Ninjacart Case Study

About Ninjacart

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.

Problem statement

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.

Objective

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

Data

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

The train data contains the following number of images Tomato : 789 Potato : 898 Onion : 849 Indian market : 599

The train data contains the following number of images Tomato : 106 potato : 83 onion : 81 Indian market : 81

Downloading and unzipping raw data

```
!gdown 1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT
```

```
Downloading...
From: https://drive.google.com/uc?id=1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT
To: /content/ninjacart_data.zip
100% 275M/275M [00:08<00:00, 32.4MB/s]
```

```
!unzip -q /content/ninjacart_data.zip
```

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import shutil
import glob
import sklearn
from sklearn import metrics
from sklearn.model_selection import train_test_split
import tensorflow as tf
import random
!pip install optuna
import optuna
from optuna.visualization.matplotlib import plot_param_importances
!pip install tensorboard

random.seed(21)
np.random.seed(21)
tf.random.set_seed(21)
```

```
Collecting optuna
  Downloading optuna-3.4.0-py3-none-any.whl (409 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 409.6/409.6 kB 5.8 MB/s eta
0:00:00
bic>=1.5.0 (from optuna)
  Downloading alembic-1.12.1-py3-none-any.whl (226 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.8/226.8 kB 12.4 MB/s eta
0:00:00
 optuna)
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from optuna) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (23.2)
Requirement already satisfied: sqlalchemy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (2.0.23)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from optuna) (4.66.1)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from optuna) (6.0.1)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.0-py3-none-any.whl (78 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.6/78.6 kB 10.8 MB/s eta
0:00:00
ent already satisfied: typing-extensions>=4 in
/usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna)
(4.5.0)
```

```
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0-
>optuna) (3.0.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0-
>optuna) (2.1.3)
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.0 alembic-1.12.1 colorlog-6.7.0
optuna-3.4.0
Requirement already satisfied: tensorboard in
/usr/local/lib/python3.10/dist-packages (2.14.1)
Requirement already satisfied: absl-py>=0.4 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (1.4.0)
Requirement already satisfied: grpcio>=1.48.2 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (1.59.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (3.5.1)
Requirement already satisfied: numpy>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (1.23.5)
Requirement already satisfied: protobuf>=3.19.6 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (2.31.0)
Requirement already satisfied: setuptools>=41.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (67.7.2)
Requirement already satisfied: six>1.9 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (1.16.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from tensorboard) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
```

```
>tensorboard) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard) (3.2.2)
```

Defining static variables

```
# defining static values

# directories of the raw and processed data
ROOT_PATH = os.getcwd()
RAW_DATA_PATH = os.path.join(ROOT_PATH,'ninjacart_data')
DATA_PATH = os.path.join(ROOT_PATH,'data')

# folders for train, val and cv
TRAIN_FOLDER = 'train'
VAL_FOLDER = 'val'
TEST_FOLDER = 'test'
```

Defining utility functions

```
# function to make a directory for processed data
def
createDataDirectory(data_path,train_folder,val_folder,test_folder,cate
gory_names):

    # creating the parent directory
    if os.path.isdir(DATA_PATH):
        pass
    else:
        os.mkdir(DATA_PATH)

    for folder in [train_folder,val_folder,test_folder]:

        # creating train, test and val directories
```

```
        if os.path.isdir(os.path.join(DATA_PATH,folder)):
            pass
        else:
            os.mkdir(os.path.join(DATA_PATH,folder))

        # creating category directories in train, test and val folders
        for cat in category_names:
            path = os.path.join(DATA_PATH,folder,cat)
            if os.path.isdir(path):
                pass
            else:
                os.mkdir(path)

# function to split the train images into train and val with 80% train
and 20% val
def splitData(image_names):
    train_images, val_images =
train_test_split(image_names,test_size=0.2,random_state=21)
    return train_images,val_images
```

Creating train and val split for the images (one time run)

```
# creating directories for the splitted images
category_names = os.listdir(os.path.join(RAW_DATA_PATH,TRAIN_FOLDER))
createDataDirectory(DATA_PATH,TRAIN_FOLDER,VAL_FOLDER,TEST_FOLDER,cate
gory_names)

# splitting train images into train and test

# getting all the image names from train folder categories
for cat in category_names:

    # defining paths
    from_path = os.path.join(RAW_DATA_PATH,TRAIN_FOLDER,cat)
    to_path_train = os.path.join(DATA_PATH,TRAIN_FOLDER,cat)
    to_path_val = os.path.join(DATA_PATH,VAL_FOLDER,cat)
    from_path_test = os.path.join(RAW_DATA_PATH,TEST_FOLDER,cat)
    to_path_test = os.path.join(DATA_PATH,TEST_FOLDER,cat)

    # extracting image names from the train folder
    _,_,image_names = list(os.walk(from_path))[0]

    # splitting the images into train and val
    train_images, val_images = splitData(image_names)

    # copying the images from old to new directory
```

```
    # 1. train images
    for image in train_images:
        shutil.copyfile(
            os.path.join(from_path,image),
            os.path.join(to_path_train,image)
        )
    # 2. val images
    for image in val_images:
        shutil.copyfile(
            os.path.join(from_path,image),
            os.path.join(to_path_val,image)
        )
    # 2. test images
    # extracting image names from the test folder
    _,_,test_images = list(os.walk(from_path_test))[0]
    for image in test_images:
        shutil.copyfile(
            os.path.join(from_path_test,image),
            os.path.join(to_path_test,image)
        )
```

Visualizing the images

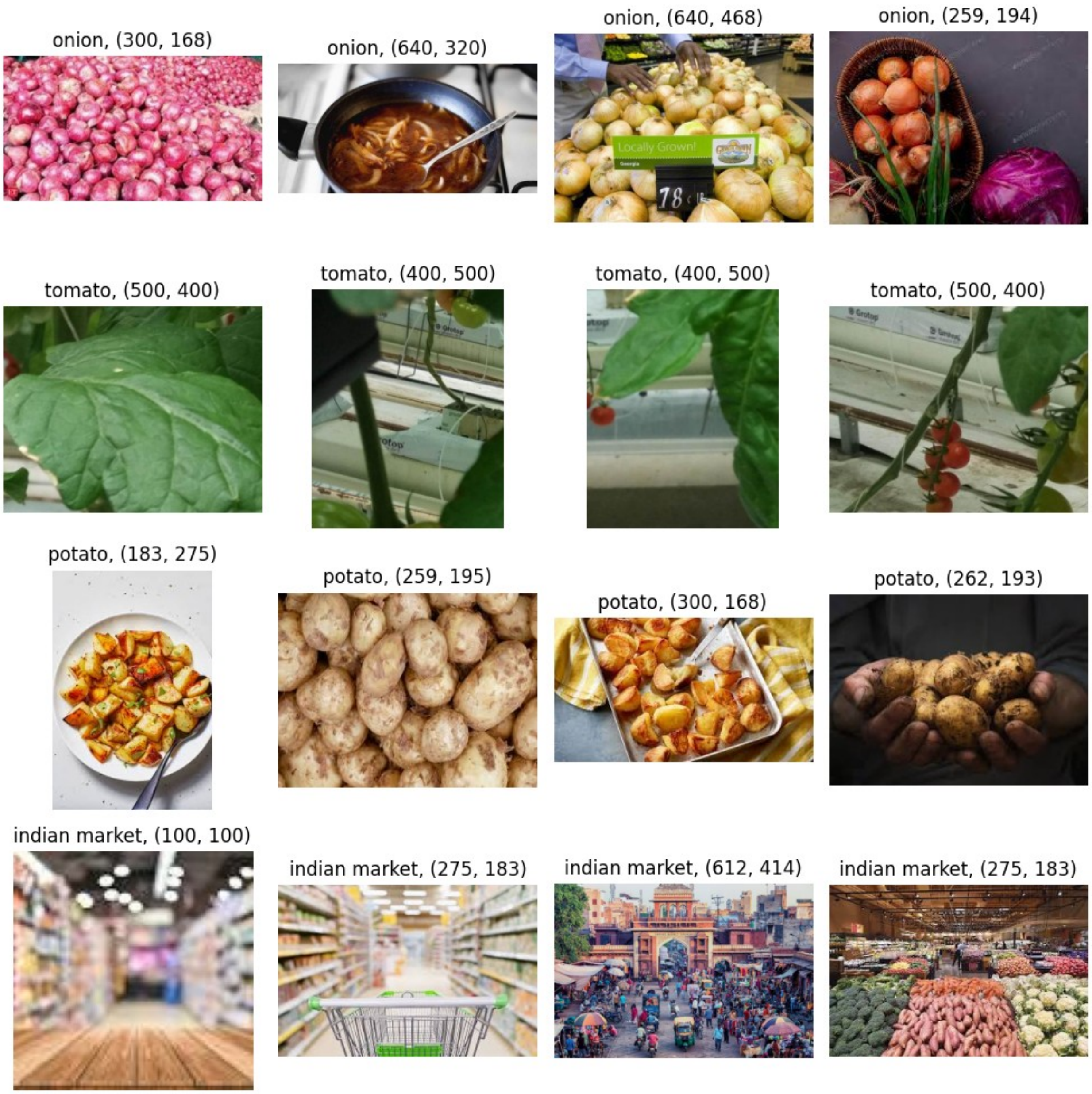
Displaying sample images from each category

```
category_names = os.listdir(os.path.join(DATA_PATH,TRAIN_FOLDER))
count_dict = dict()
image_dict = dict()
size_list = list()

for cat in category_names:
    dir_path = os.path.join(DATA_PATH,TRAIN_FOLDER,cat)
    filenames = glob.glob(dir_path+'/*')
    count_dict[cat] = len(filenames)
    images = np.random.choice(filenames,size=(4,),replace=False)
    image_dict[cat] = [tf.keras.utils.load_img(img) for img in images]
    size_list.extend([tf.keras.utils.load_img(img).size for img in
filenames])
```

Sample images

```
plt.figure(figsize=(10,10))

for i,(cat,images) in enumerate(image_dict.items()):
    for j,img in enumerate(images):
        plt.subplot(4,4,4*i+j+1)
        plt.imshow(img)
```

```
        plt.title(f"{cat}, {img.size}")
        plt.axis('off')
plt.tight_layout()
plt.show()
```



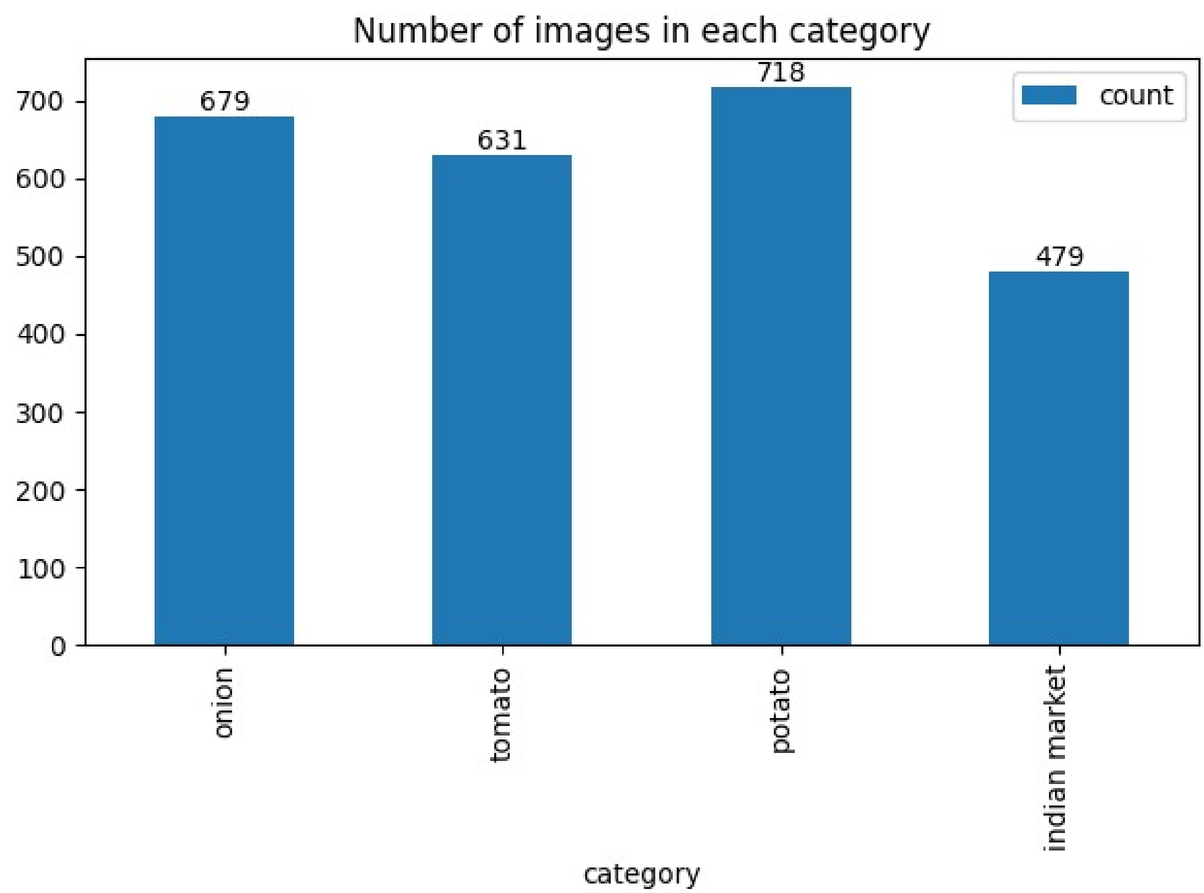
Observations

- The images of tomato are very vague. In some images, tomato is barely visible. So probably the neural network will learn how the leaves look and predict whether it is tomato or not.
- In some of the onion images, the onions looks like potato due to their color. So it might be confusion for the network to distinguish between potato and onion for these kind of images.

- Some of the market images contains vegetables. If these vegetables are either potato, tomato or onion, the network is going to have a hard time distinguishing them.

Count of images in each category

