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

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

The main aim is to classify the correct vegetable from different images.

Double-click (or enter) to edit

```
import os
import PIL
import glob
import random
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objs as go
import seaborn as sns

import sklearn.metrics as metrics
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay

import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from keras.models import load_model
from tensorflow.random import set_seed
```

```
gpus = tf.config.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu,True)
```

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

def ConfusionMatrix(model, ds, label_list):
# Note: This logic doesn't work with shuffled datasets
    plt.figure(figsize=(15,15))
    y_pred = model.predict(ds)
    predicted_categories = tf.argmax(y_pred, axis=1)
    true_categories = tf.concat([y for x, y in ds], axis=0)
    cm = metrics.confusion_matrix(true_categories,predicted_categories) # last batch
    sns.heatmap(cm, annot=True, xticklabels=label_list, yticklabels=label_list, cmap="YlGnBu", fmt='g')
    plt.show()

def testAccuracy(model):
    true_categories = tf.concat([y for x, y in test_ds1], axis=0)
    images = tf.concat([x for x, y in test_ds1], axis=0)
    y_pred = model.predict(test_ds1)
    class_names = test_ds.class_names
    predicted_categories = tf.argmax(y_pred, axis=1)
    test_acc = metrics.accuracy_score(true_categories, predicted_categories) * 100
    print(f'\nTest Accuracy: {test_acc:.2f}%\n')

def plot_image(pred_array, true_label, img, class_names):
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(pred_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ".format(class_names[predicted_label],
                               100*np.max(pred_array),
                               ),
                               color=color)

def predictions(model):
    true_categories = tf.concat([y for x, y in test_ds1], axis=0)
    images = tf.concat([x for x, y in test_ds1], axis=0)
    y_pred = model.predict(test_ds1)
    class_names = test_ds.class_names
    # Randomly sample 15 test images and plot it with their predicted labels, and the true labels.
    indices = random.sample(range(len(images)), 15)
    # Color correct predictions in blue and incorrect predictions in red.
    num_rows = 5
    num_cols = 3
```

```
    num_images = num_rows*num_cols
    plt.figure(figsize=(4*num_cols, 2*num_rows))
    for i,index in enumerate(indices):
      plt.subplot(num_rows, num_cols, i+1)
      plot_image(y_pred[index], true_categories[index], images[index],class_names)
    plt.tight_layout()
    plt.show()
```

```
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint("final_model.h5", save_best_only=True)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",patience=5, restore_best_weights=True
)
```

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
train_path = '/content/drive/My Drive/Scaler Case studies/ninjacart_data/train'
```

```
test_path = '/content/drive/My Drive/Scaler Case studies/ninjacart_data/test'
```

```
training_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_directory(
                    train_path,
                    target_size=(150,150),
                    class_mode='categorical',
                    batch_size=126
)

test_generator = test_datagen.flow_from_directory(
                            test_path,
                            target_size=(150,150),
                            class_mode='categorical',
                            batch_size=126
)
```

```
Found 3135 images belonging to 4 classes.
Found 351 images belonging to 4 classes.
```

```
class_dirs = os.listdir("../content/drive/My Drive/Scaler Case studies/ninjacart_data/train") # list all directories inside "train" folder

image_dict = {} # dict to store image array(key) for every class(value)

count_dict = {} # dict to store count of files(key) for every class(value)

# iterate over all class_dirs
for cls in class_dirs:
    # get list of all paths inside the subdirectory
    file_paths = glob.glob(f'../content/drive/My Drive/Scaler Case studies/ninjacart_data/train/{cls}/*')
    # count number of files in each class and add it to count_dict
    count_dict[cls] = len(file_paths)
    # select random item from list of image paths
    image_path = random.choice(file_paths)
    # load image using keras utility function and save it in image_dict
    image_dict[cls] = tf.keras.utils.load_img(image_path)
```

```
batch_size = 32
img_height = 180
img_width = 180
```

Splitting the dataset into train, validation, and test set

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  "/content/drive/My Drive/Scaler Case studies/ninjacart_data/train",
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  "/content/drive/My Drive/Scaler Case studies/ninjacart_data/train",
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

```
class_names = train_ds.class_names
print(class_names)
```

```
['indian market', 'onion', 'potato', 'tomato']
```

```
print('\nLoading Test Data...')
test_ds = tf.keras.utils.image_dataset_from_directory(
    "/content/drive/My Drive/Scaler Case studies/ninjacart_data/test", shuffle = False,
)
```

```
Loading Test Data...
Found 351 files belonging to 4 classes.
```

