**#To create an algorithm that can identify objects in an image**

import tensorflow as tf

from tensorflow.keras.applications.vgg16 import VGG16

from tensorflow.keras.applications.vgg16 import preprocess\_input

from tensorflow.keras.preprocessing import image

import numpy as np

import pandas as pd

# load pre-trained VGG16 model

model = VGG16(weights='imagenet')

# load pre-existing dataset of item descriptions

item\_data = pd.read\_csv('item\_descriptions.csv')

# define function to predict object in image and additional item information

def predict\_item(image\_path, brand=None, outlet=None):

# load image and resize

img = image.load\_img(image\_path, target\_size=(224, 224))

# convert image to array

x = image.img\_to\_array(img)

# preprocess input image

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)

# make prediction using VGG16 model

predictions = model.predict(x)

# decode prediction results

decoded\_predictions = tf.keras.applications.vgg16.decode\_predictions(predictions, top=1)[0]

# get predicted object label

predicted\_object = decoded\_predictions[0][1]

# search for additional item information in dataset

item\_info = item\_data[item\_data['item'] == predicted\_object]

# filter by brand and outlet if specified

if brand:

item\_info = item\_info[item\_info['brand'] == brand]

if outlet:

item\_info = item\_info[item\_info['outlet'] == outlet]

# return filtered item information

return item\_info

# example usage

image\_path = 'example\_image.jpg'

brand = 'Zara'

outlet = 'Online'

predicted\_item = predict\_item(image\_path, brand=brand, outlet=outlet)

print("Predicted item:", predicted\_item)

# **We can use web scraping libraries in Python such as BeautifulSoup and Selenium to build an algorithm that identifies online stores based on brand information and scrapes pricing information.**

import requests

from bs4 import BeautifulSoup

url = 'https://www.examplestore.com/nike-running-shoe'

# Send a GET request to the URL

response = requests.get(url)

# Parse the HTML content using BeautifulSoup

soup = BeautifulSoup(response.content, 'html.parser')

# Find the HTML element that contains the price

price\_element = soup.find('span', {'class': 'price'})

# Extract the price from the text of the element

price\_text = price\_element.text.strip()

price = float(price\_text.replace('$', ''))

**Explanation**

1) We first load the training and validation data using the ImageDataGenerator class, which automatically resizes and augments the images as needed. We then define the CNN architecture using the Sequential class, which allows us to stack layers on top of each other. The layers in this model include convolutional layers with ReLU activation, max pooling layers, a flatten layer, and two dense layers with ReLU and softmax activation, respectively.

We compile the model using the compile method, which specifies the optimizer, loss function, and evaluation metrics. We then train the model using the fit method, which takes the training and validation data as input and runs the specified number of epochs.

Finally, we save the trained model to a file using the save method, which allows us to later load the model and use it for prediction on new images.

2) We load the pre-trained VGG-16 model using the VGG16 class from Keras. We then freeze the weights of the pre-trained layers by setting the trainable attribute to False.

Next, we add new layers on top of the pre-trained model for fine-tuning. In this example, we add a flatten layer, a dense layer with 1024 units, and a final dense layer with softmax activation for classification.

Finally, we create the new model using the Model class from Keras, which takes the inputs and outputs of the model as arguments. We then compile the model as before and can train it on our dataset using the fit method.

3) We define two input layers, one for the image input and one for the brand input. We then load the pre-trained VGG-16 model and apply it to the image input to extract features.

We define a dense layer for the brand input, then concatenate the image and brand features using the concatenate layer. We apply another dense layer to the concatenated features and add the output layer for classification.

When we train this model on our dataset, we can pass both the image and brand information as inputs using a list.

4) To build an algorithm that identifies online stores based on brand information and scrapes pricing information, we can use web scraping libraries in Python such as BeautifulSoup and Selenium.

First, we need to collect a list of online stores that sell the item based on the brand information provided by the user. We can do this by using a search engine like Google and using keywords related to the brand and item. For example, if the item is a Nike running shoe, we can search for "Nike running shoe online stores" or "buy Nike running shoe". We can then scrape the search engine results to get a list of online stores that sell the item.

Next, we can use web scraping to extract pricing information from each online store. We can use libraries like BeautifulSoup to extract the relevant HTML elements that contain the price information, and then parse the text to extract the price.

Top of Form