```
plt.figure(figsize=(6,6))
ax = pd.DataFrame(
    data = {
        'category':count_dict.keys(),
        'count':count_dict.values()
    }
).plot(kind='bar',x='category',y='count')

ax.bar_label(ax.containers[0])
plt.title('Number of images in each category')
plt.tight_layout()
plt.show()

<Figure size 600x600 with 0 Axes>
```

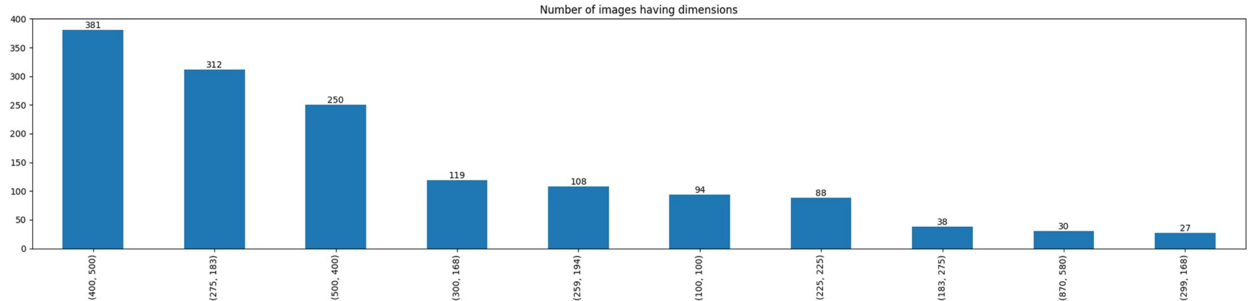


Observations

- The count of images of each class is very close to each other
- The model can perform well without class balancing

Different sizes of images

```
plt.figure(figsize=(20,5))
ax = pd.Series(size_list).value_counts()
[0:10].plot(kind='bar',x='dimension',y='count')
ax.bar_label(ax.containers[0])
plt.title('Number of images having dimensions')
plt.tight_layout()
plt.show()
```



Observations

- There are different sizes of images present in the dataset
- We need to resize each images before feeding it to the model
- This can be achieved in the image generator
- The images which have a very high or very low aspect ratio will be hampered since by resizing so much, the information in them might be corrupted

Verifying the number of datapoints given in the problem statement

```
verification = dict()
for folder in [TRAIN_FOLDER,TEST_FOLDER]:
    verification[folder] = dict()
    for cat in category_names:
        verification[folder][cat] =
len(glob.glob(os.path.join(RAW_DATA_PATH,folder,cat)+"/*"))

for folder in verification.keys():
    print(folder)
    for key,value in verification[folder].items():
        print(f"{key}: {value}")
    print()

train
onion: 849
tomato: 789
potato: 898
indian market: 599

test
onion: 83
```

```
tomato: 106
potato: 81
indian market: 81
```

The number of images for each category in the train dataset given is correct The number of images given in the test dataset is incorrect for onion and potato. They should be interchanged.

Modelling

Utility functions

```
# a function to plot the loss curve after training
# it also prints the loss and accuracy after training for train and
val data to make it easier to infer
def plot_loss(history):
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10,2),
dpi=150)
    ax = axes.ravel()

    index = np.argmax(history.history['val_accuracy'])

    print(f"Max train accuracy = {history.history['accuracy']
[index]}")
    print(f"Max val accuracy = {history.history['val_accuracy']
[index]}")
    print(f"Min train loss = {history.history['loss'][index]}")
    print(f"Min val loss = {history.history['val_loss'][index]}")

    # accuracy graph
    ax[0].plot(range(len(history.history['accuracy'])), [acc * 100 for
acc in history.history['accuracy']], label='Train', color='b')
    ax[0].plot(range(len(history.history['val_accuracy'])), [acc * 100
for acc in history.history['val_accuracy']], label='Val', color='r')
    ax[0].set_title('Accuracy vs. epoch', fontsize=10)
    ax[0].set_ylabel('Accuracy', fontsize=5)
    ax[0].set_xlabel('epoch', fontsize=5)
    ax[0].legend()

    #loss graph
    ax[1].plot(range(len(history.history['loss'])),
history.history['loss'], label='Train', color='b')
    ax[1].plot(range(len(history.history['val_loss'])),
history.history['val_loss'], label='Val', color='r')
    ax[1].set_title('Loss vs. epoch', fontsize=7)
    ax[1].set_ylabel('Loss', fontsize=5)
    ax[1].set_xlabel('epoch', fontsize=5)
    ax[1].legend()
```

```
    #display the graph
    plt.show()

# this function plots the confusion matrix, precision matrix and
recall matrix
def ConfusionMatrix(y_true, y_pred, label_list):

    fig, axes = plt.subplots(1,3,figsize=(20,5),dpi=150)
    ax = axes.ravel()
    cm = metrics.confusion_matrix(y_true,y_pred)

    # plotting confusion matrix
    sns.heatmap(cm, annot=True, xticklabels=label_list,
yticklabels=label_list, cmap="YlGnBu", fmt='g',ax=ax[0])
    ax[0].set_title('Confusion matrix')

    # plotting precision matric
    pr = cm / np.sum(cm,axis=0)
    sns.heatmap(pr, annot=True, xticklabels=label_list,
yticklabels=label_list, cmap="YlGnBu", fmt='.2f',ax=ax[1])
    ax[1].set_title('Precision matrix')

    # plotting recall matric
    rc = cm / np.sum(cm,axis=1).reshape(-1,1)
    sns.heatmap(rc, annot=True, xticklabels=label_list,
yticklabels=label_list, cmap="YlGnBu", fmt='.2f',ax=ax[2])
    ax[2].set_title('Recall matrix')

    plt.tight_layout()
    plt.show()

# image size for the input to the model
WIDTH = 224
HEIGHT = 224

# defining a function to create a generator object for reading dataset
def readData(name, batch_size=32):
    if name == 'train':
        return
tf.keras.utils.image_dataset_from_directory(DATA_PATH+'/'+TRAIN_FOLDER
, shuffle=True, image_size=(400,400), batch_size=batch_size)
    elif name == 'val':
        return
tf.keras.utils.image_dataset_from_directory(DATA_PATH+'/'+VAL_FOLDER,
shuffle=False, image_size=(400,400), batch_size=batch_size)
    elif name == 'test':
        return
tf.keras.utils.image_dataset_from_directory(DATA_PATH+'/'+TEST_FOLDER,
shuffle=False, image_size=(400,400), batch_size=batch_size)
```

```
    else:
        print("Please input either one of these: 'train', 'val',
'test'")

# Define the objective function for Optuna
def createObjective(name,getModel):
    def objective(trial):
        # Define the hyperparameters to optimize
        batch_size = trial.suggest_categorical('batch_size', [16, 32])
        dropout_rate = trial.suggest_categorical("dropout_rate",
[0.0,0.2,0.4])
        alpha = trial.suggest_float("alpha", 1e-8, 0.1, log=True)
        reg = 'l2'
        is_augmentation = True

        # pick the regularizer
        def pickRegularizer(reg_type,alpha):
            if reg_type == 'l1':
                return tf.keras.regularizers.L1(l1=alpha)
            else:
                return tf.keras.regularizers.L2(l2=alpha)
        regularizer = pickRegularizer(reg,alpha)

        # getting the dataset with selected batch_size and
preprocessing it
        train_ds = readData('train',batch_size=batch_size)
        val_ds = readData('val',batch_size=batch_size)
        test_ds = readData('test',batch_size=batch_size)

        # calling the model
        model = getModel(name,dropout_rate,regularizer)

        # compile the model
        model.compile(
            optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
            metrics = ['accuracy']
        )

        # defining callbacks
        callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
        ]

        # Train the model
```

```
        history = model.fit(train_ds, validation_data=val_ds,
epochs=25, verbose=0, callbacks=callbacks)

        return min(history.history['val_loss'])
    return objective
```

CNN from scratch

We will create a very simple neural network for our use case. Since the number of datapoints is very small, it is very easy for the model to overfit. So we will deploy regularization and dropout in the beginning itseld. On top of that, we will also apply data augmentation in order to increase the number of datapoints, and also to remove multiple variances like positional variance, zoom variance, etc. from the model.

```
def getModel(name,dropout_rate,reg):

    model = tf.keras.Sequential(
        name = name,
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/255),

tf.keras.layers.Conv2D(filters=16,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.GlobalAveragePooling2D(),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(units=128,activation='relu',kernel_regularizer=r
eg,bias_regularizer=reg),
```

```
            tf.keras.layers.Dropout(rate=dropout_rate),
            tf.keras.layers.Dense(units=4,activation='softmax')
        ]
    )
    return model

# Define the hyperparameters to optimize
batch_size = 32
dropout_rate = 0.2
alpha = 0.0001
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('custom_cnn',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# setting up tensorboard
log_dir = "custom_cnn_logs"
!rm -rf log_dir

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001),
    tf.keras.callbacks.TensorBoard(log_dir=log_dir)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30,
verbose=1, callbacks=callbacks)
```

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Epoch 1/30
```

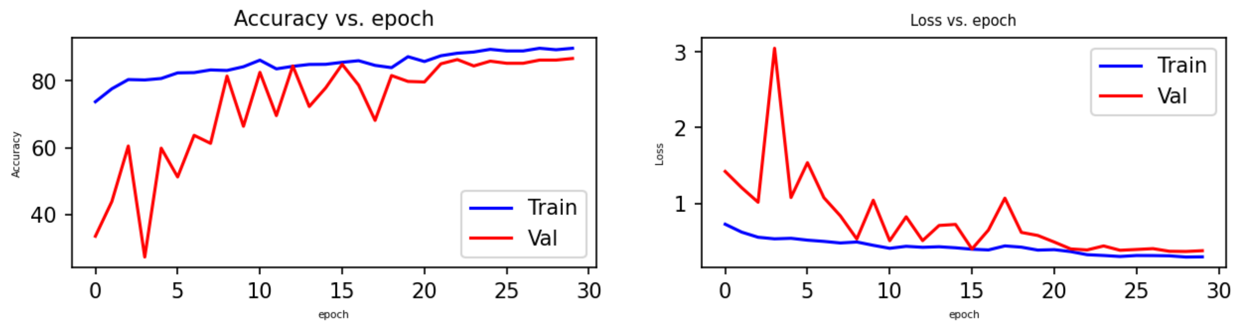
```
79/79 [==============================] - 34s 204ms/step - loss: 0.7248
- accuracy: 0.7371 - val_loss: 1.4209 - val_accuracy: 0.3360 - lr:
0.0010
Epoch 2/30
79/79 [==============================] - 21s 257ms/step - loss: 0.6206
- accuracy: 0.7758 - val_loss: 1.2072 - val_accuracy: 0.4395 - lr:
0.0010
Epoch 3/30
79/79 [==============================] - 28s 338ms/step - loss: 0.5529
- accuracy: 0.8034 - val_loss: 1.0145 - val_accuracy: 0.6051 - lr:
0.0010
Epoch 4/30
79/79 [==============================] - 19s 225ms/step - loss: 0.5319
- accuracy: 0.8022 - val_loss: 3.0461 - val_accuracy: 0.2739 - lr:
0.0010
Epoch 5/30
79/79 [==============================] - 24s 298ms/step - loss: 0.5390
- accuracy: 0.8065 - val_loss: 1.0768 - val_accuracy: 0.5987 - lr:
0.0010
Epoch 6/30
79/79 [==============================] - 17s 200ms/step - loss: 0.5147
- accuracy: 0.8233 - val_loss: 1.5369 - val_accuracy: 0.5127 - lr:
0.0010
Epoch 7/30
79/79 [==============================] - 17s 202ms/step - loss: 0.4983
- accuracy: 0.8241 - val_loss: 1.0744 - val_accuracy: 0.6369 - lr:
0.0010
Epoch 8/30
79/79 [==============================] - 16s 190ms/step - loss: 0.4773
- accuracy: 0.8321 - val_loss: 0.8353 - val_accuracy: 0.6131 - lr:
0.0010
Epoch 9/30
79/79 [==============================] - 18s 221ms/step - loss: 0.4902
- accuracy: 0.8305 - val_loss: 0.5299 - val_accuracy: 0.8137 - lr:
0.0010
Epoch 10/30
79/79 [==============================] - 15s 186ms/step - loss: 0.4463
- accuracy: 0.8416 - val_loss: 1.0411 - val_accuracy: 0.6640 - lr:
0.0010
Epoch 11/30
79/79 [==============================] - 16s 189ms/step - loss: 0.4070
- accuracy: 0.8612 - val_loss: 0.5068 - val_accuracy: 0.8248 - lr:
0.0010
Epoch 12/30
79/79 [==============================] - 18s 223ms/step - loss: 0.4334
- accuracy: 0.8353 - val_loss: 0.8229 - val_accuracy: 0.6959 - lr:
0.0010
Epoch 13/30
79/79 [==============================] - 20s 231ms/step - loss: 0.4202
```

