# Business Case

# Ninjacart: CV Classification

Suman Debnath

# Introduction

## Problem Statement

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

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

### Dataset Link

Download the dataset

### Context

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

### Data Collection

The images in this dataset were scraped from Google.

### Content

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

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

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

- **Tomato**: 106 images
- **Potato**: 83 images
- **Onion**: 81 images
- **Indian market**: 81 images

### Inspiration

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

### Concepts Tested:

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

# Detailed Analysis

## Importing all the `libs`

In [1]:
```
# Import common libraries
import os
import glob
import random
import numpy as np
import pandas as pd
import sklearn.metrics as metrics
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

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

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

plt.rcParams.update({'font.size': 14})
```

## Downloading the `dataset`

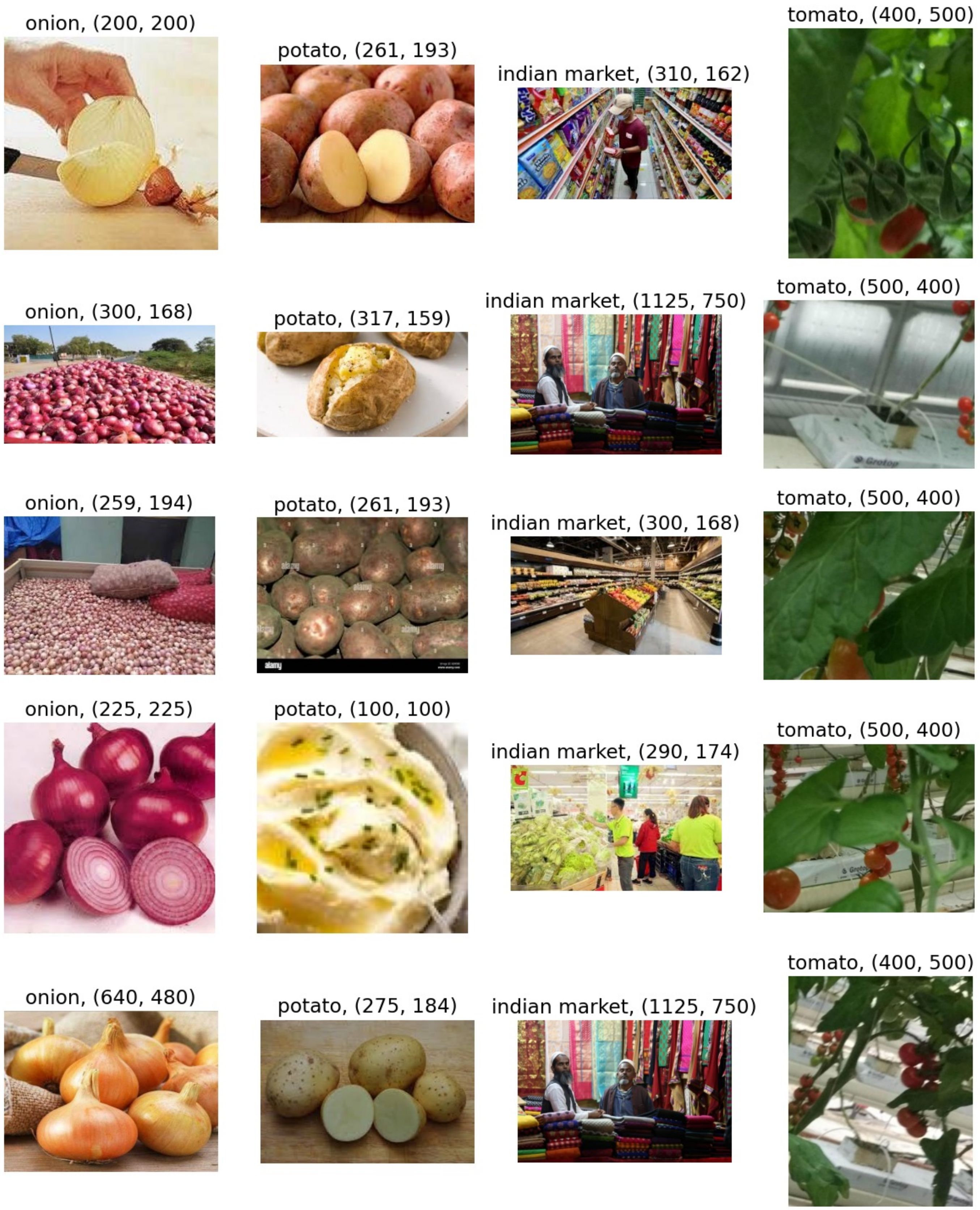
In [2]:
```
#!gdown 1clZX-lV_MLxKHSyeyTheX5OCQtNCUcqT
```

In [3]:
```
#!unzip ninjacart_data.zip
```

## Data Exploration

In [4]:
```
dataset_path = '../../dataset/ninjacart_data'

train_path = f"{dataset_path}/train/"
test_path = f"{dataset_path}/test/"
```

In [5]:
```
class_dirs = os.listdir(f"{train_path}")
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)

for _ in range(5):
    # iterate over all class_dirs
    for cls in class_dirs:
        # get list of all paths inside the subdirectory
        file_paths = glob.glob(f'{train_path}/{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)

    ## 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(3, 4, i + 1)
        # plot each image
        plt.imshow(img)
        # set "class name" along with "image size" as title
        plt.title(f'{cls}, {img.size}')
        plt.axis("off")

    plt.show()
```




- Notice that every Image has different dimension!

In [6]:
```
for product, n in count_dict.items():
    print(f"{product}: {n}")
```

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

In [7]:
```
# Creating the DataFrame
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
})

# Plotting with size adjustment
```

```python
ax = df_count_train.plot.bar(x='class', y='count', title="Training Data Count per class", figsize=(10, 6))

# Adding labels on top of each bar
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 3., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')

plt.show()
```


- Note that the train/test split already provided, we needn't create it explicitly!
- Also, we can observe that the data almost balanced (except the fact that there is little less data available for `india market` class)

## Load the `dataset`

In [9]:
```python
# Specifying the image size to resize all images
image_size = (256, 256)
image_shape = (256, 256, 3)

train_data = tf.keras.utils.image_dataset_from_directory(directory = train_path,
                                                         label_mode = 'categorical',
                                                         shuffle = True,
                                                         seed= 2024,
                                                         validation_split = 0.2,
                                                         subset = 'training',
                                                         image_size = image_size,
                                                         batch_size = 32)

val_data = tf.keras.utils.image_dataset_from_directory(directory = train_path,
                                                       label_mode = 'categorical',
                                                       shuffle = True,
                                                       seed= 2024,
                                                       validation_split = 0.2,
                                                       subset = 'validation',
                                                       image_size = image_size,
                                                       batch_size = 32)

test_data = tf.keras.utils.image_dataset_from_directory(directory = test_path,
                                                        label_mode = 'categorical',
                                                        shuffle = False,
```

```
                                                  seed=2024,
                                                  image_size = image_size,
                                                  batch_size = 32)
```

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

In [10]:
```python
# Get the first batch of images and labels from the train dataset
for images, labels in train_data.take(1):
    first_image = images[0]
    first_label = labels[0]
    print("Shape of the first batch of images in train_data:", images.shape)
    print("Shape of the first batch of labels in train_data:", labels.shape)
```

```
Shape of the first batch of images in train_data: (32, 256, 256, 3)
Shape of the first batch of labels in train_data: (32, 4)
2024-06-19 10:30:18.333755: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting wi
th status: OUT_OF_RANGE: End of sequence
```

## Building the model using `Transfer Learning`

### Image Preprocessing ( `Rescaling` )

In [11]:
```python
from tensorflow.keras import layers

# Define data preprocessing pipeline
data_preprocess = tf.keras.Sequential(
                                        name="data_preprocess",
                                        layers=[layers.Rescaling(1.0/255)]
                                     )

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

### VGG-19 ( `without` Data Augmentation)

In [12]:
```python
image_shape
```

Out[12]:
```
(256, 256, 3)
```

In [40]:
```python
pretrained_model = tf.keras.applications.VGG19(
                                                weights='imagenet',
                                                include_top=False,
                                                input_shape=image_shape)

pretrained_model.trainable=False

vgg19_model = tf.keras.Sequential([
                                        layers.Input(shape=image_shape),
                                        data_preprocess,
                                        pretrained_model,
                                        tf.keras.layers.GlobalAveragePooling2D(),
                                        tf.keras.layers.Dropout(rate = 0.1),
                                        tf.keras.layers.Dense(4, activation='softmax')
                                    ])
```

In [41]:
```python
vgg19_model.summary()
```

**Model: "sequential_7"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| data_preprocess (Sequential) | (None, 256, 256, 3) | 0 |
| vgg19 (Functional) | (None, 8, 8, 512) | 20,024,384 |
| global_average_pooling2d_7 (GlobalAveragePooling2D) | (None, 512) | 0 |
| dropout_7 (Dropout) | (None, 512) | 0 |
| dense_7 (Dense) | (None, 4) | 2,052 |

**Total params:** 20,026,436 (76.39 MB)

**Trainable params:** 2,052 (8.02 KB)

**Non-trainable params:** 20,024,384 (76.39 MB)

In [42]:
```
log_dir = "../../logs/VGG19"

tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("VGG19-1.keras", save_best_only=True)

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

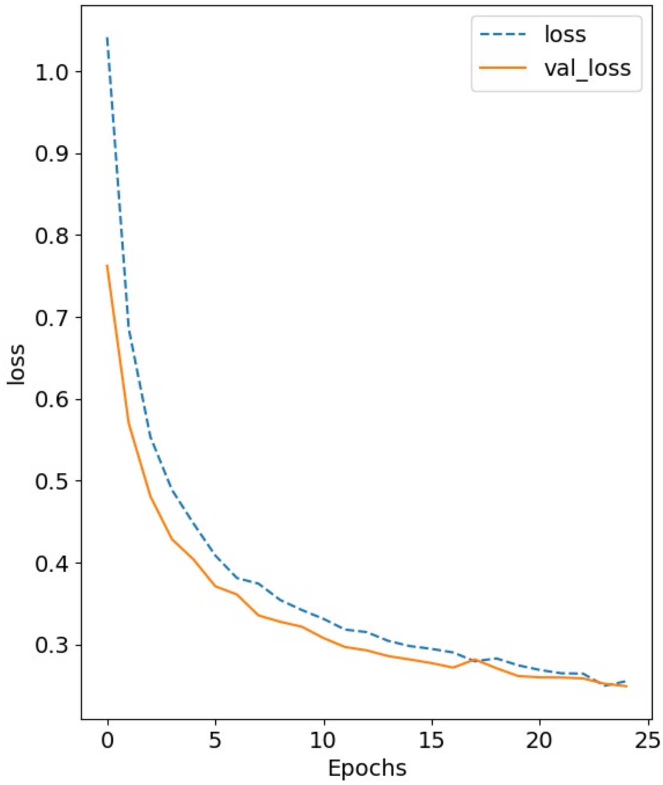
In [43]:
```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.002)
vgg19_model.compile(
                        optimizer=opt,
                        loss = 'categorical_crossentropy',
                        metrics=['accuracy']
                    )

# print(vgg19_model.optimizer.get_config())

model_fit = vgg19_model.fit(
                            train_data,
                            epochs=25,
                            validation_data=val_data,
                            callbacks=[tensorboard_cb,
                                       checkpoint_cb,
                                       early_stopping_cb]
                          )
```

