AERIAL CACTUS IDENTIFICATION

BY ANDRÉS RUBIO RAMOS (rubioand) NIE-MVI

DESCRIPTION

This is a Kaggle competition of machine learning. The task description is as follows:

To assess the impact of climate change on Earth's flora and fauna, it is vital to quantify how human activities such as logging, mining, and agriculture are impacting our protected natural areas. Researchers in Mexico have created the VIGIA project, which aims to build a system for autonomous surveillance of protected areas. A first step in such an effort is the ability to recognize the vegetation inside the protected areas. In this competition, you are tasked with creation of an algorithm that can distinguish cactus in aerial imagery.

ABOUT DATASET

Nowadays, climate change is affecting the life on Earth. In our VIGIA project we want to build a system for autonomous surveillance of protected areas. A first step is to recognize the vegetation inside the protected areas.

In this dataset, they are presenting more than 16,000 examples of a columnar cacti for plant recognition or classification.

More details about this dataset can be found at their research paper https://doi.org/10.1016/j.ecoinf.2019.05.005

More info here: Will Cukierski. (2019). Aerial Cactus Identification. Kaggle.

https://kaggle.com/competitions/aerial-cactus-identification

WHY THIS PROJECT ?

I have chosen a machine learning project focused on identifying cacti from the air. Interested in neural networks, I decided to investigate convolutional neural networks (CNN) and apply them to this task. I think the complex patterns in aerial imagery present an exciting challenge for CNN. Additionally, the visual nature of the project aligns perfectly with my interest for image-based machine learning. The Aerial Cactus dataset provides a rich set of diverse images, making it an ideal playing field for experimenting with different CNN architectures. Additionally, this project allows me to delve deeper into the nuances of image classification and feature extraction. I'm interested to see how well CNN is able to detect the nuances and complexities of these aerial images.

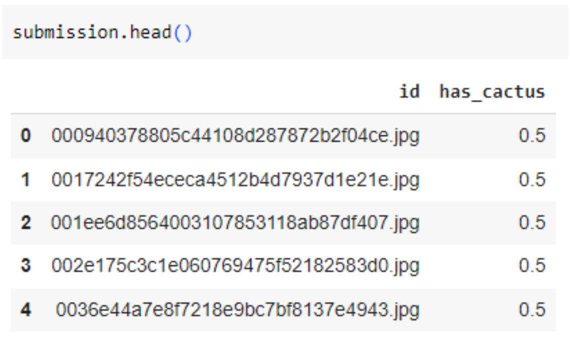
EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) serves as the foundational step in understanding the characteristics and patterns within our aerial cactus dataset.

Through EDA, we aim to gain insights into the distribution, relationships, and potential anomalies present in the data before diving into the convolutional neural network (CNN) model development.

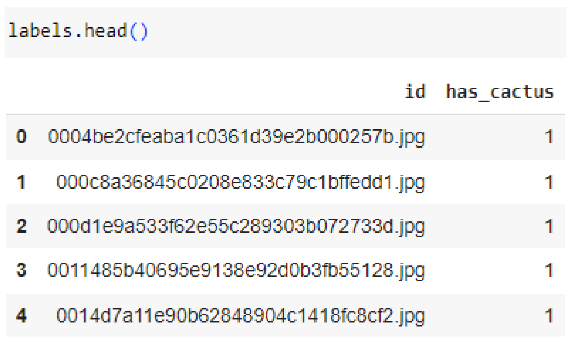
By visualizing and summarizing key statistics of the dataset during EDA, we can make informed decisions regarding preprocessing steps and feature engineering for optimal model performance.

Load the data



```
submission.head()
```

	id	has_cactus
0	000940378805c44108d287872b2f04ce.jpg	0.5
1	0017242f54ececa4512b4d7937d1e21e.jpg	0.5
2	001ee6d8564003107853118ab87df407.jpg	0.5
3	002e175c3c1e060769475f52182583d0.jpg	0.5
4	0036e44a7e8f7218e9bc7bf8137e4943.jpg	0.5



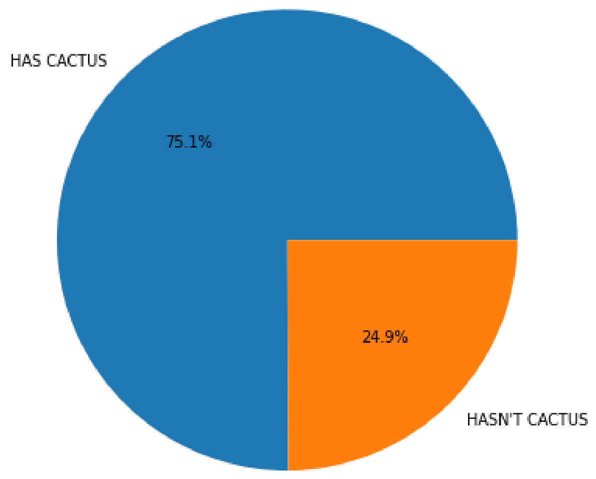
```
labels.head()
```

	id	has_cactus
0	0004be2cfeaba1c0361d39e2b000257b.jpg	1
1	000c8a36845c0208e833c79c1bffedd1.jpg	1
2	000d1e9a533f62e55c289303b072733d.jpg	1
3	0011485b40695e9138e92d0b3fb55128.jpg	1
4	0014d7a11e90b62848904c1418fc8cf2.jpg	1