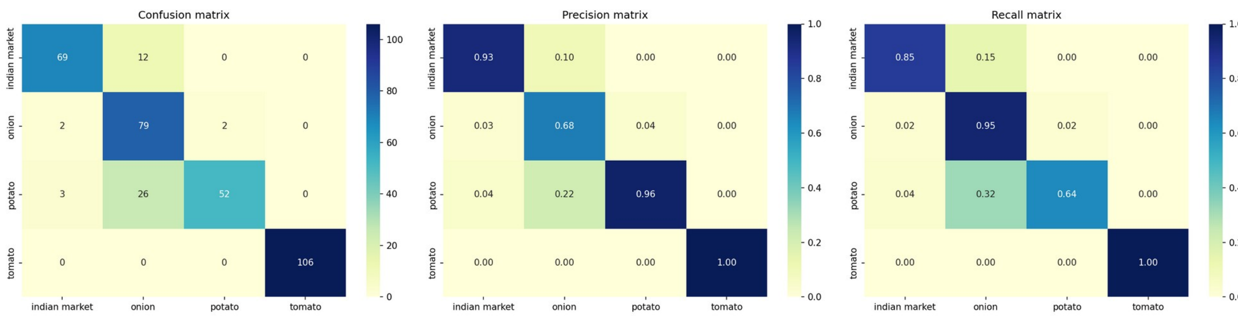
```
- accuracy: 0.8428 - val_loss: 0.5085 - val_accuracy: 0.8439 - lr:
0.0010
Epoch 14/30
79/79 [==============================] - 19s 225ms/step - loss: 0.4276
- accuracy: 0.8484 - val_loss: 0.7083 - val_accuracy: 0.7229 - lr:
0.0010
Epoch 15/30
79/79 [==============================] - 16s 190ms/step - loss: 0.4138
- accuracy: 0.8488 - val_loss: 0.7220 - val_accuracy: 0.7787 - lr:
0.0010
Epoch 16/30
79/79 [==============================] - 18s 222ms/step - loss: 0.3949
- accuracy: 0.8548 - val_loss: 0.3979 - val_accuracy: 0.8487 - lr:
0.0010
Epoch 17/30
79/79 [==============================] - 18s 215ms/step - loss: 0.3854
- accuracy: 0.8596 - val_loss: 0.6480 - val_accuracy: 0.7866 - lr:
0.0010
Epoch 18/30
79/79 [==============================] - 18s 224ms/step - loss: 0.4386
- accuracy: 0.8456 - val_loss: 1.0685 - val_accuracy: 0.6815 - lr:
0.0010
Epoch 19/30
79/79 [==============================] - 17s 196ms/step - loss: 0.4221
- accuracy: 0.8389 - val_loss: 0.6168 - val_accuracy: 0.8153 - lr:
0.0010
Epoch 20/30
79/79 [==============================] - 16s 192ms/step - loss: 0.3834
- accuracy: 0.8716 - val_loss: 0.5763 - val_accuracy: 0.7978 - lr:
0.0010
Epoch 21/30
79/79 [==============================] - 15s 180ms/step - loss: 0.3892
- accuracy: 0.8572 - val_loss: 0.4898 - val_accuracy: 0.7962 - lr:
0.0010
Epoch 22/30
79/79 [==============================] - 15s 177ms/step - loss: 0.3635
- accuracy: 0.8744 - val_loss: 0.3986 - val_accuracy: 0.8503 - lr:
1.0000e-04
Epoch 23/30
79/79 [==============================] - 18s 205ms/step - loss: 0.3226
- accuracy: 0.8819 - val_loss: 0.3854 - val_accuracy: 0.8631 - lr:
1.0000e-04
Epoch 24/30
79/79 [==============================] - 19s 225ms/step - loss: 0.3122
- accuracy: 0.8855 - val_loss: 0.4381 - val_accuracy: 0.8439 - lr:
1.0000e-04
Epoch 25/30
79/79 [==============================] - 15s 178ms/step - loss: 0.2990
- accuracy: 0.8935 - val_loss: 0.3811 - val_accuracy: 0.8583 - lr:
```

```
1.0000e-04
Epoch 26/30
79/79 [==============================] - 16s 198ms/step - loss: 0.3120
- accuracy: 0.8887 - val_loss: 0.3915 - val_accuracy: 0.8519 - lr:
1.0000e-04
Epoch 27/30
79/79 [==============================] - 19s 226ms/step - loss: 0.3112
- accuracy: 0.8887 - val_loss: 0.4026 - val_accuracy: 0.8519 - lr:
1.0000e-04
Epoch 28/30
79/79 [==============================] - 17s 212ms/step - loss: 0.3076
- accuracy: 0.8967 - val_loss: 0.3677 - val_accuracy: 0.8615 - lr:
1.0000e-04
Epoch 29/30
79/79 [==============================] - 15s 175ms/step - loss: 0.2917
- accuracy: 0.8923 - val_loss: 0.3652 - val_accuracy: 0.8615 - lr:
1.0000e-04
Epoch 30/30
79/79 [==============================] - 16s 189ms/step - loss: 0.2943
- accuracy: 0.8967 - val_loss: 0.3753 - val_accuracy: 0.8662 - lr:
1.0000e-04
```

```
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)

# Max train accuracy = 0.9226166605949402
# Max val accuracy = 0.9283439517021179
# Min train loss = 0.2065521776676178
# Min val loss = 0.2142053097486496
```

```
11/11 [==============================] - 2s 161ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.8966892957687378
Max val accuracy = 0.8662420511245728
Min train loss = 0.2943452298641205
Min val loss = 0.3753196597099304
```






```
%load_ext tensorboard
%tensorboard --logdir={log_dir}

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

Observations

- We can see that the methods to reduce overfitting are doing a great job in the first try itself
- The model's performance is very good given that we are training a network from scratch with such a small dataset.
- The model is not overfitting at all.
- Though it might be possible that our model is underfitting, since the accuracy is below 90% and we have used a sime architecture.

Increasing model complexity to improve the performance

Let's try to increase the model complexity in order to check whether we can extract more performance from the model or not. We will add a few more convolution layers, increase the feature map size.

```
def getModel(name,dropout_rate,reg):

    model = tf.keras.Sequential(
        name = name,
```

```
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/255),

tf.keras.layers.Conv2D(filters=16,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activat
ion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),

tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),

tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),

tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activa
tion='relu',kernel_regularizer=reg,bias_regularizer=reg),
            tf.keras.layers.GlobalAveragePooling2D(),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(units=256,activation='relu',kernel_regularizer=r
eg,bias_regularizer=reg),
            tf.keras.layers.Dropout(rate=dropout_rate),
            tf.keras.layers.Dense(units=4,activation='softmax')
        ]
    )
    return model
```

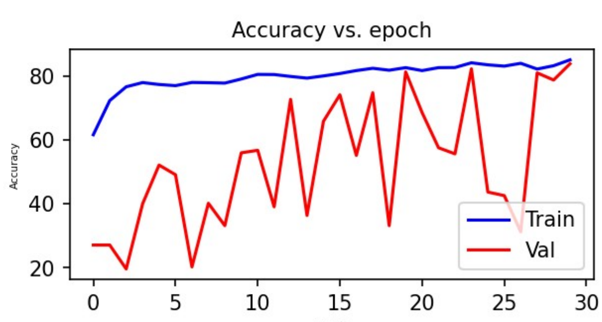
```python
# Define the hyperparameters to optimize
batch_size = 32
dropout_rate = 0.2
alpha = 0.0001
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('custom_cnn',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# setting up tensorboard
log_dir = "custom_cnn_logs"
!rm -rf log_dir

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001),
    tf.keras.callbacks.TensorBoard(log_dir=log_dir)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30,
verbose=1, callbacks=callbacks)
```

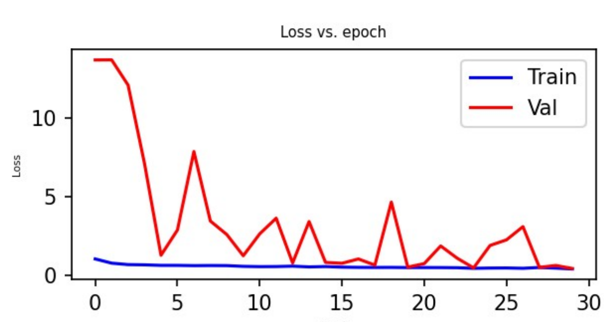
```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Epoch 1/30
79/79 [==============================] - 23s 210ms/step - loss: 1.0372
- accuracy: 0.6155 - val_loss: 13.6579 - val_accuracy: 0.2707 - lr:
0.0010
Epoch 2/30
79/79 [==============================] - 19s 218ms/step - loss: 0.7689
- accuracy: 0.7224 - val_loss: 13.6631 - val_accuracy: 0.2707 - lr:
```

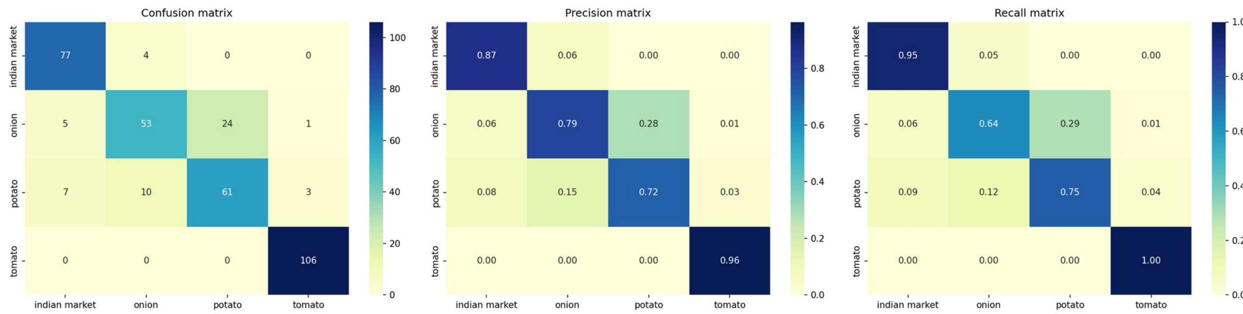
```
0.0010
Epoch 3/30
79/79 [==============================] - 16s 196ms/step - loss: 0.6772
- accuracy: 0.7655 - val_loss: 12.0614 - val_accuracy: 0.1959 - lr:
0.0010
Epoch 4/30
79/79 [==============================] - 17s 201ms/step - loss: 0.6621
- accuracy: 0.7786 - val_loss: 7.0726 - val_accuracy: 0.3997 - lr:
0.0010
Epoch 5/30
79/79 [==============================] - 21s 258ms/step - loss: 0.6284
- accuracy: 0.7726 - val_loss: 1.2696 - val_accuracy: 0.5207 - lr:
0.0010
Epoch 6/30
79/79 [==============================] - 19s 224ms/step - loss: 0.6276
- accuracy: 0.7690 - val_loss: 2.8720 - val_accuracy: 0.4904 - lr:
0.0010
Epoch 7/30
79/79 [==============================] - 18s 219ms/step - loss: 0.6111
- accuracy: 0.7790 - val_loss: 7.8484 - val_accuracy: 0.2022 - lr:
0.0010
Epoch 8/30
79/79 [==============================] - 18s 216ms/step - loss: 0.6182
- accuracy: 0.7782 - val_loss: 3.4401 - val_accuracy: 0.4013 - lr:
0.0010
Epoch 9/30
79/79 [==============================] - 15s 183ms/step - loss: 0.6124
- accuracy: 0.7770 - val_loss: 2.5833 - val_accuracy: 0.3312 - lr:
0.0010
Epoch 10/30
79/79 [==============================] - 15s 181ms/step - loss: 0.5658
- accuracy: 0.7898 - val_loss: 1.2383 - val_accuracy: 0.5589 - lr:
0.0010
Epoch 11/30
79/79 [==============================] - 16s 197ms/step - loss: 0.5478
- accuracy: 0.8041 - val_loss: 2.6269 - val_accuracy: 0.5669 - lr:
0.0010
Epoch 12/30
79/79 [==============================] - 18s 224ms/step - loss: 0.5536
- accuracy: 0.8037 - val_loss: 3.6216 - val_accuracy: 0.3901 - lr:
0.0010
Epoch 13/30
79/79 [==============================] - 15s 180ms/step - loss: 0.5783
- accuracy: 0.7978 - val_loss: 0.7798 - val_accuracy: 0.7261 - lr:
0.0010
Epoch 14/30
79/79 [==============================] - 16s 195ms/step - loss: 0.5308
- accuracy: 0.7926 - val_loss: 3.4061 - val_accuracy: 0.3631 - lr:
0.0010
```