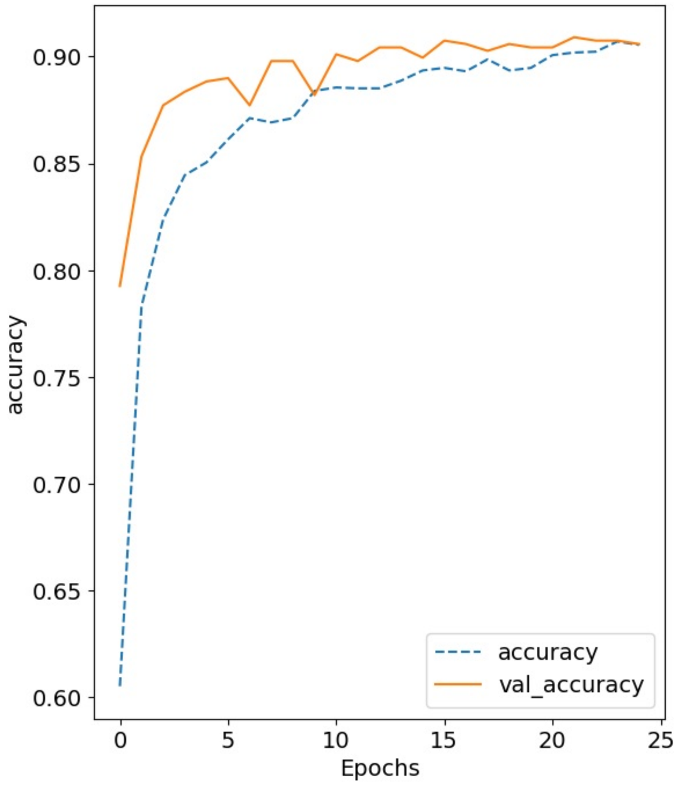
```
Epoch 1/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 38s 472ms/step - accuracy: 0.4785 - loss: 1.2239 - val_accuracy: 0.7927 - val_loss: 0
.7622
Epoch 2/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 464ms/step - accuracy: 0.7732 - loss: 0.7304 - val_accuracy: 0.8533 - val_loss: 0
.5700
Epoch 3/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 467ms/step - accuracy: 0.8146 - loss: 0.5792 - val_accuracy: 0.8772 - val_loss: 0
.4808
Epoch 4/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 465ms/step - accuracy: 0.8468 - loss: 0.5022 - val_accuracy: 0.8836 - val_loss: 0
.4283
Epoch 5/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 463ms/step - accuracy: 0.8466 - loss: 0.4604 - val_accuracy: 0.8884 - val_loss: 0
.4035
Epoch 6/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 466ms/step - accuracy: 0.8613 - loss: 0.4095 - val_accuracy: 0.8900 - val_loss: 0
.3709
Epoch 7/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 464ms/step - accuracy: 0.8754 - loss: 0.3804 - val_accuracy: 0.8772 - val_loss: 0
.3608
Epoch 8/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 463ms/step - accuracy: 0.8684 - loss: 0.3782 - val_accuracy: 0.8979 - val_loss: 0
.3353
Epoch 9/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 467ms/step - accuracy: 0.8711 - loss: 0.3567 - val_accuracy: 0.8979 - val_loss: 0
.3275
Epoch 10/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 465ms/step - accuracy: 0.8949 - loss: 0.3338 - val_accuracy: 0.8820 - val_loss: 0
.3215
Epoch 11/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 466ms/step - accuracy: 0.8872 - loss: 0.3287 - val_accuracy: 0.9011 - val_loss: 0
.3077
Epoch 12/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 465ms/step - accuracy: 0.8869 - loss: 0.3188 - val_accuracy: 0.8979 - val_loss: 0
.2967
Epoch 13/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 467ms/step - accuracy: 0.8867 - loss: 0.3136 - val_accuracy: 0.9043 - val_loss: 0
.2926
Epoch 14/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 472ms/step - accuracy: 0.8875 - loss: 0.3091 - val_accuracy: 0.9043 - val_loss: 0
.2856
Epoch 15/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 37s 470ms/step - accuracy: 0.8944 - loss: 0.2939 - val_accuracy: 0.8995 - val_loss: 0
.2815
Epoch 16/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 495ms/step - accuracy: 0.9037 - loss: 0.2883 - val_accuracy: 0.9075 - val_loss: 0
.2770
Epoch 17/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 498ms/step - accuracy: 0.8938 - loss: 0.2887 - val_accuracy: 0.9059 - val_loss: 0
.2715
Epoch 18/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 506ms/step - accuracy: 0.9044 - loss: 0.2767 - val_accuracy: 0.9027 - val_loss: 0
.2815
Epoch 19/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 497ms/step - accuracy: 0.8925 - loss: 0.2829 - val_accuracy: 0.9059 - val_loss: 0
.2708
Epoch 20/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 495ms/step - accuracy: 0.8943 - loss: 0.2758 - val_accuracy: 0.9043 - val_loss: 0
.2613
Epoch 21/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 518ms/step - accuracy: 0.9088 - loss: 0.2620 - val_accuracy: 0.9043 - val_loss: 0
.2596
Epoch 22/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 501ms/step - accuracy: 0.9107 - loss: 0.2582 - val_accuracy: 0.9091 - val_loss: 0
.2595
Epoch 23/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 513ms/step - accuracy: 0.9017 - loss: 0.2644 - val_accuracy: 0.9075 - val_loss: 0
.2584
Epoch 24/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 504ms/step - accuracy: 0.9151 - loss: 0.2373 - val_accuracy: 0.9075 - val_loss: 0
.2518
Epoch 25/25
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 518ms/step - accuracy: 0.9063 - loss: 0.2515 - val_accuracy: 0.9059 - val_loss: 0
.2488
```

In [44]:

```
def training_plot(metrics, history):
  f, ax = plt.subplots(1, len(metrics), figsize=(7*len(metrics), 8))
  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])


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



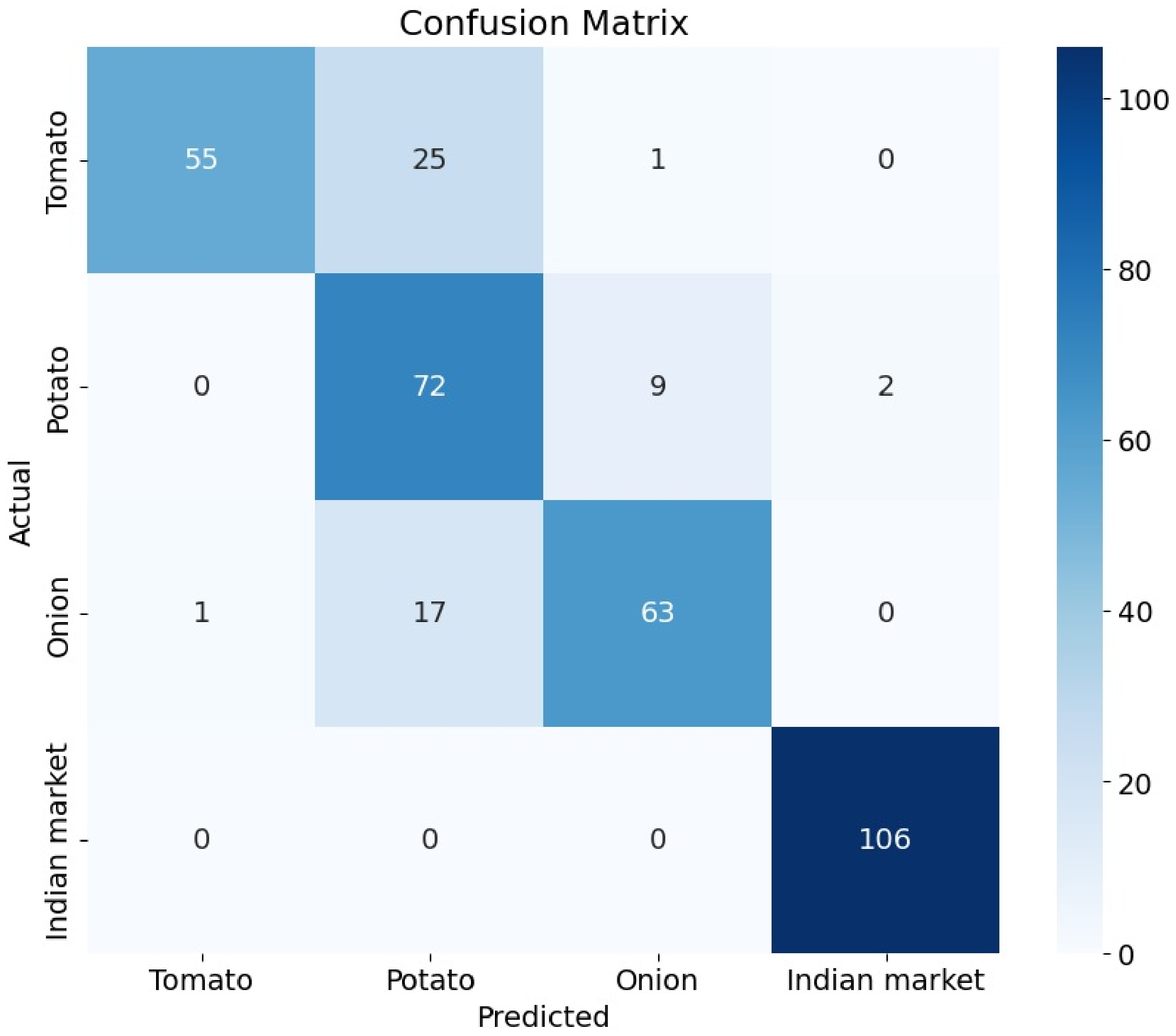




In [45]:
```
# Evaluate the model
loss, acc = vgg19_model.evaluate(test_data, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
11/11 - 4s - 369ms/step - accuracy: 0.8433 - loss: 0.4245
Restored model, accuracy: 84.33%
```

In [46]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate the model on the test set
test_loss, test_acc = vgg19_model.evaluate(test_data, verbose=2)
print(f"Test accuracy: {test_acc}")

# Get predictions from the model
predictions = vgg19_model.predict(test_data)
pred_labels = np.argmax(predictions, axis=1)

# Create a mapping for class names
classes = ['Tomato', 'Potato', 'Onion', 'Indian market']
label_map = {label: idx for idx, label in enumerate(classes)}
reverse_label_map = {v: k for k, v in label_map.items()}

# Extract actual labels from the test dataset
actual_labels = []
for _, label in test_data.unbatch():
    actual_labels.append(label.numpy())

actual_labels = np.array(actual_labels)
actual_labels = np.argmax(actual_labels, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(actual_labels, pred_labels)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print(classification_report(actual_labels, pred_labels, target_names=classes))
```

```
11/11 - 4s - 369ms/step - accuracy: 0.8433 - loss: 0.4245
Test accuracy: 0.8433048725128174
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 354ms/step
2024-06-19 10:57:54.935328: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting wi
th status: OUT_OF_RANGE: End of sequence
```




```
               precision    recall  f1-score   support

       Tomato       0.98      0.68      0.80        81
       Potato       0.63      0.87      0.73        83
        Onion       0.86      0.78      0.82        81
Indian market       0.98      1.00      0.99       106

     accuracy                           0.84       351
    macro avg       0.86      0.83      0.84       351
 weighted avg       0.87      0.84      0.85       351
```