Exploratory Data Analysis

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


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

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

Plotting class distribution & Visualizing Image dimensions with their plots

```
# draw a bar plot using pandas in-built plotting function
plt.figure(figsize=(15,12))
df_count_train.plot.bar(x='class', y='count', title="Training Data Count per class")
plt.show()
```

```
<Figure size 1500x1200 with 0 Axes>
```

Data Preprocessing

```
height, width = 227, 227

# Data Processing Stage with resizing and rescaling operations
data_preprocess = keras.Sequential(
    name="data_preprocess",
    layers=[
        layers.Resizing(height, width), # Shape Preprocessing
        layers.Rescaling(1.0/255), # Value Preprocessing
    ]
)

# Perform Data Processing on the train, val, test dataset
train_ds1 = train_ds.map(lambda x, y: (data_preprocess(x), y))
val_ds1 = val_ds.map(lambda x, y: (data_preprocess(x), y))
test_ds1 = test_ds.map(lambda x, y: (data_preprocess(x), y))
```

Creating model architecture and training

```
num_classes = 4
hidden_size_1 = 1024
hidden_size_2 = 256
```

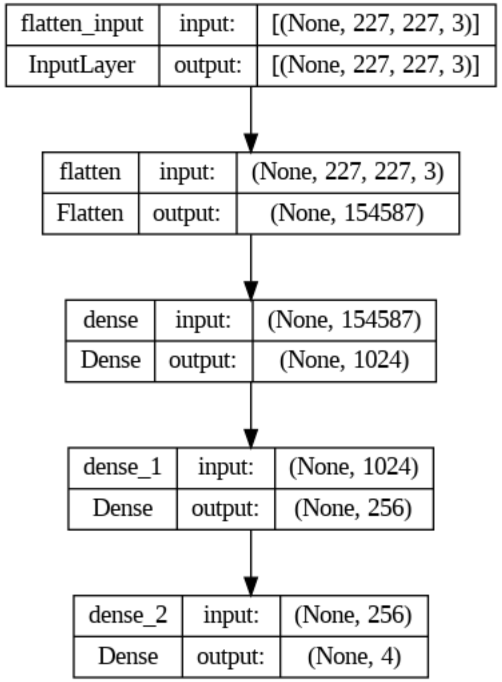
Training Simple Neural Network (MLP)

```
model_ann = keras.Sequential(
    name="model_ann",
    layers=[
        layers.Flatten(input_shape=(height, width, 3)),
        layers.Dense(units=hidden_size_1, activation='relu'), # hidden layer 1
        layers.Dense(units=hidden_size_2, activation='relu'), # hidden layer 2
        layers.Dense(units=num_classes, activation='softmax'), # output layer
    ]
)
```

```
model_ann.summary()
```

```
Model: "model_ann"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 154587)            0

 dense (Dense)               (None, 1024)              158298112

 dense_1 (Dense)             (None, 256)               262400

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 158561540 (604.86 MB)
Trainable params: 158561540 (604.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
tf.keras.utils.plot_model(model_ann, to_file="model_ann.png", show_shapes=True)
```




```
model_ann.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
early_stopping = keras.callbacks.EarlyStopping(patience=5)
```

```
# epochs = 25
# model_fit = model_ann.fit(train_ds1, validation_data=val_ds1, epochs=epochs,callbacks=early_stopping)
```

```
#training_plot(['loss', 'accuracy'], model_fit)
```

```
#testAccuracy(model_ann)
#model_ann.evaluate(test_ds1)
```

```
#ConfusionMatrix(model_ann, test_ds1, test_ds.class_names)
```

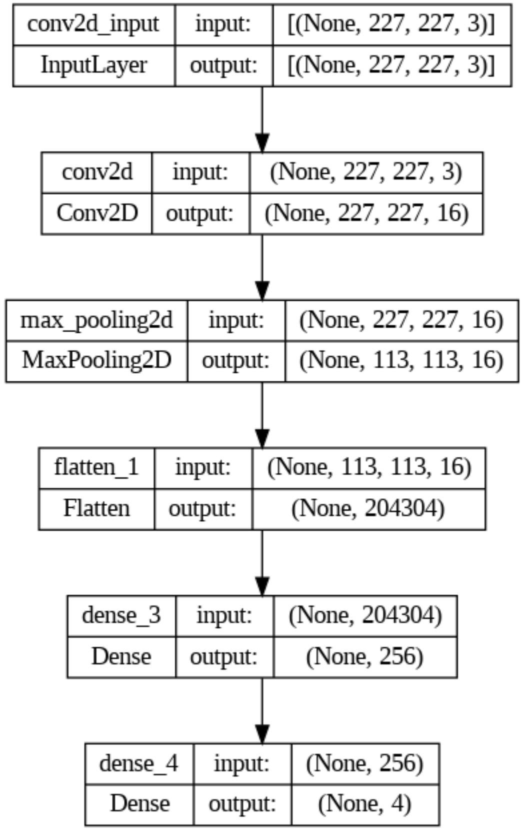
Training a simple CNN (Baseline Model)

```
model_base_cnn = keras.Sequential(
    name="model_base_cnn",
    layers=[
        layers.Conv2D(filters=16, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(units=256, activation='relu'),
        layers.Dense(units=num_classes, activation='softmax')
    ]
)
```

```
model_base_cnn.summary()
```

```
Model: "model_base_cnn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 227, 227, 16)      448

 max_pooling2d (MaxPooling2  (None, 113, 113, 16)      0
 D)

 flatten_1 (Flatten)         (None, 204304)            0

 dense_3 (Dense)             (None, 256)               52302080

 dense_4 (Dense)             (None, 4)                 1028

=================================================================
Total params: 52303556 (199.52 MB)
Trainable params: 52303556 (199.52 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
tf.keras.utils.plot_model(model_base_cnn, to_file="model_cnn.png", show_shapes=True)
```




```
model_base_cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

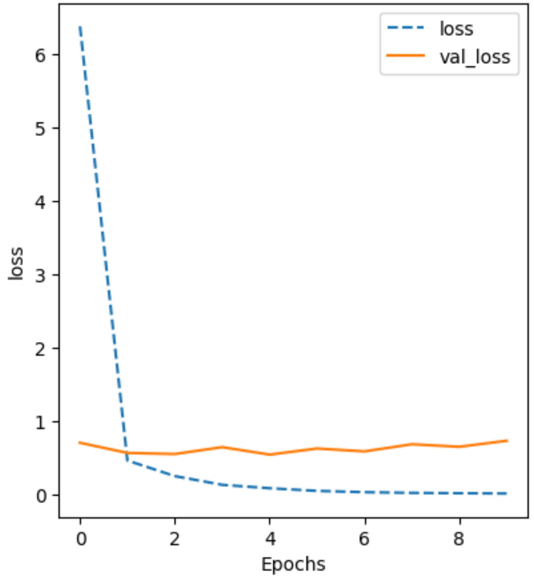
```
early_stopping = keras.callbacks.EarlyStopping(patience=5)
```

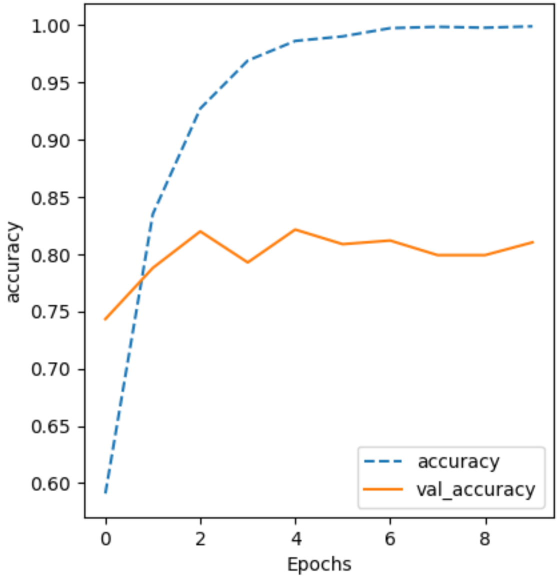
```
epochs = 32
model_base_cnn_fit = model_base_cnn.fit(train_ds1,validation_data=val_ds1,epochs=epochs,callbacks=early_stopping)
```

```
Epoch 1/32
79/79 [==============================] - 483s 5s/step - loss: 6.3814 - accuracy: 0.5909 - val_loss: 0.7022 - val_accuracy: 0.7432
Epoch 2/32
79/79 [==============================] - 174s 2s/step - loss: 0.4602 - accuracy: 0.8349 - val_loss: 0.5638 - val_accuracy: 0.7879
Epoch 3/32
79/79 [==============================] - 164s 2s/step - loss: 0.2477 - accuracy: 0.9270 - val_loss: 0.5490 - val_accuracy: 0.8198
Epoch 4/32
79/79 [==============================] - 172s 2s/step - loss: 0.1287 - accuracy: 0.9689 - val_loss: 0.6424 - val_accuracy: 0.7927
Epoch 5/32
79/79 [==============================] - 181s 2s/step - loss: 0.0821 - accuracy: 0.9860 - val_loss: 0.5411 - val_accuracy: 0.8214
Epoch 6/32
```

```
79/79 [==============================] - 171s 2s/step - loss: 0.0462 - accuracy: 0.9900 - val_loss: 0.6240 - val_accuracy: 0.8086
Epoch 7/32
79/79 [==============================] - 171s 2s/step - loss: 0.0282 - accuracy: 0.9972 - val_loss: 0.5848 - val_accuracy: 0.8118
Epoch 8/32
79/79 [==============================] - 180s 2s/step - loss: 0.0189 - accuracy: 0.9984 - val_loss: 0.6819 - val_accuracy: 0.7990
Epoch 9/32
79/79 [==============================] - 168s 2s/step - loss: 0.0145 - accuracy: 0.9976 - val_loss: 0.6477 - val_accuracy: 0.7990
Epoch 10/32
79/79 [==============================] - 169s 2s/step - loss: 0.0101 - accuracy: 0.9988 - val_loss: 0.7295 - val_accuracy: 0.8102
```

```
training_plot(['loss', 'accuracy'], model_base_cnn_fit)
```






```
model_base_cnn.evaluate(test_ds1)
```

Start coding or generate with AI.

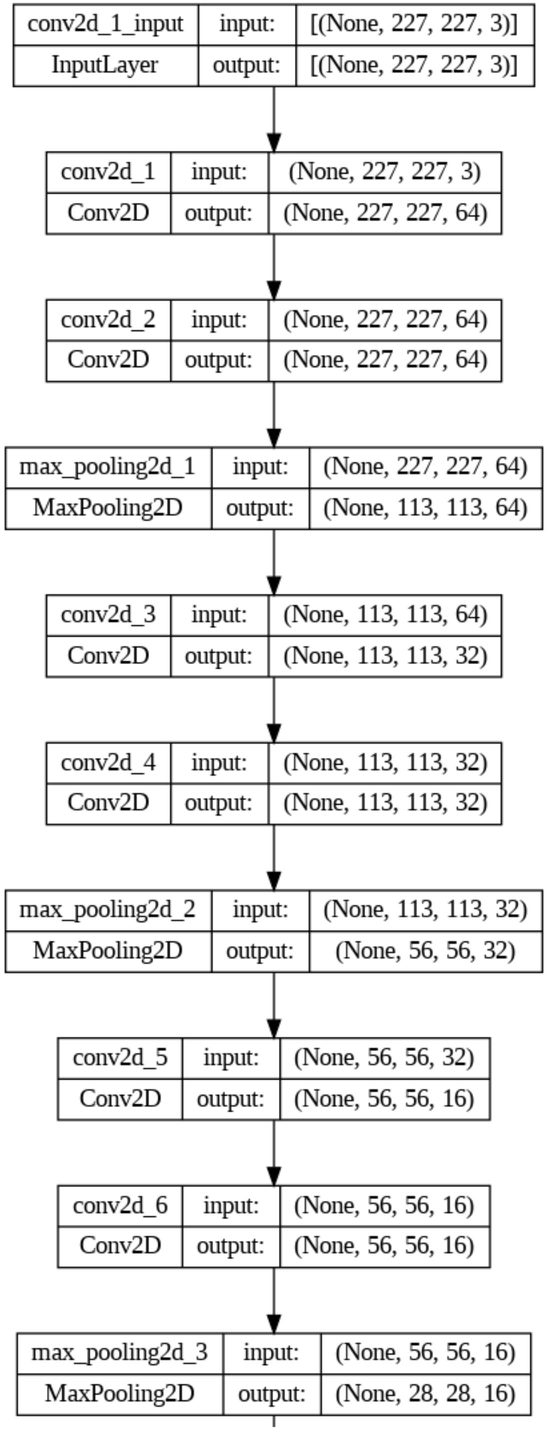
Improving Baseline Model

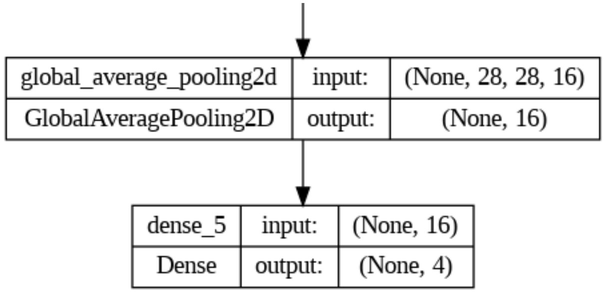
```
model_impv_cnn = keras.Sequential(
    name="model_impv_cnn_2",
    layers=[
        layers.Conv2D(filters=64, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.Conv2D(filters=64, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.MaxPooling2D(),
        layers.Conv2D(filters=32, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.Conv2D(filters=32, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.MaxPooling2D(),
        layers.Conv2D(filters=16, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.Conv2D(filters=16, kernel_size=3, strides=1, padding="same", activation='relu', input_shape=(height, width, 3)),
        layers.MaxPooling2D(),
        layers.GlobalAveragePooling2D(),
        layers.Dense(units=num_classes, activation='softmax')
    ]
)
```

```
model_impv_cnn.summary()
```

```
Model: "model_impv_cnn_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 227, 227, 64)      1792

 conv2d_2 (Conv2D)           (None, 227, 227, 64)      36928

 max_pooling2d_1 (MaxPoolin  (None, 113, 113, 64)      0
 g2D)

 conv2d_3 (Conv2D)           (None, 113, 113, 32)      18464

 conv2d_4 (Conv2D)           (None, 113, 113, 32)      9248

 max_pooling2d_2 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 56, 56, 16)        4624

 conv2d_6 (Conv2D)           (None, 56, 56, 16)        2320

 max_pooling2d_3 (MaxPoolin  (None, 28, 28, 16)        0
 g2D)

 global_average_pooling2d (  (None, 16)                0
 GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 4)                 68

=================================================================
Total params: 73444 (286.89 KB)
Trainable params: 73444 (286.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
tf.keras.utils.plot_model(model_impv_cnn, to_file="model_cnn_2.png", show_shapes=True)
```


conv2d_1_input | input: | [(None, 227, 227, 3)]
InputLayer | output: | [(None, 227, 227, 3)]
conv2d_1 | input: | (None, 227, 227, 3)
Conv2D | output: | (None, 227, 227, 64)
conv2d_2 | input: | (None, 227, 227, 64)
Conv2D | output: | (None, 227, 227, 64)
max_pooling2d_1 | input: | (None, 227, 227, 64)
MaxPooling2D | output: | (None, 113, 113, 64)
conv2d_3 | input: | (None, 113, 113, 64)
Conv2D | output: | (None, 113, 113, 32)
conv2d_4 | input: | (None, 113, 113, 32)
Conv2D | output: | (None, 113, 113, 32)
max_pooling2d_2 | input: | (None, 113, 113, 32)
MaxPooling2D | output: | (None, 56, 56, 32)
conv2d_5 | input: | (None, 56, 56, 32)
Conv2D | output: | (None, 56, 56, 16)
conv2d_6 | input: | (None, 56, 56, 16)
Conv2D | output: | (None, 56, 56, 16)
max_pooling2d_3 | input: | (None, 56, 56, 16)
MaxPooling2D | output: | (None, 28, 28, 16)




```
model_impv_cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
epochs = 32
model_fit = model_impv_cnn.fit(train_ds1, validation_data=val_ds1, epochs=epochs)
```

```
Epoch 1/32
28/79 [=========>....................] - ETA: 14:06 - loss: 0.7449 - accuracy: 0.7154
```

```
training_plot(['loss', 'accuracy'], model_fit)
```

```
#testAccuracy(model_impv_cnn)
model_base_cnn.evaluate(test_ds1)
```

```
#ConfusionMatrix(model_impv_cnn, test_ds1, test_ds.class_names)
```

Transfer Learning

VGG16

```
pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=[height,width, 3])
pretrained_model.trainable=False
vgg16_model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(15, activation='softmax')
])

vgg16_model.summary()


vgg16_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


#history_vgg16 = vgg16_model.fit(train_ds1, epochs=25, validation_data=val_ds1,callbacks=[checkpoint_callback,early_stopping_callback])
import functools
from tensorflow.keras.optimizers import Adam,SGD
top5_acc = functools.partial(tf.keras.metrics.SparseTopKCategoricalAccuracy())

opt = SGD(learning_rate=0.005, momentum=0.99)
vgg16_model.compile(
    optimizer=opt,
    loss = 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
print(vgg16_model.optimizer.get_config())
```