```
Epoch 15/30
79/79 [==============================] - 16s 194ms/step - loss: 0.5531
- accuracy: 0.7994 - val_loss: 0.8165 - val_accuracy: 0.6576 - lr:
0.0010
Epoch 16/30
79/79 [==============================] - 16s 192ms/step - loss: 0.5166
- accuracy: 0.8069 - val_loss: 0.7601 - val_accuracy: 0.7404 - lr:
0.0010
Epoch 17/30
79/79 [==============================] - 17s 203ms/step - loss: 0.5012
- accuracy: 0.8161 - val_loss: 1.0337 - val_accuracy: 0.5510 - lr:
0.0010
Epoch 18/30
79/79 [==============================] - 17s 196ms/step - loss: 0.4957
- accuracy: 0.8233 - val_loss: 0.6443 - val_accuracy: 0.7468 - lr:
0.0010
Epoch 19/30
79/79 [==============================] - 17s 199ms/step - loss: 0.4992
- accuracy: 0.8173 - val_loss: 4.6414 - val_accuracy: 0.3312 - lr:
0.0010
Epoch 20/30
79/79 [==============================] - 16s 197ms/step - loss: 0.4883
- accuracy: 0.8253 - val_loss: 0.5160 - val_accuracy: 0.8121 - lr:
0.0010
Epoch 21/30
79/79 [==============================] - 19s 230ms/step - loss: 0.4901
- accuracy: 0.8161 - val_loss: 0.7396 - val_accuracy: 0.6847 - lr:
0.0010
Epoch 22/30
79/79 [==============================] - 17s 196ms/step - loss: 0.4880
- accuracy: 0.8253 - val_loss: 1.8601 - val_accuracy: 0.5748 - lr:
0.0010
Epoch 23/30
79/79 [==============================] - 18s 207ms/step - loss: 0.4786
- accuracy: 0.8257 - val_loss: 1.0845 - val_accuracy: 0.5557 - lr:
0.0010
Epoch 24/30
79/79 [==============================] - 16s 197ms/step - loss: 0.4412
- accuracy: 0.8404 - val_loss: 0.4681 - val_accuracy: 0.8217 - lr:
0.0010
Epoch 25/30
79/79 [==============================] - 16s 196ms/step - loss: 0.4581
- accuracy: 0.8341 - val_loss: 1.8906 - val_accuracy: 0.4363 - lr:
0.0010
Epoch 26/30
79/79 [==============================] - 16s 196ms/step - loss: 0.4637
- accuracy: 0.8301 - val_loss: 2.2467 - val_accuracy: 0.4252 - lr:
0.0010
Epoch 27/30
```

```
79/79 [==============================] - 16s 195ms/step - loss: 0.4424
- accuracy: 0.8389 - val_loss: 3.0850 - val_accuracy: 0.3121 - lr:
0.0010
Epoch 28/30
79/79 [==============================] - 16s 193ms/step - loss: 0.4943
- accuracy: 0.8209 - val_loss: 0.5029 - val_accuracy: 0.8089 - lr:
0.0010
Epoch 29/30
79/79 [==============================] - 17s 198ms/step - loss: 0.4535
- accuracy: 0.8313 - val_loss: 0.6208 - val_accuracy: 0.7866 - lr:
0.0010
Epoch 30/30
79/79 [==============================] - 16s 190ms/step - loss: 0.4007
- accuracy: 0.8496 - val_loss: 0.4312 - val_accuracy: 0.8376 - lr:
1.0000e-04

y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)

# Max train accuracy = 0.9226166605949402
# Max val accuracy = 0.9283439517021179
# Min train loss = 0.2065521776676178
# Min val loss = 0.2142053097486496

11/11 [==============================] - 2s 152ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.849621057510376
Max val accuracy = 0.837579607963562
Min train loss = 0.4006928503513336
Min val loss = 0.4312150180339813
```








```
%load_ext tensorboard
%tensorboard --logdir={log_dir}

The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Reusing TensorBoard on port 6006 (pid 6213), started 0:13:08 ago. (Use
'!kill 6213' to kill it.)

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

Observations

- There is a huge fluctuation in the validation accuracy and loss
- This might be happening becuase of the distribution of training and val data might be different
- The overall performance of the model is worse than that of the simpler model.

Pretrained VGG16

Now let's try some of the pre-trained models, which are trained on image net dataset. We will remove the top layers (classification laters) and add our own layers to train the classification.

Defining the model

```
# downloading the model
pretrained_vgg16 =
tf.keras.applications.VGG16(include_top=False,weights='imagenet',input
_shape=(HEIGHT,WIDTH,3),pooling='avg')

# setting trainable = False
pretrained_vgg16.trainable = False

def getModel(name,dropout_rate,reg):
    seq = tf.keras.Sequential(
        name = name,
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/255),
```

```
            pretrained_vgg16,
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(256,activation='relu',kernel_regularizer=reg,bia
s_regularizer=reg),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(4,activation='softmax',kernel_regularizer=reg,bi
as_regularizer=reg)
        ]
    )

    return seq
```

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

Searching for the best hyperparameter

```
# creating the objective function given the model
obj = createObjective('pretrained_VGG16',getModel)

# Create a study and optimize the objective function
study = optuna.create_study(direction='minimize')
study.optimize(obj, n_trials=5, show_progress_bar=True)

# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)
```

```
[I 2023-11-25 17:32:05,190] A new study created in memory with name:
no-name-9a192d70-2df1-4f92-8216-027fb8de81d2
```

{"model_id":"89613f70d9324fa3825dfb2a392aa230","version_major":2,"vers
ion_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 17:39:52,898] Trial 0 finished with value:
0.1957613080739975 and parameters: {'batch_size': 32, 'dropout_rate':
0.2, 'alpha': 2.1283116433800812e-07}. Best is trial 0 with value:
0.1957613080739975.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 17:46:49,378] Trial 1 finished with value:
0.41897183656692505 and parameters: {'batch_size': 16, 'dropout_rate':
0.0, 'alpha': 0.029922618237130903}. Best is trial 0 with value:
```

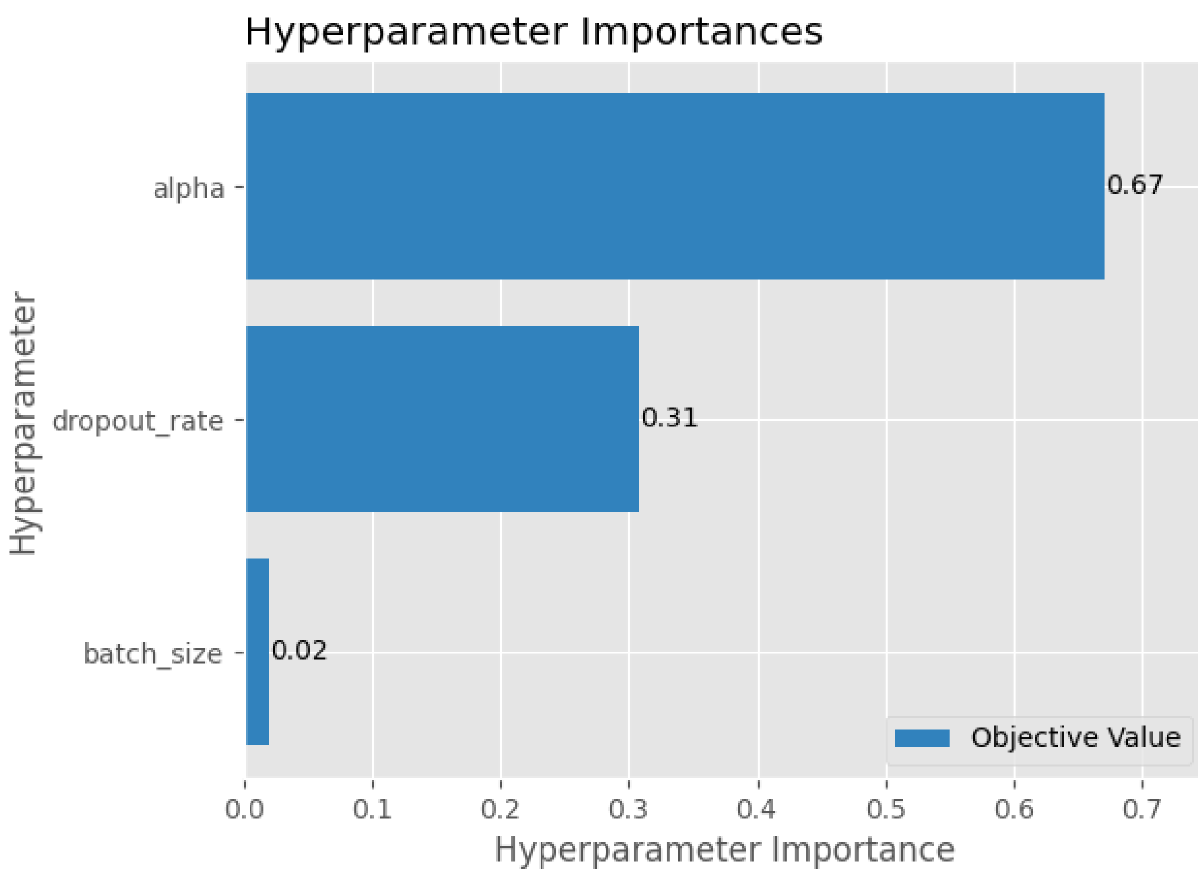
```
0.1957613080739975.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 17:53:56,292] Trial 2 finished with value:
0.2338872104883194 and parameters: {'batch_size': 32, 'dropout_rate':
0.4, 'alpha': 6.659239601542189e-05}. Best is trial 0 with value:
0.1957613080739975.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:00:44,946] Trial 3 finished with value:
0.3165602684020996 and parameters: {'batch_size': 16, 'dropout_rate':
0.0, 'alpha': 0.0005528284036060066}. Best is trial 0 with value:
0.1957613080739975.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
```

```
study.optimize(obj, n_trials=5, show_progress_bar=True)
```

{"model_id":"9a409c6e4f2b4019ad93fae1a1ce041c","version_major":2,"version_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:15:04,858] Trial 5 finished with value:
0.41390398144721985 and parameters: {'batch_size': 32, 'dropout_rate':
0.4, 'alpha': 0.01075654492337728}. Best is trial 4 with value:
0.19540159404277802.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:22:14,378] Trial 6 finished with value:
0.3762601315975189 and parameters: {'batch_size': 32, 'dropout_rate':
0.4, 'alpha': 0.0026051024053338366}. Best is trial 4 with value:
0.19540159404277802.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:28:56,182] Trial 7 finished with value:
0.5302455425262451 and parameters: {'batch_size': 16, 'dropout_rate':
0.2, 'alpha': 0.061164048081515786}. Best is trial 4 with value:
0.19540159404277802.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:35:34,909] Trial 8 finished with value:
0.23763445019721985 and parameters: {'batch_size': 16, 'dropout_rate':
```

```
0.0, 'alpha': 4.979196784345964 6e-08}. Best is trial 4 with value:
0.19540159404277802.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:42:42,808] Trial 9 finished with value:
0.5327568650245667 and parameters: {'batch_size': 32, 'dropout_rate':
0.2, 'alpha': 0.04784153417749709}. Best is trial 4 with value:
0.19540159404277802.

# plotting hyperparameter importances
plot_param_importances(study)

<ipython-input-14-4f25878d1741>:2: ExperimentalWarning:
plot_param_importances is experimental (supported from v2.2.0). The
interface can change in the future.
  plot_param_importances(study)

<Axes: title={'left': 'Hyperparameter Importances'},
xlabel='Hyperparameter Importance', ylabel='Hyperparameter'>
```



Observations

- As we can see, the regularization rate is the most important hyperparameter

- batch size does not have much effect on the performance, so from now on we will use a batch size of 32.
- dropout rate also does not have a lot of effect on the model performance compared to regularization. So we will use a constant dropout rate of 0.0

```
# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)

Best hyperparameters: {'batch_size': 32, 'dropout_rate': 0.4, 'alpha': 2.30419806532844e-07}
```