## VGG-19 (`with` Data Augmentation)

In [47]:
```
from tensorflow.keras import layers

# Define data augmentation pipeline
data_augmentation = tf.keras.Sequential(
                                            name="data_augmentation",
                                            layers=[
                                                layers.RandomFlip("horizontal_and_vertical"),
                                                layers.RandomRotation(0.2),
                                                layers.RandomTranslation(height_factor=0.2, width_factor=0.2)
                                            ]
                                        )

# Define data preprocessing pipeline
data_preprocess = tf.keras.Sequential(
                                        name="data_preprocess",
                                        layers=[layers.Rescaling(1.0/255)]
                                    )
```

In [48]:
```
# Load pre-trained VGG19 model
pretrained_model = tf.keras.applications.VGG19(
```

```
                                                    weights='imagenet',
                                                    include_top=False,
                                                    input_shape=image_shape
                                                )
pretrained_model.trainable = False

# Create the Sequential model
vgg19_model = tf.keras.Sequential([
                                    layers.Input(shape=image_shape),
                                    data_augmentation,
                                    data_preprocess,
                                    pretrained_model,
                                    layers.GlobalAveragePooling2D(),
                                    layers.Dropout(rate=0.1),
                                    layers.Dense(4, activation='softmax')
                                ])
```

In [49]:
```
vgg19_model.summary()
```

**Model: "sequential_8"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| data_augmentation (Sequential) | (None, 256, 256, 3) | 0 |
| data_preprocess (Sequential) | (None, 256, 256, 3) | 0 |
| vgg19 (Functional) | (None, 8, 8, 512) | 20,024,384 |
| global_average_pooling2d_8 (GlobalAveragePooling2D) | (None, 512) | 0 |
| dropout_8 (Dropout) | (None, 512) | 0 |
| dense_8 (Dense) | (None, 4) | 2,052 |

**Total params:** 20,026,436 (76.39 MB)

**Trainable params:** 2,052 (8.02 KB)

**Non-trainable params:** 20,024,384 (76.39 MB)

In [50]:
```
log_dir = "../../logs/VGG19"

tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("VGG19.keras", save_best_only=True)

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

In [51]:
```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.002)
vgg19_model.compile(
                        optimizer=opt,
                        loss = 'categorical_crossentropy',
                        metrics=['accuracy']
                    )

# print(vgg19_model.optimizer.get_config())

model_fit = vgg19_model.fit(
                            train_data,
                            epochs=30,
                            validation_data=val_data,
                            callbacks=[tensorboard_cb,
                                       checkpoint_cb,
                                       early_stopping_cb]
                            )
```

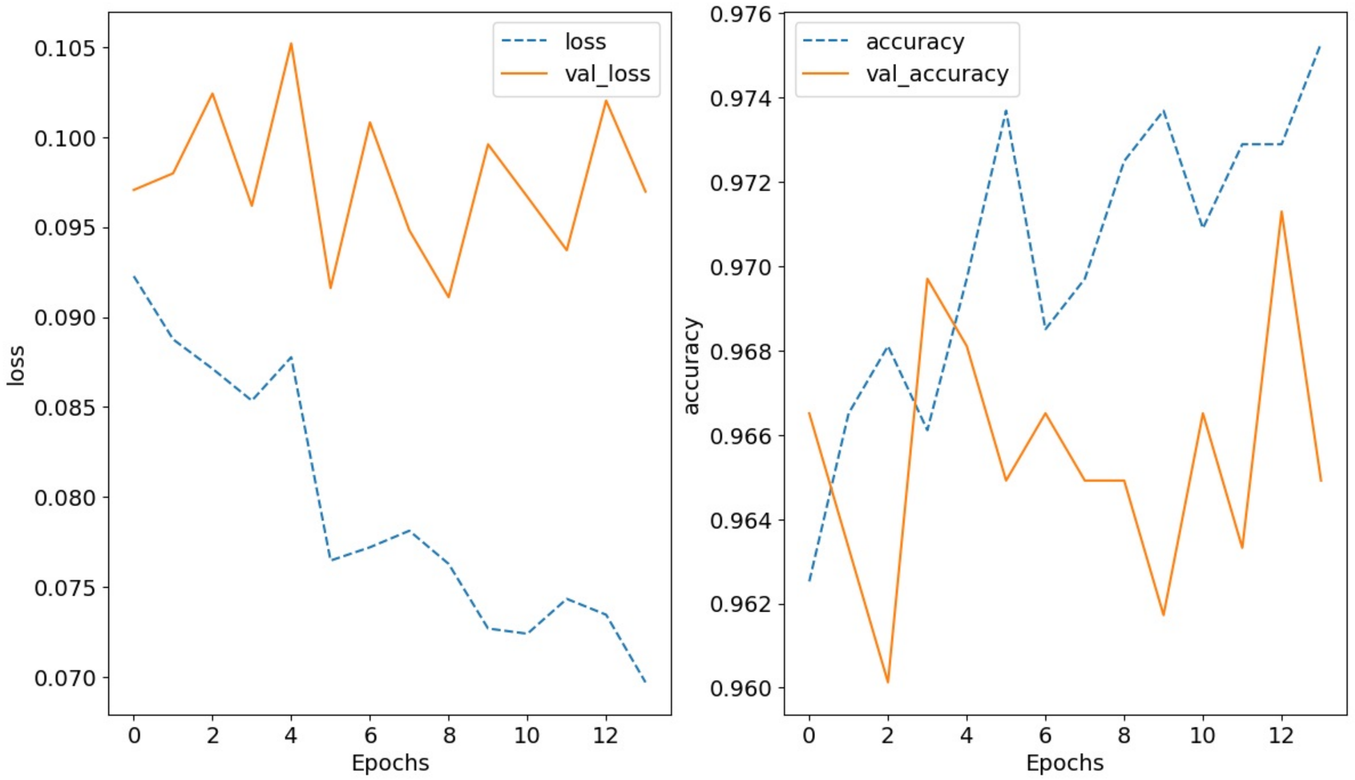
```
Epoch 1/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 510ms/step - accuracy: 0.4979 - loss: 1.1859 - val_accuracy: 0.7735 - val_loss: 0
.7932
Epoch 2/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 499ms/step - accuracy: 0.7434 - loss: 0.7884 - val_accuracy: 0.8102 - val_loss: 0
.6220
Epoch 3/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 498ms/step - accuracy: 0.7898 - loss: 0.6569 - val_accuracy: 0.8389 - val_loss: 0
.5284
Epoch 4/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 501ms/step - accuracy: 0.7984 - loss: 0.5930 - val_accuracy: 0.8708 - val_loss: 0
```

```
.4769
Epoch 5/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 497ms/step - accuracy: 0.8134 - loss: 0.5356 - val_accuracy: 0.8676 - val_loss: 0
.4450
Epoch 6/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 498ms/step - accuracy: 0.8169 - loss: 0.5064 - val_accuracy: 0.8740 - val_loss: 0
.4193
Epoch 7/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 497ms/step - accuracy: 0.8234 - loss: 0.4821 - val_accuracy: 0.8836 - val_loss: 0
.3912
Epoch 8/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 498ms/step - accuracy: 0.8136 - loss: 0.4815 - val_accuracy: 0.8884 - val_loss: 0
.3710
Epoch 9/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 501ms/step - accuracy: 0.8441 - loss: 0.4422 - val_accuracy: 0.8915 - val_loss: 0
.3672
Epoch 10/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 500ms/step - accuracy: 0.8554 - loss: 0.4329 - val_accuracy: 0.8963 - val_loss: 0
.3613
Epoch 11/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 517ms/step - accuracy: 0.8447 - loss: 0.4305 - val_accuracy: 0.8900 - val_loss: 0
.3410
Epoch 12/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 42s 526ms/step - accuracy: 0.8521 - loss: 0.4110 - val_accuracy: 0.8963 - val_loss: 0
.3439
Epoch 13/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 542ms/step - accuracy: 0.8502 - loss: 0.4063 - val_accuracy: 0.8915 - val_loss: 0
.3253
Epoch 14/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 551ms/step - accuracy: 0.8555 - loss: 0.4066 - val_accuracy: 0.8931 - val_loss: 0
.3250
Epoch 15/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 44s 557ms/step - accuracy: 0.8536 - loss: 0.3948 - val_accuracy: 0.8931 - val_loss: 0
.3190
Epoch 16/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 569ms/step - accuracy: 0.8690 - loss: 0.3776 - val_accuracy: 0.8979 - val_loss: 0
.3189
Epoch 17/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 46s 585ms/step - accuracy: 0.8452 - loss: 0.3984 - val_accuracy: 0.8963 - val_loss: 0
.3051
Epoch 18/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 607ms/step - accuracy: 0.8660 - loss: 0.3715 - val_accuracy: 0.8915 - val_loss: 0
.3035
Epoch 19/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 576ms/step - accuracy: 0.8518 - loss: 0.3814 - val_accuracy: 0.9059 - val_loss: 0
.3100
Epoch 20/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 601ms/step - accuracy: 0.8500 - loss: 0.3838 - val_accuracy: 0.9011 - val_loss: 0
.2952
Epoch 21/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 593ms/step - accuracy: 0.8601 - loss: 0.3705 - val_accuracy: 0.9107 - val_loss: 0
.2948
Epoch 22/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 593ms/step - accuracy: 0.8606 - loss: 0.3779 - val_accuracy: 0.9059 - val_loss: 0
.2849
Epoch 23/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 600ms/step - accuracy: 0.8735 - loss: 0.3469 - val_accuracy: 0.8995 - val_loss: 0
.2819
Epoch 24/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 628ms/step - accuracy: 0.8767 - loss: 0.3385 - val_accuracy: 0.9107 - val_loss: 0
.2817
Epoch 25/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 631ms/step - accuracy: 0.8787 - loss: 0.3465 - val_accuracy: 0.9027 - val_loss: 0
.2794
Epoch 26/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 623ms/step - accuracy: 0.8729 - loss: 0.3480 - val_accuracy: 0.9043 - val_loss: 0
.2758
Epoch 27/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 601ms/step - accuracy: 0.8740 - loss: 0.3330 - val_accuracy: 0.9011 - val_loss: 0
.2784
Epoch 28/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 596ms/step - accuracy: 0.8719 - loss: 0.3416 - val_accuracy: 0.9011 - val_loss: 0
.2691
Epoch 29/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 603ms/step - accuracy: 0.8667 - loss: 0.3466 - val_accuracy: 0.9027 - val_loss: 0
.2731
Epoch 30/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 611ms/step - accuracy: 0.8552 - loss: 0.3483 - val_accuracy: 0.9059 - val_loss: 0
.2760
```

In [80]:

```
def training_plot(metrics, history):
  f, ax = plt.subplots(1, len(metrics), figsize=(7*len(metrics), 8))
  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])


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



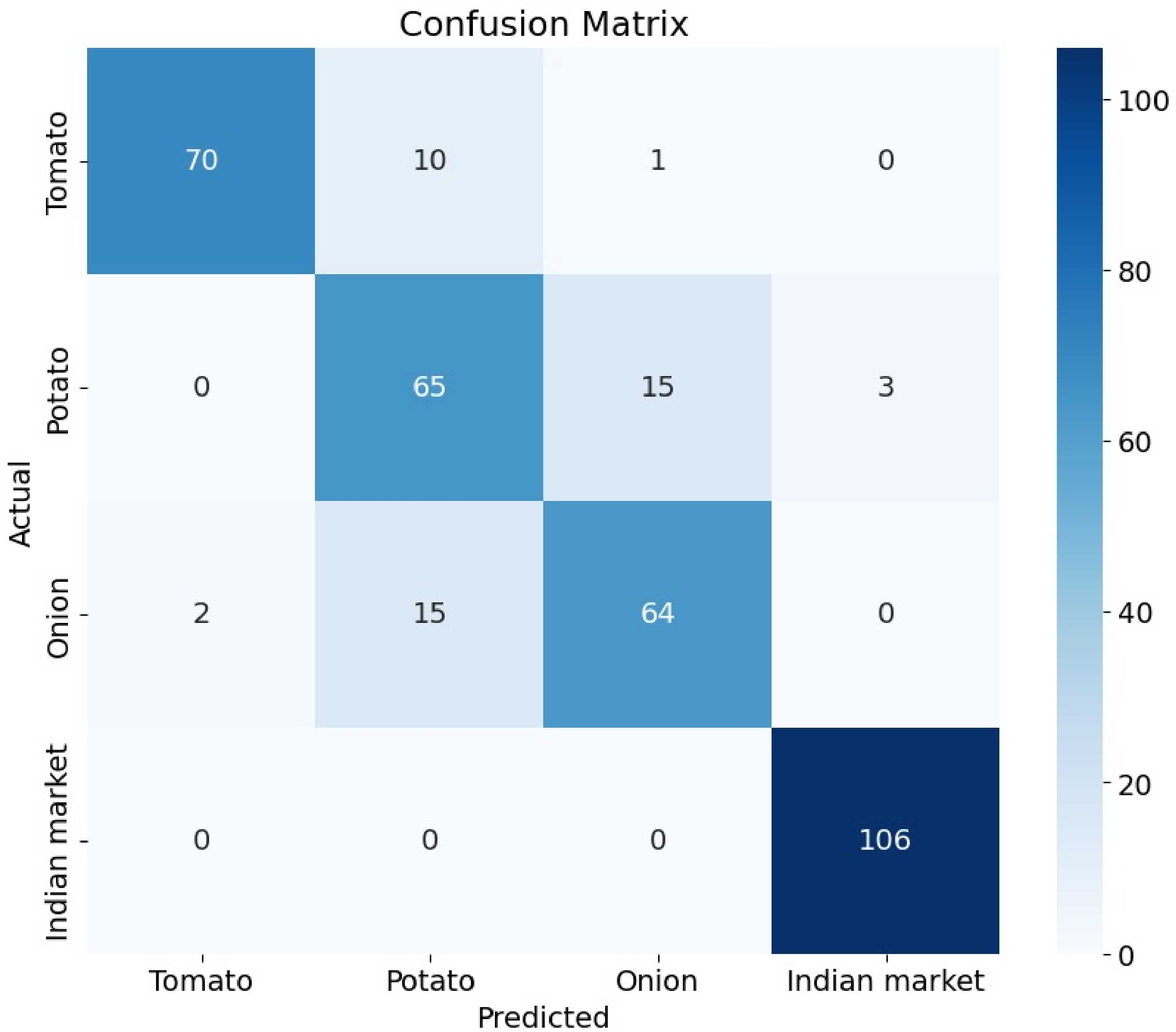

In [53]:
```
# Evaluate the model
loss, acc = vgg19_model.evaluate(test_data, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
11/11 - 5s - 416ms/step - accuracy: 0.8689 - loss: 0.4170
Restored model, accuracy: 86.89%
```

In [54]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate the model on the test set
test_loss, test_acc = vgg19_model.evaluate(test_data, verbose=2)
print(f"Test accuracy: {test_acc}")

# Get predictions from the model
predictions = vgg19_model.predict(test_data)
pred_labels = np.argmax(predictions, axis=1)

# Create a mapping for class names
classes = ['Tomato', 'Potato', 'Onion', 'Indian market']
label_map = {label: idx for idx, label in enumerate(classes)}
reverse_label_map = {v: k for k, v in label_map.items()}

# Extract actual labels from the test dataset
actual_labels = []
for _, label in test_data.unbatch():
    actual_labels.append(label.numpy())

actual_labels = np.array(actual_labels)
actual_labels = np.argmax(actual_labels, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(actual_labels, pred_labels)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```python
# Print classification report
print(classification_report(actual_labels, pred_labels, target_names=classes))
```

```
11/11 - 4s - 378ms/step - accuracy: 0.8689 - loss: 0.4170
Test accuracy: 0.8689458966255188
11/11 ━━━━━━━━━━━━━━━━━━━━ 5s 504ms/step
2024-06-19 11:21:15.861238: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting wi
th status: OUT_OF_RANGE: End of sequence
```




```
               precision    recall  f1-score   support

       Tomato       0.97      0.86      0.92        81
       Potato       0.72      0.78      0.75        83
        Onion       0.80      0.79      0.80        81
Indian market       0.97      1.00      0.99       106

     accuracy                           0.87       351
    macro avg       0.87      0.86      0.86       351
 weighted avg       0.87      0.87      0.87       351
```

### ResNet-50 ( `with` Data Augmentation)

In [59]:
```python
# Load pre-trained VGG19 model
pretrained_model = tf.keras.applications.ResNet152V2(
                                                    weights='imagenet',
                                                    include_top=False,
                                                    input_shape=image_shape
                                                )

pretrained_model.trainable = False

# Create the Sequential model
resnet152v2_model = tf.keras.Sequential([
                                    layers.Input(shape=image_shape),
                                    data_augmentation,
                                    data_preprocess,
                                    pretrained_model,
                                    layers.GlobalAveragePooling2D(),
                                    layers.Dropout(rate=0.1),
                                    layers.Dense(4, activation='softmax')
                                ])
```

```
resnet152v2_model.summary()
```

**Model: "sequential_11"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| data_augmentation (Sequential) | (None, 256, 256, 3) | 0 |
| data_preprocess (Sequential) | (None, 256, 256, 3) | 0 |
| resnet152v2 (Functional) | (None, 8, 8, 2048) | 58,331,648 |
| global_average_pooling2d_11 (GlobalAveragePooling2D) | (None, 2048) | 0 |
| dropout_11 (Dropout) | (None, 2048) | 0 |
| dense_11 (Dense) | (None, 4) | 8,196 |

**Total params:** 58,339,844 (222.55 MB)

**Trainable params:** 8,196 (32.02 KB)

**Non-trainable params:** 58,331,648 (222.52 MB)

In [60]:

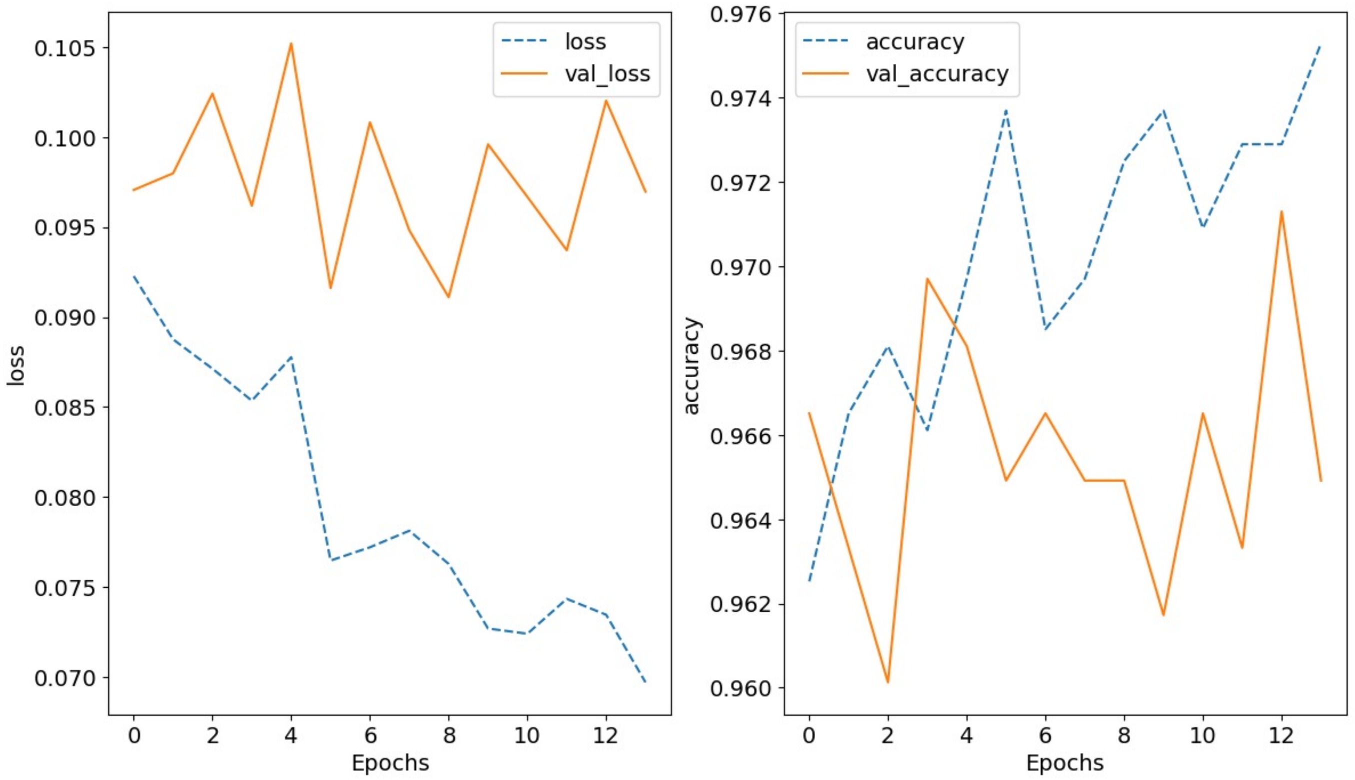
```
log_dir = "../../logs/resnet152v2_model"

tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("resnet152v2_model.keras", save_best_only=True)

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

In [68]:

```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.001)
resnet152v2_model.compile(
                        optimizer=opt,
                        loss = 'categorical_crossentropy',
                        metrics=['accuracy']
                    )

model_fit = resnet152v2_model.fit(
                                train_data,
                                epochs=30,
                                validation_data=val_data,
                                callbacks=[tensorboard_cb,
                                           checkpoint_cb,
                                           early_stopping_cb]
                              )
```

```
Epoch 1/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 92s 949ms/step - accuracy: 0.9690 - loss: 0.0823 - val_accuracy: 0.9665 - val_loss: 0
.0971
Epoch 2/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 604ms/step - accuracy: 0.9665 - loss: 0.0984 - val_accuracy: 0.9633 - val_loss: 0
.0980
Epoch 3/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 597ms/step - accuracy: 0.9696 - loss: 0.0775 - val_accuracy: 0.9601 - val_loss: 0
.1024
Epoch 4/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 627ms/step - accuracy: 0.9694 - loss: 0.0796 - val_accuracy: 0.9697 - val_loss: 0
.0962
Epoch 5/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 613ms/step - accuracy: 0.9747 - loss: 0.0725 - val_accuracy: 0.9681 - val_loss: 0
.1052
Epoch 6/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 615ms/step - accuracy: 0.9728 - loss: 0.0784 - val_accuracy: 0.9649 - val_loss: 0
.0916
Epoch 7/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 604ms/step - accuracy: 0.9692 - loss: 0.0733 - val_accuracy: 0.9665 - val_loss: 0
.1008
Epoch 8/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 614ms/step - accuracy: 0.9674 - loss: 0.0834 - val_accuracy: 0.9649 - val_loss: 0
.0948
Epoch 9/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 640ms/step - accuracy: 0.9763 - loss: 0.0682 - val_accuracy: 0.9649 - val_loss: 0
.0911
Epoch 10/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 606ms/step - accuracy: 0.9731 - loss: 0.0746 - val_accuracy: 0.9617 - val_loss: 0
.0996
Epoch 11/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 619ms/step - accuracy: 0.9721 - loss: 0.0633 - val_accuracy: 0.9665 - val_loss: 0
.0967
Epoch 12/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 633ms/step - accuracy: 0.9739 - loss: 0.0730 - val_accuracy: 0.9633 - val_loss: 0
.0937
Epoch 13/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 621ms/step - accuracy: 0.9754 - loss: 0.0647 - val_accuracy: 0.9713 - val_loss: 0
.1020
Epoch 14/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 643ms/step - accuracy: 0.9745 - loss: 0.0710 - val_accuracy: 0.9649 - val_loss: 0
.0970
```

In [69]:
```
training_plot(['loss', 'accuracy'], model_fit)
```




In [70]:
```
# Evaluate the model
loss, acc = resnet152v2_model.evaluate(test_data, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

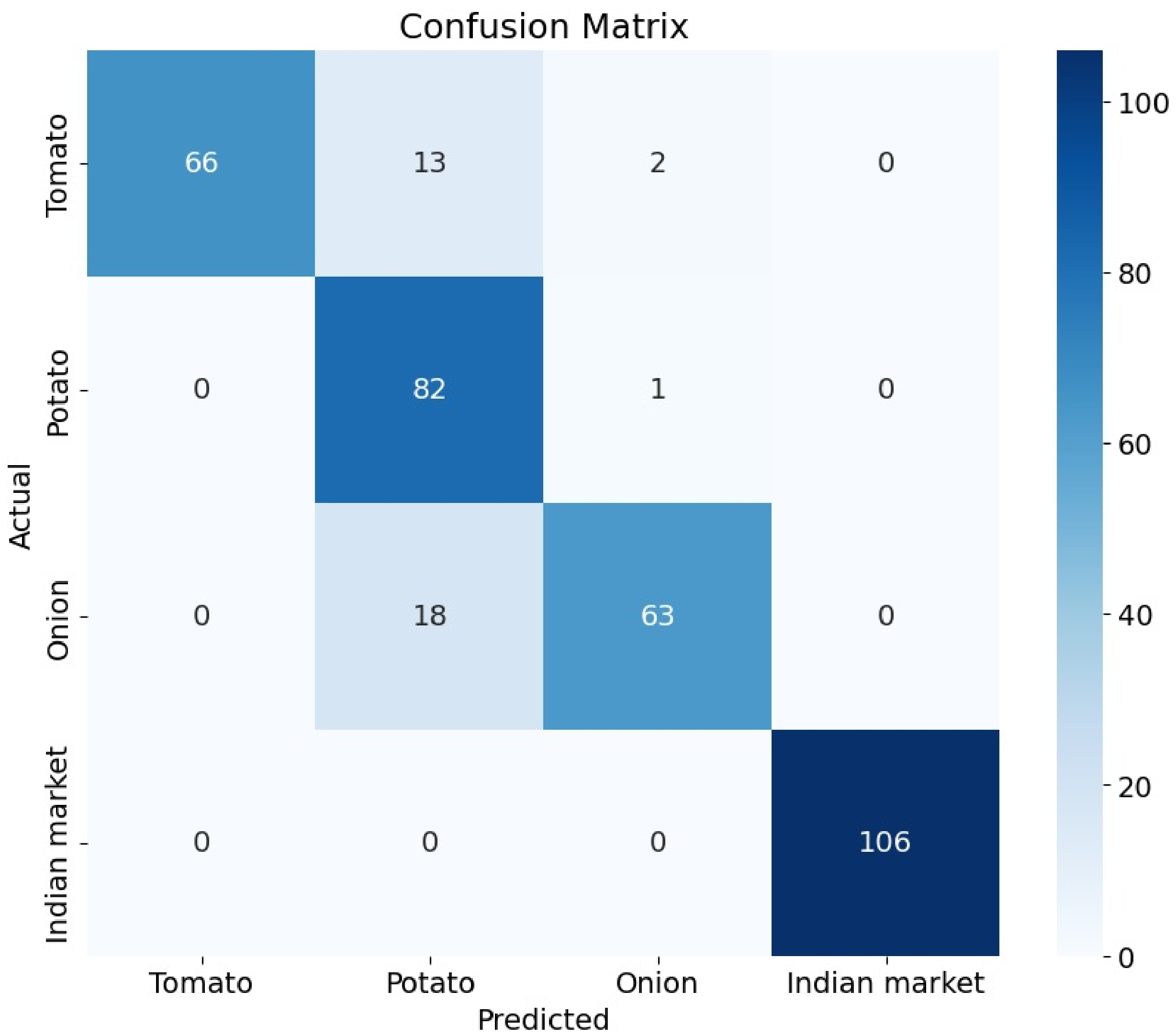
```
11/11 - 11s - 1s/step - accuracy: 0.9031 - loss: 0.3192
Restored model, accuracy: 90.31%
```

In [71]:
```
import numpy as np
```

```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate the model on the test set
test_loss, test_acc = resnet152v2_model.evaluate(test_data, verbose=2)
print(f"Test accuracy: {test_acc}")

# Get predictions from the model
predictions = resnet152v2_model.predict(test_data)
pred_labels = np.argmax(predictions, axis=1)

# Create a mapping for class names
classes = ['Tomato', 'Potato', 'Onion', 'Indian market']
label_map = {label: idx for idx, label in enumerate(classes)}
reverse_label_map = {v: k for k, v in label_map.items()}

# Extract actual labels from the test dataset
actual_labels = []
for _, label in test_data.unbatch():
    actual_labels.append(label.numpy())

actual_labels = np.array(actual_labels)
actual_labels = np.argmax(actual_labels, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(actual_labels, pred_labels)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print(classification_report(actual_labels, pred_labels, target_names=classes))
```

```
11/11 - 5s - 446ms/step - accuracy: 0.9031 - loss: 0.3192
Test accuracy: 0.9031339287757874
11/11 ━━━━━━━━━━━━━━━━━━━━ 42s 3s/step
```

```
2024-06-19 11:54:11.778776: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting wi
th status: OUT_OF_RANGE: End of sequence
```


Confusion Matrix
Tomato
Potato
Onion
Indian market
Actual
Predicted
66
13
2
0
82
1
18
63
106
0
20
40
60
80
100

```
               precision    recall  f1-score   support

       Tomato       1.00      0.81      0.90        81
       Potato       0.73      0.99      0.84        83
        Onion       0.95      0.78      0.86        81
Indian market       1.00      1.00      1.00       106

     accuracy                           0.90       351
    macro avg       0.92      0.90      0.90       351
 weighted avg       0.92      0.90      0.90       351
```

### InceptionResNetV2 (`with` Data Augmentation)

In [90]:
```
# Load pre-trained MobileNetV3Large model
pretrained_model = tf.keras.applications.InceptionResNetV2(
                                                    weights='imagenet',
                                                    include_top=False,
                                                    input_shape=image_shape
                                                )
pretrained_model.trainable = False

