_se_squeeze[0][0]']

 block4b_se_reduce (Conv2D)     (None, 1, 1, 24)     9240
['block4b_se_reshape[0][0]']

 block4b_se_expand (Conv2D)     (None, 1, 1, 384)    9600
['block4b_se_reduce[0][0]']

 block4b_se_excite (Multiply)   (None, 14, 14, 384)  0
['block4b_activation[0][0]',
'block4b_se_expand[0][0]']

 block4b_project_conv (Conv2D)  (None, 14, 14, 96)   36864
['block4b_se_excite[0][0]']

 block4b_project_bn (BatchNorma  (None, 14, 14, 96)  384
['block4b_project_conv[0][0]']
 lization)

 block4b_drop (Dropout)         (None, 14, 14, 96)   0
['block4b_project_bn[0][0]']

 block4b_add (Add)              (None, 14, 14, 96)   0
['block4b_drop[0][0]',
'block4a_project_bn[0][0]']

 block4c_expand_conv (Conv2D)   (None, 14, 14, 384)  36864
['block4b_add[0][0]']

 block4c_expand_bn (BatchNormal  (None, 14, 14, 384)  1536
['block4c_expand_conv[0][0]']
 ization)

 block4c_expand_activation (Act  (None, 14, 14, 384)  0
['block4c_expand_bn[0][0]']
 ivation)

 block4c_dwconv2 (DepthwiseConv  (None, 14, 14, 384)  3456
['block4c_expand_activation[0][0]
 2D)                                                              ']

 block4c_bn (BatchNormalization  (None, 14, 14, 384)  1536
['block4c_dwconv2[0][0]']
 )

 block4c_activation (Activation  (None, 14, 14, 384)  0
['block4c_bn[0][0]']
 )
```

```
 block4c_se_squeeze (GlobalAver  (None, 384)         0
['block4c_activation[0][0]']
 agePooling2D)

 block4c_se_reshape (Reshape)   (None, 1, 1, 384)    0
['block4c_se_squeeze[0][0]']

 block4c_se_reduce (Conv2D)     (None, 1, 1, 24)     9240
['block4c_se_reshape[0][0]']

 block4c_se_expand (Conv2D)     (None, 1, 1, 384)    9600
['block4c_se_reduce[0][0]']

 block4c_se_excite (Multiply)   (None, 14, 14, 384)  0
['block4c_activation[0][0]',
'block4c_se_expand[0][0]']

 block4c_project_conv (Conv2D)  (None, 14, 14, 96)   36864
['block4c_se_excite[0][0]']

 block4c_project_bn (BatchNorma  (None, 14, 14, 96)  384
['block4c_project_conv[0][0]']
 lization)

 block4c_drop (Dropout)         (None, 14, 14, 96)   0
['block4c_project_bn[0][0]']

 block4c_add (Add)              (None, 14, 14, 96)   0
['block4c_drop[0][0]',
'block4b_add[0][0]']

 block5a_expand_conv (Conv2D)   (None, 14, 14, 576)  55296
['block4c_add[0][0]']

 block5a_expand_bn (BatchNormal  (None, 14, 14, 576)  2304
['block5a_expand_conv[0][0]']
 ization)

 block5a_expand_activation (Act  (None, 14, 14, 576)  0
['block5a_expand_bn[0][0]']
 ivation)

 block5a_dwconv2 (DepthwiseConv  (None, 14, 14, 576)  5184
['block5a_expand_activation[0][0]
 2D)                                                              ']

 block5a_bn (BatchNormalization  (None, 14, 14, 576)  2304
```

```
['block5a_dwconv2[0][0]']
 )

 block5a_activation (Activation  (None, 14, 14, 576)  0
['block5a_bn[0][0]']
 )

 block5a_se_squeeze (GlobalAver  (None, 576)         0
['block5a_activation[0][0]']
 agePooling2D)

 block5a_se_reshape (Reshape)   (None, 1, 1, 576)    0
['block5a_se_squeeze[0][0]']

 block5a_se_reduce (Conv2D)     (None, 1, 1, 24)     13848
['block5a_se_reshape[0][0]']

 block5a_se_expand (Conv2D)     (None, 1, 1, 576)    14400
['block5a_se_reduce[0][0]']

 block5a_se_excite (Multiply)   (None, 14, 14, 576)  0
['block5a_activation[0][0]',
'block5a_se_expand[0][0]']

 block5a_project_conv (Conv2D)  (None, 14, 14, 112)  64512
['block5a_se_excite[0][0]']

 block5a_project_bn (BatchNorma  (None, 14, 14, 112)  448
['block5a_project_conv[0][0]']
 lization)

 block5b_expand_conv (Conv2D)   (None, 14, 14, 672)  75264
['block5a_project_bn[0][0]']

 block5b_expand_bn (BatchNormal  (None, 14, 14, 672)  2688
['block5b_expand_conv[0][0]']
 ization)

 block5b_expand_activation (Act  (None, 14, 14, 672)  0
['block5b_expand_bn[0][0]']
 ivation)

 block5b_dwconv2 (DepthwiseConv  (None, 14, 14, 672)  6048
['block5b_expand_activation[0][0]
 2D)                                                                ']

 block5b_bn (BatchNormalization  (None, 14, 14, 672)  2688
['block5b_dwconv2[0][0]']
```

```
 )

 block5b_activation (Activation  (None, 14, 14, 672)  0
['block5b_bn[0][0]']
 )

 block5b_se_squeeze (GlobalAver  (None, 672)         0
['block5b_activation[0][0]']
 agePooling2D)

 block5b_se_reshape (Reshape)   (None, 1, 1, 672)    0
['block5b_se_squeeze[0][0]']

 block5b_se_reduce (Conv2D)     (None, 1, 1, 28)     18844
['block5b_se_reshape[0][0]']

 block5b_se_expand (Conv2D)     (None, 1, 1, 672)    19488
['block5b_se_reduce[0][0]']

 block5b_se_excite (Multiply)   (None, 14, 14, 672)  0
['block5b_activation[0][0]',
'block5b_se_expand[0][0]']

 block5b_project_conv (Conv2D)  (None, 14, 14, 112)  75264
['block5b_se_excite[0][0]']

 block5b_project_bn (BatchNorma  (None, 14, 14, 112)  448
['block5b_project_conv[0][0]']
 lization)

 block5b_drop (Dropout)         (None, 14, 14, 112)  0
['block5b_project_bn[0][0]']

 block5b_add (Add)              (None, 14, 14, 112)  0
['block5b_drop[0][0]',
'block5a_project_bn[0][0]']

 block5c_expand_conv (Conv2D)   (None, 14, 14, 672)  75264
['block5b_add[0][0]']

 block5c_expand_bn (BatchNormal  (None, 14, 14, 672)  2688
['block5c_expand_conv[0][0]']
 ization)

 block5c_expand_activation (Act  (None, 14, 14, 672)  0
['block5c_expand_bn[0][0]']
 ivation)
```

```
 block5c_dwconv2 (DepthwiseConv  (None, 14, 14, 672)  6048
['block5c_expand_activation[0][0]
 2D)                                                              ']

 block5c_bn (BatchNormalization  (None, 14, 14, 672)  2688
['block5c_dwconv2[0][0]']
 )

 block5c_activation (Activation  (None, 14, 14, 672)  0
['block5c_bn[0][0]']
 )

 block5c_se_squeeze (GlobalAver  (None, 672)         0
['block5c_activation[0][0]']
 agePooling2D)

 block5c_se_reshape (Reshape)   (None, 1, 1, 672)    0
['block5c_se_squeeze[0][0]']

 block5c_se_reduce (Conv2D)     (None, 1, 1, 28)     18844
['block5c_se_reshape[0][0]']

 block5c_se_expand (Conv2D)     (None, 1, 1, 672)    19488
['block5c_se_reduce[0][0]']

 block5c_se_excite (Multiply)   (None, 14, 14, 672)  0
['block5c_activation[0][0]',
'block5c_se_expand[0][0]']

 block5c_project_conv (Conv2D)  (None, 14, 14, 112)  75264
['block5c_se_excite[0][0]']

 block5c_project_bn (BatchNorma  (None, 14, 14, 112)  448
['block5c_project_conv[0][0]']
 lization)

 block5c_drop (Dropout)         (None, 14, 14, 112)  0
['block5c_project_bn[0][0]']

 block5c_add (Add)              (None, 14, 14, 112)  0
['block5c_drop[0][0]',
'block5b_add[0][0]']

 block5d_expand_conv (Conv2D)   (None, 14, 14, 672)  75264
['block5c_add[0][0]']

 block5d_expand_bn (BatchNormal  (None, 14, 14, 672)  2688
['block5d_expand_conv[0][0]']
```

```
 ization)

 block5d_expand_activation (Act  (None, 14, 14, 672)  0
['block5d_expand_bn[0][0]']
 ivation)

 block5d_dwconv2 (DepthwiseConv  (None, 14, 14, 672)  6048
['block5d_expand_activation[0][0]
 2D)                                                              ']

 block5d_bn (BatchNormalization  (None, 14, 14, 672)  2688
['block5d_dwconv2[0][0]']
 )

 block5d_activation (Activation  (None, 14, 14, 672)  0
['block5d_bn[0][0]']
 )

 block5d_se_squeeze (GlobalAver  (None, 672)         0
['block5d_activation[0][0]']
 agePooling2D)

 block5d_se_reshape (Reshape)   (None, 1, 1, 672)    0
['block5d_se_squeeze[0][0]']

 block5d_se_reduce (Conv2D)     (None, 1, 1, 28)     18844
['block5d_se_reshape[0][0]']

 block5d_se_expand (Conv2D)     (None, 1, 1, 672)    19488
['block5d_se_reduce[0][0]']

 block5d_se_excite (Multiply)   (None, 14, 14, 672)  0
['block5d_activation[0][0]',
'block5d_se_expand[0][0]']

 block5d_project_conv (Conv2D)  (None, 14, 14, 112)  75264
['block5d_se_excite[0][0]']

 block5d_project_bn (BatchNorma  (None, 14, 14, 112)  448
['block5d_project_conv[0][0]']
 lization)

 block5d_drop (Dropout)         (None, 14, 14, 112)  0
['block5d_project_bn[0][0]']

 block5d_add (Add)              (None, 14, 14, 112)  0
['block5d_drop[0][0]',
'block5c_add[0][0]']
```

```
 block5e_expand_conv (Conv2D)   (None, 14, 14, 672)  75264
['block5d_add[0][0]']

 block5e_expand_bn (BatchNormal  (None, 14, 14, 672)  2688
['block5e_expand_conv[0][0]']
 ization)

 block5e_expand_activation (Act  (None, 14, 14, 672)  0
['block5e_expand_bn[0][0]']
 ivation)

 block5e_dwconv2 (DepthwiseConv  (None, 14, 14, 672)  6048
['block5e_expand_activation[0][0]
 2D)                                                              ']

 block5e_bn (BatchNormalization  (None, 14, 14, 672)  2688
['block5e_dwconv2[0][0]']
 )

 block5e_activation (Activation  (None, 14, 14, 672)  0
['block5e_bn[0][0]']
 )

 block5e_se_squeeze (GlobalAver  (None, 672)         0
['block5e_activation[0][0]']
 agePooling2D)

 block5e_se_reshape (Reshape)   (None, 1, 1, 672)    0
['block5e_se_squeeze[0][0]']

 block5e_se_reduce (Conv2D)     (None, 1, 1, 28)     18844
['block5e_se_reshape[0][0]']

 block5e_se_expand (Conv2D)     (None, 1, 1, 672)    19488
['block5e_se_reduce[0][0]']

 block5e_se_excite (Multiply)   (None, 14, 14, 672)  0
['block5e_activation[0][0]',
'block5e_se_expand[0][0]']

 block5e_project_conv (Conv2D)  (None, 14, 14, 112)  75264
['block5e_se_excite[0][0]']

 block5e_project_bn (BatchNorma  (None, 14, 14, 112)  448
['block5e_project_conv[0][0]']
 lization)
```

```
 block5e_drop (Dropout)         (None, 14, 14, 112)  0
['block5e_project_bn[0][0]']

 block5e_add (Add)              (None, 14, 14, 112)  0
['block5e_drop[0][0]',
'block5d_add[0][0]']

 block6a_expand_conv (Conv2D)   (None, 14, 14, 672)  75264
['block5e_add[0][0]']

 block6a_expand_bn (BatchNormal  (None, 14, 14, 672)  2688
['block6a_expand_conv[0][0]']
 ization)

 block6a_expand_activation (Act  (None, 14, 14, 672)  0
['block6a_expand_bn[0][0]']
 ivation)

 block6a_dwconv2 (DepthwiseConv  (None, 7, 7, 672)   6048
['block6a_expand_activation[0][0]
 2D)                                                              ']

 block6a_bn (BatchNormalization  (None, 7, 7, 672)   2688
['block6a_dwconv2[0][0]']
 )

 block6a_activation (Activation  (None, 7, 7, 672)   0
['block6a_bn[0][0]']
 )

 block6a_se_squeeze (GlobalAver  (None, 672)         0
['block6a_activation[0][0]']
 agePooling2D)

 block6a_se_reshape (Reshape)   (None, 1, 1, 672)    0
['block6a_se_squeeze[0][0]']

 block6a_se_reduce (Conv2D)     (None, 1, 1, 28)     18844
['block6a_se_reshape[0][0]']

 block6a_se_expand (Conv2D)     (None, 1, 1, 672)    19488
['block6a_se_reduce[0][0]']

 block6a_se_excite (Multiply)   (None, 7, 7, 672)    0
['block6a_activation[0][0]',
'block6a_se_expand[0][0]']

 block6a_project_conv (Conv2D)  (None, 7, 7, 192)    129024
```

```
['block6a_se_excite[0][0]']

 block6a_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6a_project_conv[0][0]']
 lization)

 block6b_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6a_project_bn[0][0]']

 block6b_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6b_expand_conv[0][0]']
 ization)

 block6b_expand_activation (Act  (None, 7, 7, 1152)  0
['block6b_expand_bn[0][0]']
 ivation)

 block6b_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6b_expand_activation[0][0]
 2D)                                                              ']

 block6b_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6b_dwconv2[0][0]']
 )

 block6b_activation (Activation  (None, 7, 7, 1152)  0
['block6b_bn[0][0]']
 )

 block6b_se_squeeze (GlobalAver  (None, 1152)        0
['block6b_activation[0][0]']
 agePooling2D)

 block6b_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6b_se_squeeze[0][0]']

 block6b_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6b_se_reshape[0][0]']

 block6b_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6b_se_reduce[0][0]']

 block6b_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6b_activation[0][0]',
'block6b_se_expand[0][0]']

 block6b_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6b_se_excite[0][0]']
```

```
 block6b_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6b_project_conv[0][0]']
 lization)

 block6b_drop (Dropout)         (None, 7, 7, 192)    0
['block6b_project_bn[0][0]']

 block6b_add (Add)              (None, 7, 7, 192)    0
['block6b_drop[0][0]',
'block6a_project_bn[0][0]']

 block6c_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6b_add[0][0]']

 block6c_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6c_expand_conv[0][0]']
 ization)

 block6c_expand_activation (Act  (None, 7, 7, 1152)  0
['block6c_expand_bn[0][0]']
 ivation)

 block6c_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6c_expand_activation[0][0]
 2D)                                                              ']

 block6c_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6c_dwconv2[0][0]']
 )

 block6c_activation (Activation  (None, 7, 7, 1152)  0
['block6c_bn[0][0]']
 )

 block6c_se_squeeze (GlobalAver  (None, 1152)        0
['block6c_activation[0][0]']
 agePooling2D)

 block6c_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6c_se_squeeze[0][0]']

 block6c_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6c_se_reshape[0][0]']

 block6c_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6c_se_reduce[0][0]']
```

```
 block6c_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6c_activation[0][0]',
'block6c_se_expand[0][0]']

 block6c_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6c_se_excite[0][0]']

 block6c_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6c_project_conv[0][0]']
 lization)

 block6c_drop (Dropout)         (None, 7, 7, 192)    0
['block6c_project_bn[0][0]']

 block6c_add (Add)              (None, 7, 7, 192)    0
['block6c_drop[0][0]',
'block6b_add[0][0]']

 block6d_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6c_add[0][0]']

 block6d_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6d_expand_conv[0][0]']
 ization)

 block6d_expand_activation (Act  (None, 7, 7, 1152)  0
['block6d_expand_bn[0][0]']
 ivation)

 block6d_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6d_expand_activation[0][0]
 2D)                                                              ']

 block6d_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6d_dwconv2[0][0]']
 )

 block6d_activation (Activation  (None, 7, 7, 1152)  0
['block6d_bn[0][0]']
 )

 block6d_se_squeeze (GlobalAver  (None, 1152)        0
['block6d_activation[0][0]']
 agePooling2D)

 block6d_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6d_se_squeeze[0][0]']
```

```
 block6d_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6d_se_reshape[0][0]']

 block6d_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6d_se_reduce[0][0]']

 block6d_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6d_activation[0][0]',
'block6d_se_expand[0][0]']

 block6d_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6d_se_excite[0][0]']

 block6d_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6d_project_conv[0][0]']
 lization)

 block6d_drop (Dropout)         (None, 7, 7, 192)    0
['block6d_project_bn[0][0]']

 block6d_add (Add)              (None, 7, 7, 192)    0
['block6d_drop[0][0]',
'block6c_add[0][0]']

 block6e_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6d_add[0][0]']

 block6e_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6e_expand_conv[0][0]']
 ization)

 block6e_expand_activation (Act  (None, 7, 7, 1152)  0
['block6e_expand_bn[0][0]']
 ivation)

 block6e_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6e_expand_activation[0][0]
 2D)                                                              ']

 block6e_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6e_dwconv2[0][0]']
 )

 block6e_activation (Activation  (None, 7, 7, 1152)  0
['block6e_bn[0][0]']
 )

 block6e_se_squeeze (GlobalAver  (None, 1152)        0
```

```
['block6e_activation[0][0]']
 agePooling2D)

 block6e_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6e_se_squeeze[0][0]']

 block6e_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6e_se_reshape[0][0]']

 block6e_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6e_se_reduce[0][0]']

 block6e_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6e_activation[0][0]',
'block6e_se_expand[0][0]']

 block6e_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6e_se_excite[0][0]']

 block6e_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6e_project_conv[0][0]']
 lization)

 block6e_drop (Dropout)         (None, 7, 7, 192)    0
['block6e_project_bn[0][0]']

 block6e_add (Add)              (None, 7, 7, 192)    0
['block6e_drop[0][0]',
'block6d_add[0][0]']

 block6f_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6e_add[0][0]']

 block6f_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6f_expand_conv[0][0]']
 ization)

 block6f_expand_activation (Act  (None, 7, 7, 1152)  0
['block6f_expand_bn[0][0]']
 ivation)

 block6f_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6f_expand_activation[0][0]
 2D)                                                              ']

 block6f_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6f_dwconv2[0][0]']
 )
```

```
 block6f_activation (Activation  (None, 7, 7, 1152)  0
['block6f_bn[0][0]']
 )

 block6f_se_squeeze (GlobalAver  (None, 1152)        0
['block6f_activation[0][0]']
 agePooling2D)

 block6f_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6f_se_squeeze[0][0]']

 block6f_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6f_se_reshape[0][0]']

 block6f_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6f_se_reduce[0][0]']

 block6f_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6f_activation[0][0]',
'block6f_se_expand[0][0]']

 block6f_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6f_se_excite[0][0]']

 block6f_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6f_project_conv[0][0]']
 lization)

 block6f_drop (Dropout)         (None, 7, 7, 192)    0
['block6f_project_bn[0][0]']

 block6f_add (Add)              (None, 7, 7, 192)    0
['block6f_drop[0][0]',
'block6e_add[0][0]']

 block6g_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6f_add[0][0]']

 block6g_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6g_expand_conv[0][0]']
 ization)

 block6g_expand_activation (Act  (None, 7, 7, 1152)  0
['block6g_expand_bn[0][0]']
 ivation)

 block6g_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
```

```
['block6g_expand_activation[0][0]
 2D)                                                                ']

 block6g_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6g_dwconv2[0][0]']
 )

 block6g_activation (Activation  (None, 7, 7, 1152)  0
['block6g_bn[0][0]']
 )

 block6g_se_squeeze (GlobalAver  (None, 1152)        0
['block6g_activation[0][0]']
 agePooling2D)

 block6g_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6g_se_squeeze[0][0]']

 block6g_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6g_se_reshape[0][0]']

 block6g_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6g_se_reduce[0][0]']

 block6g_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6g_activation[0][0]',
'block6g_se_expand[0][0]']

 block6g_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6g_se_excite[0][0]']

 block6g_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6g_project_conv[0][0]']
 lization)

 block6g_drop (Dropout)         (None, 7, 7, 192)    0
['block6g_project_bn[0][0]']

 block6g_add (Add)              (None, 7, 7, 192)    0
['block6g_drop[0][0]',
'block6f_add[0][0]']

 block6h_expand_conv (Conv2D)   (None, 7, 7, 1152)   221184
['block6g_add[0][0]']

 block6h_expand_bn (BatchNormal  (None, 7, 7, 1152)  4608
['block6h_expand_conv[0][0]']
 ization)
```