Training the model with best hyperparameters

```
# Define the hyperparameters to optimize
batch_size = 32
dropout_rate = 0.4
alpha = 2.30e-07
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('pretrainedVGG16',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,patience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,patience=5,min_delta=0.0001,min_lr=0.000001)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30, verbose=1, callbacks=callbacks)

Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
```

```
Found 351 files belonging to 4 classes.
Epoch 1/30
79/79 [==============================] - 24s 234ms/step - loss: 0.7095
- accuracy: 0.7327 - val_loss: 0.8078 - val_accuracy: 0.7882 - lr:
0.0010
Epoch 2/30
79/79 [==============================] - 20s 244ms/step - loss: 0.4476
- accuracy: 0.8333 - val_loss: 0.5693 - val_accuracy: 0.8965 - lr:
0.0010
Epoch 3/30
79/79 [==============================] - 21s 227ms/step - loss: 0.3989
- accuracy: 0.8588 - val_loss: 0.3987 - val_accuracy: 0.8965 - lr:
0.0010
Epoch 4/30
79/79 [==============================] - 18s 221ms/step - loss: 0.3815
- accuracy: 0.8568 - val_loss: 0.3133 - val_accuracy: 0.8965 - lr:
0.0010
Epoch 5/30
79/79 [==============================] - 18s 214ms/step - loss: 0.3559
- accuracy: 0.8684 - val_loss: 0.2632 - val_accuracy: 0.9013 - lr:
0.0010
Epoch 6/30
79/79 [==============================] - 19s 224ms/step - loss: 0.3471
- accuracy: 0.8640 - val_loss: 0.2345 - val_accuracy: 0.9061 - lr:
0.0010
Epoch 7/30
79/79 [==============================] - 17s 197ms/step - loss: 0.3231
- accuracy: 0.8763 - val_loss: 0.2287 - val_accuracy: 0.9236 - lr:
0.0010
Epoch 8/30
79/79 [==============================] - 17s 203ms/step - loss: 0.2983
- accuracy: 0.8847 - val_loss: 0.2272 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 9/30
79/79 [==============================] - 17s 208ms/step - loss: 0.2906
- accuracy: 0.8863 - val_loss: 0.2323 - val_accuracy: 0.9108 - lr:
0.0010
Epoch 10/30
79/79 [==============================] - 19s 229ms/step - loss: 0.3008
- accuracy: 0.8799 - val_loss: 0.2255 - val_accuracy: 0.9252 - lr:
0.0010
Epoch 11/30
79/79 [==============================] - 17s 204ms/step - loss: 0.2782
- accuracy: 0.8947 - val_loss: 0.2323 - val_accuracy: 0.9124 - lr:
0.0010
Epoch 12/30
79/79 [==============================] - 19s 233ms/step - loss: 0.3004
- accuracy: 0.8915 - val_loss: 0.2249 - val_accuracy: 0.9236 - lr:
0.0010
```

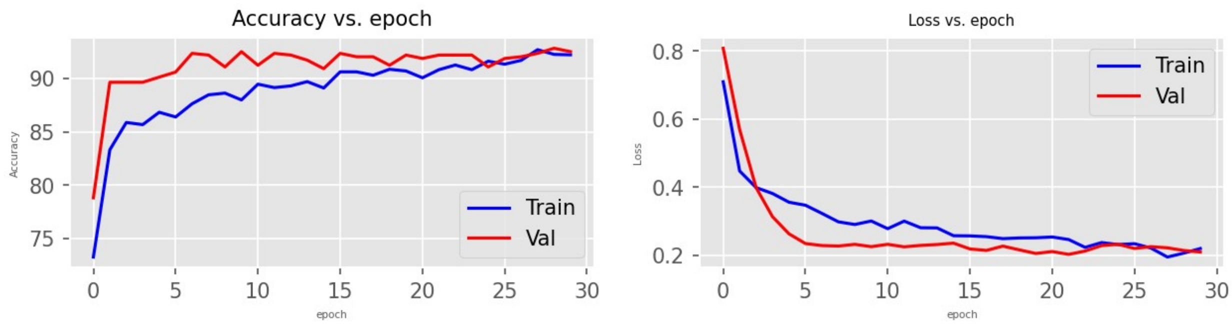
```
Epoch 13/30
79/79 [==============================] - 19s 227ms/step - loss: 0.2810
- accuracy: 0.8931 - val_loss: 0.2291 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 14/30
79/79 [==============================] - 18s 212ms/step - loss: 0.2805
- accuracy: 0.8971 - val_loss: 0.2317 - val_accuracy: 0.9172 - lr:
0.0010
Epoch 15/30
79/79 [==============================] - 17s 201ms/step - loss: 0.2576
- accuracy: 0.8911 - val_loss: 0.2358 - val_accuracy: 0.9092 - lr:
0.0010
Epoch 16/30
79/79 [==============================] - 17s 208ms/step - loss: 0.2571
- accuracy: 0.9063 - val_loss: 0.2187 - val_accuracy: 0.9236 - lr:
0.0010
Epoch 17/30
79/79 [==============================] - 19s 217ms/step - loss: 0.2549
- accuracy: 0.9063 - val_loss: 0.2144 - val_accuracy: 0.9204 - lr:
0.0010
Epoch 18/30
79/79 [==============================] - 17s 209ms/step - loss: 0.2490
- accuracy: 0.9031 - val_loss: 0.2270 - val_accuracy: 0.9204 - lr:
0.0010
Epoch 19/30
79/79 [==============================] - 17s 204ms/step - loss: 0.2511
- accuracy: 0.9087 - val_loss: 0.2161 - val_accuracy: 0.9124 - lr:
0.0010
Epoch 20/30
79/79 [==============================] - 18s 212ms/step - loss: 0.2516
- accuracy: 0.9071 - val_loss: 0.2055 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 21/30
79/79 [==============================] - 17s 202ms/step - loss: 0.2538
- accuracy: 0.9007 - val_loss: 0.2111 - val_accuracy: 0.9188 - lr:
0.0010
Epoch 22/30
79/79 [==============================] - 17s 206ms/step - loss: 0.2463
- accuracy: 0.9083 - val_loss: 0.2028 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 23/30
79/79 [==============================] - 18s 210ms/step - loss: 0.2233
- accuracy: 0.9126 - val_loss: 0.2126 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 24/30
79/79 [==============================] - 20s 237ms/step - loss: 0.2375
- accuracy: 0.9083 - val_loss: 0.2286 - val_accuracy: 0.9220 - lr:
0.0010
Epoch 25/30
```

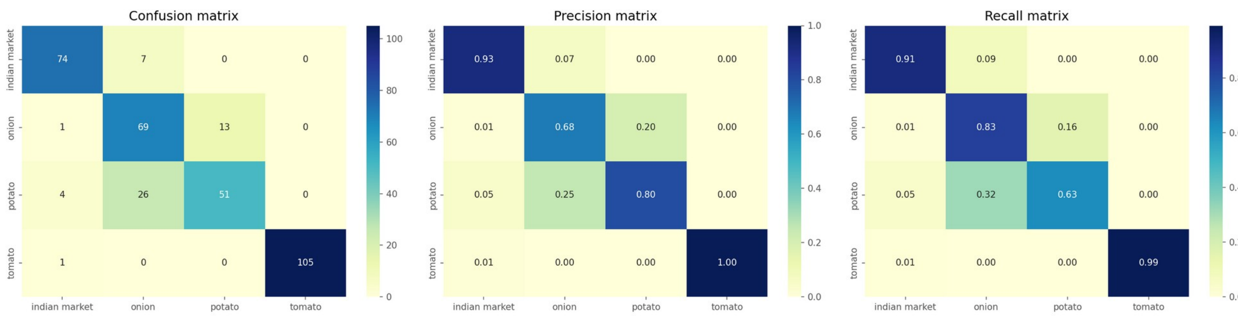
```
79/79 [==============================] - 17s 202ms/step - loss: 0.2317
- accuracy: 0.9162 - val_loss: 0.2325 - val_accuracy: 0.9108 - lr:
0.0010
Epoch 26/30
79/79 [==============================] - 17s 209ms/step - loss: 0.2341
- accuracy: 0.9134 - val_loss: 0.2202 - val_accuracy: 0.9188 - lr:
0.0010
Epoch 27/30
79/79 [==============================] - 17s 207ms/step - loss: 0.2213
- accuracy: 0.9170 - val_loss: 0.2255 - val_accuracy: 0.9204 - lr:
0.0010
Epoch 28/30
79/79 [==============================] - 18s 218ms/step - loss: 0.1950
- accuracy: 0.9270 - val_loss: 0.2220 - val_accuracy: 0.9236 - lr:
1.0000e-04
Epoch 29/30
79/79 [==============================] - 17s 200ms/step - loss: 0.2066
- accuracy: 0.9226 - val_loss: 0.2142 - val_accuracy: 0.9283 - lr:
1.0000e-04
Epoch 30/30
79/79 [==============================] - 17s 207ms/step - loss: 0.2199
- accuracy: 0.9222 - val_loss: 0.2097 - val_accuracy: 0.9252 - lr:
1.0000e-04
```

Checking model performance

```
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)
```

```
11/11 [==============================] - 6s 507ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.9226166605949402
Max val accuracy = 0.9283439517021179
Min train loss = 0.2065521776676178
Min val loss = 0.2142053097486496
```





Observations

- Pre-trained VGG16 is able to out perform our custom model

Redefining the objective function for hyperparmeter tuning

```
# Define the objective function for Optuna
def createObjective(name,getModel):
    def objective(trial):
        # Define the hyperparameters to optimize
        batch_size = 32
        dropout_rate = 0.4
        alpha = trial.suggest_float("alpha", 1e-8, 0.1, log=True)
        reg = 'l2'

        # pick the regularizer
        def pickRegularizer(reg_type,alpha):
            if reg_type == 'l1':
                return tf.keras.regularizers.L1(l1=alpha)
            else:
                return tf.keras.regularizers.L2(l2=alpha)
        regularizer = pickRegularizer(reg,alpha)

        # getting the dataset with selected batch_size and
preprocessing it
        train_ds = readData('train',batch_size=batch_size)
        val_ds = readData('val',batch_size=batch_size)
        test_ds = readData('test',batch_size=batch_size)
```

```
        # calling the model
        model = getModel(name,dropout_rate,regularizer)

        # compile the model
        model.compile(
            optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
            metrics = ['accuracy']
        )

        # defining callbacks
        callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
        ]

        # Train the model
        history = model.fit(train_ds, validation_data=val_ds,
epochs=20, verbose=0) #, callbacks=callbacks

        return min(history.history['val_loss'])
    return objective
```

Pretrained ResNet50

Defining the model

```
# downloading the model
pretrained_resnet =
tf.keras.applications.resnet50.ResNet50(include_top=False,weights='ima
genet',input_shape=(HEIGHT,WIDTH,3),pooling='avg')

# setting trainable = False
pretrained_resnet.trainable = False

def getModel(name,dropout_rate,reg):
    seq = tf.keras.Sequential(
        name = name,
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/255),
            pretrained_resnet,
            tf.keras.layers.BatchNormalization(),
```

```
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(256,activation='relu',kernel_regularizer=reg,bia
s_regularizer=reg),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(4,activation='softmax',kernel_regularizer=reg,bi
as_regularizer=reg)
        ]
    )

    return seq
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

Searching for the best hyperparameter

```
# creating the objective function given the model
obj = createObjective('pretrained_Resnet',getModel)

# Create a study and optimize the objective function
study = optuna.create_study(direction='minimize')
study.optimize(obj, n_trials=5, show_progress_bar=True)

# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)
```

```
[I 2023-11-25 18:53:13,775] A new study created in memory with name:
no-name-03046de0-9ed5-430e-a659-a37e990a8c1f
```

{"model_id":"214323debc2c44e3b7cc6ff86e3cc20f","version_major":2,"vers
ion_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 18:58:48,257] Trial 0 finished with value:
0.781572699546814 and parameters: {'alpha': 0.00042037948167177774}.
Best is trial 0 with value: 0.781572699546814.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:04:21,328] Trial 1 finished with value:
0.8749873638153076 and parameters: {'alpha': 0.00190228267952609 8}.
Best is trial 0 with value: 0.781572699546814.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
```

```
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:10:00,195] Trial 2 finished with value:
0.7300605773925781 and parameters: {'alpha': 3.836304789880116e-05}.
Best is trial 2 with value: 0.7300605773925781.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:15:33,744] Trial 3 finished with value:
0.9495513439178467 and parameters: {'alpha': 0.004389587979478359}.
Best is trial 2 with value: 0.7300605773925781.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
```

```python
study.optimize(obj, n_trials=5, show_progress_bar=True)

# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)
```

{"model_id":"6ca3c1d5519e443b9ffa471a020ce515","version_major":2,"version_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:26:34,743] Trial 5 finished with value:
0.7138864398002625 and parameters: {'alpha': 1.0795200776464035e-05}.
Best is trial 5 with value: 0.7138864398002625.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:31:50,706] Trial 6 finished with value:
0.7138316035270691 and parameters: {'alpha': 4.618209148211667e-06}.
Best is trial 6 with value: 0.7138316035270691.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:37:13,732] Trial 7 finished with value:
0.7025340795516968 and parameters: {'alpha': 8.657613951058189e-08}.
Best is trial 7 with value: 0.7025340795516968.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-25 19:42:45,355] Trial 8 finished with value:
0.7013677954673767 and parameters: {'alpha': 3.593016795857026e-08}.
Best is trial 8 with value: 0.7013677954673767.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
```

```
[I 2023-11-25 19:48:09,190] Trial 9 finished with value:
0.9779605865478516 and parameters: {'alpha': 0.010294322271876868}.
Best is trial 8 with value: 0.7013677954673767.
Best hyperparameters: {'alpha': 3.593016795857026e-08}
```

Training the model with best hyperparameters

```
# Define the hyperparameters to optimize
batch_size = 32
dropout_rate = 0.4
alpha = 3.59e-08
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('pretrained_Resnet',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30,
verbose=1, callbacks=callbacks)

Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Epoch 1/30
79/79 [==============================] - 62s 307ms/step - loss: 1.2100
- accuracy: 0.4882 - val_loss: 1.3559 - val_accuracy: 0.3169 - lr:
0.0010
Epoch 2/30
```

```
79/79 [==============================] - 26s 322ms/step - loss: 1.0387
- accuracy: 0.5568 - val_loss: 1.1844 - val_accuracy: 0.4475 - lr:
0.0010
Epoch 3/30
79/79 [==============================] - 19s 235ms/step - loss: 0.9599
- accuracy: 0.5931 - val_loss: 1.0972 - val_accuracy: 0.5143 - lr:
0.0010
Epoch 4/30
79/79 [==============================] - 19s 233ms/step - loss: 0.9413
- accuracy: 0.6003 - val_loss: 1.0642 - val_accuracy: 0.5844 - lr:
0.0010
Epoch 5/30
79/79 [==============================] - 17s 199ms/step - loss: 0.8954
- accuracy: 0.6203 - val_loss: 0.9903 - val_accuracy: 0.5844 - lr:
0.0010
Epoch 6/30
79/79 [==============================] - 17s 203ms/step - loss: 0.8854
- accuracy: 0.6314 - val_loss: 0.9144 - val_accuracy: 0.6354 - lr:
0.0010
Epoch 7/30
79/79 [==============================] - 25s 310ms/step - loss: 0.8548
- accuracy: 0.6434 - val_loss: 0.8807 - val_accuracy: 0.6401 - lr:
0.0010
Epoch 8/30
79/79 [==============================] - 17s 205ms/step - loss: 0.8430
- accuracy: 0.6558 - val_loss: 0.8155 - val_accuracy: 0.6688 - lr:
0.0010
Epoch 9/30
79/79 [==============================] - 16s 193ms/step - loss: 0.8485
- accuracy: 0.6470 - val_loss: 0.8181 - val_accuracy: 0.6417 - lr:
0.0010
Epoch 10/30
79/79 [==============================] - 19s 236ms/step - loss: 0.8726
- accuracy: 0.6374 - val_loss: 0.7753 - val_accuracy: 0.6815 - lr:
0.0010
Epoch 11/30
79/79 [==============================] - 17s 202ms/step - loss: 0.8196
- accuracy: 0.6534 - val_loss: 0.7611 - val_accuracy: 0.6863 - lr:
0.0010
Epoch 12/30
79/79 [==============================] - 17s 205ms/step - loss: 0.8164
- accuracy: 0.6693 - val_loss: 0.7627 - val_accuracy: 0.6927 - lr:
0.0010
Epoch 13/30
79/79 [==============================] - 16s 197ms/step - loss: 0.8048
- accuracy: 0.6586 - val_loss: 0.7205 - val_accuracy: 0.7054 - lr:
0.0010
Epoch 14/30
79/79 [==============================] - 19s 225ms/step - loss: 0.8023
```

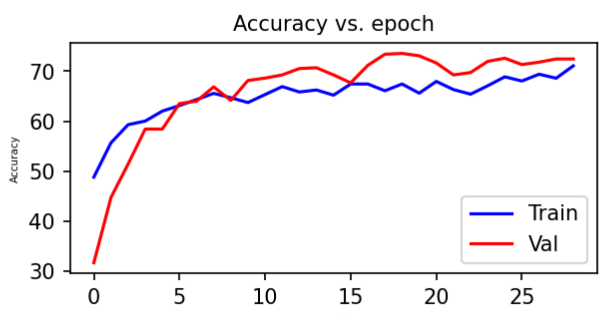
```
- accuracy: 0.6625 - val_loss: 0.7620 - val_accuracy: 0.7070 - lr:
0.0010
Epoch 15/30
79/79 [==============================] - 17s 201ms/step - loss: 0.8241
- accuracy: 0.6522 - val_loss: 0.7357 - val_accuracy: 0.6927 - lr:
0.0010
Epoch 16/30
79/79 [==============================] - 16s 191ms/step - loss: 0.8048
- accuracy: 0.6745 - val_loss: 0.7859 - val_accuracy: 0.6768 - lr:
0.0010
Epoch 17/30
79/79 [==============================] - 17s 206ms/step - loss: 0.7944
- accuracy: 0.6745 - val_loss: 0.7468 - val_accuracy: 0.7118 - lr:
0.0010
Epoch 18/30
79/79 [==============================] - 19s 236ms/step - loss: 0.8072
- accuracy: 0.6609 - val_loss: 0.6905 - val_accuracy: 0.7341 - lr:
0.0010
Epoch 19/30
79/79 [==============================] - 17s 198ms/step - loss: 0.7891
- accuracy: 0.6745 - val_loss: 0.6811 - val_accuracy: 0.7357 - lr:
0.0010
Epoch 20/30
79/79 [==============================] - 19s 234ms/step - loss: 0.8090
- accuracy: 0.6562 - val_loss: 0.6992 - val_accuracy: 0.7309 - lr:
0.0010
Epoch 21/30
79/79 [==============================] - 17s 209ms/step - loss: 0.7710
- accuracy: 0.6797 - val_loss: 0.7151 - val_accuracy: 0.7166 - lr:
0.0010
Epoch 22/30
79/79 [==============================] - 17s 199ms/step - loss: 0.7911
- accuracy: 0.6633 - val_loss: 0.7387 - val_accuracy: 0.6927 - lr:
0.0010
Epoch 23/30
79/79 [==============================] - 17s 203ms/step - loss: 0.8080
- accuracy: 0.6542 - val_loss: 0.7255 - val_accuracy: 0.6975 - lr:
0.0010
Epoch 24/30
79/79 [==============================] - 17s 204ms/step - loss: 0.7790
- accuracy: 0.6709 - val_loss: 0.7009 - val_accuracy: 0.7197 - lr:
0.0010
Epoch 25/30
79/79 [==============================] - 17s 208ms/step - loss: 0.7622
- accuracy: 0.6889 - val_loss: 0.6999 - val_accuracy: 0.7261 - lr:
1.0000e-04
Epoch 26/30
79/79 [==============================] - 18s 215ms/step - loss: 0.7622
- accuracy: 0.6805 - val_loss: 0.7038 - val_accuracy: 0.7134 - lr:
```

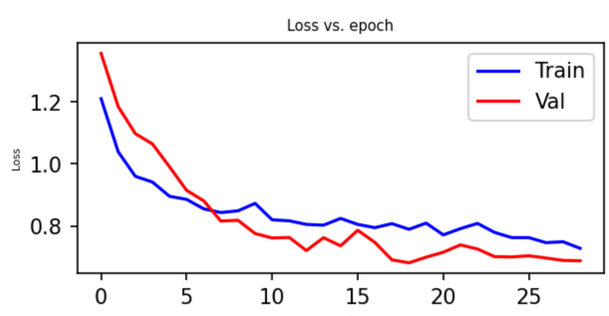
```
1.0000e-04
Epoch 27/30
79/79 [==============================] - 17s 194ms/step - loss: 0.7460
- accuracy: 0.6941 - val_loss: 0.6965 - val_accuracy: 0.7182 - lr:
1.0000e-04
Epoch 28/30
79/79 [==============================] - 17s 200ms/step - loss: 0.7488
- accuracy: 0.6861 - val_loss: 0.6887 - val_accuracy: 0.7245 - lr:
1.0000e-04
Epoch 29/30
79/79 [==============================] - 17s 207ms/step - loss: 0.7276
- accuracy: 0.7108 - val_loss: 0.6876 - val_accuracy: 0.7245 - lr:
1.0000e-04
```

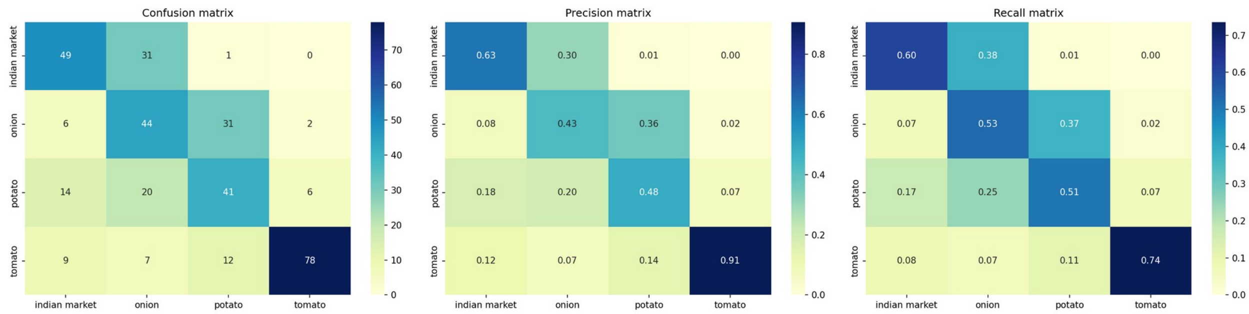
Checking model performance

```
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)
```

```
11/11 [==============================] - 4s 276ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.6745113730430603
Max val accuracy = 0.7356687784194946
Min train loss = 0.7890511751174927
Min val loss = 0.6810575127601624
```







Observations

- The pre-trained res net model is not able to beat even our custom model

Pretrained MobileNetV3Small

Defining the model

```
# downloading the model
pretrained_mobilenet =
tf.keras.applications.MobileNetV3Small(input_shape=(HEIGHT,WIDTH,3),in
clude_top=False,weights='imagenet',pooling='avg',)

# setting trainable = False
pretrained_mobilenet.trainable = False

def getModel(name,dropout_rate,reg):
    seq = tf.keras.Sequential(
        name = name,
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/127.5, offset=-1),
            pretrained_mobilenet,
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(256,activation='relu',kernel_regularizer=reg,bia
s_regularizer=reg),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(4,activation='softmax',kernel_regularizer=reg,bi
as_regularizer=reg)
        ]
    )

    return seq
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v3/
```

```
weights_mobilenet_v3_small_224_1.0_float_no_top_v2.h5
4334752/4334752 [==============================] - 0s 0us/step
```

Searching for the best hyperparameter

```python
# creating the objective function given the model
obj = createObjective('pretrained_mobilenet',getModel)

# Create a study and optimize the objective function
study = optuna.create_study(direction='minimize')
study.optimize(obj, n_trials=5, show_progress_bar=True)

# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)
```

```
[I 2023-11-26 18:09:58,706] A new study created in memory with name: no-name-8625fc0c-30ee-460a-9b43-336f1f7b16d3
```

{"model_id":"d933e263901c463c9e5ace264d27f1cb","version_major":2,"version_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:15:03,679] Trial 0 finished with value: 0.5672311186790466 and parameters: {'alpha': 3.940718311712661e-07}. Best is trial 0 with value: 0.5672311186790466.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:19:50,404] Trial 1 finished with value: 0.6737157702445984 and parameters: {'alpha': 0.000315962759430424}. Best is trial 0 with value: 0.5672311186790466.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:24:52,465] Trial 2 finished with value: 0.5915345549583435 and parameters: {'alpha': 8.469555934330742e-08}. Best is trial 0 with value: 0.5672311186790466.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:30:04,421] Trial 3 finished with value: 0.6439839005470276 and parameters: {'alpha': 0.0002591593718759926}. Best is trial 0 with value: 0.5672311186790466.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:35:07,505] Trial 4 finished with value:
```

```
0.5589413046836853 and parameters: {'alpha': 1.1250003215250747e-05}.
Best is trial 4 with value: 0.5589413046836853.
Best hyperparameters: {'alpha': 1.1250003215250747e-05}
```

```
study.optimize(obj, n_trials=5, show_progress_bar=True)

# # Get the best hyperparameters
best_params = study.best_params
print("Best hyperparameters:", best_params)
```

{"model_id":"221f469eea6d4c478c3c6e5541b534e2","version_major":2,"version_minor":0}

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:39:58,092] Trial 5 finished with value:
0.8080242872238159 and parameters: {'alpha': 0.010217552605706478}.
Best is trial 4 with value: 0.5589413046836853.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:44:44,130] Trial 6 finished with value:
0.5599690675735474 and parameters: {'alpha': 8.647549026998659e-08}.
Best is trial 4 with value: 0.5589413046836853.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:49:29,361] Trial 7 finished with value:
0.7064862847328186 and parameters: {'alpha': 0.0007957720253456414}.
Best is trial 4 with value: 0.5589413046836853.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 18:54:53,946] Trial 8 finished with value:
0.7257596850395203 and parameters: {'alpha': 0.0011242535977567657}.
Best is trial 4 with value: 0.5589413046836853.
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
[I 2023-11-26 19:00:07,074] Trial 9 finished with value:
0.8955201506614685 and parameters: {'alpha': 0.02006339605386468}.
Best is trial 4 with value: 0.5589413046836853.
Best hyperparameters: {'alpha': 1.1250003215250747e-05}
```

Training the model with best hyperparameters

```
# Define the hyperparameters to optimize
batch_size = 16
dropout_rate = 0.4
```

```
alpha = 1.12e-05
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('pretrained_mobilenet',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30,
verbose=1, callbacks=callbacks)
```

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Epoch 1/30
157/157 [==============================] - 30s 100ms/step - loss:
1.0875 - accuracy: 0.5437 - val_loss: 1.2909 - val_accuracy: 0.2261 -
lr: 0.0010
Epoch 2/30
157/157 [==============================] - 15s 94ms/step - loss:
0.9602 - accuracy: 0.5931 - val_loss: 1.0473 - val_accuracy: 0.6178 -
lr: 0.0010
Epoch 3/30
157/157 [==============================] - 15s 94ms/step - loss:
0.8961 - accuracy: 0.6274 - val_loss: 0.8363 - val_accuracy: 0.7086 -
lr: 0.0010
Epoch 4/30
157/157 [==============================] - 15s 94ms/step - loss:
0.8859 - accuracy: 0.6227 - val_loss: 0.7063 - val_accuracy: 0.7309 -
```

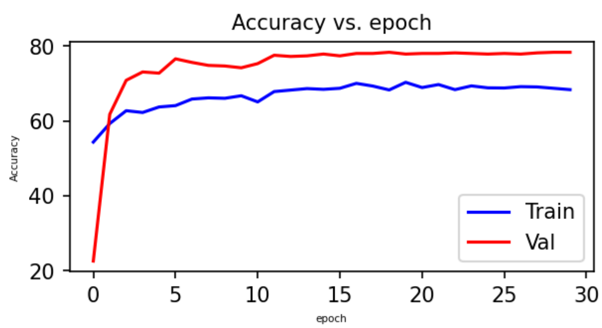
```
lr: 0.0010
Epoch 5/30
157/157 [==============================] - 14s 88ms/step - loss:
0.8570 - accuracy: 0.6374 - val_loss: 0.6485 - val_accuracy: 0.7277 -
lr: 0.0010
Epoch 6/30
157/157 [==============================] - 15s 96ms/step - loss:
0.8637 - accuracy: 0.6410 - val_loss: 0.6326 - val_accuracy: 0.7659 -
lr: 0.0010
Epoch 7/30
157/157 [==============================] - 14s 86ms/step - loss:
0.8322 - accuracy: 0.6586 - val_loss: 0.6363 - val_accuracy: 0.7564 -
lr: 0.0010
Epoch 8/30
157/157 [==============================] - 15s 94ms/step - loss:
0.8227 - accuracy: 0.6617 - val_loss: 0.6385 - val_accuracy: 0.7484 -
lr: 0.0010
Epoch 9/30
157/157 [==============================] - 14s 89ms/step - loss:
0.8192 - accuracy: 0.6606 - val_loss: 0.6338 - val_accuracy: 0.7468 -
lr: 0.0010
Epoch 10/30
157/157 [==============================] - 14s 86ms/step - loss:
0.8051 - accuracy: 0.6673 - val_loss: 0.6502 - val_accuracy: 0.7420 -
lr: 0.0010
Epoch 11/30
157/157 [==============================] - 15s 91ms/step - loss:
0.8010 - accuracy: 0.6510 - val_loss: 0.6443 - val_accuracy: 0.7532 -
lr: 0.0010
Epoch 12/30
157/157 [==============================] - 15s 94ms/step - loss:
0.7632 - accuracy: 0.6785 - val_loss: 0.6223 - val_accuracy: 0.7755 -
lr: 1.0000e-04
Epoch 13/30
157/157 [==============================] - 14s 90ms/step - loss:
0.7713 - accuracy: 0.6825 - val_loss: 0.6144 - val_accuracy: 0.7723 -
lr: 1.0000e-04
Epoch 14/30
157/157 [==============================] - 15s 95ms/step - loss:
0.7562 - accuracy: 0.6865 - val_loss: 0.6102 - val_accuracy: 0.7739 -
lr: 1.0000e-04
Epoch 15/30
157/157 [==============================] - 14s 87ms/step - loss:
0.7636 - accuracy: 0.6845 - val_loss: 0.6048 - val_accuracy: 0.7787 -
lr: 1.0000e-04
Epoch 16/30
157/157 [==============================] - 15s 94ms/step - loss:
0.7693 - accuracy: 0.6873 - val_loss: 0.6020 - val_accuracy: 0.7739 -
lr: 1.0000e-04
```

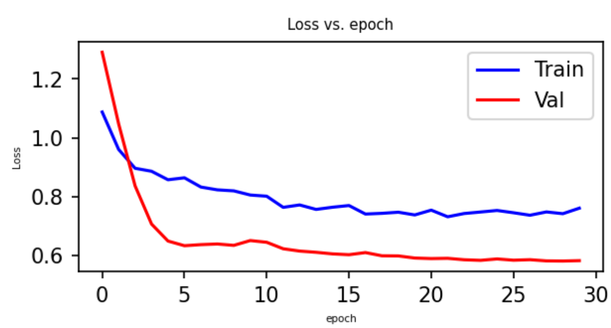
```
Epoch 17/30
157/157 [==============================] - 14s 88ms/step - loss:
0.7404 - accuracy: 0.7004 - val_loss: 0.6093 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 18/30
157/157 [==============================] - 14s 88ms/step - loss:
0.7428 - accuracy: 0.6933 - val_loss: 0.5984 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 19/30
157/157 [==============================] - 15s 94ms/step - loss:
0.7466 - accuracy: 0.6829 - val_loss: 0.5979 - val_accuracy: 0.7834 -
lr: 1.0000e-04
Epoch 20/30
157/157 [==============================] - 15s 93ms/step - loss:
0.7373 - accuracy: 0.7032 - val_loss: 0.5907 - val_accuracy: 0.7787 -
lr: 1.0000e-04
Epoch 21/30
157/157 [==============================] - 15s 91ms/step - loss:
0.7537 - accuracy: 0.6893 - val_loss: 0.5889 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 22/30
157/157 [==============================] - 14s 89ms/step - loss:
0.7311 - accuracy: 0.6972 - val_loss: 0.5897 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 23/30
157/157 [==============================] - 15s 90ms/step - loss:
0.7422 - accuracy: 0.6837 - val_loss: 0.5849 - val_accuracy: 0.7818 -
lr: 1.0000e-04
Epoch 24/30
157/157 [==============================] - 15s 93ms/step - loss:
0.7473 - accuracy: 0.6937 - val_loss: 0.5829 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 25/30
157/157 [==============================] - 14s 88ms/step - loss:
0.7526 - accuracy: 0.6885 - val_loss: 0.5877 - val_accuracy: 0.7787 -
lr: 1.0000e-04
Epoch 26/30
157/157 [==============================] - 14s 88ms/step - loss:
0.7448 - accuracy: 0.6881 - val_loss: 0.5833 - val_accuracy: 0.7803 -
lr: 1.0000e-04
Epoch 27/30
157/157 [==============================] - 14s 87ms/step - loss:
0.7363 - accuracy: 0.6917 - val_loss: 0.5851 - val_accuracy: 0.7787 -
lr: 1.0000e-04
Epoch 28/30
157/157 [==============================] - 15s 94ms/step - loss:
0.7476 - accuracy: 0.6909 - val_loss: 0.5810 - val_accuracy: 0.7818 -
lr: 1.0000e-04
Epoch 29/30
```

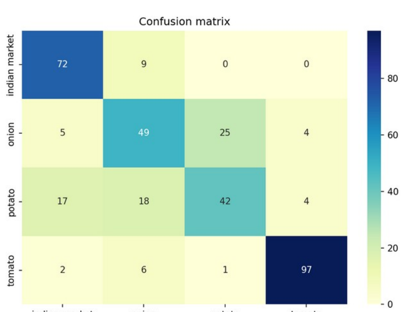
```
157/157 [==============================] - 14s 89ms/step - loss:
0.7416 - accuracy: 0.6873 - val_loss: 0.5805 - val_accuracy: 0.7834 -
lr: 1.0000e-04
Epoch 30/30
157/157 [==============================] - 14s 89ms/step - loss:
0.7603 - accuracy: 0.6837 - val_loss: 0.5818 - val_accuracy: 0.7834 -
lr: 1.0000e-04
```

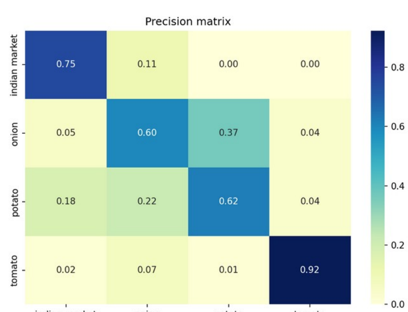
Checking model performance

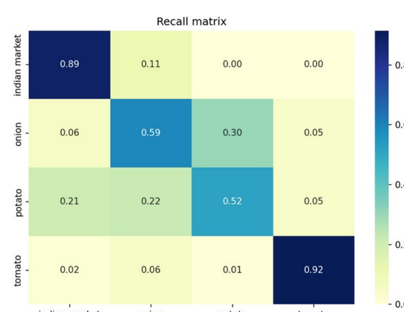
```
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)

22/22 [==============================] - 3s 87ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.6828879117965698
Max val accuracy = 0.7834395170211792
Min train loss = 0.7465966939926147
Min val loss = 0.5979008078575134
```











Observations

- Pre-trained mobile is not able to beat our custom model

Out of all the models, we can see that the pretrained VGG16 model performs the best out of all. So we will pick the VGG16 model, train it with the layers frozen, and post that we will try to fine tune the model by unfreezing the layers, in order to extract more performance for our use case

Pretrained VGG16 (best performer)

Defining the model

```
# downloading the model
pretrained_vgg16 =
tf.keras.applications.VGG16(include_top=False,weights='imagenet',input
_shape=(HEIGHT,WIDTH,3),pooling='avg')

# setting trainable = False
pretrained_vgg16.trainable = False

def getModel(name,dropout_rate,reg):
    seq = tf.keras.Sequential(
        name = name,
        layers = [
            tf.keras.layers.Resizing(300, 300),
            tf.keras.layers.RandomCrop(HEIGHT, WIDTH),
            tf.keras.layers.RandomRotation(factor=(-0.1,0.1)),
            tf.keras.layers.Rescaling(1.0/255),
            pretrained_vgg16,
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(256,activation='relu',kernel_regularizer=reg,bia
s_regularizer=reg),
            tf.keras.layers.Dropout(rate=dropout_rate),

tf.keras.layers.Dense(4,activation='softmax',kernel_regularizer=reg,bi
as_regularizer=reg)
        ]
    )

    return seq
```

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

Searching for the best hyperparameter

Training the model with best hyperparameters

```
# Define the hyperparameters to optimize
batch_size = 32
dropout_rate = 0.4
alpha = 2.30e-07
reg = tf.keras.regularizers.L2(l2=alpha)

# getting the dataset with selected batch_size and preprocessing it
train_ds = readData('train',batch_size=batch_size)
val_ds = readData('val',batch_size=batch_size)
test_ds = readData('test',batch_size=batch_size)

# calling the model
model = getModel('pretrainedVGG16',dropout_rate,reg)

# compile the model
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=30,
verbose=1, callbacks=callbacks)
```

```
Found 2507 files belonging to 4 classes.
Found 628 files belonging to 4 classes.
Found 351 files belonging to 4 classes.
Epoch 1/30
79/79 [==============================] - 52s 337ms/step - loss: 0.7060
- accuracy: 0.7423 - val_loss: 0.8665 - val_accuracy: 0.7245 - lr:
0.0010
Epoch 2/30
79/79 [==============================] - 17s 204ms/step - loss: 0.4556
- accuracy: 0.8313 - val_loss: 0.6089 - val_accuracy: 0.8360 - lr:
0.0010
Epoch 3/30
```

```
79/79 [==============================] - 17s 204ms/step - loss: 0.3749
- accuracy: 0.8580 - val_loss: 0.4139 - val_accuracy: 0.8806 - lr:
0.0010
Epoch 4/30
79/79 [==============================] - 18s 219ms/step - loss: 0.3752
- accuracy: 0.8576 - val_loss: 0.3467 - val_accuracy: 0.8806 - lr:
0.0010
Epoch 5/30
79/79 [==============================] - 21s 255ms/step - loss: 0.3259
- accuracy: 0.8787 - val_loss: 0.2948 - val_accuracy: 0.8997 - lr:
0.0010
Epoch 6/30
79/79 [==============================] - 28s 329ms/step - loss: 0.3382
- accuracy: 0.8791 - val_loss: 0.2728 - val_accuracy: 0.9013 - lr:
0.0010
Epoch 7/30
79/79 [==============================] - 19s 232ms/step - loss: 0.3145
- accuracy: 0.8835 - val_loss: 0.2691 - val_accuracy: 0.9029 - lr:
0.0010
Epoch 8/30
79/79 [==============================] - 16s 199ms/step - loss: 0.3262
- accuracy: 0.8712 - val_loss: 0.2607 - val_accuracy: 0.9108 - lr:
0.0010
Epoch 9/30
79/79 [==============================] - 18s 221ms/step - loss: 0.3074
- accuracy: 0.8815 - val_loss: 0.2533 - val_accuracy: 0.9092 - lr:
0.0010
Epoch 10/30
79/79 [==============================] - 19s 234ms/step - loss: 0.2825
- accuracy: 0.8867 - val_loss: 0.2648 - val_accuracy: 0.9013 - lr:
0.0010
Epoch 11/30
79/79 [==============================] - 18s 211ms/step - loss: 0.2985
- accuracy: 0.8843 - val_loss: 0.2690 - val_accuracy: 0.9092 - lr:
0.0010
Epoch 12/30
79/79 [==============================] - 17s 208ms/step - loss: 0.2663
- accuracy: 0.8975 - val_loss: 0.2465 - val_accuracy: 0.9140 - lr:
0.0010
Epoch 13/30
79/79 [==============================] - 19s 232ms/step - loss: 0.2641
- accuracy: 0.8931 - val_loss: 0.2646 - val_accuracy: 0.9092 - lr:
0.0010
Epoch 14/30
79/79 [==============================] - 17s 207ms/step - loss: 0.2757
- accuracy: 0.8927 - val_loss: 0.2585 - val_accuracy: 0.9108 - lr:
0.0010
Epoch 15/30
79/79 [==============================] - 17s 209ms/step - loss: 0.2572
```

```
- accuracy: 0.8951 - val_loss: 0.2703 - val_accuracy: 0.9124 - lr:
0.0010
Epoch 16/30
79/79 [==============================] - 18s 218ms/step - loss: 0.2656
- accuracy: 0.8963 - val_loss: 0.2513 - val_accuracy: 0.9156 - lr:
0.0010
Epoch 17/30
79/79 [==============================] - 17s 204ms/step - loss: 0.2606
- accuracy: 0.8971 - val_loss: 0.2725 - val_accuracy: 0.9045 - lr:
0.0010
Epoch 18/30
79/79 [==============================] - 19s 235ms/step - loss: 0.2294
- accuracy: 0.9091 - val_loss: 0.2618 - val_accuracy: 0.9156 - lr:
1.0000e-04
Epoch 19/30
79/79 [==============================] - 19s 222ms/step - loss: 0.2322
- accuracy: 0.9138 - val_loss: 0.2521 - val_accuracy: 0.9172 - lr:
1.0000e-04
Epoch 20/30
79/79 [==============================] - 16s 199ms/step - loss: 0.2510
- accuracy: 0.9007 - val_loss: 0.2492 - val_accuracy: 0.9172 - lr:
1.0000e-04
Epoch 21/30
79/79 [==============================] - 17s 208ms/step - loss: 0.2427
- accuracy: 0.9047 - val_loss: 0.2435 - val_accuracy: 0.9188 - lr:
1.0000e-04
Epoch 22/30
79/79 [==============================] - 17s 210ms/step - loss: 0.2147
- accuracy: 0.9178 - val_loss: 0.2428 - val_accuracy: 0.9204 - lr:
1.0000e-04
Epoch 23/30
79/79 [==============================] - 18s 222ms/step - loss: 0.2340
- accuracy: 0.9146 - val_loss: 0.2404 - val_accuracy: 0.9188 - lr:
1.0000e-04
Epoch 24/30
79/79 [==============================] - 17s 199ms/step - loss: 0.2353
- accuracy: 0.9043 - val_loss: 0.2368 - val_accuracy: 0.9204 - lr:
1.0000e-04
Epoch 25/30
79/79 [==============================] - 19s 232ms/step - loss: 0.2144
- accuracy: 0.9170 - val_loss: 0.2378 - val_accuracy: 0.9220 - lr:
1.0000e-04
Epoch 26/30
79/79 [==============================] - 17s 202ms/step - loss: 0.2263
- accuracy: 0.9194 - val_loss: 0.2474 - val_accuracy: 0.9188 - lr:
1.0000e-04
Epoch 27/30
79/79 [==============================] - 17s 209ms/step - loss: 0.2202
- accuracy: 0.9170 - val_loss: 0.2437 - val_accuracy: 0.9204 - lr:
```

```
1.0000e-04
Epoch 28/30
79/79 [==============================] - 17s 207ms/step - loss: 0.2201
- accuracy: 0.9186 - val_loss: 0.2436 - val_accuracy: 0.9204 - lr:
1.0000e-04
Epoch 29/30
79/79 [==============================] - 18s 209ms/step - loss: 0.2156
- accuracy: 0.9194 - val_loss: 0.2530 - val_accuracy: 0.9156 - lr:
1.0000e-04
Epoch 30/30
79/79 [==============================] - 17s 201ms/step - loss: 0.2257
- accuracy: 0.9162 - val_loss: 0.2525 - val_accuracy: 0.9188 - lr:
1.0000e-05
```

Fine tuning the model with a very small learning rate and 10 epochs

```
model.trainable = True

# compile the model after setting trainable = True

opt = tf.keras.optimizers.Adam(learning_rate=1e-5,)
model.compile(
    optimizer = opt,
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

# defining callbacks
callbacks = [

tf.keras.callbacks.EarlyStopping(monitor="val_loss",min_delta=0.0001,p
atience=10,restore_best_weights=True,start_from_epoch=10),

tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",factor=0.1,pat
ience=5,min_delta=0.0001,min_lr=0.000001)
]

# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10,
verbose=1, callbacks=callbacks)
```

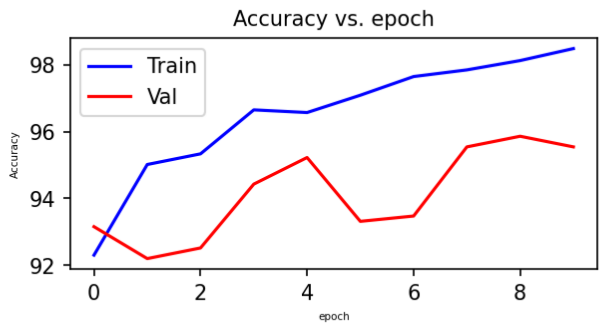
```
Epoch 1/10
79/79 [==============================] - 54s 502ms/step - loss: 0.1963
- accuracy: 0.9230 - val_loss: 0.2513 - val_accuracy: 0.9315 - lr:
1.0000e-05
Epoch 2/10
79/79 [==============================] - 39s 481ms/step - loss: 0.1392
- accuracy: 0.9501 - val_loss: 0.2556 - val_accuracy: 0.9220 - lr:
1.0000e-05
Epoch 3/10
```

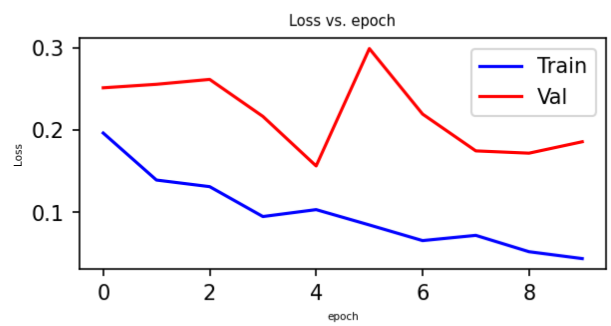
```
79/79 [==============================] - 41s 504ms/step - loss: 0.1312
- accuracy: 0.9533 - val_loss: 0.2615 - val_accuracy: 0.9252 - lr:
1.0000e-05
Epoch 4/10
79/79 [==============================] - 40s 493ms/step - loss: 0.0948
- accuracy: 0.9665 - val_loss: 0.2165 - val_accuracy: 0.9443 - lr:
1.0000e-05
Epoch 5/10
79/79 [==============================] - 40s 498ms/step - loss: 0.1032
- accuracy: 0.9657 - val_loss: 0.1563 - val_accuracy: 0.9522 - lr:
1.0000e-05
Epoch 6/10
79/79 [==============================] - 42s 508ms/step - loss: 0.0846
- accuracy: 0.9709 - val_loss: 0.2990 - val_accuracy: 0.9331 - lr:
1.0000e-05
Epoch 7/10
79/79 [==============================] - 40s 498ms/step - loss: 0.0656
- accuracy: 0.9765 - val_loss: 0.2194 - val_accuracy: 0.9347 - lr:
1.0000e-05
Epoch 8/10
79/79 [==============================] - 40s 496ms/step - loss: 0.0720
- accuracy: 0.9785 - val_loss: 0.1746 - val_accuracy: 0.9554 - lr:
1.0000e-05
Epoch 9/10
79/79 [==============================] - 42s 518ms/step - loss: 0.0521
- accuracy: 0.9813 - val_loss: 0.1718 - val_accuracy: 0.9586 - lr:
1.0000e-05
Epoch 10/10
79/79 [==============================] - 41s 511ms/step - loss: 0.0437
- accuracy: 0.9848 - val_loss: 0.1858 - val_accuracy: 0.9554 - lr:
1.0000e-05
```

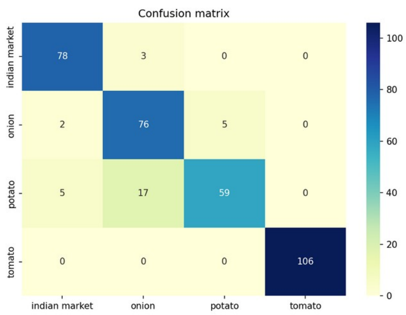
Checking model performance

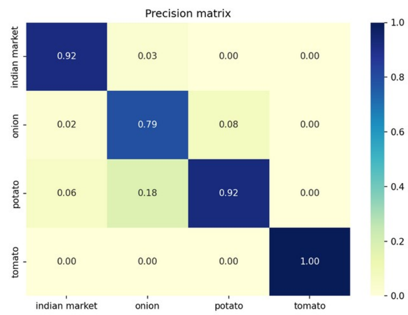
```python
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

plot_loss(history)
ConfusionMatrix(y_true,y_pred,class_names)
```

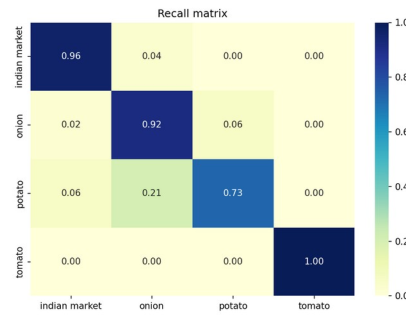
```
11/11 [==============================] - 6s 514ms/step
Found 351 files belonging to 4 classes.
Max train accuracy = 0.9812524914741516
Max val accuracy = 0.9585987329483032
Min train loss = 0.05207398533821106
Min val loss = 0.17184984683990479
```











Observations

- The model quickly started to overfit after the first epoch
- This is the reason why we have to be very careful when fine tuning the complete model
- We have restored the best weight of the model where the validation loss is minimum
- We are able to bosst the performance of the model by fine tuning it

Testing on the test set

```
y_prob = model.predict(test_ds)
y_pred = tf.argmax(y_prob, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)
x_true = tf.concat([x for x, y in test_ds], axis=0)
class_names = readData('test').class_names

print(f"Test accuracy: {metrics.accuracy_score(y_true,y_pred)*100:.4f}
%")

11/11 [==============================] - 2s 204ms/step
Found 351 files belonging to 4 classes.
Test accuracy: 90.8832%
```

Conclusion

- We created the train and validation split since the dataset did not have a separate folder for validation images

- We observed that some of the onion images were similar to potato, which might confuse the model between the two classes
- Also, some of the market images contains vegetables. If the vegetables are either potato, tomato or onion, the model might confuse between the classes
- We trained a CNN from scratch, with proper regularization, dropout and data augmentation, since the dataset size is very small and the model is prone to overfit
- The custom CNN model was able to outperform other pre-trained model except VGG16, without fine-tuning.
- The other pretrained models might performa better with proper fine-tuning
- VGG16 model without fine-tuning was performaing the best
- After fine tuning the pretrained VGG16 model, we were able to significantly boost the performance both in terms of crossentropy and accuracy
- In the fine tuning step, the model quickly started to overfit despite having a very low learning rate and few number of epochs, hence it is advisable to be very careful while tuning the whole model.
- The confusion metrics looks healthy except for potato and onion classes. As we discussed that the images are similar, this might be causing the model to confuse between the two clases.
- The final test accuracy of the best model comes out to be 90.88%