Convolutional Neural Networks - Codealong

Introduction

In this codealong, we will reinvestigate our previous Santa image classification example. To do this, we will review loading a dataset from a nested directory structure and building a baseline model. From there, we'll build a CNN and demonstrate its improved performance on image recognition tasks. It is recommended you run the cells in order to further explore variables and investigate the code snippets themselves. However, please note that some cells (particularly training cells later on) may take several minutes to run. (On a Macbook pro the entire notebook took ~15 minutes to run.)

Objectives

You will be able to:

- Load images from a hierarchical file structure using an image datagenerator
- Explain why one might augment image data when training a neural network
- Apply data augmentation to image files before training a neural network
- Build a CNN using Keras

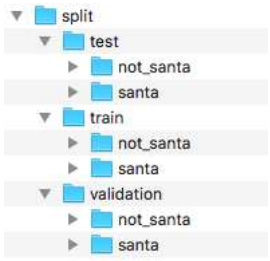
Properly store your images

When you're analyzing your image data, file management is important. We will be using the santa images again, but this time, they are stored in two folders: `santa` and `not_santa`. We want to work with a `train`, `validation`, and `test` datasets now, as we know by now that this is the best way to obtain unbiased estimate of your model performance.

Let's import libraries `os` and `shutil`, as we'll need them to create the new folders and move the new files in there.

In [1]:
```
import os, shutil
```

Below we create three objects representing the existing directories: `data/santa/` as `data_santa_dir` and `data/not_santa/` as `data_not_santa_dir`. We will create a new directory `split/` as `new_dir`, where we will split the dataset in three groups (or three subdirectories): `train`, `test`, and `validation`, each containing `santa` and `not_santa` subfolders. The final desired structure is represented below:



In [2]:
```
data_santa_dir = 'data/santa/'
data_not_santa_dir = 'data/not_santa/'
new_dir = 'split/'
```

You can use `os.listdir()` to create an object that stores all the relevant image names.

In [3]:
```
imgs_santa = [file for file in os.listdir(data_santa_dir) if file.endswith('.jpg')]
```

In [4]:
```
imgs_santa[0:10]
```

Out[4]:
```
['00000129.jpg',
 '00000378.jpg',
 '00000184.jpg',
 '00000117.jpg',
 '00000100.jpg',
 '00000295.jpg',
 '00000283.jpg',
 '00000204.jpg',
 '00000079.jpg',
 '00000037.jpg']
```

Let's see how many images there are in the `santa` directory.

In [5]:
```
print('There are', len(imgs_santa), 'santa images')
```

```
There are 461 santa images
```

Now, repeat this for the `not_santa` directory:

In [6]:
```
imgs_not_santa = [file for file in os.listdir(data_not_santa_dir) if file.endswith('.jpg')]
```

In [7]:
```
print('There are', len(imgs_not_santa), 'images without santa')
```

```
There are 461 images without santa
```

Create all the folders and subfolders in order to get the structure represented above. You can use `os.path.join()` to create strings that will be used later on to generate new directories.

In [8]:
```
os.mkdir(new_dir)
```

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
Cell In [8], line 1
----> 1 os.mkdir(new_dir)

FileExistsError: [Errno 17] File exists: 'split/'
```

In [9]:
```
train_folder = os.path.join(new_dir, 'train')
train_santa = os.path.join(train_folder, 'santa')
train_not_santa = os.path.join(train_folder, 'not_santa')

test_folder = os.path.join(new_dir, 'test')
test_santa = os.path.join(test_folder, 'santa')
test_not_santa = os.path.join(test_folder, 'not_santa')

val_folder = os.path.join(new_dir, 'validation')
val_santa = os.path.join(val_folder, 'santa')
val_not_santa = os.path.join(val_folder, 'not_santa')
```

In [10]:
```
train_santa
```

Out[10]:
```
'split/train/santa'
```

Now use all the path strings you created to make new directories. You can use `os.mkdir()` to do this. Go have a look at your directory and see if this worked!

In [11]:
```
os.mkdir(test_folder)
os.mkdir(test_santa)
os.mkdir(test_not_santa)

os.mkdir(train_folder)
os.mkdir(train_santa)
os.mkdir(train_not_santa)

os.mkdir(val_folder)
os.mkdir(val_santa)
os.mkdir(val_not_santa)
```

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
Cell In [11], line 1
----> 1 os.mkdir(test_folder)
      2 os.mkdir(test_santa)
      3 os.mkdir(test_not_santa)

FileExistsError: [Errno 17] File exists: 'split/test'
```

Copy the Santa images in the three santa subfolders. Let's put the first 271 images in the training set, the next 100 images in the validation set and the final 90 images in the test set.

In [12]:
```
# train santa
imgs = imgs_santa[:271]
for img in imgs:
    origin = os.path.join(data_santa_dir, img)
    destination = os.path.join(train_santa, img)
    shutil.copyfile(origin, destination)
```

In [13]:
```
# validation santa
imgs = imgs_santa[271:371]
for img in imgs:
    origin = os.path.join(data_santa_dir, img)
    destination = os.path.join(val_santa, img)
    shutil.copyfile(origin, destination)
```

In [14]:
```
# test santa
imgs = imgs_santa[371:]
for img in imgs:
    origin = os.path.join(data_santa_dir, img)
    destination = os.path.join(test_santa, img)
    shutil.copyfile(origin, destination)
```

Now, repeat all this for the `not_santa` images!

In [15]:
```
# train not_santa
imgs = imgs_not_santa[:271]
for img in imgs:
    origin = os.path.join(data_not_santa_dir, img)
    destination = os.path.join(train_not_santa, img)
    shutil.copyfile(origin, destination)
# validation not_santa
imgs = imgs_not_santa[271:371]
for img in imgs:
    origin = os.path.join(data_not_santa_dir, img)
    destination = os.path.join(val_not_santa, img)
    shutil.copyfile(origin, destination)
# test not_santa
imgs = imgs_not_santa[371:]
for img in imgs:
    origin = os.path.join(data_not_santa_dir, img)
    destination = os.path.join(test_not_santa, img)
    shutil.copyfile(origin, destination)
```

Let's print out how many images we have in each directory so we know for sure our numbers are right!

In [16]:
```
print('There are', len(os.listdir(train_santa)), 'santa images in the training set')
```

```
There are 387 santa images in the training set
```

In [17]:
```
print('There are', len(os.listdir(val_santa)), 'santa images in the validation set')
```

```
There are 176 santa images in the validation set
```

In [18]:
```
print('There are', len(os.listdir(test_santa)), 'santa images in the test set')
```

```
There are 158 santa images in the test set
```

In [19]:
```
print('There are', len(os.listdir(train_not_santa)), 'images without santa in the train set')
```

```
There are 385 images without santa in the train set
```

In [20]:
```
print('There are', len(os.listdir(val_not_santa)), 'images without santa in the validation set')
```

```
There are 176 images without santa in the validation set
```

In [21]:
```
print('There are', len(os.listdir(test_not_santa)), 'images without santa in the test set')
```

```
There are 159 images without santa in the test set
```

Use a densely connected network as a baseline

Now that we've a handle on our data, we can easily use Keras' module with image-processing tools. Let's import the necessary libraries below.

In [22]:
```
import time
import matplotlib.pyplot as plt
import scipy
import numpy as np
from PIL import Image
from scipy import ndimage
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

np.random.seed(123)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In [22], line 7
      5 from PIL import Image
      6 from scipy import ndimage
----> 7 from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
      9 np.random.seed(123)

ImportError: cannot import name 'array_to_img' from 'keras.preprocessing.image' (/opt/saturncloud/envs/saturn/lib/python3.10/si
te-packages/keras/preprocessing/image.py)
```

In [ ]:
```
# get all the data in the directory split/test (180 images), and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_folder,
        target_size=(64, 64), batch_size = 180)

# get all the data in the directory split/validation (200 images), and reshape them
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_folder,
        target_size=(64, 64), batch_size = 200)

# get all the data in the directory split/train (542 images), and reshape them
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        train_folder,
        target_size=(64, 64), batch_size=542)
```

In [ ]:
```
# create the data sets
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)
val_images, val_labels = next(val_generator)
```

In [ ]:
```
# Explore your dataset again
m_train = train_images.shape[0]
num_px = train_images.shape[1]
m_test = test_images.shape[0]
m_val = val_images.shape[0]

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
print ("train_images shape: " + str(train_images.shape))
print ("train_labels shape: " + str(train_labels.shape))
print ("test_images shape: " + str(test_images.shape))
print ("test_labels shape: " + str(test_labels.shape))
print ("val_images shape: " + str(val_images.shape))
print ("val_labels shape: " + str(val_labels.shape))
```

In [ ]:
```
train_img = train_images.reshape(train_images.shape[0], -1)
test_img = test_images.reshape(test_images.shape[0], -1)
val_img = val_images.reshape(val_images.shape[0], -1)

print(train_img.shape)
print(test_img.shape)
print(val_img.shape)
```

In [ ]:
```
train_y = np.reshape(train_labels[:,0], (542,1))
test_y = np.reshape(test_labels[:,0], (180,1))
val_y = np.reshape(val_labels[:,0], (200,1))
```

In [ ]:
```
# Build a baseline fully connected model
from keras import models
from keras import layers
np.random.seed(123)
model = models.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(12288,))) # 2 hidden layers
model.add(layers.Dense(7, activation='relu'))
model.add(layers.Dense(5, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])

histoire = model.fit(train_img,
                     train_y,
                     epochs=50,
                     batch_size=32,
                     validation_data=(val_img, val_y))
```

In [ ]:
```
results_train = model.evaluate(train_img, train_y)
```

In [ ]:
```
results_test = model.evaluate(test_img, test_y)
```

In [ ]:
```
results_train
```

In [ ]:
```
results_test
```

Remember that, in our previous lab on building deeper neural networks from scratch, we obtained a training accuracy of 95%, and a test set accuracy of 74.23%.

This result is similar to what we got building our manual "deeper" dense model. The results are not entirely different. This is not a surprise!

- Before, we only had a training and a validation set (which was at the same time the test set). Now we have split up the data 3-ways.
- We didn't use minibatches before, yet we used mini-batches of 32 units here.

Build a CNN

In [ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(64 ,64,  3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (4, 4), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['acc'])
```

In [ ]:
```
history = model.fit(train_images,
                    train_y,
                    epochs=30,
                    batch_size=32,
                    validation_data=(val_images, val_y))
```

In [ ]:
```
results_train = model.evaluate(train_images, train_y)
```

In [ ]:
```
results_test = model.evaluate(test_images, test_y)
```

In [ ]:
```
results_train
```

In [ ]:
```
results_test
```

Data Augmentation

`ImageDataGenerator()` becomes really useful when we *actually* want to generate more data. We'll show you how this works.

In [ ]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.3,
                                   zoom_range=0.1,
                                   horizontal_flip=False)
```

In [ ]:
```
names = [os.path.join(train_santa, name) for name in os.listdir(train_santa)]
img_path = names[91]
img = load_img(img_path, target_size=(64, 64))

reshape_img = img_to_array(img)
reshape_img = reshape_img.reshape((1,) + reshape_img.shape)
i=0
for batch in train_datagen.flow(reshape_img, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(array_to_img(batch[0]))
    i += 1
    if i % 3 == 0:
        break
plt.show()
```

In [ ]:
```
# get all the data in the directory split/test (180 images), and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_folder,
        target_size=(64, 64),
        batch_size = 180,
        class_mode='binary')

# get all the data in the directory split/validation (200 images), and reshape them
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_folder,
        target_size=(64, 64),
        batch_size = 32,
        class_mode='binary')

# get all the data in the directory split/train (542 images), and reshape them
train_generator = train_datagen.flow_from_directory(
        train_folder,
        target_size=(64, 64),
        batch_size = 32,
        class_mode='binary')
```

In [ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(64 ,64,  3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (4, 4), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [ ]:
```
history_2 = model.fit_generator(train_generator,
                                steps_per_epoch=25,
                                epochs=30,
                                validation_data=val_generator,
                                validation_steps=25)
```

In [ ]:
```
test_x, test_y = next(test_generator)
```

In [ ]:
```
results_test = model.evaluate(test_x, test_y)
```

In [ ]:
```
results_test
```

Summary

In this code along lab, we looked again at some of the preprocessing techniques needed in order to organize our data prior to building a model using Keras. Afterwards, we investigated new code in order to build a CNN for image recognition.