# Create the Sequential model
inceptionResNetV2_model = tf.keras.Sequential([
                                    layers.Input(shape=image_shape),
                                    data_augmentation,
                                    data_preprocess,
                                    pretrained_model,
                                    layers.GlobalAveragePooling2D(),
                                    layers.Dropout(rate=0.1),
                                    layers.Dense(4, activation='softmax')
                                ])

inceptionResNetV2_model.summary()
```

**Model: "sequential_15"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| data_augmentation (Sequential) | (None, 256, 256, 3) | 0 |
| data_preprocess (Sequential) | (None, 256, 256, 3) | 0 |
| inception_resnet_v2 (Functional) | (None, 6, 6, 1536) | 54,336,736 |
| global_average_pooling2d_13 (GlobalAveragePooling2D) | (None, 1536) | 0 |
| dropout_13 (Dropout) | (None, 1536) | 0 |
| dense_15 (Dense) | (None, 4) | 6,148 |

**Total params:** 54,342,884 (207.30 MB)

**Trainable params:** 6,148 (24.02 KB)

**Non-trainable params:** 54,336,736 (207.28 MB)

In [91]:
```
log_dir = "../../logs/inceptionResNetV2_model"

tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("inceptionResNetV2_model.keras", save_best_only=True)

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

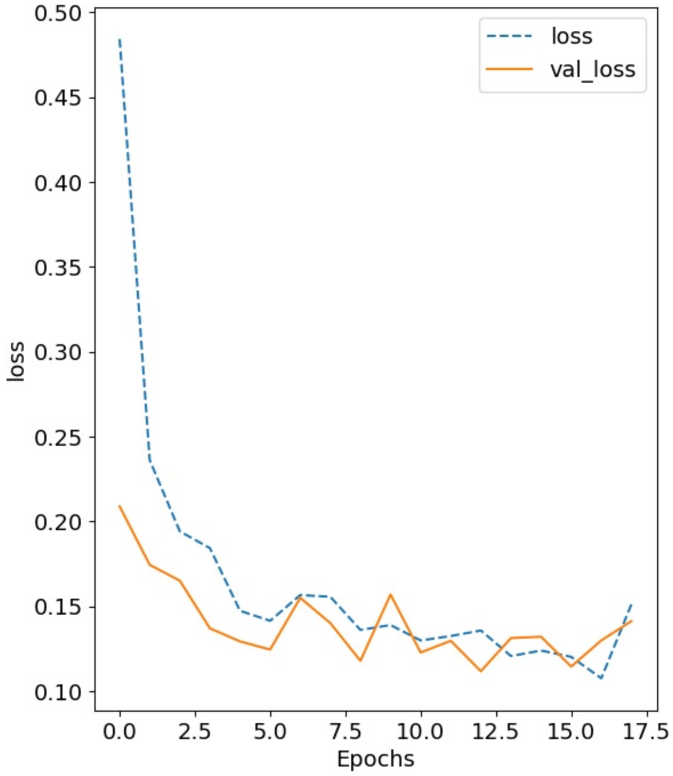
In [92]:
```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.002)
inceptionResNetV2_model.compile(
                                optimizer=opt,
                                loss = 'categorical_crossentropy',
                                metrics=['accuracy']
                            )

model_fit = inceptionResNetV2_model.fit(
                                        train_data,
                                        epochs=30,
```

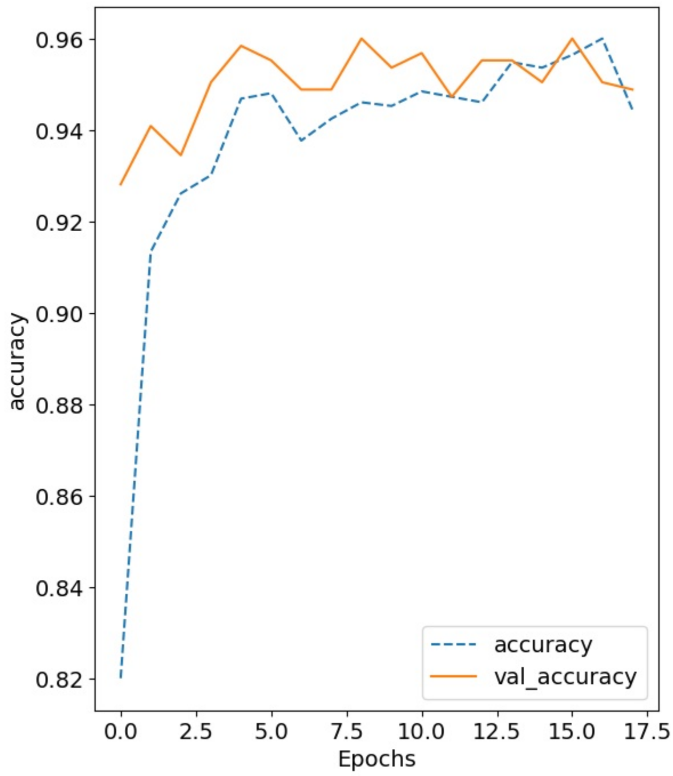
```
                                            validation_data=val_data,
                                            callbacks=[tensorboard_cb,
                                                       checkpoint_cb,
                                                       early_stopping_cb]
                                          )
```

```
Epoch 1/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 160s 2s/step - accuracy: 0.6949 - loss: 0.8091 - val_accuracy: 0.9282 - val_loss: 0.2
089
Epoch 2/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 542ms/step - accuracy: 0.9139 - loss: 0.2327 - val_accuracy: 0.9410 - val_loss: 0
.1745
Epoch 3/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 604ms/step - accuracy: 0.9218 - loss: 0.1955 - val_accuracy: 0.9346 - val_loss: 0
.1652
Epoch 4/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 46s 579ms/step - accuracy: 0.9218 - loss: 0.2022 - val_accuracy: 0.9506 - val_loss: 0
.1371
Epoch 5/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 591ms/step - accuracy: 0.9478 - loss: 0.1467 - val_accuracy: 0.9585 - val_loss: 0
.1294
Epoch 6/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 609ms/step - accuracy: 0.9527 - loss: 0.1353 - val_accuracy: 0.9553 - val_loss: 0
.1246
Epoch 7/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 46s 589ms/step - accuracy: 0.9459 - loss: 0.1372 - val_accuracy: 0.9490 - val_loss: 0
.1550
Epoch 8/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 540ms/step - accuracy: 0.9505 - loss: 0.1379 - val_accuracy: 0.9490 - val_loss: 0
.1402
Epoch 9/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 44s 563ms/step - accuracy: 0.9497 - loss: 0.1289 - val_accuracy: 0.9601 - val_loss: 0
.1179
Epoch 10/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 42s 531ms/step - accuracy: 0.9446 - loss: 0.1350 - val_accuracy: 0.9537 - val_loss: 0
.1570
Epoch 11/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 42s 526ms/step - accuracy: 0.9517 - loss: 0.1240 - val_accuracy: 0.9569 - val_loss: 0
.1229
Epoch 12/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 541ms/step - accuracy: 0.9479 - loss: 0.1262 - val_accuracy: 0.9474 - val_loss: 0
.1297
Epoch 13/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 44s 552ms/step - accuracy: 0.9501 - loss: 0.1288 - val_accuracy: 0.9553 - val_loss: 0
.1118
Epoch 14/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 42s 523ms/step - accuracy: 0.9535 - loss: 0.1261 - val_accuracy: 0.9553 - val_loss: 0
.1314
Epoch 15/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 521ms/step - accuracy: 0.9536 - loss: 0.1310 - val_accuracy: 0.9506 - val_loss: 0
.1321
Epoch 16/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 524ms/step - accuracy: 0.9563 - loss: 0.1177 - val_accuracy: 0.9601 - val_loss: 0
.1146
Epoch 17/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 518ms/step - accuracy: 0.9609 - loss: 0.1067 - val_accuracy: 0.9506 - val_loss: 0
.1300
Epoch 18/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 39s 492ms/step - accuracy: 0.9514 - loss: 0.1360 - val_accuracy: 0.9490 - val_loss: 0
.1413
```

In [93]:

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



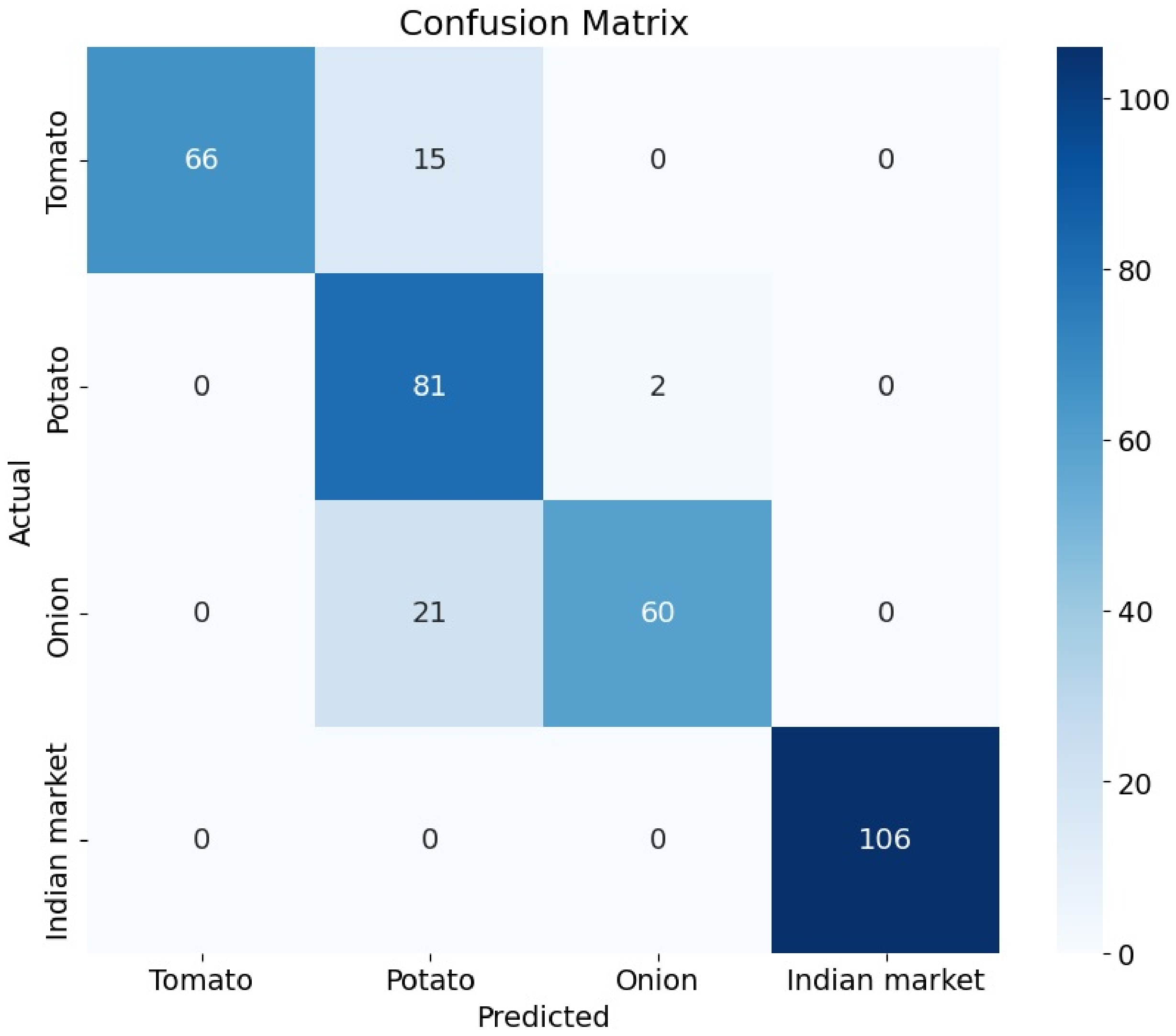




In [96]:
```
# Evaluate the model
loss, acc = inceptionResNetV2_model.evaluate(test_data, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
11/11 - 19s - 2s/step - accuracy: 0.8917 - loss: 0.3820
Restored model, accuracy: 89.17%
```