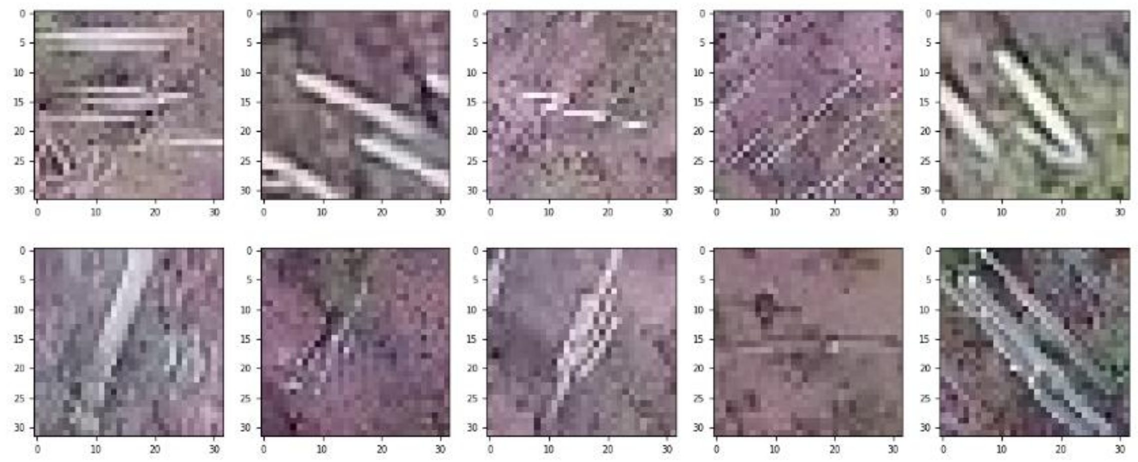
The id is the image filename of the test or training data. has_cactus is the target value. It indicates whether an image with that filename contains a cactus. A value of 0 means it doesn't have a cactus, and a value of 1 means it does.

Data Visualization

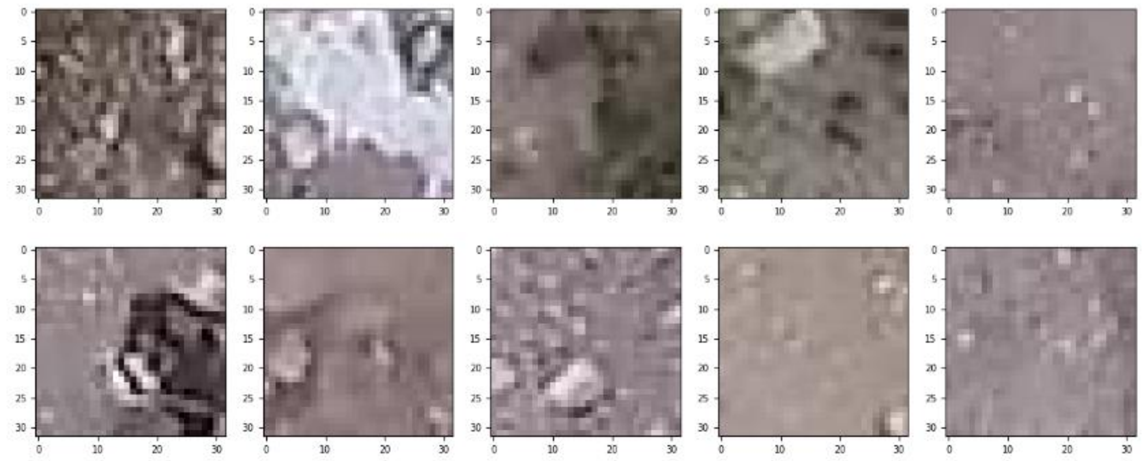
With this circular graph of labels data we can see that there are three times as many photos with cactus compared to those without cactus. If we consider a deterministic model that always answered that if the photo has cactus, it would have an accuracy of 75.1% if we assume that the training data reflects very well the proportion of the data that we would find in real practice.



The amount of train images is 17500 and the amount of test images is 4000. Having 17,500 training images and 4,000 test images is considered a substantial dataset for many machine learning tasks, especially for image classification tasks using convolutional neural networks (CNNs).



These are 10 images from the labels file (train.csv) in which it says that there are cactus, which we can see more or less in all the photos.



These are 10 images from the labels file (train.csv) in which it says that there is no presence of cactus. They appear to be lands of earth and stone, some with a little Grass.

Upon inspection, we observe that the image dimensions are 32 pixels in width and 32 pixels in height, encompassing three distinct color channels. This tri-channel structure indicates that the image is in color format, specifically representing the red, green, and blue color spectrums.

Approach to building the model

-The image files are low resolution color images.

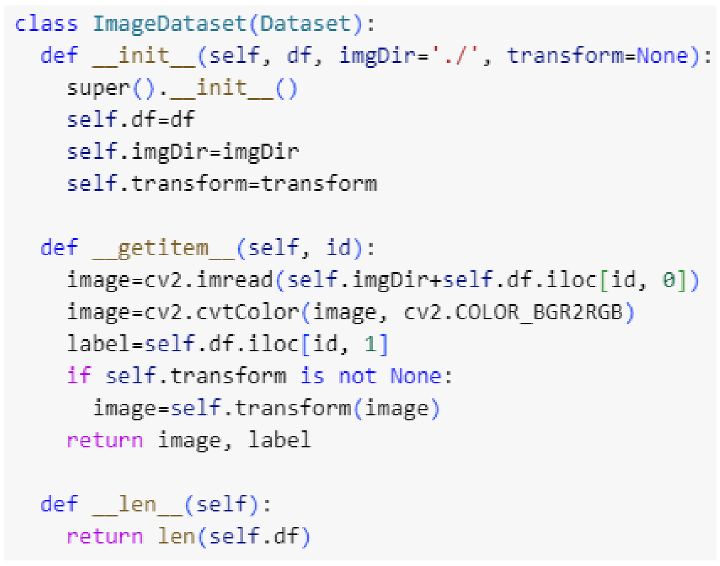
-The ID of the CSV file is the image filename. You can get the location of the file by simply adding the file's pathname.

With these characteristics and seeing the problem to be solved, I think it would be very good to try to use a CNN to solve it. Given the characteristics of the images (low resolution, color) and the inherent advantages of CNNs in image processing and analysis, it is a good decision to use a CNN to develop a cactus classifier. The ability of CNN to efficiently capture and utilize relevant features in low-resolution, low-color images provides a good solution to address this specific image classification problem.

MODELING

First I will separe the train data in train and validation data. I have decided that validation data is 10 percent of the total. So the number of train data will be 15750 and the number of validation data will be 1750.

This ImageDataset class provides a structured way to access and transform images and their corresponding labels, making it compatible with PyTorch's DataLoader for efficient training and validation of machine learning models.



```python
class ImageDataset(Dataset):
  def __init__(self, df, imgDir='./', transform=None):
    super().__init__()
    self.df=df
    self.imgDir=imgDir
    self.transform=transform

  def __getitem__(self, id):
    image=cv2.imread(self.imgDir+self.df.iloc[id, 0])
    image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    label=self.df.iloc[id, 1]
    if self.transform is not None:
      image=self.transform(image)
    return image, label

  def __len__(self):
    return len(self.df)
```

```
trainTransform= transforms.Compose([transforms.ToTensor(),
                                      transforms.Pad(32, padding_mode='symmetric'),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomVerticalFlip(),
                                      transforms.RandomRotation(10),
                                      transforms.Normalize((0.485, 0.456, 0.406),
                                                          (0.229, 0.224, 0.225))])

testTransform= transforms.Compose([transforms.ToTensor(),
                                      transforms.Pad(32, padding_mode='symmetric'),
                                      transforms.Normalize((0.485, 0.456, 0.406),
                                                          (0.229, 0.224, 0.225))])
```

These transformations are applied to increase the diversity and robustness of the data set during training. Transformations such as padding, flipping, and rotating introduce variability into images, while normalization ensures that pixel values are on a scale appropriate for model training. These data augmentation techniques are essential to improve the model's ability to generalize and improve its performance on unseen cactus images.

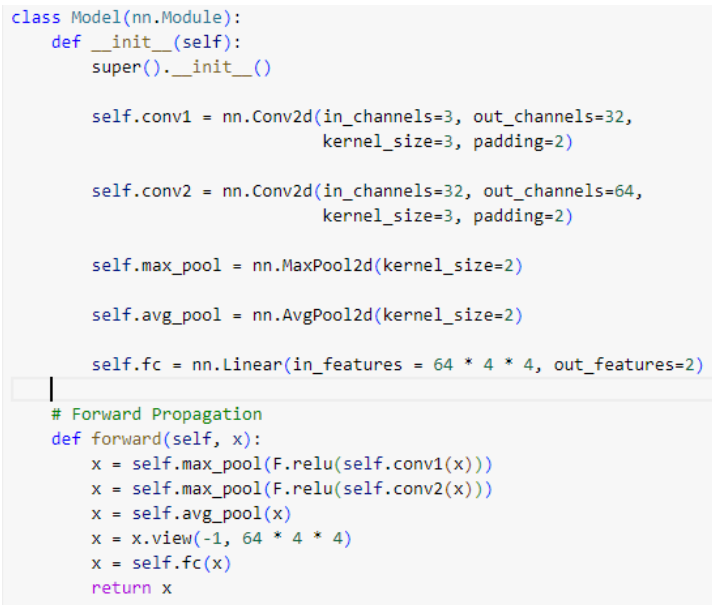
```
trainDataset=ImageDataset(trainData, 'train/', trainTransform)
validationDataset=ImageDataset(validData, 'train/', testTransform)
```

```
from torch.utils.data import DataLoader

trainLoader= DataLoader(dataset=trainDataset, batch_size=32, shuffle=True)
validLoader= DataLoader(dataset=validationDataset, batch_size=32, shuffle=False)
```

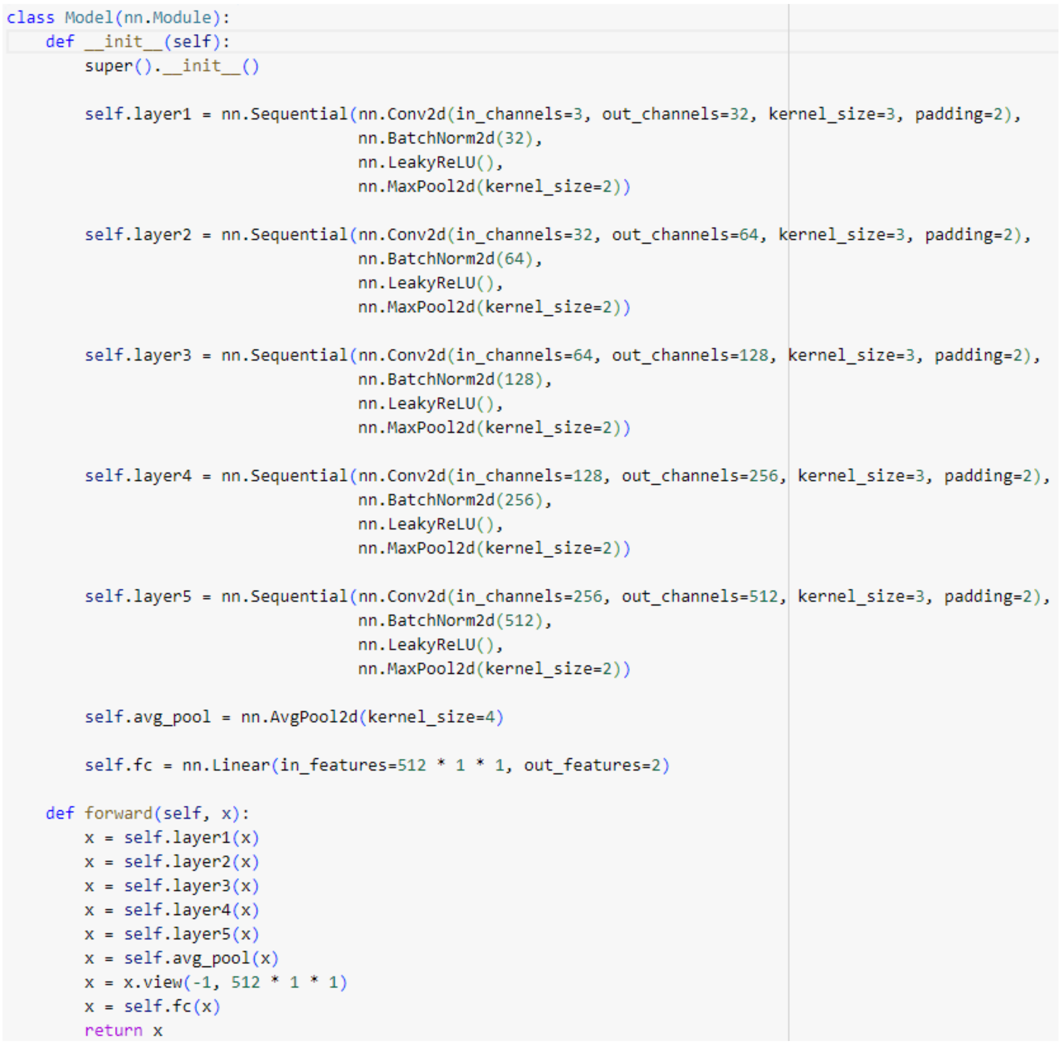
This code sets up the training and validation data loaders, which will be used to efficiently feed batches of data to the model during the training and validation phases of your machine learning pipeline.

CONVOLUTIONAL NEURAL NETWORK MODEL

First I based myself on the example of a practical class that we did in the subject to make my first attempt. But the error was too large for validation, about 10 percent. So I started looking for improvements to make on different blogs and discovered the Sequential method that improved the results considerably.



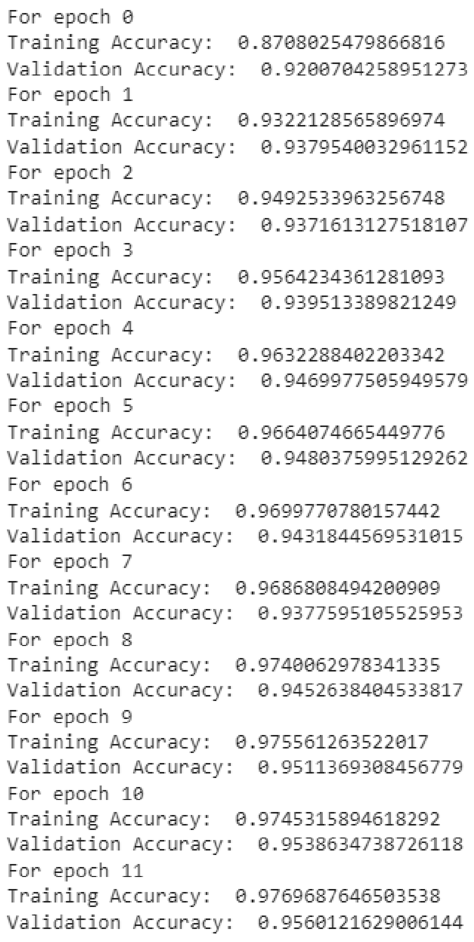
```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
                               kernel_size=3, padding=2)

        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
                               kernel_size=3, padding=2)

        self.max_pool = nn.MaxPool2d(kernel_size=2)

        self.avg_pool = nn.AvgPool2d(kernel_size=2)

        self.fc = nn.Linear(in_features = 64 * 4 * 4, out_features=2)

    # Forward Propagation
    def forward(self, x):
        x = self.max_pool(F.relu(self.conv1(x)))
        x = self.max_pool(F.relu(self.conv2(x)))
        x = self.avg_pool(x)
        x = x.view(-1, 64 * 4 * 4)
        x = self.fc(x)
        return x
```



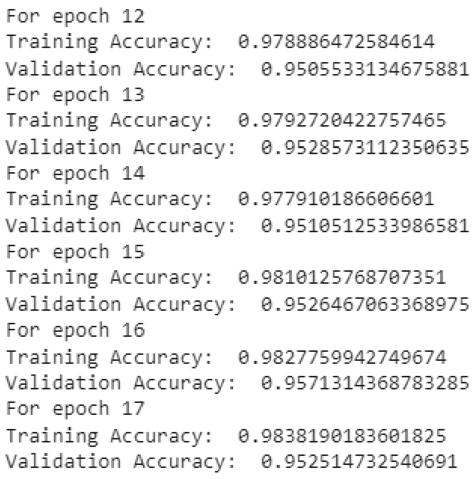
```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.layer1 = nn.Sequential(nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=2),
                                    nn.BatchNorm2d(32),
                                    nn.LeakyReLU(),
                                    nn.MaxPool2d(kernel_size=2))

        self.layer2 = nn.Sequential(nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=2),
                                    nn.BatchNorm2d(64),
                                    nn.LeakyReLU(),
                                    nn.MaxPool2d(kernel_size=2))

        self.layer3 = nn.Sequential(nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=2),
                                    nn.BatchNorm2d(128),
                                    nn.LeakyReLU(),
                                    nn.MaxPool2d(kernel_size=2))

        self.layer4 = nn.Sequential(nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=2),
                                    nn.BatchNorm2d(256),
                                    nn.LeakyReLU(),
                                    nn.MaxPool2d(kernel_size=2))

        self.layer5 = nn.Sequential(nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=2),
                                    nn.BatchNorm2d(512),
                                    nn.LeakyReLU(),
                                    nn.MaxPool2d(kernel_size=2))

        self.avg_pool = nn.AvgPool2d(kernel_size=4)

        self.fc = nn.Linear(in_features=512 * 1 * 1, out_features=2)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.avg_pool(x)
        x = x.view(-1, 512 * 1 * 1)
        x = self.fc(x)
        return x
```

The second model introduces several improvements compared to the first model. It incorporates batch normalization after each convolutional layer, which can speed up training and enhance model performance. Additionally, it uses Leaky ReLU activation instead of regular ReLU activation, which helps address the dying ReLU problem. The second model also has a deeper network with five convolutional layers, allowing it to capture more complex patterns. It applies max pooling after each convolutional layer until the final layer, where average pooling is used to further reduce the spatial dimensions. Furthermore, the second model has a dynamic input size for the fully connected layer based on the output of the final convolutional layer. Overall, these enhancements make the second model more complex but potentially enable it to learn more intricate patterns and improve performance.

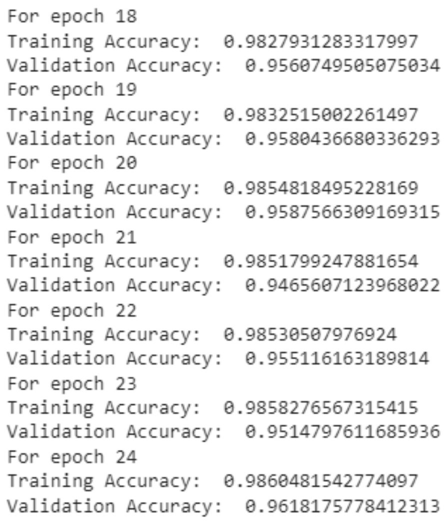
CNN TRAINING AND VALIDATION



```
For epoch 0
Training Accuracy:  0.8708025479866816
Validation Accuracy:  0.9200704258951273
For epoch 1
Training Accuracy:  0.9322128565896974
Validation Accuracy:  0.9379540032961152
For epoch 2
Training Accuracy:  0.9492533963256748
Validation Accuracy:  0.9371613127518107
For epoch 3
Training Accuracy:  0.9564234361281093
Validation Accuracy:  0.939513389821249
For epoch 4
Training Accuracy:  0.9632288402203342
Validation Accuracy:  0.9469977505949579
For epoch 5
Training Accuracy:  0.9664074665449776
Validation Accuracy:  0.9480375995129262
For epoch 6
Training Accuracy:  0.9699770780157442
Validation Accuracy:  0.9431844569531015
For epoch 7
Training Accuracy:  0.9686808494200909
Validation Accuracy:  0.9377595105525953
For epoch 8
Training Accuracy:  0.9740062978341335
Validation Accuracy:  0.9452638404533817
For epoch 9
Training Accuracy:  0.975561263522017
Validation Accuracy:  0.9511369308456779
For epoch 10
Training Accuracy:  0.9745315894618292
Validation Accuracy:  0.9538634738726118
For epoch 11
Training Accuracy:  0.9769687646503538
Validation Accuracy:  0.9560121629006144
```

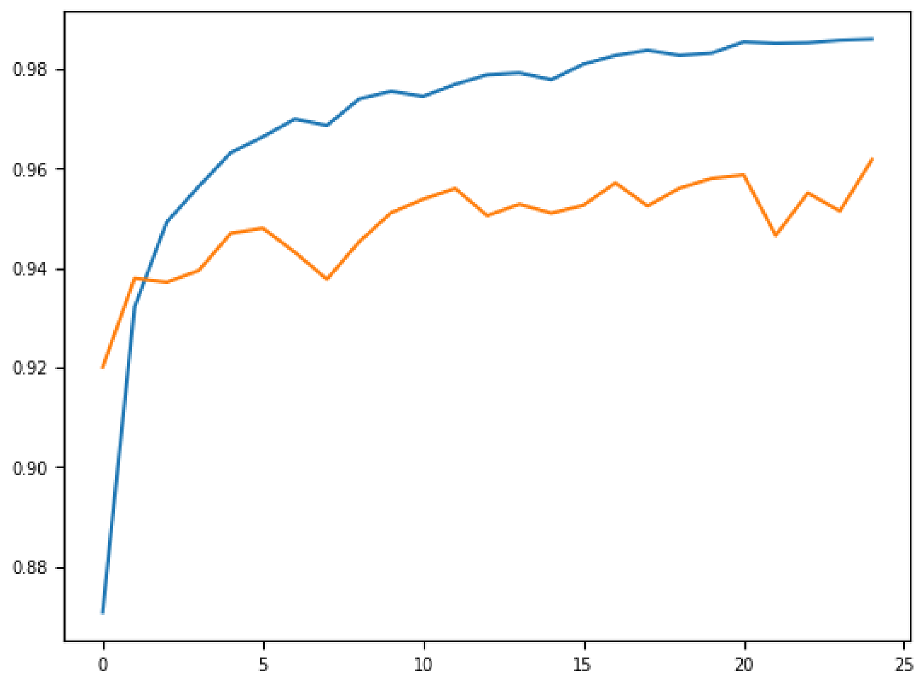


```
For epoch 12
Training Accuracy:  0.978886472584614
Validation Accuracy:  0.9505533134675881
For epoch 13
Training Accuracy:  0.9792720422757465
Validation Accuracy:  0.9528573112350635
For epoch 14
Training Accuracy:  0.977910186606601
Validation Accuracy:  0.9510512533986581
For epoch 15
Training Accuracy:  0.9810125768707351
Validation Accuracy:  0.9526467063368975
For epoch 16
Training Accuracy:  0.9827759942749674
Validation Accuracy:  0.9571314368783285
For epoch 17
Training Accuracy:  0.9838190183601825
Validation Accuracy:  0.952514732540691
```



```
For epoch 18
Training Accuracy:  0.9827931283317997
Validation Accuracy:  0.9560749505075034
For epoch 19
Training Accuracy:  0.9832515002261497
Validation Accuracy:  0.9580436680336293
For epoch 20
Training Accuracy:  0.9854818495228169
Validation Accuracy:  0.9587566309169315
For epoch 21
Training Accuracy:  0.9851799247881654
Validation Accuracy:  0.9465607123968022
For epoch 22
Training Accuracy:  0.98530507976924
Validation Accuracy:  0.955116163189814
For epoch 23
Training Accuracy:  0.9858276567315415
Validation Accuracy:  0.9514797611685936
For epoch 24
Training Accuracy:  0.9860481542774097
Validation Accuracy:  0.9618175778412313
```

After training using the optimizer Adamax and the criterion CrossEntropyLoss I have obtained these validation results of 96.18 % which I consider to be a fairly high result.



This would be the graph showing the evolution of our model during training. Which is increasing for training as we expected and for validation it also grows although it is not uniform.

CNN TEST

I am going to develop two different tests.

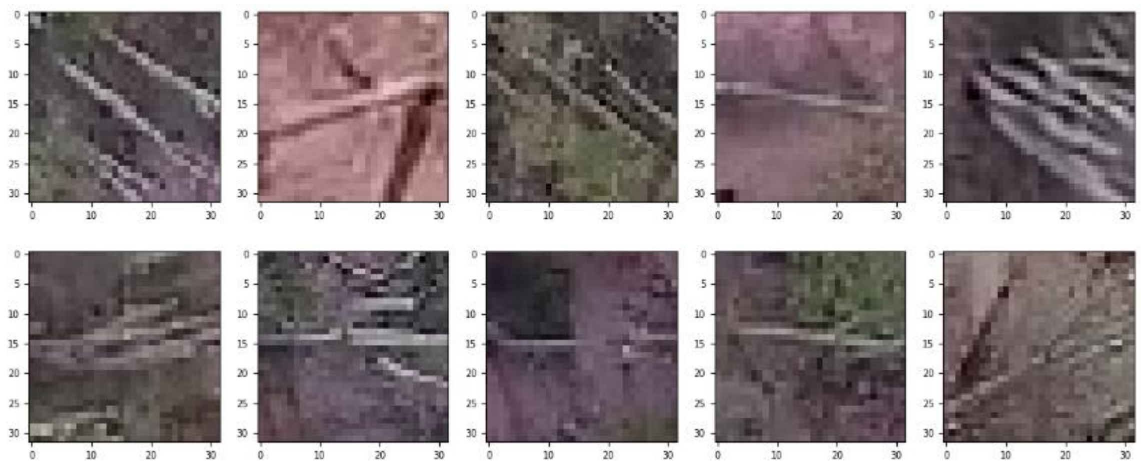
Sample submission test

The first will be seeing the result that my model would give to the Kaggle contest submission file, and we will visually check if we would give the same answer to check how good the response would be.

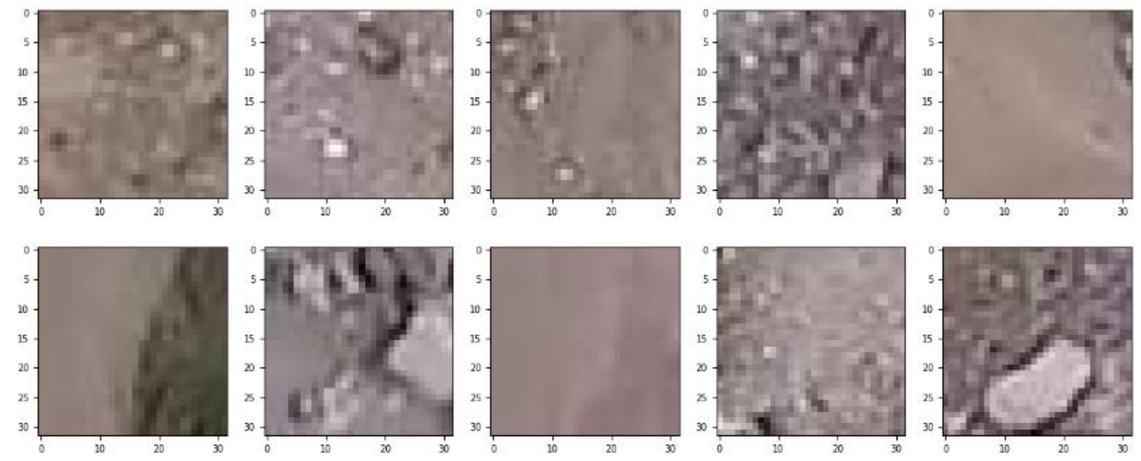
```
testDataset = ImageDataset(df=submission, imgDir='test/', transform=testTransform)
testLoader = DataLoader(dataset=testDataset, batch_size=32, shuffle=False)
```

```
model.eval()

preds=[]
with torch.no_grad():
    for images, _ in testLoader:
        images = images.to(device)

        outputs = model(images)
        preds_part = torch.softmax(outputs.cpu(), dim=1)[:, 1].tolist()

        preds.extend([1 if x >= 0.5 else 0 for x in preds_part])
```

First let's see photos that according to our model there are cactus present and then we will see those in which there are no cactus.

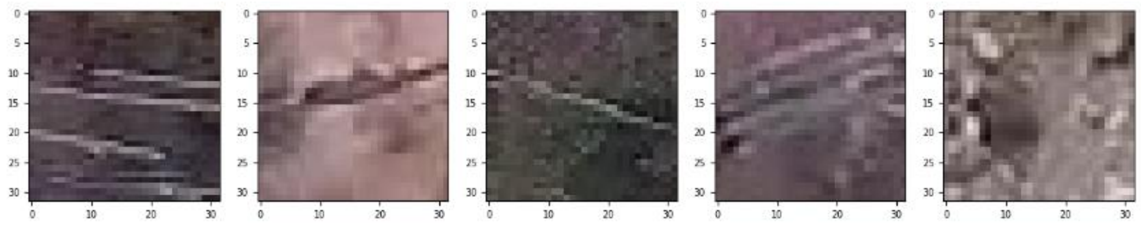


It seems that in these 10 photos I would have been correct in saying that there are cactus in all the photos. The only ones we could doubt would be in the second and tenth, since not even we ourselves could be sure if that elongated figure would be a cactus or not.



For these 10 photos there is no confusion that no cactus are present, so our model would also be correct.

To finish with sample submission test, I am going to show 5 random photos and then I will show what the model said they represent.

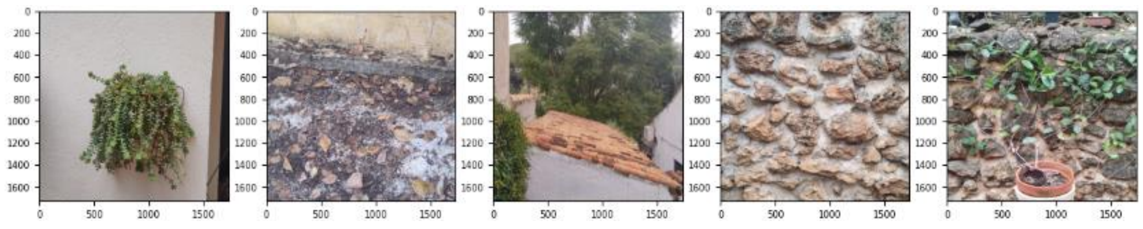


```
For image 1 the model says there is a cactus
For image 2 the model says there is a cactus
For image 3 the model says there is a cactus
For image 4 the model says there is a cactus
For image 5 the model says there is no cactus
```

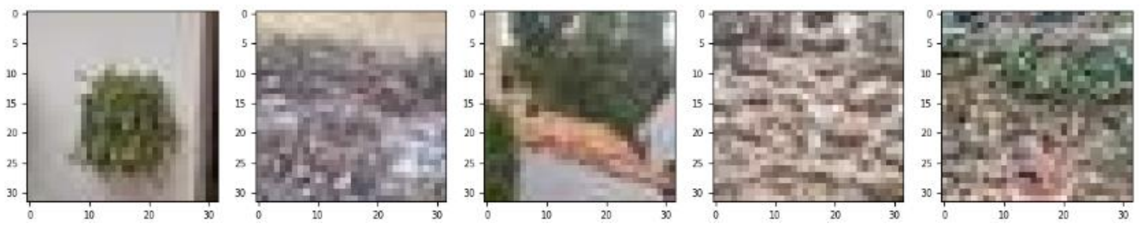
Cactus photos made by me test

In this test I am going to take photos of different cactus and things that are not cactus such as other plants or terrains. We will convert these photos to 32 x 32 x 3 format and run them through the model to see how useful it would be to determine photos that anyone could take to see if there is a cactus in your surroundings.

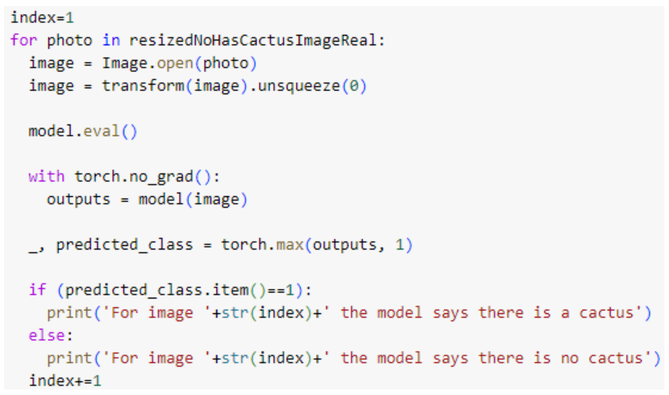
First let's start with images that don't have cactus.



We convert it to 32x32x3.



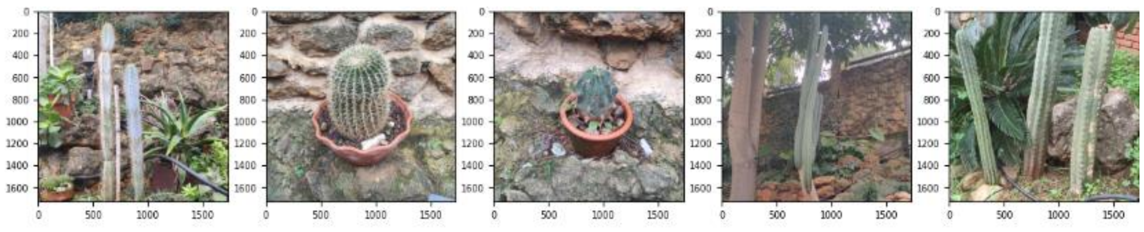
```
For image 1 the model says
there is no cactus
For image 2 the model says
there is no cactus
For image 3 the model says
there is no cactus
For image 4 the model says
there is no cactus
For image 5 the model says
there is no cactus
```



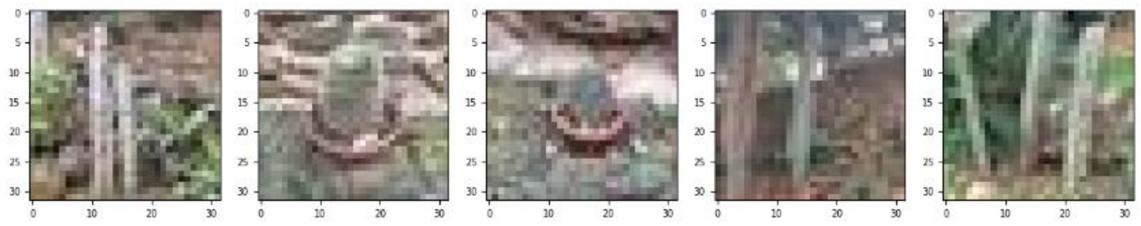
```
index=1
for photo in resizedNoHasCactusImageReal:
  image = Image.open(photo)
  image = transform(image).unsqueeze(0)

  model.eval()

  with torch.no_grad():
    outputs = model(image)

  _, predicted_class = torch.max(outputs, 1)

  if (predicted_class.item()==1):
    print('For image '+str(index)+' the model says there is a cactus')
  else:
    print('For image '+str(index)+' the model says there is no cactus')
  index+=1
```

As we can see given some photos of things that are not cactus such as other plants, a rock wall, a roof... the model has been able to differentiate and deduce that there are no cactus present in the photos.

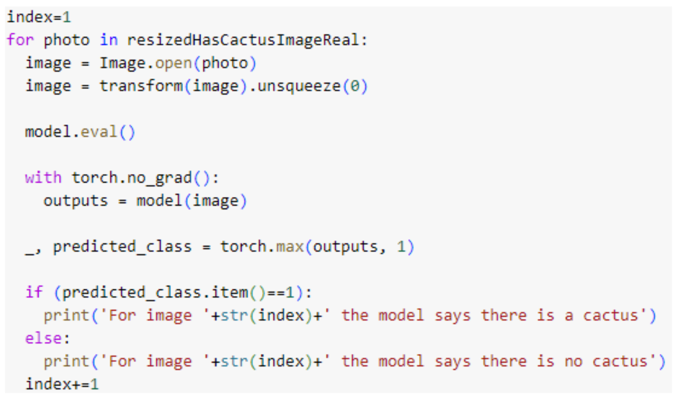
Now let's see photos that do have cactus.



We convert it to 32x32x3.



```
For image 1 the model says
there is a cactus
For image 2 the model says
there is a cactus
For image 3 the model says
there is a cactus
For image 4 the model says
there is a cactus
For image 5 the model says
there is a cactus
```



```
index=1
for photo in resizedHasCactusImageReal:
  image = Image.open(photo)
  image = transform(image).unsqueeze(0)

  model.eval()

  with torch.no_grad():
    outputs = model(image)

  _, predicted_class = torch.max(outputs, 1)

  if (predicted_class.item()==1):
    print('For image '+str(index)+' the model says there is a cactus')
  else:
    print('For image '+str(index)+' the model says there is no cactus')
  index+=1
```

As we expected, the result is positive, so it recognizes that there are cactus. Even differentiating more elongated cactus from smaller ones.

ALL FILES USED

sample_submission.csv: file with the name of the photos files used for the test and submission on Kaggle.

train.csv: file with the name of the photos files used for training and validation with its output 0 or 1 deciding if there is or not a cactus.

test.zip: zip with all the photos files refered in sample_submission.csv.

train.zip: zip with all the photos files refered in train.csv.

aerialCactusModel.pth: file with the final trained model.

hasCactus0-4: square photos of cactus that I have in my house.

noHasCactus0-4: square photos of land and other plants other than cactus.

CONCLUSIONS

This Kaggle competition not only highlighted the challenges of aerial image classification but also demonstrated the efficacy of CNNs in addressing such tasks. The developed model, with its enhanced architecture and high validation accuracy, stands as a testament to the potential of machine learning in environmental applications.

In practical tests, the model exhibited promising capabilities. It successfully distinguished between cacti and other elements such as different plants, rock walls, and roofs. Notably, the model's ability to differentiate between various types of cacti, such as elongated versus smaller varieties, highlights its potential usefulness in real-world scenarios.

It is the first large machine learning project I have carried out and I am very satisfied with the project carried out. and with my learning in convolutional neural networks.

BIBLIOGRAPHY

-https://kaggle.com/competitions/aerial-cactus-identification

-https://towardsdatascience.com/coding-a-convolutional-neural-network-cnn-using-keras-sequential-api-ec5211126875

-https://www.tensorflow.org/tutorials/images/cnn?hl=es-419

-https://victorzhou.com/blog/keras-cnn-tutorial/

-https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/