convolutional_neural_network

June 21, 2021

1 Convolutional Neural Network

Work by: Raghavendra Tapas

1.1 Importing the libraries

Keras is wrapper on Tensorflow which contains ImageDataGenerator

[2]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

[3]:
```
tf.__version__
```

[3]:
```
'2.5.0'
```

1.2 Part 1 - Data Preprocessing

1.2.1 Pre-Processing the Training Set

Source: We need to augument/transform the images, so that the machine doesn't overlearn the images.

Keras Deep Learning Library: https://keras.io/api/preprocessing/

- **Feature Scaling:** Each pixel takes a value between 0 to 255. So we need to divide each pixel by 255.
- **Image Transformation:** Shear, zoom and horizontal flips are the image transformations.
- **target_size:** Final size of the images before going into the training.
- **batch_size:** 32 batch size in the batch gradient descent.
- **class_mode:** In this specific case, we are trying to predict whether image is cat or dog. Therefore `binary`.

[4]:
```
train_datagen = ImageDataGenerator(rescale = 1./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.
```

1.2.2 Preprocessing the Test set

[5]:
```
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.
```

1.3 Part 2 - Building the CNN

1.3.1 Initialising the CNN

[8]:
```
cnn = tf.keras.models.Sequential()
```

1.3.2 Step 1 - Adding First Convolution layer

- `filters`: number of feature detectors
- `activation`: rectifier activation function for all layers except output layers. Reduces linearity.
- `input_shape`: 64*64 image size, 3 if RGB, 2 if black and white

[9]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',␣
↪input_shape=[64, 64, 3]))
```

1.3.3 Step 2 - Pooling

Pooling reduces the information to be processed and yet be able to detect the required features. We are also reducing the number of parameters. * `pool_size`: *(22) size of the frame, note that we only specify width (i.e. pool_size = 2)* `strides`: the frame shifts to the right by 2 pixels.

[10]:
```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

1.3.4 Adding a second convolutional layer

[11]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

1.3.5 Step 3 - Flattening

Flattening creates a 1 dimensional vector. Flattening of a `64*64` image matrix into a `4096 * 1` vector.

[13]:
```
cnn.add(tf.keras.layers.Flatten())
```

1.3.6 Step 4 - Full Connection

- units = hidden number of neurons, 128
- activation function used: Re

[15]:
```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

1.3.7 Step 5 - Output Layer

- Binary classification - `sigmoid` .i.e. Cat or Dog?
- multi-class classification - softmax

[ ]:
```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

1.4 Part 3 - Training the CNN

1.4.1 Compiling the CNN

- `Adam optimizer:` Adaptive Moment Estimation or Adam optimizer combines two gradient descent methodologies. Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters.
- `Binary Cross-Entropy:` Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. Since we're trying to compute a loss, we need to penalize bad predictions. If the probability associated with the true class is 1.0, we need its loss to be zero. Conversely, if that probability is low, say, 0.01, we need its loss to be huge.

[17]:
```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
```

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

- x = training set
- validation_data = test_set
- epochs = 25 After trying 10, 15, 20... the accuracies were not up to the par. So 25 epochs.

