index

May 18, 2022

1 Convolutional Neural Networks - Codealong

1.1 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.)

1.2 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

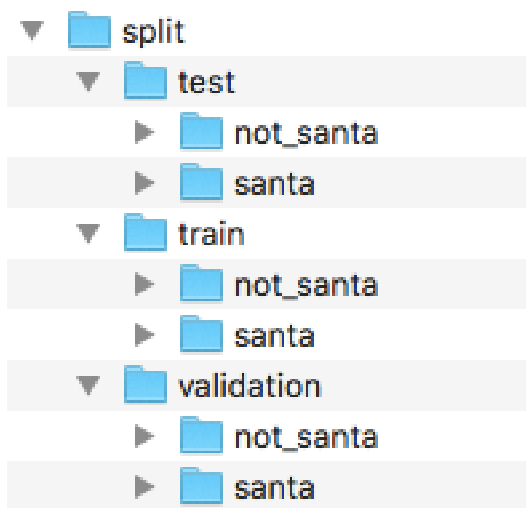
1.3 Properly store your images

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.

[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:



[2]:
```python
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.

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

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

[4]:
```
['00000428.jpg',
 '00000400.jpg',
 '00000366.jpg',
 '00000372.jpg',
 '00000414.jpg',
 '00000399.jpg',
 '00000158.jpg',
 '00000164.jpg',
 '00000170.jpg',
 '00000038.jpg']
```

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

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

```
There are 461 santa images
```

Now, repeat this for the `not_santa` directory:

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

[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.

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

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-8-0d1f8d3cb245> in <module>
----> 1 os.mkdir(new_dir)

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

[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')
```

[10]:
```
train_santa
```

[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!

[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)
<ipython-input-11-9ef278089501> in <module>
----> 1 os.mkdir(test_folder)
      2 os.mkdir(test_santa)
      3 os.mkdir(test_not_santa)
      4
      5 os.mkdir(train_folder)

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.

[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)
```

[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)
```

[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!

[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!

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

```
There are 271 santa images in the training set
```

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

```
There are 100 santa images in the validation set
```

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

```
There are 90 santa images in the test set
```

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

```
There are 271 images without santa in the train set
```

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

```
There are 100 images without santa in the validation set
```

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

```
There are 90 images without santa in the test set
```

1.4 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.

[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)
```

[23]:
```
# 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)
```

```
Found 180 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Found 542 images belonging to 2 classes.
```

[24]:
```
# 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)
```

[25]:
```
# 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))
```

```
Number of training samples: 542
Number of testing samples: 180
Number of validation samples: 200
train_images shape: (542, 64, 64, 3)
train_labels shape: (542, 2)
test_images shape: (180, 64, 64, 3)
test_labels shape: (180, 2)
val_images shape: (200, 64, 64, 3)
val_labels shape: (200, 2)
```

[26]:
```
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)
```

```
(542, 12288)
(180, 12288)
(200, 12288)
```

[27]:
```
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))
```

[28]:
```
# 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'))
```

[29]:
```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])

histoire = model.fit(train_img,
                    train_y,
                    epochs=50,
```

```python
                   batch_size=32,
                   validation_data=(val_img, val_y))
```

```
Epoch 1/50
17/17 [==============================] - 0s 11ms/step - loss: 0.6992 - accuracy:
0.5092 - val_loss: 0.6787 - val_accuracy: 0.5000
Epoch 2/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6823 - accuracy:
0.5055 - val_loss: 0.6757 - val_accuracy: 0.5000
Epoch 3/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6781 - accuracy:
0.5000 - val_loss: 0.6746 - val_accuracy: 0.5000
Epoch 4/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6738 - accuracy:
0.5000 - val_loss: 0.6673 - val_accuracy: 0.5000
Epoch 5/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6646 - accuracy:
0.5018 - val_loss: 0.6571 - val_accuracy: 0.5000
Epoch 6/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6430 - accuracy:
0.5166 - val_loss: 0.6381 - val_accuracy: 0.5000
Epoch 7/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6321 - accuracy:
0.5941 - val_loss: 0.6341 - val_accuracy: 0.5000
Epoch 8/50
17/17 [==============================] - 0s 3ms/step - loss: 0.6148 - accuracy:
0.6494 - val_loss: 0.6478 - val_accuracy: 0.7300
Epoch 9/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5746 - accuracy:
0.7159 - val_loss: 0.5893 - val_accuracy: 0.7550
Epoch 10/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5983 - accuracy:
0.6642 - val_loss: 0.5885 - val_accuracy: 0.5350
Epoch 11/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5538 - accuracy:
0.7232 - val_loss: 0.5679 - val_accuracy: 0.7850
Epoch 12/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5287 - accuracy:
0.7694 - val_loss: 0.5934 - val_accuracy: 0.5600
Epoch 13/50
17/17 [==============================] - 0s 4ms/step - loss: 0.5385 - accuracy:
0.7269 - val_loss: 0.5485 - val_accuracy: 0.6400
Epoch 14/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5323 - accuracy:
0.7491 - val_loss: 0.6275 - val_accuracy: 0.5050
Epoch 15/50
17/17 [==============================] - 0s 3ms/step - loss: 0.5286 - accuracy:
```

```
0.7675 - val_loss: 0.5661 - val_accuracy: 0.6550
Epoch 16/50
17/17 [==============================] - 0s 4ms/step - loss: 0.4973 - accuracy:
0.7989 - val_loss: 0.5708 - val_accuracy: 0.7750
Epoch 17/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4826 - accuracy:
0.8100 - val_loss: 0.5609 - val_accuracy: 0.6200
Epoch 18/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4879 - accuracy:
0.7860 - val_loss: 0.5140 - val_accuracy: 0.8050
Epoch 19/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4760 - accuracy:
0.8044 - val_loss: 0.5658 - val_accuracy: 0.7700
Epoch 20/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4529 - accuracy:
0.8284 - val_loss: 0.5070 - val_accuracy: 0.8100
Epoch 21/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4271 - accuracy:
0.8635 - val_loss: 0.5400 - val_accuracy: 0.7700
Epoch 22/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4721 - accuracy:
0.8044 - val_loss: 0.5943 - val_accuracy: 0.7000
Epoch 23/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4462 - accuracy:
0.8303 - val_loss: 0.5167 - val_accuracy: 0.7950
Epoch 24/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4343 - accuracy:
0.8266 - val_loss: 0.5279 - val_accuracy: 0.7800
Epoch 25/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3995 - accuracy:
0.8838 - val_loss: 0.5588 - val_accuracy: 0.6600
Epoch 26/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4290 - accuracy:
0.8339 - val_loss: 0.4698 - val_accuracy: 0.8250
Epoch 27/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3731 - accuracy:
0.8948 - val_loss: 0.6043 - val_accuracy: 0.6050
Epoch 28/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3910 - accuracy:
0.8672 - val_loss: 0.5153 - val_accuracy: 0.7350
Epoch 29/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4144 - accuracy:
0.8506 - val_loss: 0.4960 - val_accuracy: 0.7750
Epoch 30/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3464 - accuracy:
0.9077 - val_loss: 0.5142 - val_accuracy: 0.7600
Epoch 31/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3661 - accuracy:
```

```
0.8708 - val_loss: 0.5305 - val_accuracy: 0.7500
Epoch 32/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3483 - accuracy:
0.9004 - val_loss: 0.5816 - val_accuracy: 0.6750
Epoch 33/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3768 - accuracy:
0.8727 - val_loss: 0.4573 - val_accuracy: 0.8050
Epoch 34/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3126 - accuracy:
0.9225 - val_loss: 0.4814 - val_accuracy: 0.7950
Epoch 35/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3082 - accuracy:
0.9188 - val_loss: 0.5365 - val_accuracy: 0.7250
Epoch 36/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3349 - accuracy:
0.9096 - val_loss: 0.4617 - val_accuracy: 0.8150
Epoch 37/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3341 - accuracy:
0.8856 - val_loss: 0.6162 - val_accuracy: 0.6850
Epoch 38/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3131 - accuracy:
0.9225 - val_loss: 0.7309 - val_accuracy: 0.6100
Epoch 39/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4287 - accuracy:
0.8100 - val_loss: 0.4776 - val_accuracy: 0.8200
Epoch 40/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2958 - accuracy:
0.9170 - val_loss: 0.5956 - val_accuracy: 0.6950
Epoch 41/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2831 - accuracy:
0.9299 - val_loss: 0.4407 - val_accuracy: 0.8150
Epoch 42/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2908 - accuracy:
0.9262 - val_loss: 0.6198 - val_accuracy: 0.6700
Epoch 43/50
17/17 [==============================] - 0s 3ms/step - loss: 0.4364 - accuracy:
0.8026 - val_loss: 0.5318 - val_accuracy: 0.7250
Epoch 44/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3602 - accuracy:
0.8524 - val_loss: 0.4876 - val_accuracy: 0.7800
Epoch 45/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2766 - accuracy:
0.9317 - val_loss: 0.4396 - val_accuracy: 0.8400
Epoch 46/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2591 - accuracy:
0.9465 - val_loss: 0.4751 - val_accuracy: 0.8050
Epoch 47/50
17/17 [==============================] - 0s 3ms/step - loss: 0.3040 - accuracy:
```

```
0.8985 - val_loss: 0.4603 - val_accuracy: 0.8100
Epoch 48/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2421 - accuracy:
0.9502 - val_loss: 0.4408 - val_accuracy: 0.8050
Epoch 49/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2497 - accuracy:
0.9483 - val_loss: 0.4646 - val_accuracy: 0.7950
Epoch 50/50
17/17 [==============================] - 0s 3ms/step - loss: 0.2167 - accuracy:
0.9723 - val_loss: 0.4898 - val_accuracy: 0.7800
```

[30]:
```
results_train = model.evaluate(train_img, train_y)
```

```
17/17 [==============================] - 0s 1ms/step - loss: 0.2549 - accuracy:
0.9539
```

[31]:
```
results_test = model.evaluate(test_img, test_y)
```

```
6/6 [==============================] - 0s 964us/step - loss: 0.4305 - accuracy:
0.8389
```

[32]:
```
results_train
```

[32]:
```
[0.2549353837966919, 0.9538745284080505]
```

[33]:
```
results_test
```

[33]:
```
[0.43052712082862854, 0.8388888835906982]
```

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.

1.5 Build a CNN

[34]:
```
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'])
```

[35]:
```
history = model.fit(train_images,
                    train_y,
                    epochs=30,
                    batch_size=32,
                    validation_data=(val_images, val_y))
```

```
Epoch 1/30
17/17 [==============================] - 1s 81ms/step - loss: 0.6811 - acc:
0.5000 - val_loss: 0.6737 - val_acc: 0.5000
Epoch 2/30
17/17 [==============================] - 1s 81ms/step - loss: 0.6746 - acc:
0.5000 - val_loss: 0.6665 - val_acc: 0.5000
Epoch 3/30
17/17 [==============================] - 1s 88ms/step - loss: 0.6681 - acc:
0.5000 - val_loss: 0.6601 - val_acc: 0.5050
Epoch 4/30
17/17 [==============================] - 1s 74ms/step - loss: 0.6617 - acc:
0.5018 - val_loss: 0.6527 - val_acc: 0.5100
Epoch 5/30
17/17 [==============================] - 1s 74ms/step - loss: 0.6551 - acc:
0.5092 - val_loss: 0.6444 - val_acc: 0.5300
Epoch 6/30
17/17 [==============================] - 1s 73ms/step - loss: 0.6461 - acc:
0.5277 - val_loss: 0.6342 - val_acc: 0.5800
Epoch 7/30
17/17 [==============================] - 1s 72ms/step - loss: 0.6343 - acc:
0.5664 - val_loss: 0.6205 - val_acc: 0.6100
Epoch 8/30
17/17 [==============================] - 1s 71ms/step - loss: 0.6208 - acc:
0.6494 - val_loss: 0.6047 - val_acc: 0.6050
Epoch 9/30
17/17 [==============================] - 1s 71ms/step - loss: 0.6032 - acc:
0.6937 - val_loss: 0.5892 - val_acc: 0.6000
Epoch 10/30
17/17 [==============================] - 1s 72ms/step - loss: 0.5865 - acc:
0.7251 - val_loss: 0.5628 - val_acc: 0.7250
Epoch 11/30
17/17 [==============================] - 1s 73ms/step - loss: 0.5680 - acc:
```

```
0.7841 - val_loss: 0.5367 - val_acc: 0.8300
Epoch 12/30
17/17 [==============================] - 1s 71ms/step - loss: 0.5355 - acc:
0.8137 - val_loss: 0.5153 - val_acc: 0.9000
Epoch 13/30
17/17 [==============================] - 1s 73ms/step - loss: 0.5088 - acc:
0.8542 - val_loss: 0.4708 - val_acc: 0.8850
Epoch 14/30
17/17 [==============================] - 1s 76ms/step - loss: 0.4916 - acc:
0.8524 - val_loss: 0.4368 - val_acc: 0.8900
Epoch 15/30
17/17 [==============================] - 1s 73ms/step - loss: 0.4439 - acc:
0.8672 - val_loss: 0.4697 - val_acc: 0.7400
Epoch 16/30
17/17 [==============================] - 1s 73ms/step - loss: 0.4477 - acc:
0.8321 - val_loss: 0.5432 - val_acc: 0.6900
Epoch 17/30
17/17 [==============================] - 1s 71ms/step - loss: 0.4194 - acc:
0.8413 - val_loss: 0.3474 - val_acc: 0.9300
Epoch 18/30
17/17 [==============================] - 1s 73ms/step - loss: 0.4471 - acc:
0.8210 - val_loss: 0.3506 - val_acc: 0.9150
Epoch 19/30
17/17 [==============================] - 1s 73ms/step - loss: 0.3765 - acc:
0.8801 - val_loss: 0.3369 - val_acc: 0.9150
Epoch 20/30
17/17 [==============================] - 2s 94ms/step - loss: 0.3638 - acc:
0.8745 - val_loss: 0.3054 - val_acc: 0.9250
Epoch 21/30
17/17 [==============================] - 1s 82ms/step - loss: 0.3693 - acc:
0.8506 - val_loss: 0.4181 - val_acc: 0.7950
Epoch 22/30
17/17 [==============================] - 1s 75ms/step - loss: 0.3457 - acc:
0.8616 - val_loss: 0.3144 - val_acc: 0.8950
Epoch 23/30
17/17 [==============================] - 1s 76ms/step - loss: 0.2858 - acc:
0.9004 - val_loss: 0.3896 - val_acc: 0.8250
Epoch 24/30
17/17 [==============================] - 1s 84ms/step - loss: 0.3351 - acc:
0.8450 - val_loss: 0.2747 - val_acc: 0.9100
Epoch 25/30
17/17 [==============================] - 1s 82ms/step - loss: 0.3054 - acc:
0.8911 - val_loss: 0.2594 - val_acc: 0.9200
Epoch 26/30
17/17 [==============================] - 1s 79ms/step - loss: 0.2740 - acc:
0.9059 - val_loss: 0.2585 - val_acc: 0.9050
Epoch 27/30
17/17 [==============================] - 1s 79ms/step - loss: 0.2933 - acc:
```

```
0.8875 - val_loss: 0.2453 - val_acc: 0.9350
Epoch 28/30
17/17 [==============================] - 1s 83ms/step - loss: 0.2643 - acc:
0.9041 - val_loss: 0.3302 - val_acc: 0.8700
Epoch 29/30
17/17 [==============================] - 1s 87ms/step - loss: 0.2459 - acc:
0.9170 - val_loss: 0.2853 - val_acc: 0.8750
Epoch 30/30
17/17 [==============================] - 1s 81ms/step - loss: 0.2929 - acc:
0.8856 - val_loss: 0.2382 - val_acc: 0.9300
```

[36]:
```
results_train = model.evaluate(train_images, train_y)
```

```
17/17 [==============================] - 0s 12ms/step - loss: 0.2279 - acc:
0.9391
```

[37]:
```
results_test = model.evaluate(test_images, test_y)
```

```
6/6 [==============================] - 0s 12ms/step - loss: 0.2922 - acc: 0.9056
```

[38]:
```
results_train
```

[38]:
```
[0.22792600095272064, 0.9391143918037415]
```

[39]:
```
results_test
```

[39]:
```
[0.29216301441192627, 0.9055555462837219]
```

1.6 Data Augmentation

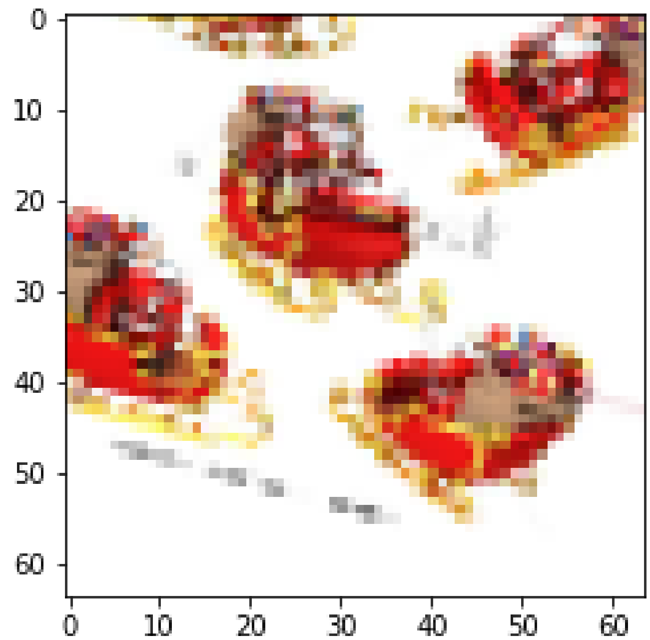
`ImageDataGenerator()` becomes really useful when we *actually* want to generate more data. We'll show you how this works.

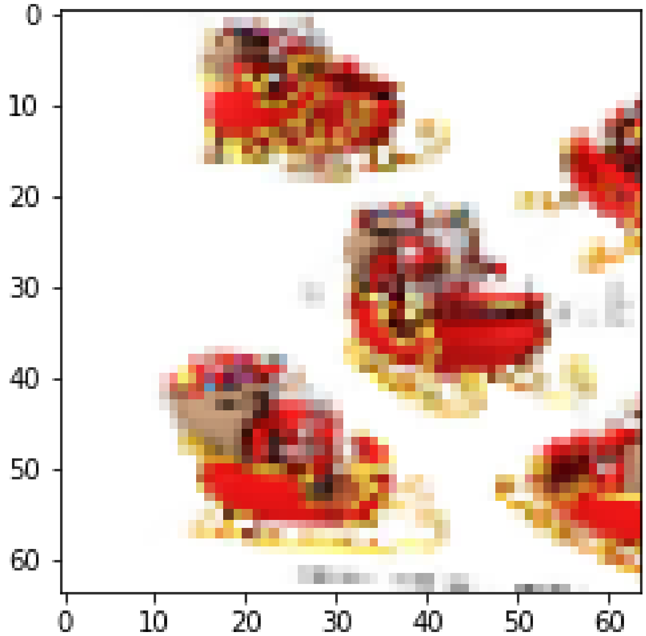
[40]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.3,
                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

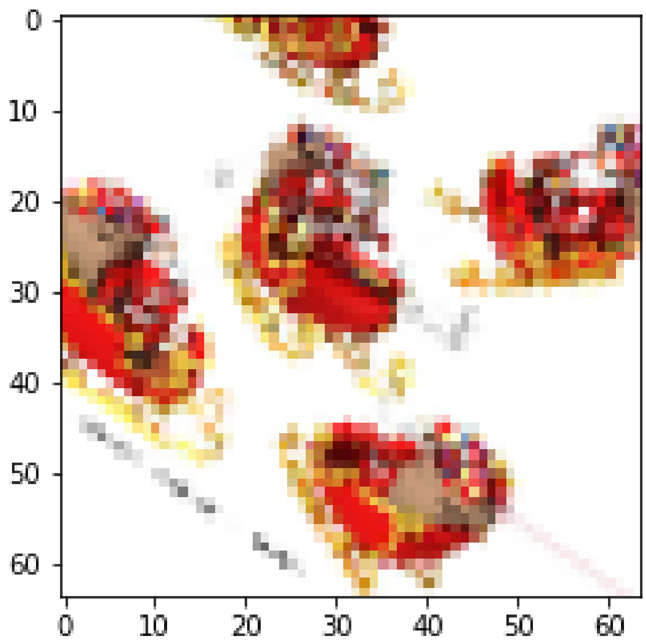
[41]:
```
names = [os.path.join(train_santa, name) for name in os.listdir(train_santa)]
img_path = names[91]
img = load_img(img_path, target_size=(64, 64))

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()
```







[42]:
```
# 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')
```

```
Found 180 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Found 542 images belonging to 2 classes.
```

[43]:
```
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'])
```

[44]:
```
history_2 = model.fit_generator(train_generator,
                                steps_per_epoch=25,
                                epochs=30,
                                validation_data=val_generator,
                                validation_steps=25)
```

```
Epoch 1/30
25/25 [==============================] - 24s 959ms/step - loss: 0.6825 - acc:
0.5088 - val_loss: 0.6784 - val_acc: 0.4931
Epoch 2/30
25/25 [==============================] - 18s 731ms/step - loss: 0.6809 - acc:
0.4889 - val_loss: 0.6705 - val_acc: 0.5057
Epoch 3/30
25/25 [==============================] - 19s 759ms/step - loss: 0.6700 - acc:
0.5099 - val_loss: 0.6649 - val_acc: 0.4986
Epoch 4/30
25/25 [==============================] - 19s 756ms/step - loss: 0.6716 - acc:
0.4925 - val_loss: 0.6545 - val_acc: 0.5142
Epoch 5/30
25/25 [==============================] - 18s 720ms/step - loss: 0.6618 - acc:
0.4953 - val_loss: 0.6480 - val_acc: 0.5412
Epoch 6/30
25/25 [==============================] - 17s 692ms/step - loss: 0.6488 - acc:
0.5585 - val_loss: 0.6350 - val_acc: 0.5426
Epoch 7/30
25/25 [==============================] - 17s 698ms/step - loss: 0.6406 - acc:
0.5986 - val_loss: 0.6131 - val_acc: 0.6179
Epoch 8/30
25/25 [==============================] - 18s 703ms/step - loss: 0.6182 - acc:
0.6848 - val_loss: 0.5917 - val_acc: 0.5701
Epoch 9/30
25/25 [==============================] - 19s 747ms/step - loss: 0.5995 - acc:
0.7053 - val_loss: 0.5610 - val_acc: 0.7159
Epoch 10/30
25/25 [==============================] - 19s 778ms/step - loss: 0.5746 - acc:
0.7722 - val_loss: 0.5229 - val_acc: 0.8736
Epoch 11/30
25/25 [==============================] - 21s 820ms/step - loss: 0.5355 - acc:
0.8202 - val_loss: 0.4920 - val_acc: 0.7500
Epoch 12/30
25/25 [==============================] - 18s 716ms/step - loss: 0.5157 - acc:
0.7957 - val_loss: 0.4750 - val_acc: 0.8558
Epoch 13/30
25/25 [==============================] - 19s 753ms/step - loss: 0.5224 - acc:
0.7517 - val_loss: 0.4509 - val_acc: 0.8679
Epoch 14/30
25/25 [==============================] - 17s 695ms/step - loss: 0.4365 - acc:
0.8322 - val_loss: 0.4074 - val_acc: 0.8608
Epoch 15/30
25/25 [==============================] - 19s 751ms/step - loss: 0.4558 - acc:
0.7927 - val_loss: 0.3604 - val_acc: 0.8626
Epoch 16/30
25/25 [==============================] - 15s 619ms/step - loss: 0.3974 - acc:
0.8244 - val_loss: 0.6614 - val_acc: 0.6222
```

```
Epoch 17/30
25/25 [==============================] - 18s 703ms/step - loss: 0.3881 - acc:
0.8430 - val_loss: 0.4041 - val_acc: 0.7926
Epoch 18/30
25/25 [==============================] - 18s 707ms/step - loss: 0.3873 - acc:
0.8208 - val_loss: 0.2955 - val_acc: 0.8991
Epoch 19/30
25/25 [==============================] - 20s 793ms/step - loss: 0.3690 - acc:
0.8472 - val_loss: 0.5001 - val_acc: 0.7418
Epoch 20/30
25/25 [==============================] - 19s 767ms/step - loss: 0.3411 - acc:
0.8567 - val_loss: 0.3544 - val_acc: 0.8693
Epoch 21/30
25/25 [==============================] - 17s 682ms/step - loss: 0.3096 - acc:
0.8796 - val_loss: 0.2478 - val_acc: 0.9190
Epoch 22/30
25/25 [==============================] - 18s 724ms/step - loss: 0.2971 - acc:
0.8921 - val_loss: 0.2591 - val_acc: 0.9217
Epoch 23/30
25/25 [==============================] - 17s 700ms/step - loss: 0.2797 - acc:
0.8945 - val_loss: 0.2508 - val_acc: 0.8963
Epoch 24/30
25/25 [==============================] - 17s 694ms/step - loss: 0.3001 - acc:
0.8614 - val_loss: 0.3669 - val_acc: 0.8585
Epoch 25/30
25/25 [==============================] - 19s 751ms/step - loss: 0.2836 - acc:
0.8869 - val_loss: 0.2718 - val_acc: 0.9233
Epoch 26/30
25/25 [==============================] - 19s 745ms/step - loss: 0.2654 - acc:
0.9030 - val_loss: 0.2096 - val_acc: 0.9272
Epoch 27/30
25/25 [==============================] - 16s 645ms/step - loss: 0.2532 - acc:
0.8948 - val_loss: 0.2557 - val_acc: 0.9176
Epoch 28/30
25/25 [==============================] - 17s 690ms/step - loss: 0.2838 - acc:
0.8935 - val_loss: 0.2240 - val_acc: 0.9261
Epoch 29/30
25/25 [==============================] - 18s 728ms/step - loss: 0.2906 - acc:
0.8950 - val_loss: 0.2873 - val_acc: 0.8984
Epoch 30/30
25/25 [==============================] - 24s 970ms/step - loss: 0.2447 - acc:
0.9083 - val_loss: 0.2566 - val_acc: 0.9048
```

[45]:
```
test_x, test_y = next(test_generator)
```

[46]:
```
results_test = model.evaluate(test_x, test_y)
```

```
180/180 [==============================] - 0s 1ms/step
```

[47]:
```
results_test
```

[47]:
```
[0.2512160725063748, 0.9277777751286824]
```

1.7 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.