Beans

LAU

Deep Learning

Larissa Abboud 202001686

Mourad Rahi 202002464

April 30

Abstract

This report presents the development, implementation, and evaluation of a convolutional neural network (CNN) model for image classification using the "Beans" dataset. The dataset comprises images of beans captured in the field, annotated into three classes: Angular Leaf Spot, Bean Rust, and healthy beans. The study aimed to accurately classify these images using deep learning techniques. The methodology involved preprocessing the dataset, constructing 2 CNN models where one is based on a standard CNN architecture and the other is based on the VGG16 architecture . training the model, and evaluating its performance. The report highlights the achieved test accuracy of 82.03% after training the model for 100 epochs for the first cnn and test accuracy of 84% after training the model for 10/10 epochs for the second cnn and ~82% for 17/50 and 5/50 epochs .

Introduction

Image classification is crucial in various domains, including agriculture, where it assists in the early detection and management of plant diseases. The "Beans" dataset offers valuable imagery of beans afflicted with Angular Leaf Spot and Bean Rust diseases, alongside healthy beans. Leveraging deep learning methods, this study aimed to develop a CNN model capable of effectively identifying these diseases from field-captured images.

Dataset

The "Beans" dataset contains images of beans taken in the field using smartphone cameras, annotated by experts from the National Crops Resources Research Institute (NaCRRI) in Uganda. These images are divided into three classes: Angular Leaf Spot, Bean Rust, and healthy beans, making it suitable for training and evaluating machine learning models for disease detection in beans.

Methodology

Building a Convolutional Neural Network (CNN) involves several key steps, from data preprocessing to model evaluation.

We designed 2 types of model to analyse the different results in each type. The first model consisted of a normal cnn architecture and another using the vgg16 base model architecture. VGG-16 is a convolutional neural network that is 16 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database . The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. For each model, different hyperparameters were chosen accordingly.

Data Collection was the first step we took to begin building our model. First we Gathered the beans dataset which consists of images of beans captured in the field .This dataset is suitable for our topic since we need to build a model that should be able to distinguish between these 3 classes with high accuracy. The data was imported from tensorflow as :

```
import tensorflow_datasets as tfds
```

And we loaded the dataset using .load function to separate the train and test datasets present:

```
dataset_name = "beans"
(train_ds, test_ds), ds_info = tfds.load(
    dataset_name,
    split=["train", "test"],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,)
```

From this we were able to obtain the following results:

```
Number of samples in train dataset: 1034
Number of samples in test dataset: 128
Number of samples in validation dataset: 133
```

Data Preprocessing was the next step to build our model. We began by preprocessing the images by A preprocessing function preprocess_image() is defined to resize each image to the input size expected by the VGG16 model (224x224 pixels) and normalise pixel values to a range between 0 and 1. This function is then applied to both the training and testing datasets using the map() function. This is also applied to the normal cnn model as well for (32x32 pixels). Then we proceeded with batching and prefetching the data. The data is first batched into batches of size 32 using the batch() function to fit the VGG16 model's input requirements. Prefetching is applied to optimise performance using TensorFlow's prefetch() function with the AUTOTUNE argument. The batch size was reduced from 128 from the normal model to 32 for the vgg16.

Our CNN Model building and Model Construction continued with designing the Model Architecture for both types.First we began by determining the input shape of the dataset using this:

For the first CNN architecture used we had the following :

```
for image_batch, label_batch in train_ds.take(1):
    input_shape = image_batch.shape[1:]  # Extract input shape from the
image batch
    print("Input shape:", input_shape)
Input shape: (500, 3)
```

Second we began building the first model architecture which consisted of:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_4 (Sequential)   (None, 32, 32, 3)         0

 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_4 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

 dropout_6 (Dropout)         (None, 16, 16, 32)        0

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_5 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_7 (Dropout)         (None, 8, 8, 64)          0

 flatten_2 (Flatten)         (None, 4096)              0

 dense_4 (Dense)             (None, 512)               2097664

 dropout_8 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 2168362 (8.27 MB)
Trainable params: 2168362 (8.27 MB)
Non-trainable params: 0 (0.00 Byte)
```

And the model for the VGG16 architect is:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 2)	8194

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_3 (Flatten)         (None, 25088)             0

 dense_6 (Dense)             (None, 512)               12845568
```

```
 dropout_9 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 10)                5130

=================================================================
Total params: 27565386 (105.15 MB)
Trainable params: 12850698 (49.02 MB)
Non-trainable params: 14714688 (56.13 MB)
```

In summary our Model Construction consisted of convolutional layers to extract features from input images,pooling layers to downsample feature maps and reduce computational complexity,the use of activation functions such as ReLU or softmax to introduce non-linearity,inclusion of dropout layers to prevent overfitting by randomly dropping units during training,additional convolutional and pooling layers to create deeper representations, Flatten the final feature maps into a 1D vector to feed into fully connected layers,fully connected (dense) layers to perform classification/regression tasks.

Before compiling, training and evaluation the model, the vgg16 architecture required to laod the base model and freeze the pretrained layer.The VGG16 model is loaded with pre-trained weights from the ImageNet dataset using the VGG16() function.The include_top=False argument excludes the fully connected layers at the top of the network, leaving only the convolutional base. input_shape=(224, 224, 3) specifies the input image shape expected by the VGG16 model (224x224 pixels with 3 colour channels for RGB images). Additionally , The pre-trained layers of the VGG16 model are frozen by setting the trainable attribute of each layer to False. This prevents their weights from being updated during training. Freezing the pre-trained layers is a common practice when using transfer learning to retain the learned features and prevent overfitting on the new dataset.

Finally the model is compiled , trained and evaluated using the following common parameters:

- Optimizer : Adam(learning_rate=0.001)
- loss function: which is based on the task which is classification where we used categorical cross-entropy.
- Metrics : accuracy

The training was done on the first model with: 100 epochs, 128 batch size. While for the vgg16 architectural model: 17/ 50 ,5/50 and 10/10 epochs , 32 batch size.

Results

For the first experiment done on the normal cnn model we obtained :
100 epochs, 128 batch:

```
Test Loss: 0.49476876854896545
Test Accuracy: 0.8203125
```

While for the vgg16 architecture model we obtained:
10 epoche , 32 batch:

```
Test Loss: 0.44001439213752747
Test Accuracy: 0.84375
```

5/50 epochs , 32 batch:

```
Test Loss: 0.39794132113456726
Test Accuracy: 0.8203125
```

17/50 epochs , 32 batch:

```
Test Loss: 0.3728514015674591
Test Accuracy: 0.828125
```

These results were obtained after initially obtaining a constant val_accuracy of 0.3359 with default batch size and learning rate. However after testing out different hyper parameter options and adding data augmentation for the 1st cnn model and early stopping for thevgg16 model where were able to receive better results.

Data augmentation is the application of differnt types of image transformation to increase the size and diversity of the training set.Early stopping is a form of regularisation that stops the training process of a neural network before it reaches the maximum number of epochs or iterations.And both were used in the complex models to obtain these results.

Reference

Convolutional Neural Network (CNN) | TensorFlow Core. (2019). Retrieved from TensorFlow website: https://www.tensorflow.org/tutorials/images/cnn

Giri, S. (2019). Image based flower species classification using CNN. *Journal of Innovations in Engineering Education*, *2*(1), 182–186. https://doi.org/10.3126/jiee.v2i1.36670

Haupt, J., Kahl, S., Kowerko, D., & Eibl, M. (2018). Large-Scale Plant Classification using Deep Convolutional Neural Networks. *CLEF (Working Notes)*.

ibean. (2023, May 19). ibean. Retrieved from GitHub website: https://github.com/AI-Lab-Makerere/ibean/

Sanad, M. (2020, February 18). Image Classification Using CNN (Convolutional Neural Networks). Retrieved from Analytics Vidhya website: https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/#h-full-code-for-the-cnn-model