In [97]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate the model on the test set
test_loss, test_acc = inceptionResNetV2_model.evaluate(test_data, verbose=2)
print(f"Test accuracy: {test_acc}")

# Get predictions from the model
predictions = inceptionResNetV2_model.predict(test_data)
pred_labels = np.argmax(predictions, axis=1)

# Create a mapping for class names
classes = ['Tomato', 'Potato', 'Onion', 'Indian market']
label_map = {label: idx for idx, label in enumerate(classes)}
reverse_label_map = {v: k for k, v in label_map.items()}

# Extract actual labels from the test dataset
actual_labels = []
for _, label in test_data.unbatch():
    actual_labels.append(label.numpy())

actual_labels = np.array(actual_labels)
actual_labels = np.argmax(actual_labels, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(actual_labels, pred_labels)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print(classification_report(actual_labels, pred_labels, target_names=classes))
```

```
11/11 - 4s - 379ms/step - accuracy: 0.8917 - loss: 0.3820
Test accuracy: 0.8917378783226013
11/11 ━━━━━━━━━━━━━━━━━━━━ 66s 4s/step
2024-06-19 12:25:08.569402: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting wi
th status: OUT_OF_RANGE: End of sequence
```




```
              precision    recall  f1-score   support

       Tomato       1.00      0.81      0.90        81
       Potato       0.69      0.98      0.81        83
        Onion       0.97      0.74      0.84        81
Indian market       1.00      1.00      1.00       106

     accuracy                           0.89       351
    macro avg       0.92      0.88      0.89       351
 weighted avg       0.92      0.89      0.89       351
```

## Custom Model

In [108…

```python
from tensorflow.keras import layers, regularizers

def custom_model(height=256, width=256,num_classes=4):
    hidden_size = 256

    model = keras.Sequential(
        name="custom_model",
        layers=[
            layers.Conv2D(filters=16,
                          kernel_size=3,
                          padding="same",
                          input_shape=(height, width, 3),
                            kernel_regularizer=regularizers.l2(1e-3)),
            layers.BatchNormalization(),
            layers.Activation("relu"),
            layers.MaxPooling2D(),
            layers.Conv2D(filters=32, kernel_size=3, padding="same",
                            kernel_regularizer=regularizers.l2(1e-3)),
            layers.BatchNormalization(),
            layers.Activation("relu"),
            layers.MaxPooling2D(),
            layers.Conv2D(filters=64, kernel_size=3, padding="same",
                            kernel_regularizer=regularizers.l2(1e-3)),
            layers.BatchNormalization(),
            layers.Activation("relu"),
```

```python
            layers.MaxPooling2D(),
            layers.Conv2D(filters=128, kernel_size=3, padding="same",
                          kernel_regularizer=regularizers.l2(1e-3)),
            layers.BatchNormalization(),
            layers.Activation("relu"),
            layers.MaxPooling2D(),
            layers.Conv2D(filters=256, kernel_size=3, padding="same",
                          kernel_regularizer=regularizers.l2(1e-3)),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            # layers.MaxPooling2D(),
            # layers.Flatten(),
            layers.GlobalAveragePooling2D(),
            layers.Dense(units=hidden_size, kernel_regularizer=regularizers.l2(1e-3)),
            layers.Activation("relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.5),
            layers.Dense(units=num_classes, activation='softmax')
        ]
    )
    return model
```

In [109…

```python
model = custom_model()
model.summary()
```

**Model: "custom_model"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_609 (Conv2D) | (None, 256, 256, 16) | 448 |
| batch_normalization_609 (BatchNormalization) | (None, 256, 256, 16) | 64 |
| activation_649 (Activation) | (None, 256, 256, 16) | 0 |
| max_pooling2d_18 (MaxPooling2D) | (None, 128, 128, 16) | 0 |
| conv2d_610 (Conv2D) | (None, 128, 128, 32) | 4,640 |
| batch_normalization_610 (BatchNormalization) | (None, 128, 128, 32) | 128 |
| activation_650 (Activation) | (None, 128, 128, 32) | 0 |
| max_pooling2d_19 (MaxPooling2D) | (None, 64, 64, 32) | 0 |
| conv2d_611 (Conv2D) | (None, 64, 64, 64) | 18,496 |
| batch_normalization_611 (BatchNormalization) | (None, 64, 64, 64) | 256 |
| activation_651 (Activation) | (None, 64, 64, 64) | 0 |
| max_pooling2d_20 (MaxPooling2D) | (None, 32, 32, 64) | 0 |
| conv2d_612 (Conv2D) | (None, 32, 32, 128) | 73,856 |
| batch_normalization_612 (BatchNormalization) | (None, 32, 32, 128) | 512 |
| activation_652 (Activation) | (None, 32, 32, 128) | 0 |
| max_pooling2d_21 (MaxPooling2D) | (None, 16, 16, 128) | 0 |
| conv2d_613 (Conv2D) | (None, 16, 16, 256) | 295,168 |
| activation_653 (Activation) | (None, 16, 16, 256) | 0 |
| batch_normalization_613 (BatchNormalization) | (None, 16, 16, 256) | 1,024 |
| global_average_pooling2d_15 (GlobalAveragePooling2D) | (None, 256) | 0 |
| dense_17 (Dense) | (None, 256) | 65,792 |
| activation_654 (Activation) | (None, 256) | 0 |
| batch_normalization_614 (BatchNormalization) | (None, 256) | 1,024 |
| dropout_15 (Dropout) | (None, 256) | 0 |
| dense_18 (Dense) | (None, 4) | 1,028 |

**Total params:** 462,436 (1.76 MB)

**Trainable params:** 460,932 (1.76 MB)

**Non-trainable params:** 1,504 (5.88 KB)

In [112…

```
def compile_train_v2(model, train_ds, val_ds, epochs=10, ckpt_path="../../logs/custom_model.weights.h5"):
    callbacks = [
                    keras.callbacks.ReduceLROnPlateau(
                        monitor="val_loss", factor=0.3, patience=5, min_lr=0.00001
                    ),
                    keras.callbacks.ModelCheckpoint(ckpt_path, save_weights_only=True, monitor='val_accuracy', 
                    keras.callbacks.EarlyStopping(
                        monitor="val_loss", patience=10, min_delta=0.001, mode='min'
                    )
                ]
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model_fit = model.fit(train_ds,
                          validation_data=val_ds,
                          epochs=epochs,
```

```
                          callbacks=callbacks)

    return model_fit
```

In [113…

```
model_fit = compile_train_v2a(model, train_data, val_data, epochs=100)
```

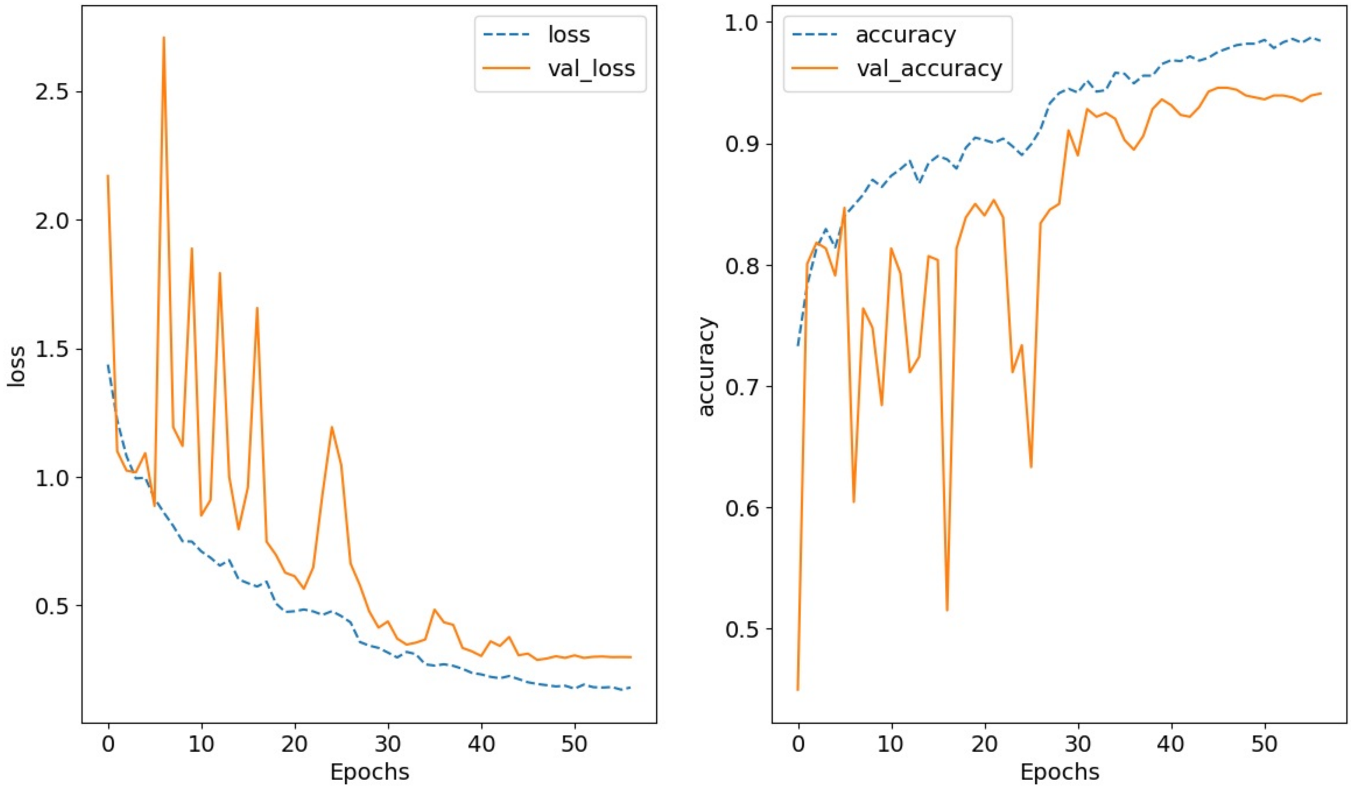
```
Epoch 1/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 60s 445ms/step - accuracy: 0.6771 - loss: 1.6646 - val_accuracy: 0.4498 - val_loss: 2
.1693 - learning_rate: 0.0010
Epoch 2/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 164ms/step - accuracy: 0.7768 - loss: 1.2973 - val_accuracy: 0.8006 - val_loss: 1
.0995 - learning_rate: 0.0010
Epoch 3/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.8086 - loss: 1.0984 - val_accuracy: 0.8182 - val_loss: 1
.0243 - learning_rate: 0.0010
Epoch 4/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.8305 - loss: 0.9929 - val_accuracy: 0.8134 - val_loss: 1
.0181 - learning_rate: 0.0010
Epoch 5/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.8063 - loss: 1.0230 - val_accuracy: 0.7911 - val_loss: 1
.0922 - learning_rate: 0.0010
Epoch 6/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 165ms/step - accuracy: 0.8420 - loss: 0.9190 - val_accuracy: 0.8469 - val_loss: 0
.8868 - learning_rate: 0.0010
Epoch 7/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.8397 - loss: 0.8886 - val_accuracy: 0.6045 - val_loss: 2
.7078 - learning_rate: 0.0010
Epoch 8/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.8563 - loss: 0.8125 - val_accuracy: 0.7640 - val_loss: 1
.1929 - learning_rate: 0.0010
Epoch 9/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 166ms/step - accuracy: 0.8580 - loss: 0.7775 - val_accuracy: 0.7480 - val_loss: 1
.1199 - learning_rate: 0.0010
Epoch 10/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.8485 - loss: 0.7709 - val_accuracy: 0.6842 - val_loss: 1
.8877 - learning_rate: 0.0010
Epoch 11/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.8669 - loss: 0.7200 - val_accuracy: 0.8134 - val_loss: 0
.8495 - learning_rate: 0.0010
Epoch 12/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.8802 - loss: 0.6824 - val_accuracy: 0.7927 - val_loss: 0
.9102 - learning_rate: 0.0010
Epoch 13/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.8898 - loss: 0.6531 - val_accuracy: 0.7113 - val_loss: 1
.7923 - learning_rate: 0.0010
Epoch 14/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 14s 172ms/step - accuracy: 0.8641 - loss: 0.6886 - val_accuracy: 0.7241 - val_loss: 0
.9999 - learning_rate: 0.0010
Epoch 15/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.8679 - loss: 0.6420 - val_accuracy: 0.8070 - val_loss: 0
.7963 - learning_rate: 0.0010
Epoch 16/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.8880 - loss: 0.6030 - val_accuracy: 0.8038 - val_loss: 0
.9591 - learning_rate: 0.0010
Epoch 17/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.8963 - loss: 0.5434 - val_accuracy: 0.5152 - val_loss: 1
.6564 - learning_rate: 0.0010
Epoch 18/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.8655 - loss: 0.6235 - val_accuracy: 0.8134 - val_loss: 0
.7484 - learning_rate: 0.0010
Epoch 19/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.8855 - loss: 0.5127 - val_accuracy: 0.8389 - val_loss: 0
.6972 - learning_rate: 0.0010
Epoch 20/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9019 - loss: 0.4789 - val_accuracy: 0.8501 - val_loss: 0
.6275 - learning_rate: 0.0010
Epoch 21/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 158ms/step - accuracy: 0.8995 - loss: 0.4965 - val_accuracy: 0.8405 - val_loss: 0
.6141 - learning_rate: 0.0010
Epoch 22/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 167ms/step - accuracy: 0.9042 - loss: 0.4760 - val_accuracy: 0.8533 - val_loss: 0
.5651 - learning_rate: 0.0010
Epoch 23/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 165ms/step - accuracy: 0.9042 - loss: 0.4742 - val_accuracy: 0.8389 - val_loss: 0
.6479 - learning_rate: 0.0010
Epoch 24/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.8960 - loss: 0.4764 - val_accuracy: 0.7113 - val_loss: 0
.9345 - learning_rate: 0.0010
Epoch 25/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 158ms/step - accuracy: 0.8827 - loss: 0.4778 - val_accuracy: 0.7337 - val_loss: 1
.1936 - learning_rate: 0.0010
Epoch 26/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.8914 - loss: 0.4725 - val_accuracy: 0.6332 - val_loss: 1
```

```
.0465 - learning_rate: 0.0010
Epoch 27/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.9019 - loss: 0.4466 - val_accuracy: 0.8341 - val_loss: 0
.6633 - learning_rate: 0.0010
Epoch 28/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.9192 - loss: 0.3814 - val_accuracy: 0.8453 - val_loss: 0
.5790 - learning_rate: 3.0000e-04
Epoch 29/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9366 - loss: 0.3519 - val_accuracy: 0.8501 - val_loss: 0
.4772 - learning_rate: 3.0000e-04
Epoch 30/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 158ms/step - accuracy: 0.9414 - loss: 0.3364 - val_accuracy: 0.9107 - val_loss: 0
.4140 - learning_rate: 3.0000e-04
Epoch 31/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.9349 - loss: 0.3255 - val_accuracy: 0.8900 - val_loss: 0
.4382 - learning_rate: 3.0000e-04
Epoch 32/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 14s 176ms/step - accuracy: 0.9565 - loss: 0.2939 - val_accuracy: 0.9282 - val_loss: 0
.3713 - learning_rate: 3.0000e-04
Epoch 33/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9408 - loss: 0.3272 - val_accuracy: 0.9219 - val_loss: 0
.3482 - learning_rate: 3.0000e-04
Epoch 34/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9446 - loss: 0.3141 - val_accuracy: 0.9250 - val_loss: 0
.3556 - learning_rate: 3.0000e-04
Epoch 35/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.9574 - loss: 0.2771 - val_accuracy: 0.9203 - val_loss: 0
.3681 - learning_rate: 3.0000e-04
Epoch 36/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.9470 - loss: 0.2831 - val_accuracy: 0.9027 - val_loss: 0
.4839 - learning_rate: 3.0000e-04
Epoch 37/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.9474 - loss: 0.2699 - val_accuracy: 0.8947 - val_loss: 0
.4349 - learning_rate: 3.0000e-04
Epoch 38/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.9477 - loss: 0.2863 - val_accuracy: 0.9059 - val_loss: 0
.4248 - learning_rate: 3.0000e-04
Epoch 39/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.9422 - loss: 0.2831 - val_accuracy: 0.9282 - val_loss: 0
.3350 - learning_rate: 9.0000e-05
Epoch 40/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.9597 - loss: 0.2426 - val_accuracy: 0.9362 - val_loss: 0
.3220 - learning_rate: 9.0000e-05
Epoch 41/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9668 - loss: 0.2424 - val_accuracy: 0.9314 - val_loss: 0
.3038 - learning_rate: 9.0000e-05
Epoch 42/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 158ms/step - accuracy: 0.9691 - loss: 0.2219 - val_accuracy: 0.9234 - val_loss: 0
.3610 - learning_rate: 9.0000e-05
Epoch 43/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.9740 - loss: 0.2106 - val_accuracy: 0.9219 - val_loss: 0
.3429 - learning_rate: 9.0000e-05
Epoch 44/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.9671 - loss: 0.2263 - val_accuracy: 0.9298 - val_loss: 0
.3779 - learning_rate: 9.0000e-05
Epoch 45/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 158ms/step - accuracy: 0.9737 - loss: 0.2154 - val_accuracy: 0.9426 - val_loss: 0
.3066 - learning_rate: 9.0000e-05
Epoch 46/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.9776 - loss: 0.2040 - val_accuracy: 0.9458 - val_loss: 0
.3132 - learning_rate: 9.0000e-05
Epoch 47/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.9698 - loss: 0.2063 - val_accuracy: 0.9458 - val_loss: 0
.2886 - learning_rate: 2.7000e-05
Epoch 48/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9799 - loss: 0.1944 - val_accuracy: 0.9442 - val_loss: 0
.2939 - learning_rate: 2.7000e-05
Epoch 49/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.9786 - loss: 0.1894 - val_accuracy: 0.9394 - val_loss: 0
.3029 - learning_rate: 2.7000e-05
Epoch 50/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 157ms/step - accuracy: 0.9804 - loss: 0.1881 - val_accuracy: 0.9378 - val_loss: 0
.2965 - learning_rate: 2.7000e-05
Epoch 51/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 12s 156ms/step - accuracy: 0.9841 - loss: 0.1744 - val_accuracy: 0.9362 - val_loss: 0
.3063 - learning_rate: 2.7000e-05
Epoch 52/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 163ms/step - accuracy: 0.9788 - loss: 0.1884 - val_accuracy: 0.9394 - val_loss: 0
.2965 - learning_rate: 2.7000e-05
Epoch 53/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.9805 - loss: 0.1842 - val_accuracy: 0.9394 - val_loss: 0
.3008 - learning_rate: 1.0000e-05
Epoch 54/100
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.9802 - loss: 0.1874 - val_accuracy: 0.9378 - val_loss: 0
.3019 - learning_rate: 1.0000e-05
Epoch 55/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 159ms/step - accuracy: 0.9886 - loss: 0.1748 - val_accuracy: 0.9346 - val_loss: 0
.2995 - learning_rate: 1.0000e-05
Epoch 56/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 160ms/step - accuracy: 0.9858 - loss: 0.1747 - val_accuracy: 0.9394 - val_loss: 0
.2998 - learning_rate: 1.0000e-05
Epoch 57/100
79/79 ━━━━━━━━━━━━━━━━━━━━ 13s 169ms/step - accuracy: 0.9798 - loss: 0.1890 - val_accuracy: 0.9410 - val_loss: 0
.2994 - learning_rate: 1.0000e-05
```

In [114…

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




In [118…

```
%load_ext tensorboard
%tensorboard --logdir ../../logs/
```

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

# Connection refused

Failed to load URL https://html2pdf.com:6007/.

QtNetwork Error 1

In [ ]: