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.

Importing the dataset and doing usual exploratory analysis steps like checking the structure & characteristics of the data (10 points)

```
Visualize the data, use the dataset directory to create a list
containing all the image paths in the training folder. You can use
matplotlib or tensorflow to plot a grid sample of the images you
fetched from the list of image paths.

Plot a few of the images of each class to check their dimensions.
[Note that the images are not all of uniform dimensions]

Verify the count of images in each train and test folder by plotting
histogram .

Check each folder to see if the number of images matches the reported
number.
```

Testing your best model so far(20 points)

```
Testing on the test set & Random image samples prediction[10]

Summary & Insights [10]
```

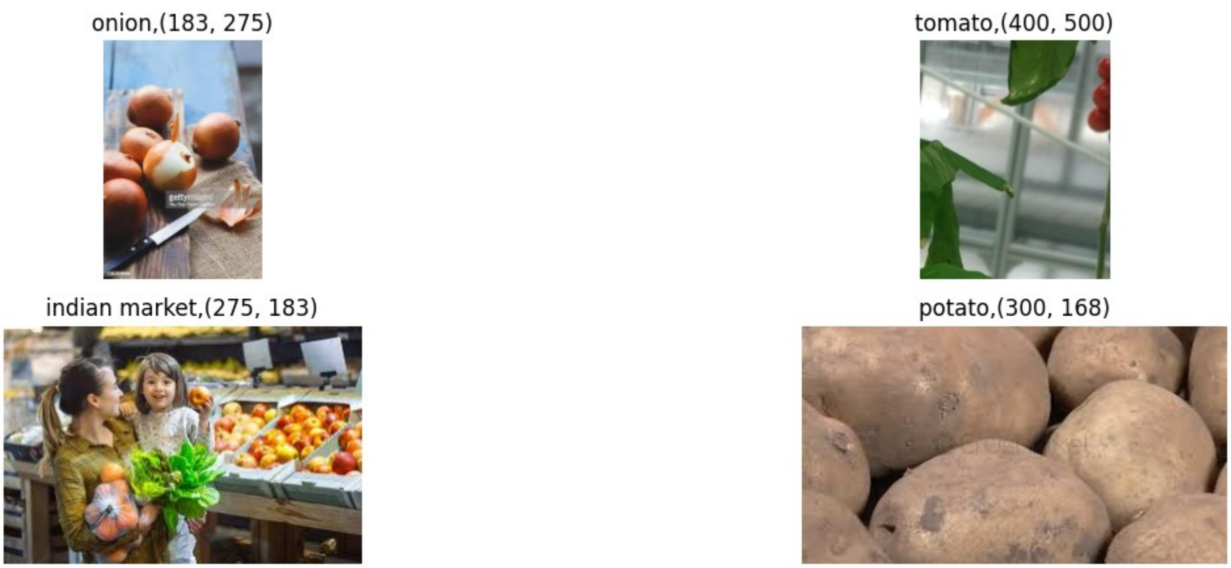
```
!gdown 1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT
#!unzip /content/ninjacart_data.zip -d /content/New_Folder/1
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1clZX-
lV_MLxKHSyeyTheX5OCQtNCUcqT
From (redirected): https://drive.google.com/uc?id=1clZX-
lV_MLxKHSyeyTheX5OCQtNCUcqT&confirm=t&uuid=f2d72ee4-94c4-437f-8222-
```

```
a2bc5a773b2c
To: /content/ninjacart_data.zip
100% 275M/275M [00:02<00:00, 99.7MB/s]

#import shutil
#shutil.rmtree('ninjacart_data/')

!unzip -q ninjacart_data.zip

#!pip install tensorflow

# Tensorflow import
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Dense, ReLU, Softmax, BatchNormalization,
Dropout
from tensorflow.random import set_seed

import os

import matplotlib.pyplot as plt
import pandas as pd
import random
import glob
import sklearn.metrics as metrics


class_dirs = os.listdir("/content/ninjacart_data/train") # list all
directories inside "train" folder
image_dict = {} # dict to store image array(key) for every
class(value)
count_dict = {} # dict to store count of files(key) for every
class(value)
count_dict_test={}
print('Total Classes : ',class_dirs)
# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'/content/ninjacart_data/train/{cls}/*')
    #print(file_paths)
    count_dict[cls]=len(file_paths)
    image_path=random.choice(file_paths)
    image_dict[cls]=tf.keras.utils.load_img(image_path)

print('Training - Total Images : \n',count_dict)
#print('Test - Total Images :',len(test_ds))

#print(image_dict.keys())
plt.figure(figsize=(15,8))
```

```
for i,(cls,img) in enumerate(image_dict.items()):
  plt.subplot(3,2,i+1)
  plt.imshow(img)
  plt.axis('off')
  plt.title(f'{cls},{img.size}')
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'/content/ninjacart_data/test/{cls}/*')
    count_dict_test[cls]=len(file_paths)

print('Test - Total Images : \n',count_dict_test)
```

```
Total Classes :  ['onion', 'tomato', 'indian market', 'potato']
Training - Total Images :
 {'onion': 849, 'tomato': 789, 'indian market': 599, 'potato': 898}
Test - Total Images :
 {'onion': 83, 'tomato': 106, 'indian market': 81, 'potato': 81}
```



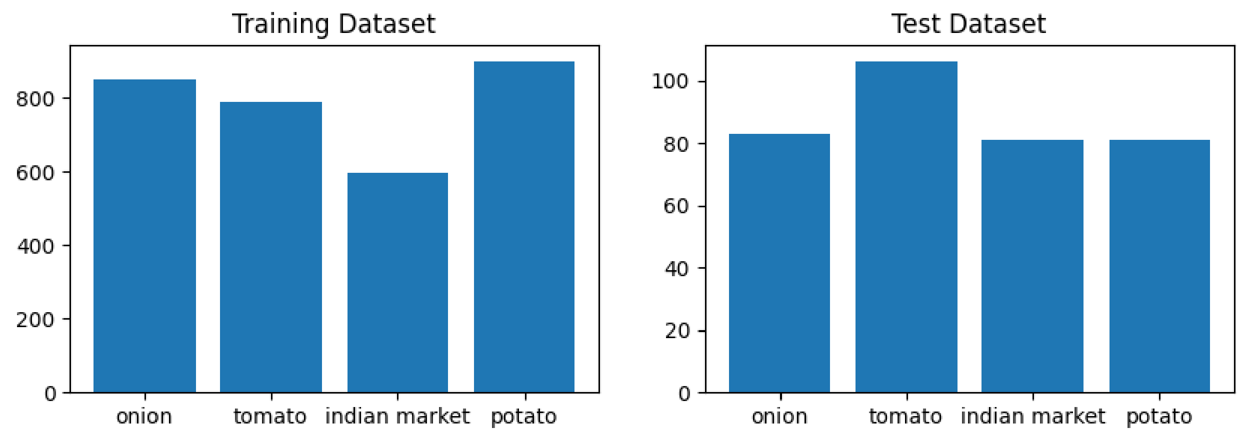
Exploratory Data Analysis. (20 points)

Plotting class distribution & Visualizing Image dimensions with their plots[10]

Splitting the dataset into train, validation, and test set[10]

```
plt.figure(figsize=(10,3))
plt.subplot(121)
plt.bar(count_dict.keys(),count_dict.values(),label=count_dict.values(
))
plt.title('Training Dataset')
plt.subplot(122)
```

```
plt.bar(count_dict_test.keys(),count_dict_test.values())
plt.title('Test Dataset')

Text(0.5, 1.0, 'Test Dataset')
```




```
Split the dataset to a train and validation set.

The provided data does not contain separate training and validation
folders. For us to do hyperparameter tuning of our models, it is
important
to divide the dataset into an 80-20 split for training and validation
respectively.

#pip install split-folders

'''import splitfolders
try:
  splitfolders.ratio("/content/ninjacart_data/train",seed=1337,
output="ninjacart_data/Train", ratio=(0.8, 0.2))
except:
  pass'''

{"type":"string"}

Before fitting data to our model, we must make sure that each image is
square-shaped so that we may resize it to the required dimensions and
also
perform rescaling which will rescale the inputs between 0-1 by
dividing each value by 255.

image_size = (128, 128)
train_ds,val_ds=tf.keras.utils.image_dataset_from_directory('/content/
ninjacart_data/train',label_mode = 'categorical',image_size =
image_size,validation_split=0.2,subset='both',shuffle=True,seed=42)
test_ds=tf.keras.utils.image_dataset_from_directory('/content/ninjacar
t_data/test',label_mode = 'categorical',image_size = image_size)
```

```
#val_ds=tf.keras.utils.image_dataset_from_directory('/content/ninjacart_data/train',image_size =
image_size,validation_split=0.2,subset='validation',shuffle=True,seed=42)

height,width=128,128
```

```
Found 3135 files belonging to 4 classes.
Using 2508 files for training.
Using 627 files for validation.
Found 351 files belonging to 4 classes.
```

```
'''train_ds=tf.keras.utils.image_dataset_from_directory('/content/
ninjacart_data/Train/train')
val_ds=tf.keras.utils.image_dataset_from_directory('/content/ninjacart_data/Train/val')
test_ds=tf.keras.utils.image_dataset_from_directory('/content/ninjacart_data/test')

height,width=128,128
data_preprocess_with_flatten=keras.Sequential(
    name='data_preprocess_with_flatten'
    ,layers= [
              layers.Resizing(height,width),
             layers.Rescaling(1/255),
             #layers.Flatten(),
             ]
)

# Perform Data Processing on the train, val, test dataset
train_ds = train_ds.map(lambda x, y: (data_preprocess_with_flatten(x), y))
val_ds = val_ds.map(lambda x, y: (data_preprocess_with_flatten(x), y))
test_ds=test_ds.map(lambda x,y :
(data_preprocess_with_flatten(x),y))'''
```

{"type":"string"}

```
sample = next(iter(train_ds))[0]
```

```
#print(sample)
```

Creating model architecture and training (50 points)

```
Defining the CNN Classifier model from scratch[10]

Improving Baseline CNN to reduce overfitting[10]

Implementing Callbacks while training the model[10]

Finetune pretrained models such as VGG, ResNet and MobileNet[10]
```

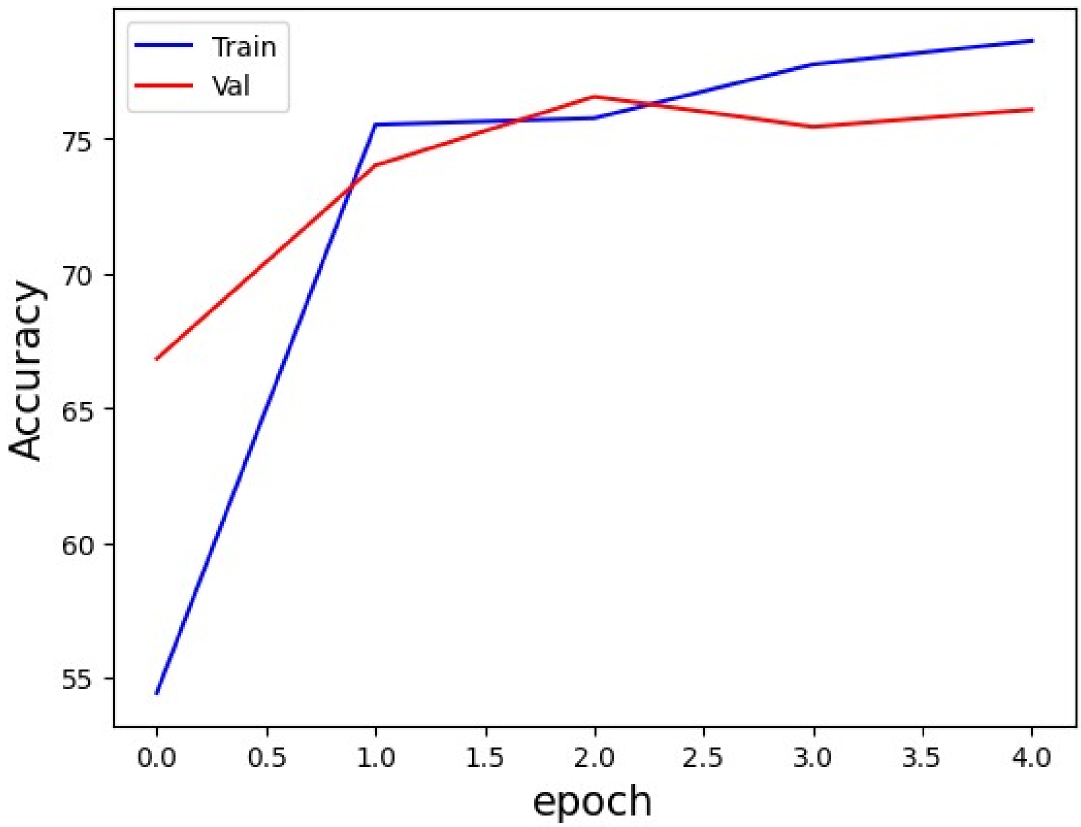
```
Plotting the model training metrics and confusion matrix[10]

#from tensorflow.keras import layers
#from tensorflow.keras import regularizers

def plot_accuracy(model_fit):
    #accuracy graph

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

import seaborn as sns
def ConfusionMatrix(model, ds, label_list):
# Note: This logic doesn't work with shuffled datasets

    # generate confusion matrix and plot it
    cm =
metrics.confusion_matrix(true_categories,predicted_categories) # last
batch
    sns.heatmap(cm, annot=True, xticklabels=label_list,
yticklabels=label_list, cmap="YlGnBu", fmt='g')
    plt.show()

noise_path = '/content/ninjacart_data/test/indian market'
onion_path = '/content/ninjacart_data/test/onion'
potato_path = '/content/ninjacart_data/test/potato'
tomato_path = '/content/ninjacart_data/test/tomato'
def classwise_accuracy(class_path, class_name, model_name) :
    paths = []
    for i in os.listdir(class_path):
        paths.append(class_path + "/" + str(i))

    correct = 0
    total = 0

    for i in range(len(paths)):
        total+= 1
```

```
        img = tf.keras.utils.load_img(paths[i])
        img = tf.keras.utils.img_to_array(img)
        img = tf.image.resize(img, (128, 128))
        img = tf.expand_dims(img, axis = 0)

        pred = model_name.predict(img,verbose=0)
        if tf.argmax(pred[0]) == class_names.index(f"{class_name}"):
            correct+= 1

    print(f"Accuracy for class {class_name} is
{round((correct/total)*100, 2)}% consisting of {len(paths)} images")

def model_evaluation_acc(model_name):
  # Evaluate the model
  loss, acc = model_name.evaluate(test_ds, verbose=2)
  print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

  y_pred = model_name.predict(test_ds)
  predicted_categories = tf.argmax(y_pred, axis=1)
  true_cat = tf.concat([y for x, y in test_ds], axis=0)
  true_categories = tf.argmax(true_cat, axis=1)

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

  classwise_accuracy(noise_path, 'indian market', model_name)
  classwise_accuracy(onion_path, 'onion', model_name)
  classwise_accuracy(potato_path, 'potato', model_name)
  classwise_accuracy(tomato_path, 'tomato', model_name)

  ConfusionMatrix(model_name, test_ds, class_dirs)

cnn_model = tf.keras.Sequential(
    name='cnn_model',
    layers=[
    layers.Rescaling(1./255),
    layers.InputLayer(input_shape = [128, 128, 3]),

    layers.Conv2D(filters = 32, kernel_size = (3,3), padding = 'Same',
activation = 'relu'),
    layers.Conv2D(filters = 32, kernel_size = (3,3), padding = 'Same',
activation = 'relu'),
    layers.MaxPool2D(pool_size = (2,2)),

    layers.Conv2D(filters = 64, kernel_size = (3,3), padding = 'Same',
activation = 'relu'),
    layers.Conv2D(filters = 64, kernel_size = (3,3), padding = 'Same',
activation = 'relu'),
```

```
    layers.MaxPool2D(pool_size = (2,2)),

    layers.Conv2D(filters = 128, kernel_size = (3,3), padding = 
'Same', activation = 'relu'),

    layers.GlobalAveragePooling2D(),
    layers.Dense(4, activation = 'softmax')
])
cnn_model.compile(optimizer='Adam',loss='categorical_crossentropy',met
rics='accuracy')
history=cnn_model.fit(train_ds,epochs=5,validation_data=val_ds)
```