```
 block6h_expand_activation (Act  (None, 7, 7, 1152)  0
['block6h_expand_bn[0][0]']
 ivation)

 block6h_dwconv2 (DepthwiseConv  (None, 7, 7, 1152)  10368
['block6h_expand_activation[0][0]
 2D)                                                              ']

 block6h_bn (BatchNormalization  (None, 7, 7, 1152)  4608
['block6h_dwconv2[0][0]']
 )

 block6h_activation (Activation  (None, 7, 7, 1152)  0
['block6h_bn[0][0]']
 )

 block6h_se_squeeze (GlobalAver  (None, 1152)        0
['block6h_activation[0][0]']
 agePooling2D)

 block6h_se_reshape (Reshape)   (None, 1, 1, 1152)   0
['block6h_se_squeeze[0][0]']

 block6h_se_reduce (Conv2D)     (None, 1, 1, 48)     55344
['block6h_se_reshape[0][0]']

 block6h_se_expand (Conv2D)     (None, 1, 1, 1152)   56448
['block6h_se_reduce[0][0]']

 block6h_se_excite (Multiply)   (None, 7, 7, 1152)   0
['block6h_activation[0][0]',
'block6h_se_expand[0][0]']

 block6h_project_conv (Conv2D)  (None, 7, 7, 192)    221184
['block6h_se_excite[0][0]']

 block6h_project_bn (BatchNorma  (None, 7, 7, 192)   768
['block6h_project_conv[0][0]']
 lization)

 block6h_drop (Dropout)         (None, 7, 7, 192)    0
['block6h_project_bn[0][0]']

 block6h_add (Add)              (None, 7, 7, 192)    0
['block6h_drop[0][0]',
'block6g_add[0][0]']
```

```
 top_conv (Conv2D)              (None, 7, 7, 1280)   245760
['block6h_add[0][0]']

 top_bn (BatchNormalization)    (None, 7, 7, 1280)   5120
['top_conv[0][0]']

 top_activation (Activation)    (None, 7, 7, 1280)   0
['top_bn[0][0]']

==================================================================================
==================
Total params: 5,919,312
Trainable params: 0
Non-trainable params: 5,919,312
----------------------------------------------------------------------------------
------------------
```

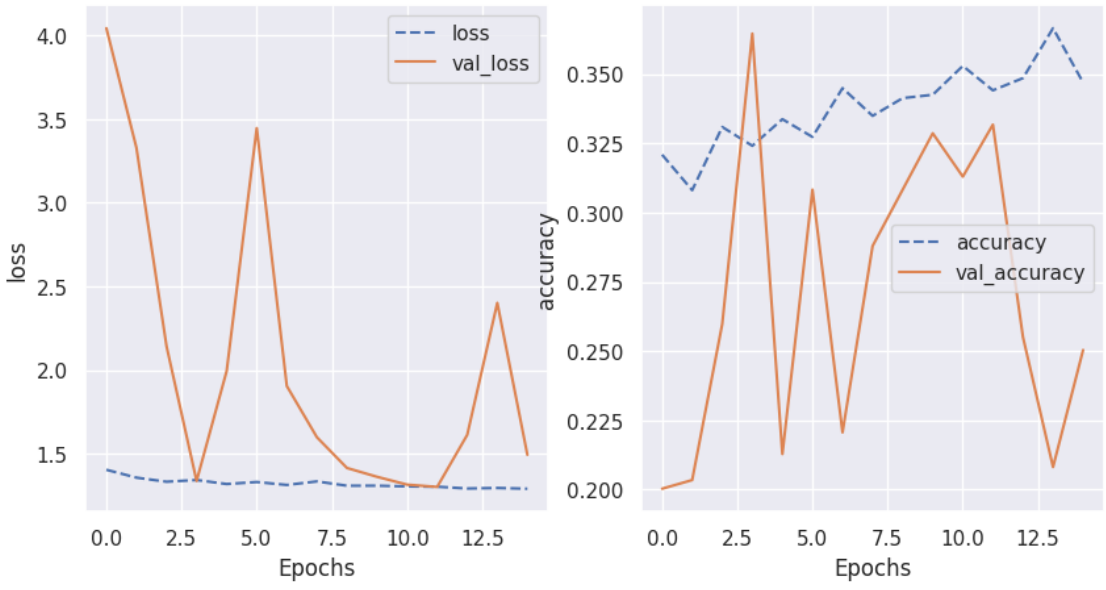
[41]:
```
model_name = 'efn_v1'

callbacks = [
  keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.3, patience=5, min_lr=0.000005
  ),
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, min_delta=0.001, mode='min',␣
  ↪restore_best_weights=True
  ),
  checkpoint_helper.getCallback(model_name)
]

optimizer = keras.optimizers.Adam(learning_rate=0.01)

efn_v1, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_efn,
          tf.keras.layers.GlobalMaxPooling2D(),
          tf.keras.layers.Dense(256, activation='relu'),
          tf.keras.layers.BatchNormalization(),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),
```

```
    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
↪val_data, epochs=15, callbacks=[]),

    #training data
    train_ds1.unbatch().batch(64),

    #validation data
    val_ds1.unbatch().batch(64),

    force_train = True
    )
```

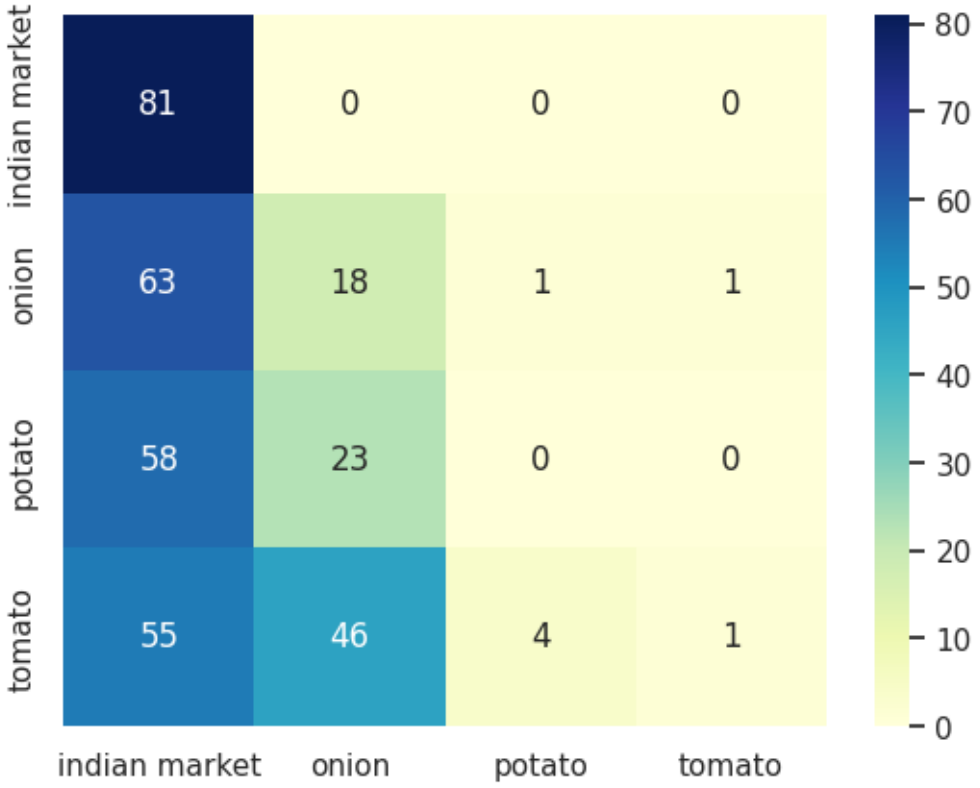
```
Epoch 1/15
39/39 [==============================] - 37s 679ms/step - loss: 1.4057 -
accuracy: 0.3209 - val_loss: 4.0430 - val_accuracy: 0.2003
Epoch 2/15
39/39 [==============================] - 33s 843ms/step - loss: 1.3587 -
accuracy: 0.3081 - val_loss: 3.3283 - val_accuracy: 0.2034
Epoch 3/15
39/39 [==============================] - 25s 628ms/step - loss: 1.3343 -
accuracy: 0.3309 - val_loss: 2.1441 - val_accuracy: 0.2598
Epoch 4/15
39/39 [==============================] - 25s 620ms/step - loss: 1.3446 -
accuracy: 0.3241 - val_loss: 1.3388 - val_accuracy: 0.3646
Epoch 5/15
39/39 [==============================] - 26s 661ms/step - loss: 1.3206 -
accuracy: 0.3337 - val_loss: 1.9956 - val_accuracy: 0.2128
Epoch 6/15
39/39 [==============================] - 24s 608ms/step - loss: 1.3326 -
accuracy: 0.3273 - val_loss: 3.4462 - val_accuracy: 0.3083
Epoch 7/15
39/39 [==============================] - 26s 641ms/step - loss: 1.3151 -
accuracy: 0.3450 - val_loss: 1.9059 - val_accuracy: 0.2207
Epoch 8/15
39/39 [==============================] - 24s 613ms/step - loss: 1.3361 -
accuracy: 0.3349 - val_loss: 1.6002 - val_accuracy: 0.2879
Epoch 9/15
39/39 [==============================] - 25s 618ms/step - loss: 1.3105 -
accuracy: 0.3413 - val_loss: 1.4164 - val_accuracy: 0.3083
Epoch 10/15
39/39 [==============================] - 26s 642ms/step - loss: 1.3107 -
accuracy: 0.3425 - val_loss: 1.3637 - val_accuracy: 0.3286
```

```
Epoch 11/15
39/39 [==============================] - 24s 604ms/step - loss: 1.3063 -
accuracy: 0.3530 - val_loss: 1.3169 - val_accuracy: 0.3130
Epoch 12/15
39/39 [==============================] - 22s 563ms/step - loss: 1.3045 -
accuracy: 0.3442 - val_loss: 1.3033 - val_accuracy: 0.3318
Epoch 13/15
39/39 [==============================] - 33s 814ms/step - loss: 1.2938 -
accuracy: 0.3486 - val_loss: 1.6143 - val_accuracy: 0.2551
Epoch 14/15
39/39 [==============================] - 24s 573ms/step - loss: 1.2971 -
accuracy: 0.3666 - val_loss: 2.4026 - val_accuracy: 0.2081
Epoch 15/15
39/39 [==============================] - 24s 604ms/step - loss: 1.2925 -
accuracy: 0.3470 - val_loss: 1.4950 - val_accuracy: 0.2504
saving model and history

WARNING:absl:Function `_wrapped_model` contains input name(s)
efficientnetv2-b0_input with unsupported characters which will be renamed to
efficientnetv2_b0_input in the SavedModel.
WARNING:absl:`efficientnetv2-b0_input` is not a valid tf.function parameter
name. Sanitizing to `efficientnetv2_b0_input`.
WARNING:absl:`efficientnetv2-b0_input` is not a valid tf.function parameter
name. Sanitizing to `efficientnetv2_b0_input`.
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 91). These functions will not be directly callable after loading.
```

[42]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(efn_v1, test_ds3, classes)
```



```
11/11 - 4s - 4s/epoch - 392ms/step
```




```
           class  precision  recall
0  indian market       0.32    1.00
1          onion       0.21    0.22
2         potato       0.00    0.00
3         tomato       0.50    0.01
Overall accuracy: 28.5%, macro precision: 0.26, macro recall: 0.31
```

Observations:

Surprisingly, our custom model based on the pretrained efficientnet V2 B0 model works very poorly. We tried several tricks: adjusting learning rate, changing batch size, applying data augmentation, adding denser fully connected layer (upto 1024), batch normalization, dropouts, etc (These were tried offline and not captured in this notebook). None of the options helped improve validation/test accuracy. The validation score always seemed to hover in the 0.2 - 0.35 range, while test score in the range 0.25-0.3. We need more investigation to understand what could be the issue for poor performance.

1.10 Extending Resnet50

Finally, in this section we will extend pretrained Resnet50V2 model. In this section, we will also use Tensorboard callback to track of training progress.

[43]:
```
from tensorflow.keras.applications import ResNet50V2

pretrained_resnet = ResNet50V2(
    input_shape = (224, 224, 3),
    include_top = False,
    weights = 'imagenet')

pretrained_resnet.trainable=False

pretrained_resnet.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760 [==============================] - 5s 0us/step
Model: "resnet50v2"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_3 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0
['input_3[0][0]']

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

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0
['conv1_conv[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0
['pool1_pad[0][0]']

 conv2_block1_preact_bn (BatchN  (None, 56, 56, 64)  256
['pool1_pool[0][0]']
 ormalization)

 conv2_block1_preact_relu (Acti  (None, 56, 56, 64)  0
['conv2_block1_preact_bn[0][0]']
```

```
 vation)

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4096
['conv2_block1_preact_relu[0][0]'
                                                                 ]

 conv2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block1_1_relu[0][0]']
 g2D)

 conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36864
['conv2_block1_2_pad[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block1_2_conv[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block1_2_bn[0][0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block1_preact_relu[0][0]'
                                                                 ]

 conv2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block1_2_relu[0][0]']

 conv2_block1_out (Add)         (None, 56, 56, 256)  0
['conv2_block1_0_conv[0][0]',
'conv2_block1_3_conv[0][0]']

 conv2_block2_preact_bn (BatchN  (None, 56, 56, 256)  1024
['conv2_block1_out[0][0]']
 ormalization)

 conv2_block2_preact_relu (Acti  (None, 56, 56, 256)  0
['conv2_block2_preact_bn[0][0]']
 vation)
```

```
 conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16384
['conv2_block2_preact_relu[0][0]'
                                                                ]

 conv2_block2_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block2_1_conv[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block2_1_bn[0][0]']
 n)

 conv2_block2_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block2_1_relu[0][0]']
 g2D)

 conv2_block2_2_conv (Conv2D)   (None, 56, 56, 64)   36864
['conv2_block2_2_pad[0][0]']

 conv2_block2_2_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block2_2_conv[0][0]']
 ization)

 conv2_block2_2_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block2_2_bn[0][0]']
 n)

 conv2_block2_3_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block2_2_relu[0][0]']

 conv2_block2_out (Add)         (None, 56, 56, 256)  0
['conv2_block1_out[0][0]',
'conv2_block2_3_conv[0][0]']

 conv2_block3_preact_bn (BatchN  (None, 56, 56, 256)  1024
['conv2_block2_out[0][0]']
 ormalization)

 conv2_block3_preact_relu (Acti  (None, 56, 56, 256)  0
['conv2_block3_preact_bn[0][0]']
 vation)

 conv2_block3_1_conv (Conv2D)   (None, 56, 56, 64)   16384
['conv2_block3_preact_relu[0][0]'
                                                                ]

 conv2_block3_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block3_1_conv[0][0]']
```

```
 ization)

 conv2_block3_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block3_1_bn[0][0]']
 n)

 conv2_block3_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block3_1_relu[0][0]']
 g2D)

 conv2_block3_2_conv (Conv2D)   (None, 28, 28, 64)   36864
['conv2_block3_2_pad[0][0]']

 conv2_block3_2_bn (BatchNormal  (None, 28, 28, 64)  256
['conv2_block3_2_conv[0][0]']
 ization)

 conv2_block3_2_relu (Activatio  (None, 28, 28, 64)  0
['conv2_block3_2_bn[0][0]']
 n)

 max_pooling2d (MaxPooling2D)   (None, 28, 28, 256)  0
['conv2_block2_out[0][0]']

 conv2_block3_3_conv (Conv2D)   (None, 28, 28, 256)  16640
['conv2_block3_2_relu[0][0]']

 conv2_block3_out (Add)         (None, 28, 28, 256)  0
['max_pooling2d[0][0]',
'conv2_block3_3_conv[0][0]']

 conv3_block1_preact_bn (BatchN  (None, 28, 28, 256)  1024
['conv2_block3_out[0][0]']
 ormalization)

 conv3_block1_preact_relu (Acti  (None, 28, 28, 256)  0
['conv3_block1_preact_bn[0][0]']
 vation)

 conv3_block1_1_conv (Conv2D)   (None, 28, 28, 128)  32768
['conv3_block1_preact_relu[0][0]'
                                                                ]

 conv3_block1_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block1_1_conv[0][0]']
 ization)

 conv3_block1_1_relu (Activatio  (None, 28, 28, 128)  0
```

```
['conv3_block1_1_bn[0][0]']
 n)

 conv3_block1_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block1_1_relu[0][0]']
 g2D)

 conv3_block1_2_conv (Conv2D)   (None, 28, 28, 128)  147456
['conv3_block1_2_pad[0][0]']

 conv3_block1_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block1_2_conv[0][0]']
 ization)

 conv3_block1_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block1_2_bn[0][0]']
 n)

 conv3_block1_0_conv (Conv2D)   (None, 28, 28, 512)  131584
['conv3_block1_preact_relu[0][0]'
                                                                 ]

 conv3_block1_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block1_2_relu[0][0]']

 conv3_block1_out (Add)         (None, 28, 28, 512)  0
['conv3_block1_0_conv[0][0]',
'conv3_block1_3_conv[0][0]']

 conv3_block2_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block1_out[0][0]']
 ormalization)

 conv3_block2_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block2_preact_bn[0][0]']
 vation)

 conv3_block2_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block2_preact_relu[0][0]'
                                                                 ]

 conv3_block2_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block2_1_conv[0][0]']
 ization)

 conv3_block2_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block2_1_bn[0][0]']
 n)
```

```
 conv3_block2_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block2_1_relu[0][0]']
 g2D)

 conv3_block2_2_conv (Conv2D)   (None, 28, 28, 128)  147456
['conv3_block2_2_pad[0][0]']

 conv3_block2_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block2_2_conv[0][0]']
 ization)

 conv3_block2_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block2_2_bn[0][0]']
 n)

 conv3_block2_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block2_2_relu[0][0]']

 conv3_block2_out (Add)         (None, 28, 28, 512)  0
['conv3_block1_out[0][0]',
'conv3_block2_3_conv[0][0]']

 conv3_block3_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block2_out[0][0]']
 ormalization)

 conv3_block3_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block3_preact_bn[0][0]']
 vation)

 conv3_block3_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block3_preact_relu[0][0]'
                                                                ]

 conv3_block3_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block3_1_conv[0][0]']
 ization)

 conv3_block3_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block3_1_bn[0][0]']
 n)

 conv3_block3_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block3_1_relu[0][0]']
 g2D)

 conv3_block3_2_conv (Conv2D)   (None, 28, 28, 128)  147456
```

```
['conv3_block3_2_pad[0][0]']

 conv3_block3_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block3_2_conv[0][0]']
 ization)

 conv3_block3_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block3_2_bn[0][0]']
 n)

 conv3_block3_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block3_2_relu[0][0]']

 conv3_block3_out (Add)         (None, 28, 28, 512)  0
['conv3_block2_out[0][0]',
'conv3_block3_3_conv[0][0]']

 conv3_block4_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block3_out[0][0]']
 ormalization)

 conv3_block4_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block4_preact_bn[0][0]']
 vation)

 conv3_block4_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block4_preact_relu[0][0]'
                                                                ]

 conv3_block4_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block4_1_conv[0][0]']
 ization)

 conv3_block4_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block4_1_bn[0][0]']
 n)

 conv3_block4_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block4_1_relu[0][0]']
 g2D)

 conv3_block4_2_conv (Conv2D)   (None, 14, 14, 128)  147456
['conv3_block4_2_pad[0][0]']

 conv3_block4_2_bn (BatchNormal  (None, 14, 14, 128)  512
['conv3_block4_2_conv[0][0]']
 ization)
```

```
 conv3_block4_2_relu (Activatio  (None, 14, 14, 128)  0
['conv3_block4_2_bn[0][0]']
 n)

 max_pooling2d_1 (MaxPooling2D)  (None, 14, 14, 512)  0
['conv3_block3_out[0][0]']

 conv3_block4_3_conv (Conv2D)   (None, 14, 14, 512)  66048
['conv3_block4_2_relu[0][0]']

 conv3_block4_out (Add)         (None, 14, 14, 512)  0
['max_pooling2d_1[0][0]',
'conv3_block4_3_conv[0][0]']

 conv4_block1_preact_bn (BatchN  (None, 14, 14, 512)  2048
['conv3_block4_out[0][0]']
 ormalization)

 conv4_block1_preact_relu (Acti  (None, 14, 14, 512)  0
['conv4_block1_preact_bn[0][0]']
 vation)

 conv4_block1_1_conv (Conv2D)   (None, 14, 14, 256)  131072
['conv4_block1_preact_relu[0][0]'
                                                                 ]

 conv4_block1_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block1_1_conv[0][0]']
 ization)

 conv4_block1_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block1_1_bn[0][0]']
 n)

 conv4_block1_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block1_1_relu[0][0]']
 g2D)

 conv4_block1_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block1_2_pad[0][0]']

 conv4_block1_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block1_2_conv[0][0]']
 ization)

 conv4_block1_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block1_2_bn[0][0]']
 n)
```

```
 conv4_block1_0_conv (Conv2D)   (None, 14, 14, 1024  525312
['conv4_block1_preact_relu[0][0]'
                               )                                 ]

 conv4_block1_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block1_2_relu[0][0]']
                               )

 conv4_block1_out (Add)         (None, 14, 14, 1024  0
['conv4_block1_0_conv[0][0]',
                               )
'conv4_block1_3_conv[0][0]']

 conv4_block2_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block1_out[0][0]']
 ormalization)                 )

 conv4_block2_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block2_preact_bn[0][0]']
 vation)                       )

 conv4_block2_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block2_preact_relu[0][0]'
                                                                 ]

 conv4_block2_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block2_1_conv[0][0]']
 ization)

 conv4_block2_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block2_1_bn[0][0]']
 n)

 conv4_block2_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block2_1_relu[0][0]']
 g2D)

 conv4_block2_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block2_2_pad[0][0]']

 conv4_block2_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block2_2_conv[0][0]']
 ization)

 conv4_block2_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block2_2_bn[0][0]']
 n)
```

```
 conv4_block2_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block2_2_relu[0][0]']
                                )

 conv4_block2_out (Add)         (None, 14, 14, 1024  0
['conv4_block1_out[0][0]',
                                )
'conv4_block2_3_conv[0][0]']

 conv4_block3_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block2_out[0][0]']
 ormalization)                  )

 conv4_block3_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block3_preact_bn[0][0]']
 vation)                        )

 conv4_block3_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block3_preact_relu[0][0]'
                                                                  ]

 conv4_block3_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block3_1_conv[0][0]']
 ization)

 conv4_block3_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block3_1_bn[0][0]']
 n)

 conv4_block3_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block3_1_relu[0][0]']
 g2D)

 conv4_block3_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block3_2_pad[0][0]']

 conv4_block3_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block3_2_conv[0][0]']
 ization)

 conv4_block3_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block3_2_bn[0][0]']
 n)

 conv4_block3_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block3_2_relu[0][0]']
                                )
```

```
 conv4_block3_out (Add)         (None, 14, 14, 1024  0
['conv4_block2_out[0][0]',
                                )
'conv4_block3_3_conv[0][0]']

 conv4_block4_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block3_out[0][0]']
 ormalization)                  )

 conv4_block4_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block4_preact_bn[0][0]']
 vation)                        )

 conv4_block4_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block4_preact_relu[0][0]'
                                                                ]

 conv4_block4_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block4_1_conv[0][0]']
 ization)

 conv4_block4_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block4_1_bn[0][0]']
 n)

 conv4_block4_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block4_1_relu[0][0]']
 g2D)

 conv4_block4_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block4_2_pad[0][0]']

 conv4_block4_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block4_2_conv[0][0]']
 ization)

 conv4_block4_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block4_2_bn[0][0]']
 n)

 conv4_block4_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block4_2_relu[0][0]']
                                )

 conv4_block4_out (Add)         (None, 14, 14, 1024  0
['conv4_block3_out[0][0]',
                                )
```

```
'conv4_block4_3_conv[0][0]']

 conv4_block5_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block4_out[0][0]']
 ormalization)                  )

 conv4_block5_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block5_preact_bn[0][0]']
 vation)                        )

 conv4_block5_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block5_preact_relu[0][0]'
                                                                 ]

 conv4_block5_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block5_1_conv[0][0]']
 ization)

 conv4_block5_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block5_1_bn[0][0]']
 n)

 conv4_block5_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block5_1_relu[0][0]']
 g2D)

 conv4_block5_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block5_2_pad[0][0]']

 conv4_block5_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block5_2_conv[0][0]']
 ization)

 conv4_block5_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block5_2_bn[0][0]']
 n)

 conv4_block5_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block5_2_relu[0][0]']
                                )

 conv4_block5_out (Add)         (None, 14, 14, 1024  0
['conv4_block4_out[0][0]',
                                )
'conv4_block5_3_conv[0][0]']

 conv4_block6_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block5_out[0][0]']
```

```
 ormalization)                  )

 conv4_block6_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block6_preact_bn[0][0]']
 vation)                        )

 conv4_block6_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block6_preact_relu[0][0]'
                                                                 ]

 conv4_block6_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block6_1_conv[0][0]']
 ization)

 conv4_block6_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block6_1_bn[0][0]']
 n)

 conv4_block6_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block6_1_relu[0][0]']
 g2D)

 conv4_block6_2_conv (Conv2D)   (None, 7, 7, 256)    589824
['conv4_block6_2_pad[0][0]']

 conv4_block6_2_bn (BatchNormal  (None, 7, 7, 256)   1024
['conv4_block6_2_conv[0][0]']
 ization)

 conv4_block6_2_relu (Activatio  (None, 7, 7, 256)   0
['conv4_block6_2_bn[0][0]']
 n)

 max_pooling2d_2 (MaxPooling2D)  (None, 7, 7, 1024)  0
['conv4_block5_out[0][0]']

 conv4_block6_3_conv (Conv2D)   (None, 7, 7, 1024)   263168
['conv4_block6_2_relu[0][0]']

 conv4_block6_out (Add)         (None, 7, 7, 1024)   0
['max_pooling2d_2[0][0]',
'conv4_block6_3_conv[0][0]']

 conv5_block1_preact_bn (BatchN  (None, 7, 7, 1024)  4096
['conv4_block6_out[0][0]']
 ormalization)

 conv5_block1_preact_relu (Acti  (None, 7, 7, 1024)  0
```

```
['conv5_block1_preact_bn[0][0]']
 vation)

 conv5_block1_1_conv (Conv2D)   (None, 7, 7, 512)    524288
['conv5_block1_preact_relu[0][0]'
                                                                 ]

 conv5_block1_1_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block1_1_conv[0][0]']
 ization)

 conv5_block1_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block1_1_bn[0][0]']
 n)

 conv5_block1_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block1_1_relu[0][0]']
 g2D)

 conv5_block1_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block1_2_pad[0][0]']

 conv5_block1_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block1_2_conv[0][0]']
 ization)

 conv5_block1_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block1_2_bn[0][0]']
 n)

 conv5_block1_0_conv (Conv2D)   (None, 7, 7, 2048)   2099200
['conv5_block1_preact_relu[0][0]'
                                                                 ]

 conv5_block1_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block1_2_relu[0][0]']

 conv5_block1_out (Add)         (None, 7, 7, 2048)   0
['conv5_block1_0_conv[0][0]',
'conv5_block1_3_conv[0][0]']

 conv5_block2_preact_bn (BatchN  (None, 7, 7, 2048)  8192
['conv5_block1_out[0][0]']
 ormalization)

 conv5_block2_preact_relu (Acti  (None, 7, 7, 2048)  0
['conv5_block2_preact_bn[0][0]']
 vation)
```

```
 conv5_block2_1_conv (Conv2D)   (None, 7, 7, 512)    1048576
['conv5_block2_preact_relu[0][0]'
                                                                 ]

 conv5_block2_1_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block2_1_conv[0][0]']
 ization)

 conv5_block2_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block2_1_bn[0][0]']
 n)

 conv5_block2_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block2_1_relu[0][0]']
 g2D)

 conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block2_2_pad[0][0]']

 conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block2_2_conv[0][0]']
 ization)

 conv5_block2_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block2_2_bn[0][0]']
 n)

 conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block2_2_relu[0][0]']

 conv5_block2_out (Add)         (None, 7, 7, 2048)   0
['conv5_block1_out[0][0]',
'conv5_block2_3_conv[0][0]']

 conv5_block3_preact_bn (BatchN  (None, 7, 7, 2048)  8192
['conv5_block2_out[0][0]']
 ormalization)

 conv5_block3_preact_relu (Acti  (None, 7, 7, 2048)  0
['conv5_block3_preact_bn[0][0]']
 vation)

 conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)    1048576
['conv5_block3_preact_relu[0][0]'
                                                                 ]

 conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)   2048
```

```
['conv5_block3_1_conv[0][0]']
 ization)

 conv5_block3_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block3_1_bn[0][0]']
 n)

 conv5_block3_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block3_1_relu[0][0]']
 g2D)

 conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block3_2_pad[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block3_2_relu[0][0]']

 conv5_block3_out (Add)         (None, 7, 7, 2048)   0
['conv5_block2_out[0][0]',
'conv5_block3_3_conv[0][0]']

 post_bn (BatchNormalization)   (None, 7, 7, 2048)   8192
['conv5_block3_out[0][0]']

 post_relu (Activation)         (None, 7, 7, 2048)   0
['post_bn[0][0]']

==================================================================================
==================
Total params: 23,564,800
Trainable params: 0
Non-trainable params: 23,564,800
----------------------------------------------------------------------------------
------------------
```

1.10.1 Resnet50 V1 - baseline model

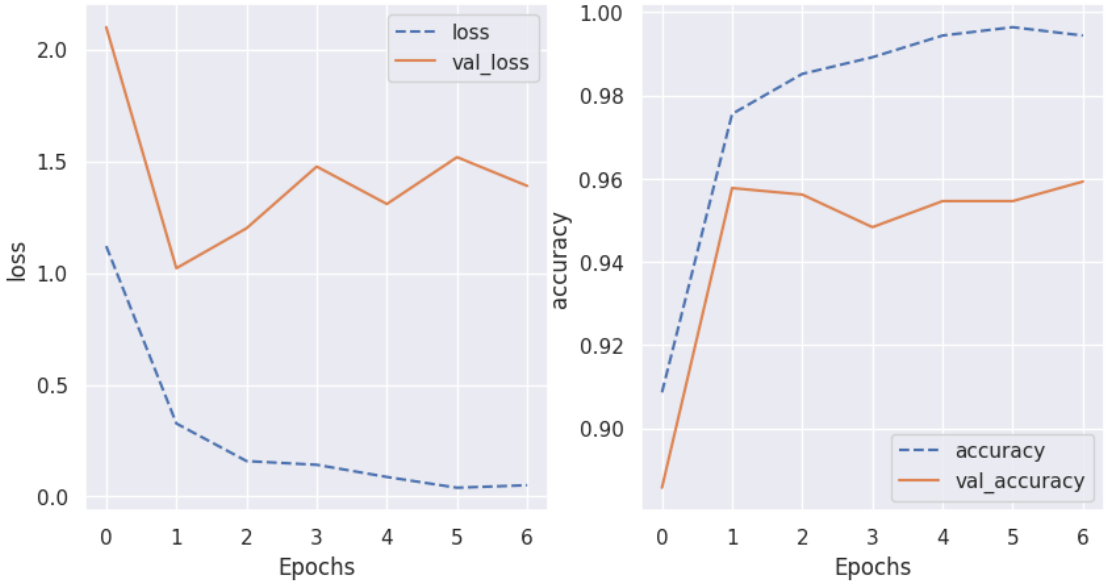
[61]:
```
#add a fully connected layer in classifier head.

model_name = 'resnet_v1'

callbacks = [
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, min_delta=0.001, mode='min',␣
 ↪restore_best_weights=True
  ),
  tf.keras.callbacks.TensorBoard(
    log_dir='logs',
    update_freq='epoch',
  ),
  checkpoint_helper.getCallback(model_name)
]

optimizer = keras.optimizers.Adam(learning_rate=0.001)

resnet_model_v1, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_resnet,
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=10, callbacks=callbacks),

    #training data
    train_ds1,

    #validation data
    val_ds1,

    force_train = True
```

```
    )
```

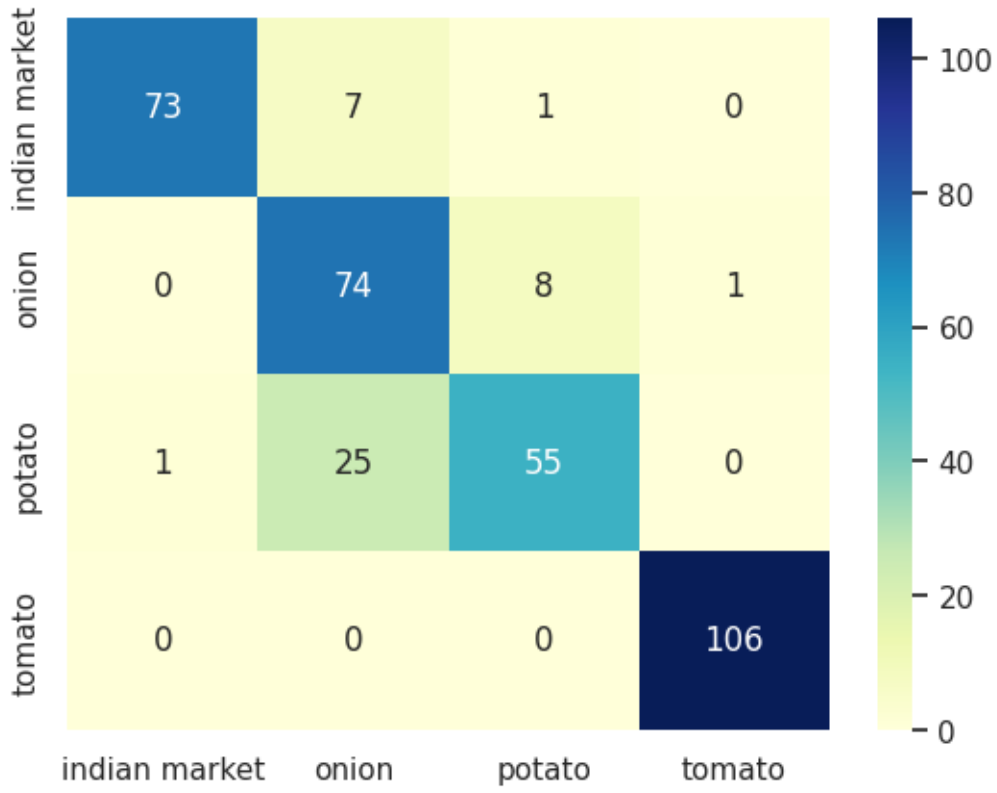
```
Epoch 1/10
78/78 [==============================] - ETA: 0s - loss: 1.4581 - accuracy:
0.8922
Epoch 1: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 46s 480ms/step - loss: 1.4581 -
accuracy: 0.8922 - val_loss: 1.8837 - val_accuracy: 0.9061
Epoch 2/10
78/78 [==============================] - ETA: 0s - loss: 0.2890 - accuracy:
0.9752
Epoch 2: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 26s 320ms/step - loss: 0.2890 -
accuracy: 0.9752 - val_loss: 0.9840 - val_accuracy: 0.9546
Epoch 3/10
78/78 [==============================] - ETA: 0s - loss: 0.1246 - accuracy:
0.9900
Epoch 3: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 33s 423ms/step - loss: 0.1246 -
accuracy: 0.9900 - val_loss: 1.1044 - val_accuracy: 0.9468
Epoch 4/10
78/78 [==============================] - ETA: 0s - loss: 0.0470 - accuracy:
0.9948
Epoch 4: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 34s 425ms/step - loss: 0.0470 -
accuracy: 0.9948 - val_loss: 0.9274 - val_accuracy: 0.9656
Epoch 5/10
78/78 [==============================] - ETA: 0s - loss: 0.0454 - accuracy:
0.9948
Epoch 5: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 32s 406ms/step - loss: 0.0454 -
accuracy: 0.9948 - val_loss: 1.7129 - val_accuracy: 0.9358
Epoch 6/10
78/78 [==============================] - ETA: 0s - loss: 0.0633 - accuracy:
0.9916
Epoch 6: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 25s 312ms/step - loss: 0.0633 -
accuracy: 0.9916 - val_loss: 1.2894 - val_accuracy: 0.9452
Epoch 7/10
78/78 [==============================] - ETA: 0s - loss: 0.0397 - accuracy:
0.9968
Epoch 7: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 25s 314ms/step - loss: 0.0397 -
accuracy: 0.9968 - val_loss: 1.1836 - val_accuracy: 0.9562
Epoch 8/10
78/78 [==============================] - ETA: 0s - loss: 0.0641 - accuracy:
0.9952
```

```
Epoch 8: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 25s 316ms/step - loss: 0.0641 -
accuracy: 0.9952 - val_loss: 1.1392 - val_accuracy: 0.9609
Epoch 9/10
78/78 [==============================] - ETA: 0s - loss: 0.0239 - accuracy:
0.9960
Epoch 9: saving model to ./checkpoints_test/resnet_v1
78/78 [==============================] - 25s 316ms/step - loss: 0.0239 -
accuracy: 0.9960 - val_loss: 1.1867 - val_accuracy: 0.9577
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 53). These functions will not be directly callable after loading.
```

[45]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(resnet_model_v1, test_ds1, classes)
```




```
11/11 - 3s - 3s/epoch - 314ms/step
```



	class	precision	recall
0	indian market	0.99	0.90
1	onion	0.70	0.89
2	potato	0.86	0.68
3	tomato	0.99	1.00

Overall accuracy: 87.7%, macro precision: 0.88, macro recall: 0.87

Observations:

1.10.2 Resnet V2

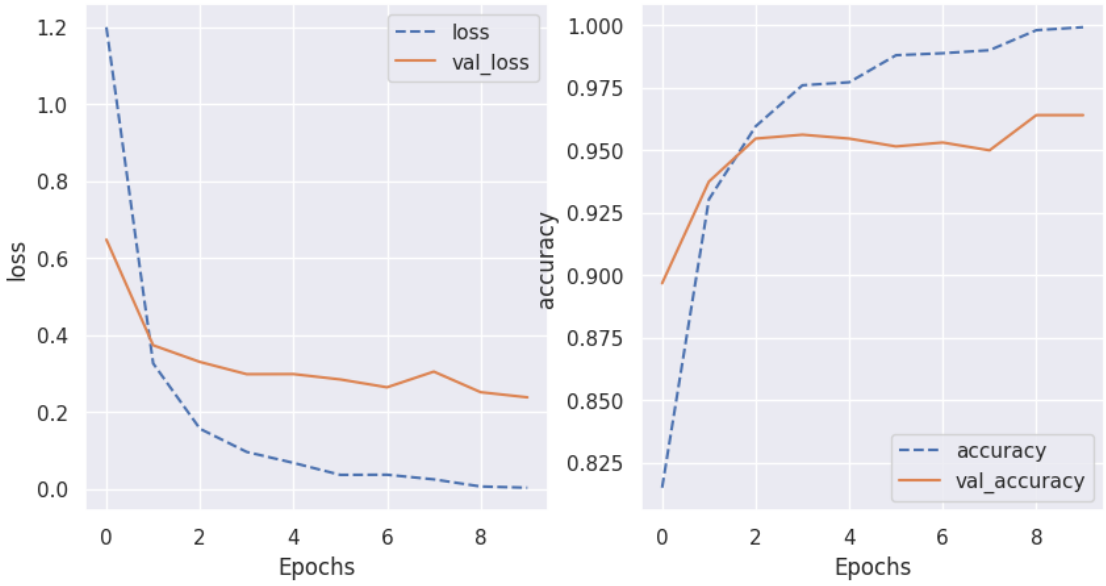
[47]:
```
model_name = 'resnet_v2'

optimizer = keras.optimizers.Adam(learning_rate=0.001)

resnet_model_v2, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
```

```
    name=model_name,
    layers=[
        pretrained_resnet,
        tf.keras.layers.GlobalMaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(4, activation='softmax')]
    ),

  # compile_model_fn
  lambda model: get_compile_fn(model, optimizer=optimizer),

  # train_model_fn
  lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
↪val_data, epochs=10, callbacks=callbacks),

  #training data
  train_ds1,

  #validation data
  val_ds1,

  force_train = True
  )
```

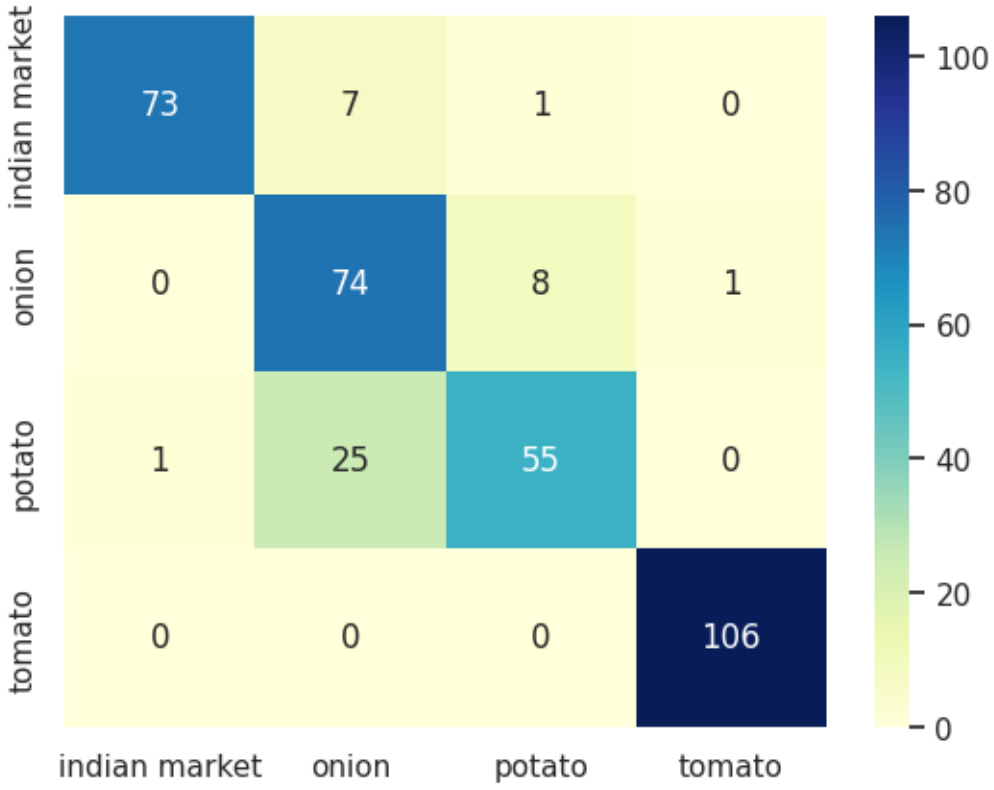
```
Epoch 1/10
78/78 [==============================] - ETA: 0s - loss: 1.1996 - accuracy:
0.8149
Epoch 1: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 41s 445ms/step - loss: 1.1996 -
accuracy: 0.8149 - val_loss: 0.6478 - val_accuracy: 0.8967 - lr: 0.0010
Epoch 2/10
78/78 [==============================] - ETA: 0s - loss: 0.3263 - accuracy:
0.9303
Epoch 2: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 34s 425ms/step - loss: 0.3263 -
accuracy: 0.9303 - val_loss: 0.3734 - val_accuracy: 0.9374 - lr: 0.0010
Epoch 3/10
78/78 [==============================] - ETA: 0s - loss: 0.1566 - accuracy:
0.9595
Epoch 3: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 26s 323ms/step - loss: 0.1566 -
accuracy: 0.9595 - val_loss: 0.3301 - val_accuracy: 0.9546 - lr: 0.0010
Epoch 4/10
78/78 [==============================] - ETA: 0s - loss: 0.0963 - accuracy:
0.9760
Epoch 4: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 26s 319ms/step - loss: 0.0963 -
```

```
accuracy: 0.9760 - val_loss: 0.2983 - val_accuracy: 0.9562 - lr: 0.0010
Epoch 5/10
78/78 [==============================] - ETA: 0s - loss: 0.0679 - accuracy:
0.9772
Epoch 5: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 33s 422ms/step - loss: 0.0679 -
accuracy: 0.9772 - val_loss: 0.2986 - val_accuracy: 0.9546 - lr: 0.0010
Epoch 6/10
78/78 [==============================] - ETA: 0s - loss: 0.0368 - accuracy:
0.9880
Epoch 6: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 34s 430ms/step - loss: 0.0368 -
accuracy: 0.9880 - val_loss: 0.2847 - val_accuracy: 0.9515 - lr: 0.0010
Epoch 7/10
78/78 [==============================] - ETA: 0s - loss: 0.0372 - accuracy:
0.9888
Epoch 7: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 26s 320ms/step - loss: 0.0372 -
accuracy: 0.9888 - val_loss: 0.2642 - val_accuracy: 0.9531 - lr: 0.0010
Epoch 8/10
78/78 [==============================] - ETA: 0s - loss: 0.0251 - accuracy:
0.9900
Epoch 8: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 33s 420ms/step - loss: 0.0251 -
accuracy: 0.9900 - val_loss: 0.3050 - val_accuracy: 0.9499 - lr: 0.0010
Epoch 9/10
78/78 [==============================] - ETA: 0s - loss: 0.0066 - accuracy:
0.9980
Epoch 9: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 25s 312ms/step - loss: 0.0066 -
accuracy: 0.9980 - val_loss: 0.2516 - val_accuracy: 0.9640 - lr: 0.0010
Epoch 10/10
78/78 [==============================] - ETA: 0s - loss: 0.0034 - accuracy:
0.9992
Epoch 10: saving model to ./checkpoints_test/efn_v1
78/78 [==============================] - 34s 426ms/step - loss: 0.0034 -
accuracy: 0.9992 - val_loss: 0.2382 - val_accuracy: 0.9640 - lr: 0.0010
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 53). These functions will not be directly callable after loading.
```

[48]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(resnet_model_v1, test_ds1, classes)
```




```
11/11 - 2s - 2s/epoch - 167ms/step
```



```
           class  precision  recall
0  indian market       0.99    0.90
1          onion       0.70    0.89
2         potato       0.86    0.68
3         tomato       0.99    1.00
Overall accuracy: 87.7%, macro precision: 0.88, macro recall: 0.87
```

1.10.3 resnet V3

[49]:
```
model_name = 'resnet_v3'

callbacks = [
  keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, min_delta=0.001, mode='min',␣
 ↪restore_best_weights=True
  ),
  tf.keras.callbacks.TensorBoard(
    log_dir='logs',
    update_freq='epoch',
  ),
  checkpoint_helper.getCallback(model_name)
]

optimizer = keras.optimizers.Adam(learning_rate=0.001)

resnet_model_v3, model_history = get_trained_model(

    model_name,

    # create_model_fn
    lambda model_name: tf.keras.Sequential(
      name=model_name,
      layers=[
          pretrained_resnet,
          tf.keras.layers.GlobalMaxPooling2D(),
          tf.keras.layers.Flatten(),
          tf.keras.layers.Dense(4, activation='softmax')]
      ),

    # compile_model_fn
    lambda model: get_compile_fn(model, optimizer=optimizer),

    # train_model_fn
    lambda model, train_data, val_data: model_fit_fn(model, train_data,␣
 ↪val_data, epochs=15, callbacks=callbacks),
```