[18]:
```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
Epoch 1/25
250/250 [==============================] - 370s 1s/step - loss: 2.0391 -
accuracy: 6.2500e-04 - val_loss: 1.8876 - val_accuracy: 5.0000e-04
Epoch 2/25
250/250 [==============================] - 80s 322ms/step - loss: 1.5692 -
accuracy: 0.0146 - val_loss: 1.4550 - val_accuracy: 0.0830
Epoch 3/25
250/250 [==============================] - 80s 319ms/step - loss: 1.3618 -
accuracy: 0.0056 - val_loss: 1.2811 - val_accuracy: 0.0000e+00
Epoch 4/25
250/250 [==============================] - 81s 326ms/step - loss: 1.0260 -
```

```
accuracy: 0.0019 - val_loss: 0.9740 - val_accuracy: 0.0000e+00
Epoch 5/25
250/250 [==============================] - 82s 328ms/step - loss: 1.0070 -
accuracy: 0.0000e+00 - val_loss: 1.0285 - val_accuracy: 0.0000e+00
Epoch 6/25
250/250 [==============================] - 85s 341ms/step - loss: 0.9786 -
accuracy: 0.0012 - val_loss: 0.9056 - val_accuracy: 0.0000e+00
Epoch 7/25
250/250 [==============================] - 75s 299ms/step - loss: 0.9135 -
accuracy: 5.0000e-04 - val_loss: 0.9512 - val_accuracy: 0.0000e+00
Epoch 8/25
250/250 [==============================] - 69s 275ms/step - loss: 0.9050 -
accuracy: 5.0000e-04 - val_loss: 0.9492 - val_accuracy: 0.0000e+00
Epoch 9/25
250/250 [==============================] - 65s 260ms/step - loss: 0.8848 -
accuracy: 7.5000e-04 - val_loss: 0.8684 - val_accuracy: 0.0015
Epoch 10/25
250/250 [==============================] - 80s 321ms/step - loss: 0.8411 -
accuracy: 0.0103 - val_loss: 0.7956 - val_accuracy: 0.0210
Epoch 11/25
250/250 [==============================] - 94s 374ms/step - loss: 0.7724 -
accuracy: 0.0077 - val_loss: 0.7898 - val_accuracy: 0.0015
Epoch 12/25
250/250 [==============================] - 57s 229ms/step - loss: 0.7534 -
accuracy: 0.0046 - val_loss: 0.6835 - val_accuracy: 0.0035
Epoch 13/25
250/250 [==============================] - 57s 227ms/step - loss: 0.6960 -
accuracy: 0.0020 - val_loss: 0.7164 - val_accuracy: 0.0040
Epoch 14/25
250/250 [==============================] - 57s 228ms/step - loss: 0.7236 -
accuracy: 0.0024 - val_loss: 0.7092 - val_accuracy: 0.0000e+00
Epoch 15/25
250/250 [==============================] - 57s 230ms/step - loss: 0.7258 -
accuracy: 0.0191 - val_loss: 0.7190 - val_accuracy: 5.0000e-04
Epoch 16/25
250/250 [==============================] - 59s 236ms/step - loss: 0.7545 -
accuracy: 0.0096 - val_loss: 0.7104 - val_accuracy: 0.0050
Epoch 17/25
250/250 [==============================] - 58s 232ms/step - loss: 0.7163 -
accuracy: 0.0043 - val_loss: 0.7162 - val_accuracy: 0.0205
Epoch 18/25
250/250 [==============================] - 57s 226ms/step - loss: 0.6989 -
accuracy: 0.0063 - val_loss: 0.7361 - val_accuracy: 0.0035
Epoch 19/25
250/250 [==============================] - 58s 232ms/step - loss: 0.6815 -
accuracy: 0.0060 - val_loss: 0.7391 - val_accuracy: 0.0015
Epoch 20/25
250/250 [==============================] - 58s 233ms/step - loss: 0.6895 -
```

```
accuracy: 0.0045 - val_loss: 0.6723 - val_accuracy: 0.0070
Epoch 21/25
250/250 [==============================] - 57s 229ms/step - loss: 0.6630 -
accuracy: 0.0039 - val_loss: 0.6776 - val_accuracy: 0.0140
Epoch 22/25
250/250 [==============================] - 58s 230ms/step - loss: 0.6666 -
accuracy: 0.0035 - val_loss: 0.6736 - val_accuracy: 0.0000e+00
Epoch 23/25
250/250 [==============================] - 58s 233ms/step - loss: 0.6401 -
accuracy: 0.0019 - val_loss: 0.6657 - val_accuracy: 0.0000e+00
Epoch 24/25
250/250 [==============================] - 58s 232ms/step - loss: 0.6790 -
accuracy: 0.0030 - val_loss: 0.7189 - val_accuracy: 0.0000e+00
Epoch 25/25
250/250 [==============================] - 59s 238ms/step - loss: 0.6916 -
accuracy: 2.5000e-04 - val_loss: 0.6885 - val_accuracy: 5.0000e-04
```

[18]:
```
<tensorflow.python.keras.callbacks.History at 0x230c09eacd0>
```

Epoch 25/25 250/250 [==============================] - 59s 238ms/step - loss: 0.6916 - accuracy: 2.5000e-04 - val_loss: 0.6885 - val_accuracy: 5.0000e-04

1.5 Part 4 - Making a single prediction

- `numpy`: to batch the test image.
- 1 –> Dog and 0 –> Cat

[38]:
```
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/255.0)
training_set.class_indices

if result[0][0]  > 0.5:
  prediction = 'It\'s a Dog'
else:
  prediction = 'It\'s a Cat'
```

1.6 Predicting Dog

1. cat_or_dog_1.jpg

[39]:
```
print(prediction)
```

```
It's a Dog
```

[43]:
```
## Function that takes path of image file as input and tells whether it is a␣
↪cat or dog.
def predictImage(path):
    test_image = image.load_img(path, 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/255.0)
    training_set.class_indices

    if result[0][0]  > 0.5:
      prediction = 'It\'s a Dog'
    else:
      prediction = 'It\'s a Cat'

    print(prediction)
```

1.7 Predicting Cat

We normalize the image by dividing the image array by 255.

2. cat_or_dog_2.jpg

[42]:
```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_2.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/255.0)
training_set.class_indices

if result[0][0]  > 0.5:
  prediction = 'It\'s a Dog'
else:
  prediction = 'It\'s a Cat'

print(prediction)
```

```
It's a Cat
```

[44]:
```
predictImage('dataset/single_prediction/cat_or_dog_2.jpg')
```

```
It's a Cat
```

1.8 Predicting 3 cats just for fun

2. cat_or_dog_3.jpg

[46]:
```
predictImage('dataset/single_prediction/cat_or_dog_3.jpg')
```

```
It's a Cat
```

1.9 What if its a Dog and a Cat?

2. cat_or_dog_4.jpg

[50]:
```
predictImage('dataset/single_prediction/cat_or_dog_4.jpg')
```

```
It's a Dog
```

[ ]:
```

```