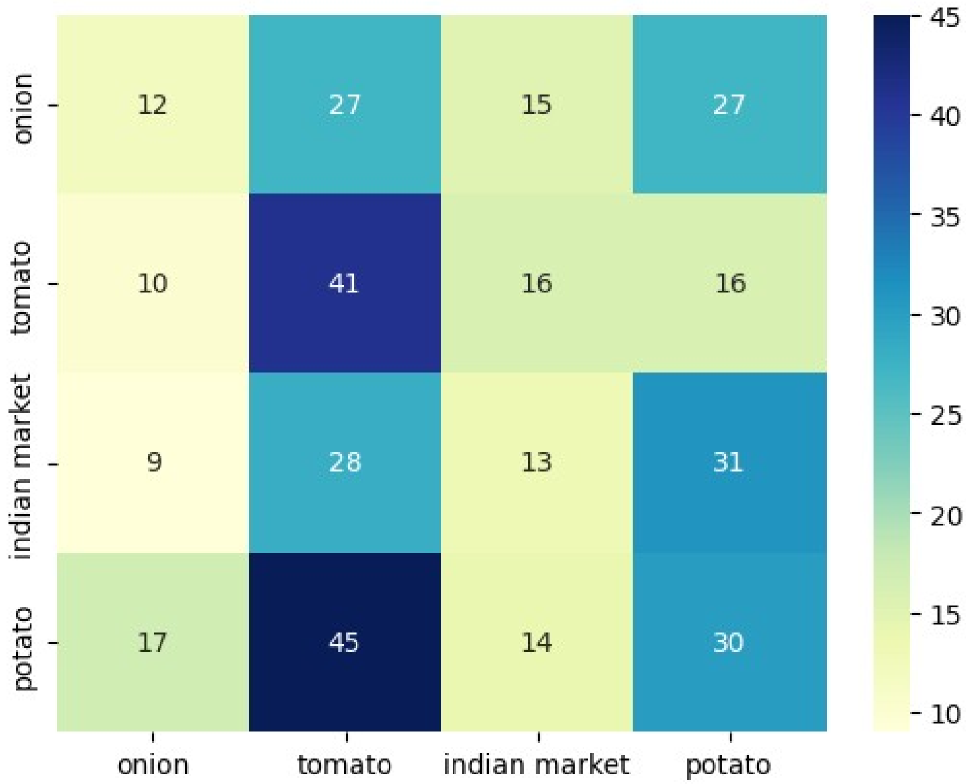
```
Epoch 1/5
79/79 [==============================] - 18s 128ms/step - loss: 1.0201 
- accuracy: 0.5443 - val_loss: 0.7741 - val_accuracy: 0.6683
Epoch 2/5
79/79 [==============================] - 12s 147ms/step - loss: 0.6591 
- accuracy: 0.7552 - val_loss: 0.6689 - val_accuracy: 0.7400
Epoch 3/5
79/79 [==============================] - 9s 111ms/step - loss: 0.6315 
- accuracy: 0.7576 - val_loss: 0.5752 - val_accuracy: 0.7656
Epoch 4/5
79/79 [==============================] - 9s 107ms/step - loss: 0.5820 
- accuracy: 0.7775 - val_loss: 0.5651 - val_accuracy: 0.7544
Epoch 5/5
79/79 [==============================] - 8s 90ms/step - loss: 0.5452 - 
accuracy: 0.7863 - val_loss: 0.5467 - val_accuracy: 0.7608
```

```
plot_accuracy(cnn_model)
```




```
# Evaluate the model
class_names = ['indian market', 'onion', 'potato', 'tomato']
loss, acc = cnn_model.evaluate(test_ds, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

y_pred = cnn_model.predict(test_ds)
predicted_categories = tf.argmax(y_pred, axis=1)
true_cat = tf.concat([y for x, y in test_ds], axis=0)
true_categories = tf.argmax(true_cat, axis=1)

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

classwise_accuracy(noise_path, 'indian market', cnn_model)
classwise_accuracy(onion_path, 'onion', cnn_model)
classwise_accuracy(potato_path, 'potato', cnn_model)
classwise_accuracy(tomato_path, 'tomato', cnn_model)

ConfusionMatrix(cnn_model, test_ds, class_dirs)

11/11 - 2s - loss: 0.6833 - accuracy: 0.6952 - 2s/epoch - 161ms/step
Restored model, accuracy: 69.52%
```

```
11/11 [==============================] - 2s 38ms/step

Test Accuracy: 27.35%

Accuracy for class indian market is 43.21% consisting of 81 images
Accuracy for class onion is 78.31% consisting of 83 images
Accuracy for class potato is 49.38% consisting of 81 images
Accuracy for class tomato is 96.23% consisting of 106 images
```



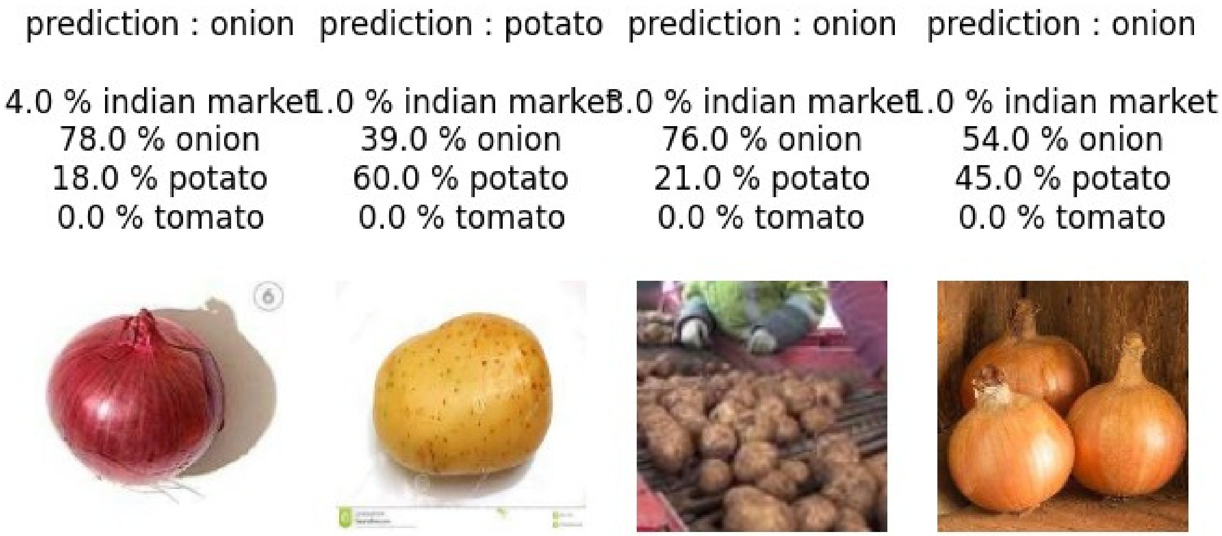

```
test_dir = '/content/ninjacart_data/test'
test_images = []
class_names = ['indian market', 'onion', 'potato', 'tomato']

for folder in os.listdir(test_dir):
  for image in os.listdir(test_dir + '/' + folder):
    test_images.append(os.path.join(test_dir, folder, image))
img_0 = tf.keras.utils.load_img(random.choice(test_images))
img_0 = tf.keras.utils.img_to_array(img_0)
img_0 = tf.image.resize(img_0, (128, 128))
img_1 = tf.expand_dims(img_0, axis = 0)

def grid_test_model(model_name):

  fig = plt.figure(1, figsize=(8, 8))
  plt.axis('off')
```

```
  n = 0
  for i in range(4):
    n += 1

    img_0 = tf.keras.utils.load_img(random.choice(test_images))
    img_0 = tf.keras.utils.img_to_array(img_0)
    img_0 = tf.image.resize(img_0, (128, 128))
    img_1 = tf.expand_dims(img_0, axis = 0)

    pred = model_name.predict(img_1)
    predicted_label = tf.argmax(pred, 1).numpy().item()

    for item in pred :
      item = tf.round((item*100))

    plt.subplot(2, 4, n)
    plt.axis('off')
    plt.title(f'prediction : {class_names[predicted_label]}\n\n'
              f'{item[0]} % {class_names[0]}\n'
              f'{item[1]} % {class_names[1]}\n'
              f'{item[2]} % {class_names[2]}\n'
              f'{item[3]} % {class_names[3]}\n')
    plt.imshow(img_0/255)
  plt.show()
grid_test_model(cnn_model)
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```

```
<ipython-input-21-40cb159e3305>:32: MatplotlibDeprecationWarning:
Auto-removal of overlapping axes is deprecated since 3.6 and will be
removed two minor releases later; explicitly call ax.remove() as
needed.
  plt.subplot(2, 4, n)
```

```
1/1 [==============================] - 0s 17ms/step
```

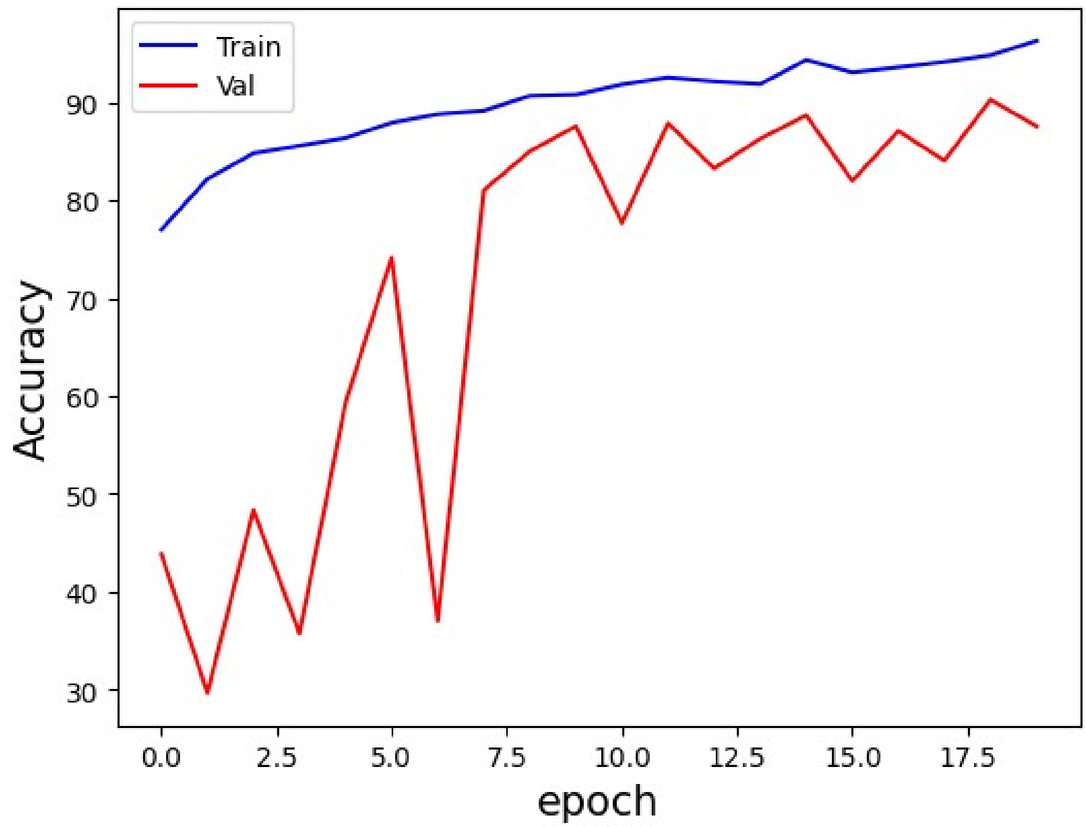



```
augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal_and_vertical"),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomTranslation(height_factor = 0.2,
width_factor=0.2)
])

aug_ds = train_ds

for image, label in aug_ds :
  image = augmentation(image)

cnn_model_imp_performance=keras.Sequential(
    name='cnn_model_imp_performance',
    layers=[
    layers.Rescaling(1./255)

    ,layers.Conv2D(filters=32,kernel_size=3,padding='same',activation=
'relu',input_shape=(128,128,3))
    ,layers.BatchNormalization()
    ,layers.MaxPooling2D()

    ,layers.Conv2D(filters=32,kernel_size=3,padding='same',activation=
'relu')
    ,layers.BatchNormalization()
    ,layers.MaxPooling2D()

    ,layers.Conv2D(filters=64,kernel_size=3,padding='same',activation=
'relu')
    ,layers.BatchNormalization()
    ,layers.MaxPooling2D()

    ,layers.Conv2D(filters=64,kernel_size=3,padding='same',activation=
```

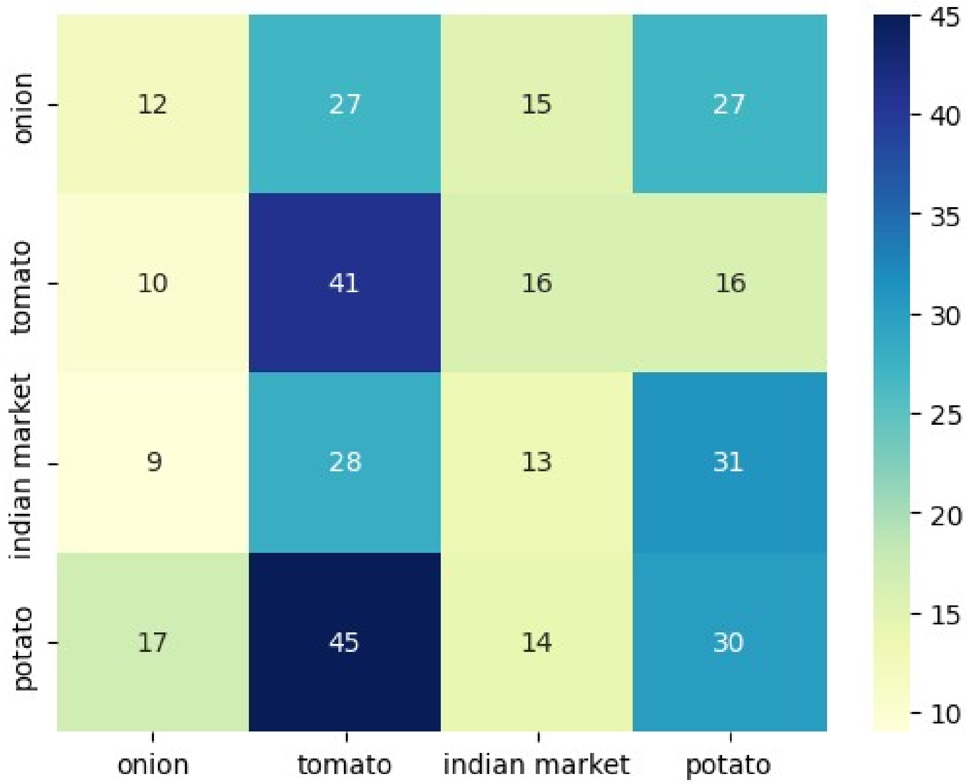
```
'relu')
    ,layers.BatchNormalization()
    ,layers.MaxPooling2D()

    ,layers.Conv2D(filters=128,kernel_size=3,padding='same',activation
='relu')
    ,layers.BatchNormalization()
    ,layers.GlobalAveragePooling2D()
#    ,layers.AveragePooling2D()
    #,layers.Flatten()
    #,layers.Dense(256,activation='relu')
    ,layers.Dropout(0.2)
    ,layers.Dense(4,activation='softmax')
])

cnn_model_imp_performance.compile(optimizer='Adam',loss='categorical_c
rossentropy',metrics='accuracy')
history=cnn_model_imp_performance.fit(train_ds,epochs=20,validation_da
ta=val_ds)
```

```
Epoch 1/20
79/79 [==============================] - 15s 125ms/step - loss: 0.6369
- accuracy: 0.7699 - val_loss: 1.2843 - val_accuracy: 0.4386
Epoch 2/20
79/79 [==============================] - 8s 93ms/step - loss: 0.4591 -
accuracy: 0.8218 - val_loss: 1.4006 - val_accuracy: 0.2967
Epoch 3/20
79/79 [==============================] - 7s 85ms/step - loss: 0.3978 -
accuracy: 0.8481 - val_loss: 1.2212 - val_accuracy: 0.4833
Epoch 4/20
79/79 [==============================] - 9s 104ms/step - loss: 0.3658
- accuracy: 0.8557 - val_loss: 1.8263 - val_accuracy: 0.3573
Epoch 5/20
79/79 [==============================] - 8s 99ms/step - loss: 0.3426 -
accuracy: 0.8636 - val_loss: 1.0784 - val_accuracy: 0.5933
Epoch 6/20
79/79 [==============================] - 7s 85ms/step - loss: 0.3233 -
accuracy: 0.8792 - val_loss: 0.6574 - val_accuracy: 0.7416
Epoch 7/20
79/79 [==============================] - 10s 118ms/step - loss: 0.3025
- accuracy: 0.8880 - val_loss: 2.5867 - val_accuracy: 0.3700
Epoch 8/20
79/79 [==============================] - 10s 116ms/step - loss: 0.2858
- accuracy: 0.8915 - val_loss: 0.4279 - val_accuracy: 0.8102
Epoch 9/20
79/79 [==============================] - 7s 85ms/step - loss: 0.2568 -
accuracy: 0.9067 - val_loss: 0.4054 - val_accuracy: 0.8501
Epoch 10/20
79/79 [==============================] - 10s 114ms/step - loss: 0.2381
- accuracy: 0.9079 - val_loss: 0.2982 - val_accuracy: 0.8756
```

```
Epoch 11/20
79/79 [==============================] - 9s 104ms/step - loss: 0.2266
- accuracy: 0.9183 - val_loss: 0.6611 - val_accuracy: 0.7767
Epoch 12/20
79/79 [==============================] - 7s 85ms/step - loss: 0.2004 -
accuracy: 0.9254 - val_loss: 0.3064 - val_accuracy: 0.8788
Epoch 13/20
79/79 [==============================] - 9s 102ms/step - loss: 0.2161
- accuracy: 0.9215 - val_loss: 0.4336 - val_accuracy: 0.8325
Epoch 14/20
79/79 [==============================] - 9s 102ms/step - loss: 0.2114
- accuracy: 0.9187 - val_loss: 0.3485 - val_accuracy: 0.8628
Epoch 15/20
79/79 [==============================] - 7s 84ms/step - loss: 0.1654 -
accuracy: 0.9434 - val_loss: 0.3499 - val_accuracy: 0.8868
Epoch 16/20
79/79 [==============================] - 7s 85ms/step - loss: 0.1667 -
accuracy: 0.9306 - val_loss: 0.5411 - val_accuracy: 0.8198
Epoch 17/20
79/79 [==============================] - 9s 103ms/step - loss: 0.1757
- accuracy: 0.9358 - val_loss: 0.3916 - val_accuracy: 0.8708
Epoch 18/20
79/79 [==============================] - 8s 95ms/step - loss: 0.1587 -
accuracy: 0.9414 - val_loss: 0.5229 - val_accuracy: 0.8405
Epoch 19/20
79/79 [==============================] - 7s 86ms/step - loss: 0.1393 -
accuracy: 0.9482 - val_loss: 0.2485 - val_accuracy: 0.9027
Epoch 20/20
79/79 [==============================] - 9s 104ms/step - loss: 0.1122
- accuracy: 0.9629 - val_loss: 0.4119 - val_accuracy: 0.8756

plot_accuracy(cnn_model_imp_performance)
```




```
model_evaluation_acc(cnn_model_imp_performance)

11/11 - 1s - loss: 0.5283 - accuracy: 0.8063 - 1s/epoch - 106ms/step
Restored model, accuracy: 80.63%
11/11 [==============================] - 1s 21ms/step

Test Accuracy: 24.50%

Accuracy for class indian market is 70.37% consisting of 81 images
Accuracy for class onion is 63.86% consisting of 83 images
Accuracy for class potato is 83.95% consisting of 81 images
Accuracy for class tomato is 98.11% consisting of 106 images
```



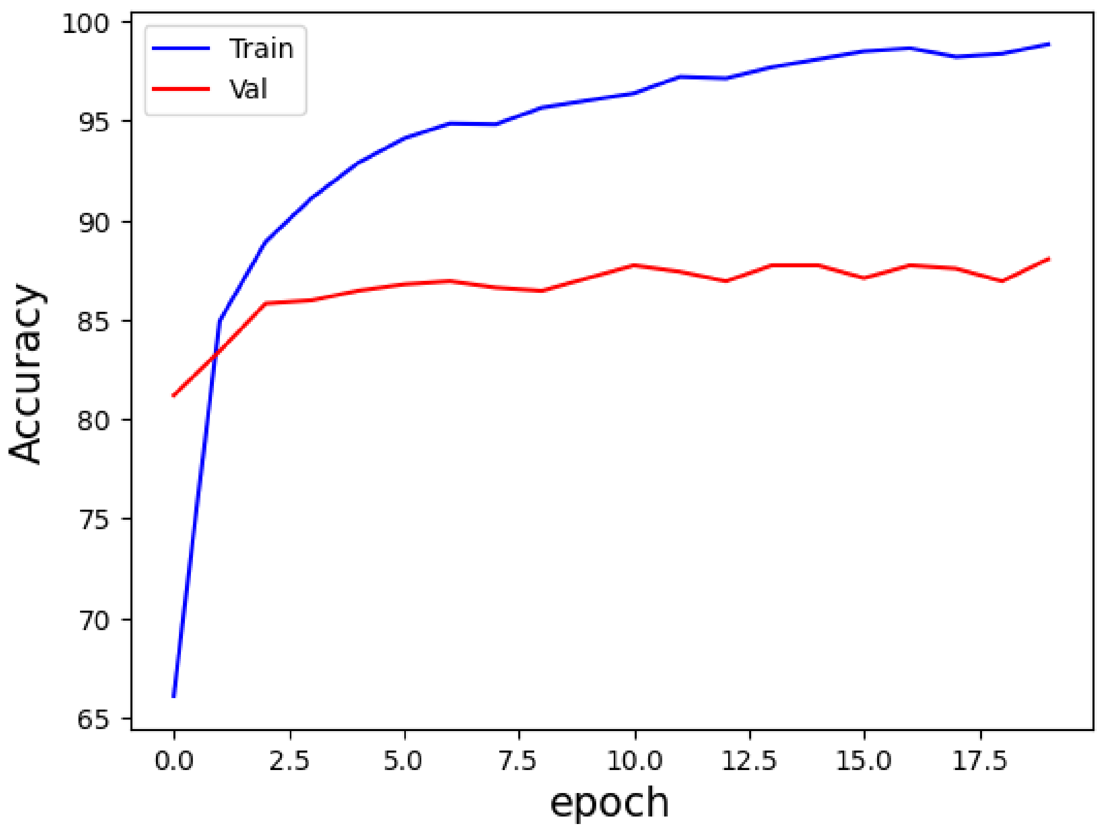
We observe an accuracy jump of ~10 % by just :

```
Applying augmentation to our data
Adding Dropout and BatchNormalization
Implementing callbacks during training

Use a model of your choice (could be vgg, resnet and mobilenet) and
train it with an appropriate batch size.

Using the pretrained weights of popular networks is a great way to do
 transfer learning, since the size of our original dataset is small.

early_stopping_cb = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss', patience = 5, restore_best_weights=True
)
pretrained_model=tf.keras.applications.VGG16(weights='imagenet',includ
e_top=False,input_shape=(128,128,3))
pretrained_model.trainable=False
VGG16_model=tf.keras.Sequential([
    pretrained_model,
    #layers.Flatten(),
    layers.GlobalAveragePooling2D(),
```

```
    layers.Dense(4,activation='softmax')
    ])
VGG16_model.compile(optimizer='Adam', metrics='accuracy',
loss='categorical_crossentropy')

history=VGG16_model.fit(train_ds,epochs=20,validation_data=val_ds,call
backs=[early_stopping_cb],verbose=0)
#pretrained_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 4s 0us/step
```

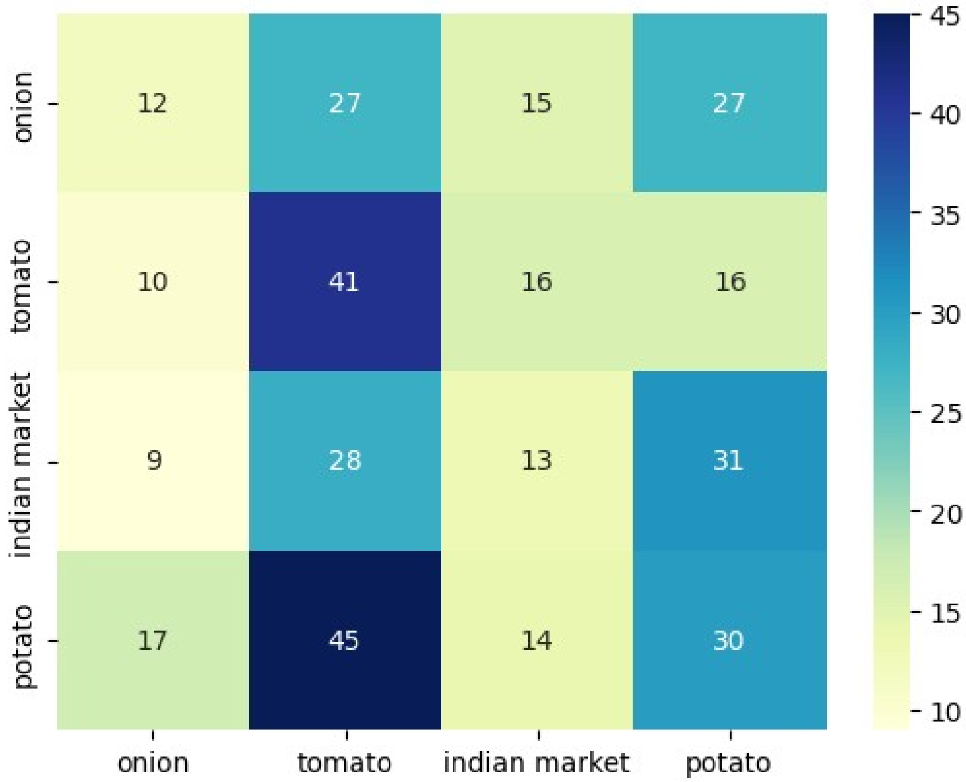
```
plot_accuracy(VGG16_model)
```




```
model_evaluation_acc(VGG16_model)
```

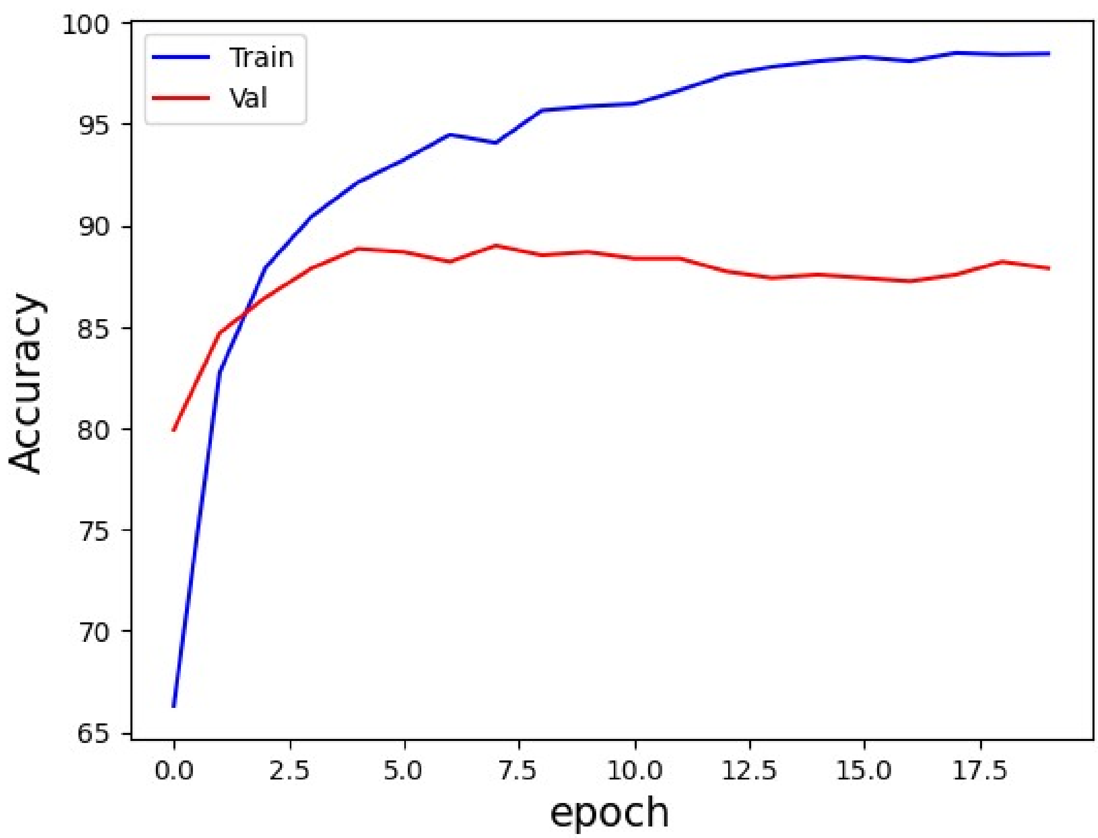
```
11/11 - 5s - loss: 1.0675 - accuracy: 0.8632 - 5s/epoch - 431ms/step
Restored model, accuracy: 86.32%
11/11 [==============================] - 1s 48ms/step

Test Accuracy: 23.65%

Accuracy for class indian market is 85.19% consisting of 81 images
```

```
Accuracy for class onion is 89.16% consisting of 83 images
Accuracy for class potato is 67.9% consisting of 81 images
Accuracy for class tomato is 99.06% consisting of 106 images
```

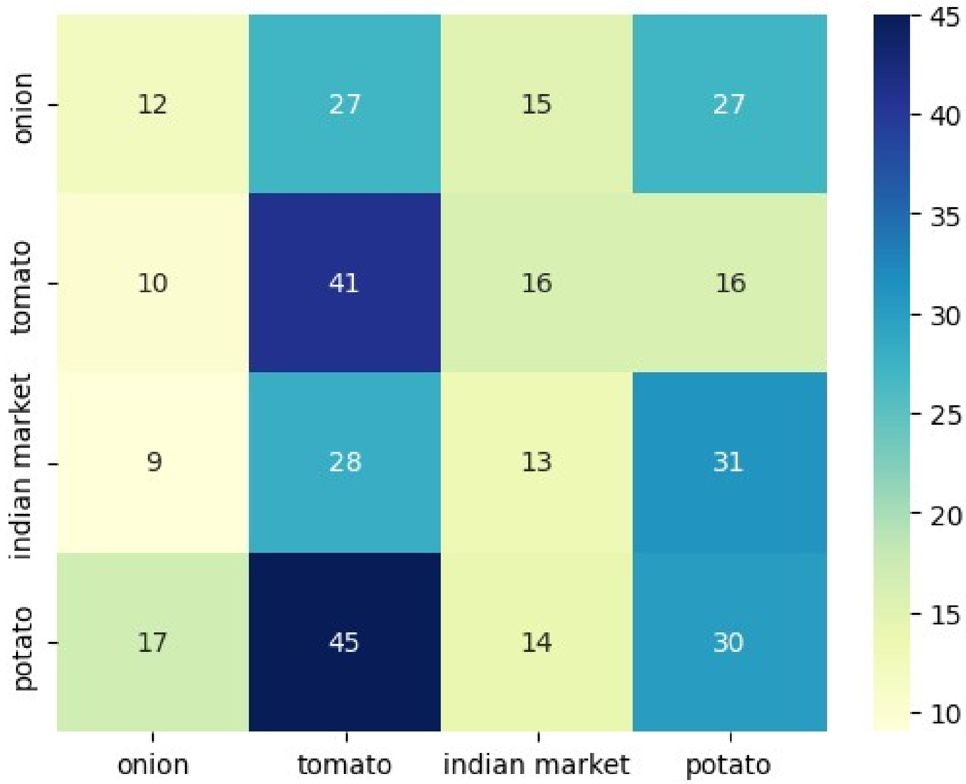



```
'''early_stopping_cb = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss', patience = 5, restore_best_weights=True
)'''
pretrained_model=tf.keras.applications.VGG19(weights='imagenet',includ
e_top=False,input_shape=(128,128,3))
pretrained_model.trainable=False
VGG19_model=tf.keras.Sequential([
    pretrained_model,
    #layers.Flatten(),
    layers.GlobalAveragePooling2D(),
    layers.Dense(4,activation='softmax')
    ])
VGG19_model.compile(optimizer='Adam', metrics='accuracy',
loss='categorical_crossentropy')

history=VGG19_model.fit(aug_ds,epochs=20,validation_data=val_ds,verbos
e=0)
#pretrained_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 5s 0us/step

plot_accuracy(VGG19_model)
model_evaluation_acc(VGG19_model)
```




```
11/11 - 1s - loss: 1.3534 - accuracy: 0.8575 - 1s/epoch - 113ms/step
Restored model, accuracy: 85.75%
11/11 [==============================] - 1s 57ms/step

Test Accuracy: 22.22%

Accuracy for class indian market is 83.95% consisting of 81 images
Accuracy for class onion is 81.93% consisting of 83 images
Accuracy for class potato is 72.84% consisting of 81 images
Accuracy for class tomato is 100.0% consisting of 106 images
```




```
'''early_stopping_cb = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss', patience = 5, restore_best_weights=True
)'''
pretrained_model=tf.keras.applications.ResNet101(weights='imagenet',in
clude_top=False,input_shape=(128,128,3))
pretrained_model.trainable=False
ResNet101_model=tf.keras.Sequential([
    pretrained_model,
    #layers.Flatten(),
    layers.GlobalAveragePooling2D(),
    layers.Dense(4,activation='softmax')
    ])
ResNet101_model.compile(optimizer='Adam',
metrics=['accuracy','Precision','Recall'],
loss='categorical_crossentropy')

history=ResNet101_model.fit(train_ds,epochs=20,validation_data=val_ds)
#pretrained_model.summary()

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet101_weights_tf_dim_ordering_tf_kernels_notop.h5
171446536/171446536 [==============================] - 9s 0us/step
Epoch 1/20
79/79 [==============================] - 21s 158ms/step - loss: 0.4259
```

```
- accuracy: 0.8361 - precision: 0.8530 - recall: 0.8170 - val_loss:
0.2283 - val_accuracy: 0.9171 - val_precision: 0.9241 - val_recall:
0.9123
Epoch 2/20
79/79 [==============================] - 9s 103ms/step - loss: 0.1343
- accuracy: 0.9498 - precision: 0.9550 - recall: 0.9482 - val_loss:
0.2086 - val_accuracy: 0.9314 - val_precision: 0.9343 - val_recall:
0.9298
Epoch 3/20
79/79 [==============================] - 8s 96ms/step - loss: 0.0904 -
accuracy: 0.9701 - precision: 0.9735 - recall: 0.9677 - val_loss:
0.2161 - val_accuracy: 0.9314 - val_precision: 0.9373 - val_recall:
0.9298
Epoch 4/20
79/79 [==============================] - 9s 97ms/step - loss: 0.0689 -
accuracy: 0.9829 - precision: 0.9832 - recall: 0.9777 - val_loss:
0.1729 - val_accuracy: 0.9410 - val_precision: 0.9469 - val_recall:
0.9394
Epoch 5/20
79/79 [==============================] - 10s 115ms/step - loss: 0.0453
- accuracy: 0.9884 - precision: 0.9888 - recall: 0.9872 - val_loss:
0.1598 - val_accuracy: 0.9522 - val_precision: 0.9521 - val_recall:
0.9506
Epoch 6/20
79/79 [==============================] - 9s 111ms/step - loss: 0.0371
- accuracy: 0.9916 - precision: 0.9920 - recall: 0.9912 - val_loss:
0.1935 - val_accuracy: 0.9426 - val_precision: 0.9501 - val_recall:
0.9410
Epoch 7/20
79/79 [==============================] - 8s 97ms/step - loss: 0.0263 -
accuracy: 0.9956 - precision: 0.9956 - recall: 0.9952 - val_loss:
0.1586 - val_accuracy: 0.9537 - val_precision: 0.9551 - val_recall:
0.9506
Epoch 8/20
79/79 [==============================] - 10s 105ms/step - loss: 0.0221
- accuracy: 0.9964 - precision: 0.9968 - recall: 0.9964 - val_loss:
0.1693 - val_accuracy: 0.9490 - val_precision: 0.9518 - val_recall:
0.9458
Epoch 9/20
79/79 [==============================] - 10s 114ms/step - loss: 0.0169
- accuracy: 0.9984 - precision: 0.9984 - recall: 0.9980 - val_loss:
0.1559 - val_accuracy: 0.9569 - val_precision: 0.9584 - val_recall:
0.9553
Epoch 10/20
79/79 [==============================] - 9s 108ms/step - loss: 0.0137
- accuracy: 0.9996 - precision: 0.9996 - recall: 0.9992 - val_loss:
0.1663 - val_accuracy: 0.9506 - val_precision: 0.9535 - val_recall:
0.9490
Epoch 11/20
```

```
79/79 [==============================] - 9s 99ms/step - loss: 0.0115 -
accuracy: 0.9996 - precision: 0.9996 - recall: 0.9992 - val_loss:
0.1670 - val_accuracy: 0.9522 - val_precision: 0.9536 - val_recall:
0.9506
Epoch 12/20
79/79 [==============================] - 10s 115ms/step - loss: 0.0100
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1633 - val_accuracy: 0.9553 - val_precision: 0.9569 - val_recall:
0.9553
Epoch 13/20
79/79 [==============================] - 11s 128ms/step - loss: 0.0100
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1774 - val_accuracy: 0.9458 - val_precision: 0.9471 - val_recall:
0.9426
Epoch 14/20
79/79 [==============================] - 10s 113ms/step - loss: 0.0082
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1691 - val_accuracy: 0.9537 - val_precision: 0.9553 - val_recall:
0.9537
Epoch 15/20
79/79 [==============================] - 9s 110ms/step - loss: 0.0078
- accuracy: 0.9996 - precision: 0.9996 - recall: 0.9996 - val_loss:
0.1687 - val_accuracy: 0.9537 - val_precision: 0.9553 - val_recall:
0.9537
Epoch 16/20
79/79 [==============================] - 10s 109ms/step - loss: 0.0062
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1743 - val_accuracy: 0.9537 - val_precision: 0.9537 - val_recall:
0.9537
Epoch 17/20
79/79 [==============================] - 10s 115ms/step - loss: 0.0056
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1700 - val_accuracy: 0.9537 - val_precision: 0.9552 - val_recall:
0.9522
Epoch 18/20
79/79 [==============================] - 9s 112ms/step - loss: 0.0050
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1785 - val_accuracy: 0.9506 - val_precision: 0.9506 - val_recall:
0.9506
Epoch 19/20
79/79 [==============================] - 9s 100ms/step - loss: 0.0047
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1692 - val_accuracy: 0.9522 - val_precision: 0.9522 - val_recall:
0.9522
Epoch 20/20
79/79 [==============================] - 10s 117ms/step - loss: 0.0042
- accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - val_loss:
0.1818 - val_accuracy: 0.9522 - val_precision: 0.9522 - val_recall:
0.9522
```

```
loss, acc,precision,recall = ResNet101_model.evaluate(test_ds,
verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 *
acc),'precision:',precision,'recall:',recall)
```

```
11/11 - 2s - loss: 0.5180 - accuracy: 0.8803 - precision: 0.8851 -
recall: 0.8775 - 2s/epoch - 209ms/step
Restored model, accuracy: 88.03% precision: 0.8850574493408203 recall:
0.8774929046630859
```

```
VGG19_model.metrics_names
```

```
['loss', 'accuracy']
```

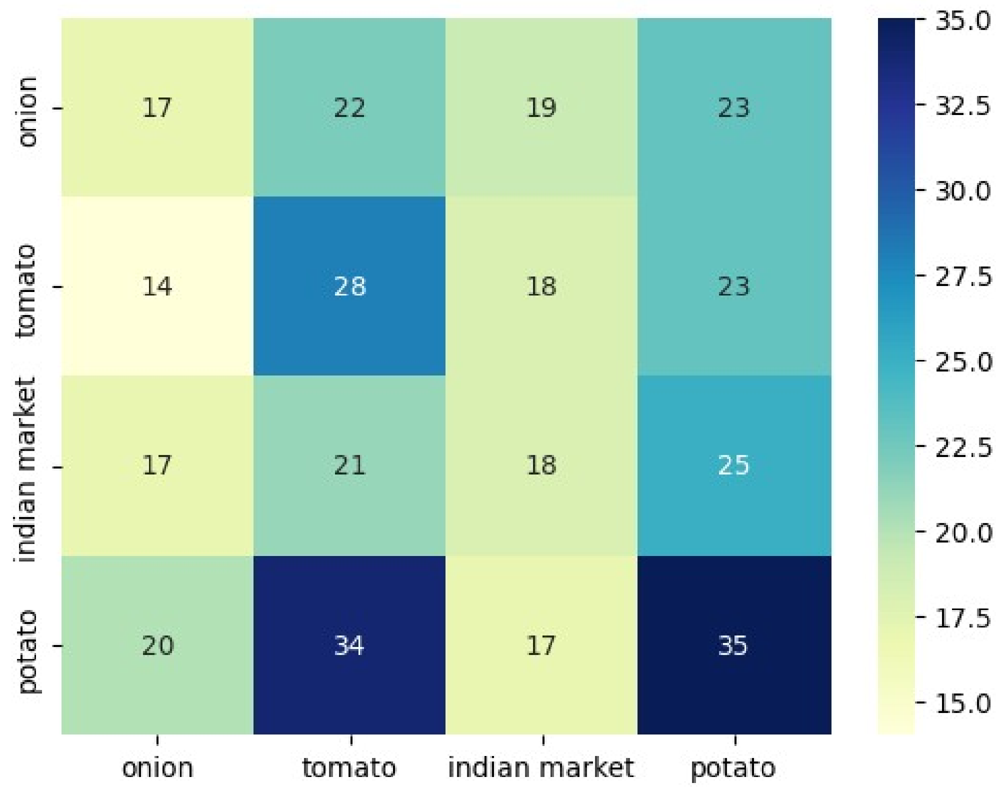
```
ResNet101_model.metrics_names
```

```
['loss', 'accuracy', 'precision', 'recall']
```

```
def plot_accuracy_ResNet(model_fit):
    #accuracy graph

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

#plot_accuracy(ResNet101_model)
loss, accuracy,precision,recall = ResNet101_model.evaluate(test_ds,
verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 *
acc),'precision:',precision,'recall:',recall)

y_pred = ResNet101_model.predict(test_ds)
predicted_categories = tf.argmax(y_pred, axis=1)
true_cat = tf.concat([y for x, y in test_ds], axis=0)
true_categories = tf.argmax(true_cat, axis=1)

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

```
classwise_accuracy(noise_path, 'indian market', ResNet101_model)
classwise_accuracy(onion_path, 'onion', ResNet101_model)
classwise_accuracy(potato_path, 'potato', ResNet101_model)
classwise_accuracy(tomato_path, 'tomato', ResNet101_model)

ConfusionMatrix(cnn_model, test_ds, class_dirs)
```

```
11/11 - 1s - loss: 0.5180 - accuracy: 0.8803 - precision: 0.8851 -
recall: 0.8775 - 1s/epoch - 122ms/step
Restored model, accuracy: 88.03% precision: 0.8850574493408203 recall:
0.8774929046630859
11/11 [==============================] - 3s 67ms/step

Test Accuracy: 27.92%

Accuracy for class indian market is 81.48% consisting of 81 images
Accuracy for class onion is 90.36% consisting of 83 images
Accuracy for class potato is 76.54% consisting of 81 images
Accuracy for class tomato is 100.0% consisting of 106 images
```



Testing our best model (ResNet)

```
ResNet achieved the highest accuracy of 94% among all the models we
trained

Let's test our Finetuned ResNet to predict on some random unseen data
to visualize how accurate it is !

ResNet101_model.summary()

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet101 (Functional)      (None, 4, 4, 2048)        42658176

 global_average_pooling2d_4  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 4)                 8196

=================================================================
Total params: 42666372 (162.76 MB)
Trainable params: 8196 (32.02 KB)
Non-trainable params: 42658176 (162.73 MB)
_________________________________________________________________

grid_test_model(ResNet101_model)

1/1 [==============================] - 0s 28ms/step

<ipython-input-21-40cb159e3305>:32: MatplotlibDeprecationWarning:
Auto-removal of overlapping axes is deprecated since 3.6 and will be
removed two minor releases later; explicitly call ax.remove() as
needed.
  plt.subplot(2, 4, n)

1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 37ms/step
```

prediction : potato

0.0 % indian market
0.0 % onion
100.0 % potato
0.0 % tomato

prediction : tomato

7.0 % indian market
0.0 % onion
1.0 % potato
92.0 % tomato

prediction : tomato

0.0 % indian market
0.0 % onion
0.0 % potato
100.0 % tomato

prediction : tomato

0.0 % indian market
0.0 % onion
0.0 % potato
100.0 % tomato