Convolutional Neural Networks for Classification of Cats and Dogs

Author: Rantete Tshinanga Keith

This is a classification problem that employs machine vision, where the CNNs, after some intense training, will be able to classify cats and dogs, given their images. The model learns from scratch, via supervised machine learning.

About 8000 and 2000 images of cats and dogs will be used in the training phase and testing phase respectively.

Optimizer:

Adaptive Moment Estimation (ADAM) so our stochastic gradient descent optimizes to a global minima. Both the weights and the feature detectors are optimized during the backpropagation optimization.

Loss function:

Binary CrossEntropy since binary encoding of the target categories will be employed.

Activation function (convolution):

Rectified Linear Unit (RELU) to introduce non-linearities to the images.

Activation function (voting synapse):

Sigmoid function, optimal for binary encoded categories.

Stages: Images inputs, convolution, max pooling, convolution, max pooling, flattening, image inputs, fully connected neurons, output (a cat or a dog)

Convolution:

Images, feature detectors, feature maps

Max pooling:

To add spatial invariance, prevent overfitting, and for dimension reduction of the images.

Flattening:

To flatten the tensors into a one-dimensional vector to serve as an input to the Fully Connected Artifical Neural Networks.

Full connection:

Full connection of neural layers and neurons respectively.

Using similar procedure, we can build an AI that can diagnose patients after diagnostic imaging tests such as X-rays, CT scan, MRI, Mammogram, Ultrasound, Fluoroscopy, and PET scans. The AI can improve with time to even make more accurate diagnosis than a human doctor can, thus removing human error in medical diagnosis.

More applications of the CNNs are in self-driving cars, where the CNNs enable car vision so it can recognize road signs, lanes, other cars and the like, which allows a car to drive itself and may help in reducing accidents caused by human error.

Importing the libraries

In [1]:

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

import sys
from PIL import Image
sys.modules['Image'] = Image
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.3.0'
```

Data Preprocessing

Preprocessing the Training set

In [3]:

```
## Image Argumentation to avoid overfitting
train_datagen = ImageDataGenerator(
        rescale=1./255,    #applies feature scaling to pixel by dividng each pixel by 255
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

training_set = train_datagen.flow_from_directory(
        'dataset/training_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 8000 images belonging to 2 classes.
```

Preprocessing the Test set

In [4]:

```
test_datagen = ImageDataGenerator(rescale=1./255)

test_set = test_datagen.flow_from_directory(
        'dataset/test_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
```

Building the CNN

Initialising the CNN

In [5]:

```
cnn = tf.keras.models.Sequential()
```

Convolution

Convolution

In [6]:

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape
=[64, 64, 3]))
```

Pooling

In [7]:

```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Adding a second convolutional layer

In [8]:

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Flattening

In [9]:

```
cnn.add(tf.keras.layers.Flatten())
```

Full Connection

In [10]:

```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

Output Layer

In [11]:

```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# sigmoid for binary classification and softmax for multiple classification
```

Training the CNN

Compiling the CNN

In [12]:

```
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training the CNN on the Training set and evaluating it on the Test set

In [25]:

```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
Epoch 1/25
250/250 [==============================] - 359s 1s/step - loss: 0.2279 - accuracy: 0.9087
- val_loss: 0.5157 - val_accuracy: 0.7960
Epoch 2/25
250/250 [==============================] - 126s 505ms/step - loss: 0.2157 - accuracy: 0.9
120 - val_loss: 0.5451 - val_accuracy: 0.7885
Epoch 3/25
```

```
250/250 [==============================] - 46s 182ms/step - loss: 0.1994 - accuracy: 0.92
04 - val_loss: 0.6090 - val_accuracy: 0.7885
Epoch 4/25
250/250 [==============================] - 47s 186ms/step - loss: 0.1957 - accuracy: 0.92
10 - val_loss: 0.6078 - val_accuracy: 0.7895
Epoch 5/25
250/250 [==============================] - 45s 178ms/step - loss: 0.1888 - accuracy: 0.92
25 - val_loss: 0.5735 - val_accuracy: 0.7895
Epoch 6/25
250/250 [==============================] - 46s 186ms/step - loss: 0.1883 - accuracy: 0.92
45 - val_loss: 0.5913 - val_accuracy: 0.7890
Epoch 7/25
250/250 [==============================] - 44s 175ms/step - loss: 0.1671 - accuracy: 0.93
44 - val_loss: 0.6003 - val_accuracy: 0.7880
Epoch 8/25
250/250 [==============================] - 45s 178ms/step - loss: 0.1626 - accuracy: 0.93
54 - val_loss: 0.6657 - val_accuracy: 0.7780
Epoch 9/25
250/250 [==============================] - 43s 171ms/step - loss: 0.1605 - accuracy: 0.93
24 - val_loss: 0.6639 - val_accuracy: 0.7845
Epoch 10/25
250/250 [==============================] - 49s 197ms/step - loss: 0.1467 - accuracy: 0.93
95 - val_loss: 0.7348 - val_accuracy: 0.7825
Epoch 11/25
250/250 [==============================] - 49s 194ms/step - loss: 0.1414 - accuracy: 0.94
56 - val_loss: 0.7184 - val_accuracy: 0.7905
Epoch 12/25
250/250 [==============================] - 53s 212ms/step - loss: 0.1302 - accuracy: 0.95
11 - val_loss: 0.6932 - val_accuracy: 0.7950
Epoch 13/25
250/250 [==============================] - 49s 195ms/step - loss: 0.1340 - accuracy: 0.94
84 - val_loss: 0.7560 - val_accuracy: 0.7815
Epoch 14/25
250/250 [==============================] - 50s 199ms/step - loss: 0.1291 - accuracy: 0.95
16 - val_loss: 0.6953 - val_accuracy: 0.8020
Epoch 15/25
250/250 [==============================] - 48s 193ms/step - loss: 0.1233 - accuracy: 0.95
07 - val_loss: 0.7381 - val_accuracy: 0.7935
Epoch 16/25
250/250 [==============================] - 47s 189ms/step - loss: 0.1130 - accuracy: 0.95
88 - val_loss: 0.7527 - val_accuracy: 0.7920
Epoch 17/25
250/250 [==============================] - 48s 191ms/step - loss: 0.1132 - accuracy: 0.95
63 - val_loss: 0.7393 - val_accuracy: 0.8070
Epoch 18/25
250/250 [==============================] - 47s 188ms/step - loss: 0.1174 - accuracy: 0.95
46 - val_loss: 0.8071 - val_accuracy: 0.7855
Epoch 19/25
250/250 [==============================] - 47s 190ms/step - loss: 0.1150 - accuracy: 0.95
71 - val_loss: 0.8004 - val_accuracy: 0.7810
Epoch 20/25
250/250 [==============================] - 48s 191ms/step - loss: 0.1174 - accuracy: 0.95
82 - val_loss: 0.7316 - val_accuracy: 0.7890
Epoch 21/25
250/250 [==============================] - 50s 201ms/step - loss: 0.1002 - accuracy: 0.96
36 - val_loss: 0.7959 - val_accuracy: 0.7945
Epoch 22/25
250/250 [==============================] - 49s 194ms/step - loss: 0.0934 - accuracy: 0.96
68 - val_loss: 0.7661 - val_accuracy: 0.7950
Epoch 23/25
250/250 [==============================] - 47s 189ms/step - loss: 0.0938 - accuracy: 0.96
63 - val_loss: 0.7641 - val_accuracy: 0.7920
Epoch 24/25
250/250 [==============================] - 47s 190ms/step - loss: 0.0952 - accuracy: 0.96
51 - val_loss: 0.8607 - val_accuracy: 0.7820
Epoch 25/25
250/250 [==============================] - 48s 191ms/step - loss: 0.0916 - accuracy: 0.96
36 - val_loss: 0.8931 - val_accuracy: 0.7880
```

Out[25]:

```
<tensorflow.python.keras.callbacks.History at 0x1c02232b610>
```

Making a prediction

In [45]:

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_3.jfif', target_size =
(64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'Dog'
else:
  prediction = 'Cat'
```

In [46]:

```
print(prediction)
```

```
Cat
```

In [47]:

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (
64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'Dog'
else:
  prediction = 'Cat'
```

In [48]:

```
print(prediction)
```

```
Dog
```

In [ ]:

```

```