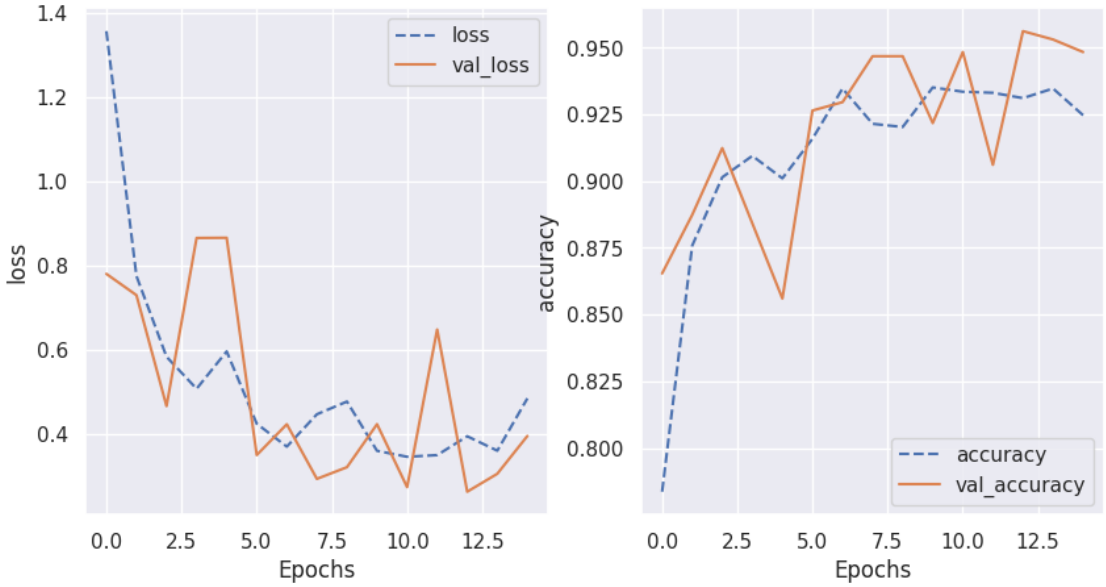
```
    #training data
    train_ds3,

    #validation data
    val_ds3
    )
```

```
Epoch 1/15
78/78 [==============================] - ETA: 0s - loss: 1.3561 - accuracy:
0.7837
Epoch 1: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 77s 916ms/step - loss: 1.3561 -
accuracy: 0.7837 - val_loss: 0.7797 - val_accuracy: 0.8654
Epoch 2/15
78/78 [==============================] - ETA: 0s - loss: 0.7745 - accuracy:
0.8758
Epoch 2: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 92s 1s/step - loss: 0.7745 - accuracy:
0.8758 - val_loss: 0.7290 - val_accuracy: 0.8873
Epoch 3/15
78/78 [==============================] - ETA: 0s - loss: 0.5828 - accuracy:
0.9014
Epoch 3: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 82s 1s/step - loss: 0.5828 - accuracy:
0.9014 - val_loss: 0.4657 - val_accuracy: 0.9124
Epoch 4/15
78/78 [==============================] - ETA: 0s - loss: 0.5068 - accuracy:
0.9095
Epoch 4: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 88s 1s/step - loss: 0.5068 - accuracy:
0.9095 - val_loss: 0.8649 - val_accuracy: 0.8842
Epoch 5/15
78/78 [==============================] - ETA: 0s - loss: 0.5956 - accuracy:
0.9010
Epoch 5: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 64s 813ms/step - loss: 0.5956 -
accuracy: 0.9010 - val_loss: 0.8654 - val_accuracy: 0.8560
Epoch 6/15
78/78 [==============================] - ETA: 0s - loss: 0.4238 - accuracy:
0.9159
Epoch 6: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 77s 960ms/step - loss: 0.4238 -
accuracy: 0.9159 - val_loss: 0.3493 - val_accuracy: 0.9264
Epoch 7/15
78/78 [==============================] - ETA: 0s - loss: 0.3694 - accuracy:
0.9347
Epoch 7: saving model to ./checkpoints_test/resnet_v3
```

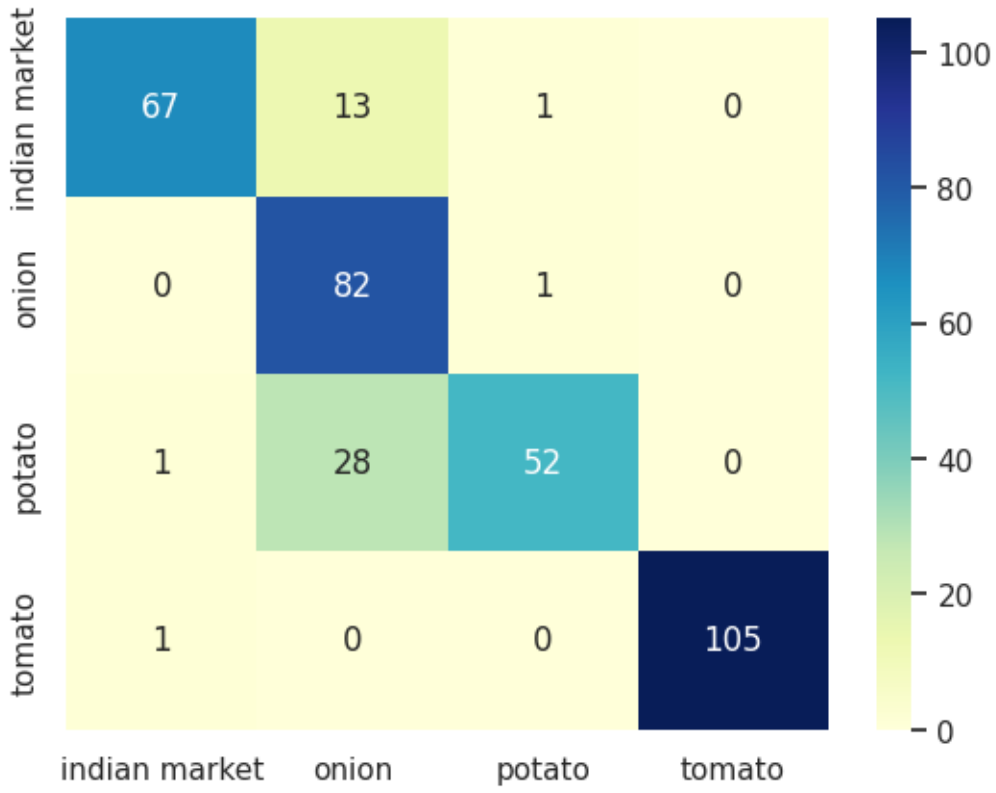
```
78/78 [==============================] - 89s 1s/step - loss: 0.3694 - accuracy:
0.9347 - val_loss: 0.4226 - val_accuracy: 0.9296
Epoch 8/15
78/78 [==============================] - ETA: 0s - loss: 0.4462 - accuracy:
0.9215
Epoch 8: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 80s 1s/step - loss: 0.4462 - accuracy:
0.9215 - val_loss: 0.2928 - val_accuracy: 0.9468
Epoch 9/15
78/78 [==============================] - ETA: 0s - loss: 0.4764 - accuracy:
0.9203
Epoch 9: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 83s 1s/step - loss: 0.4764 - accuracy:
0.9203 - val_loss: 0.3205 - val_accuracy: 0.9468
Epoch 10/15
78/78 [==============================] - ETA: 0s - loss: 0.3594 - accuracy:
0.9351
Epoch 10: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 78s 969ms/step - loss: 0.3594 -
accuracy: 0.9351 - val_loss: 0.4227 - val_accuracy: 0.9218
Epoch 11/15
78/78 [==============================] - ETA: 0s - loss: 0.3453 - accuracy:
0.9335
Epoch 11: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 66s 828ms/step - loss: 0.3453 -
accuracy: 0.9335 - val_loss: 0.2733 - val_accuracy: 0.9484
Epoch 12/15
78/78 [==============================] - ETA: 0s - loss: 0.3492 - accuracy:
0.9331
Epoch 12: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 92s 1s/step - loss: 0.3492 - accuracy:
0.9331 - val_loss: 0.6475 - val_accuracy: 0.9061
Epoch 13/15
78/78 [==============================] - ETA: 0s - loss: 0.3940 - accuracy:
0.9311
Epoch 13: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 80s 1s/step - loss: 0.3940 - accuracy:
0.9311 - val_loss: 0.2623 - val_accuracy: 0.9562
Epoch 14/15
78/78 [==============================] - ETA: 0s - loss: 0.3597 - accuracy:
0.9347
Epoch 14: saving model to ./checkpoints_test/resnet_v3
78/78 [==============================] - 90s 1s/step - loss: 0.3597 - accuracy:
0.9347 - val_loss: 0.3049 - val_accuracy: 0.9531
Epoch 15/15
78/78 [==============================] - ETA: 0s - loss: 0.4840 - accuracy:
0.9247
Epoch 15: saving model to ./checkpoints_test/resnet_v3
```

```
78/78 [==============================] - 86s 1s/step - loss: 0.4840 - accuracy:
0.9247 - val_loss: 0.3947 - val_accuracy: 0.9484
saving model and history

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 53). These functions will not be directly callable after loading.
```

[50]:
```
training_plot(['loss', 'accuracy'], model_history)
predict_and_plot_confusion_matrix(resnet_model_v3, test_ds3, classes)
```




```
11/11 - 3s - 3s/epoch - 249ms/step
```




```
           class  precision  recall
0  indian market       0.97    0.83
1          onion       0.67    0.99
2         potato       0.96    0.64
3         tomato       1.00    0.99
Overall accuracy: 87.2%, macro precision: 0.9, macro recall: 0.86
```

[64]:
```
%load_ext tensorboard
%tensorboard --logdir logs
```

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

[ ]:
```
```

1.11 Insights:

1. Our base model based on Alexnet architecture reported 57% accuracy on validation set and 52.7% accuracy on test dataset. macro precision and recall numbers were 0.53 and 0.51 respectively. In the base model we used earlystopping, LR scheduler, as well as regularization. Data augmentation had nagative impact on the performance.
2. The best VGG19 model reported validation score of 94% and test accuracy of 84.9%. We

used LR scheduler and early stopping for addressing overfitting. Data augmentation didn't help improve performance. Another version of VGG19 model using global max pooling layer reduced the number of trainable parameters, but achieved validation and test accuracy of 85% and 80% respectively. We used earlystopping and LR scheduler in this model as well.

3. Surprisingly, our custom model based on the pretrained efficientnet V2 B0 model worked very poorly (poorer than even the base model). We tried several tricks: adjusting learning rate, changing batch size, applying data augmentation, adding denser fully connected layer (upto 1024), batch normalization, dropouts, etc (These were tried offline and not captured in this notebook). None of the options helped improve validation/test accuracy. The validation score always seemed to hover in the 0.2 - 0.35 range, while test score in the range 0.25-0.3. We observed that predicitons of this model were highly biased in favour of 'indian market' class: that is, it classified majority of the images from other classes as 'indian market'. We need more investigation to understand what could be the issue for subpar performance.

4. The resnet50 based model produced highest validation and test accuracy scores of 96% and 87.7% respectively. The model performed exceedingly well for 'tomato' images (recall: 1, precision: 0.99) and 'indian market' images (recall: 0.9, precision: 0.99). However, it confused some potatos as onions which decreased overall accuracy.

5. Across models, we applied basic pre-processing operations including resize and rescaling. However, additional data augmentation techniques such as translation, random crop, and flip layers, in general, didn't help improve performance.

Conclusion: resnet50 based model produced best validation and test accuracy scores at 96% and 87.7% respectively. VGG19 produced second best results with validation and test accuracy at 94% and 84.9%. Efficientnetv2B0 based model, however, produced below par accuracy score.

[ ]: