In [1]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Dropout, Conv2D, MaxPooling2D, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop

import numpy as np
import tensorflow as tf

tf.get_logger().setLevel('ERROR')
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade (https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

In [0]:
```
num_classes = 10
```

In [3]:
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype(np.float32) / 255
x_test = x_test.astype(np.float32) / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

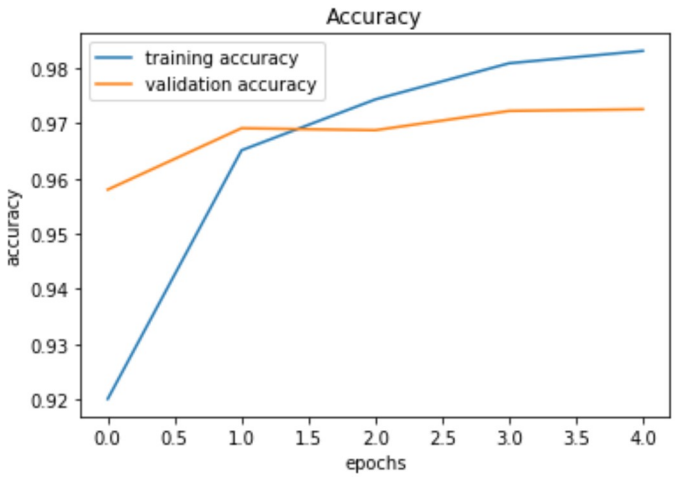
Exercise 2

In [0]:
```
model = Sequential()
model.add(Flatten(input_shape=(28,28,1)))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(Adam(lr=0.001), loss=keras.losses.categorical_crossentropy, metrics=
['accuracy'])
```

In [5]:
```
history = model.fit(x_train, y_train,
          batch_size=64,
          epochs=5,
          verbose=1,
          validation_split=0.33)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 40199 samples, validate on 19801 samples
Epoch 1/5
40199/40199 [==============================] - 4s 105us/step - loss: 0.2615 - ac
c: 0.9201 - val_loss: 0.1388 - val_acc: 0.9580
Epoch 2/5
40199/40199 [==============================] - 3s 71us/step - loss: 0.1133 - ac
c: 0.9651 - val_loss: 0.1050 - val_acc: 0.9691
Epoch 3/5
40199/40199 [==============================] - 3s 70us/step - loss: 0.0798 - ac
c: 0.9743 - val_loss: 0.1008 - val_acc: 0.9687
Epoch 4/5
40199/40199 [==============================] - 3s 72us/step - loss: 0.0607 - ac
c: 0.9808 - val_loss: 0.1013 - val_acc: 0.9722
Epoch 5/5
40199/40199 [==============================] - 3s 70us/step - loss: 0.0532 - ac
c: 0.9831 - val_loss: 0.0963 - val_acc: 0.9725
Test loss: 0.08604762646773888
Test accuracy: 0.9732
```

In [6]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'], label='training accuracy')
plt.plot(history.history['val_acc'], label='validation accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



Exercise 3

In [0]:
```
model2 = Sequential()
model2.add(Conv2D(32, (3, 3), padding='same',
                  input_shape=x_train.shape[1:]))
model2.add(Activation('relu'))
model2.add(Dropout(0.2))
model2.add(Conv2D(64, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(128, (3, 3), padding='same'))
model2.add(Activation('relu'))
model2.add(Dropout(0.2))
model2.add(Conv2D(128, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(10))
model2.add(Activation('softmax'))

model2.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])
```

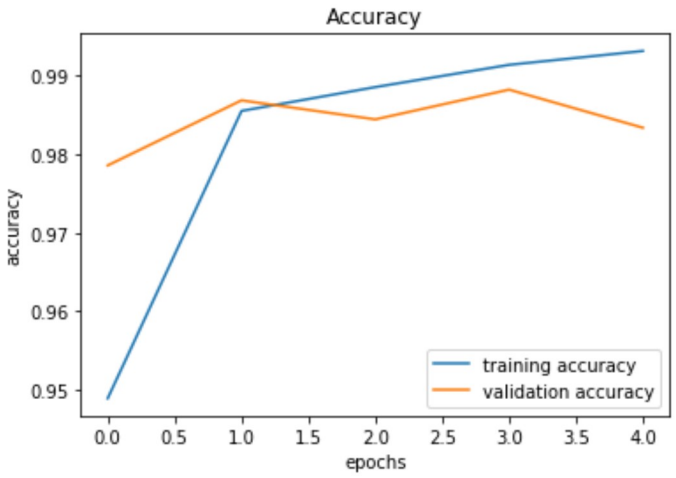
In [8]:
```
model.summary()
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
activation_4 (Activation)    (None, 28, 28, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 26, 64)        18496
_________________________________________________________________
activation_5 (Activation)    (None, 26, 26, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
activation_6 (Activation)    (None, 13, 13, 128)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 128)       147584
_________________________________________________________________
activation_7 (Activation)    (None, 11, 11, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                32010
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
=================================================================
Total params: 272,266
Trainable params: 272,266
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
history = model2.fit(x_train, y_train,
          batch_size=64,
          epochs=5,
          verbose=1,
          validation_split=0.33)
score = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 40199 samples, validate on 19801 samples
Epoch 1/5
40199/40199 [==============================] - 8s 187us/step - loss: 0.1674 - ac
c: 0.9489 - val_loss: 0.0710 - val_acc: 0.9786
Epoch 2/5
40199/40199 [==============================] - 6s 142us/step - loss: 0.0489 - ac
c: 0.9855 - val_loss: 0.0448 - val_acc: 0.9869
Epoch 3/5
40199/40199 [==============================] - 6s 142us/step - loss: 0.0357 - ac
c: 0.9886 - val_loss: 0.0492 - val_acc: 0.9844
Epoch 4/5
40199/40199 [==============================] - 6s 142us/step - loss: 0.0261 - ac
c: 0.9914 - val_loss: 0.0394 - val_acc: 0.9882
Epoch 5/5
40199/40199 [==============================] - 6s 143us/step - loss: 0.0204 - ac
c: 0.9932 - val_loss: 0.0600 - val_acc: 0.9834
Test loss: 0.04129169438488316
Test accuracy: 0.9856
```

In [10]:
```
plt.plot(history.history['acc'], label='training accuracy')
plt.plot(history.history['val_acc'], label='validation accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



Exercise 4

Calculation for Conv Layers: (Kernel_Width *Kernel_Height* Input_Channels + 1) * Number_Filters \ // + 1 is for biases

Calculation for Dense Layers: Input_Dim * Output_Dim

Exercise 5

CNNs are better suited for large image data as the amount of parameters of conv layers does not depend on the spatial input dimensions (in contrast to dense layers)

Exercise 6

Strided Convolutions: Stride defines the spatial step-size of the kernel of a convolutional layer. For a stride n > 1 (1 would be the default convolution) the kernel is only evaluated in every nth position thus resulting in a smaller spatial output dimension.

Max Pooling: Returns the single largest value within its receptive field. For non- or not fully overlapping receptive fields (e.g. use of stride) and a pool size > 1x1 the output will be smaller than its input.

CIFAR10

In [11]:
```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype(np.float32) / 255
x_test = x_test.astype(np.float32) / 255

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

In [0]:
```
model3 = Sequential()
model3.add(Conv2D(input_shape=x_train[0,:,:,:].shape, filters=32, kernel_size=(3,
3), padding="same"))
model3.add(BatchNormalization())
model3.add(Activation('relu'))
model3.add(Conv2D(filters=64, kernel_size=(3,3), padding="same"))
model3.add(BatchNormalization())
model3.add(Activation('relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Conv2D(filters=64, kernel_size=(3,3), padding="same"))
model3.add(BatchNormalization())
model3.add(Activation('relu'))
model3.add(Conv2D(filters=64, kernel_size=(3,3), padding="same"))
model3.add(BatchNormalization())
model3.add(Activation('relu'))
model3.add(Flatten())
model3.add(Dropout(0.5))
model3.add(Dense(128))
model3.add(Activation('relu'))
model3.add(Dense(num_classes, activation="softmax"))
model3.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=
['accuracy'])
```

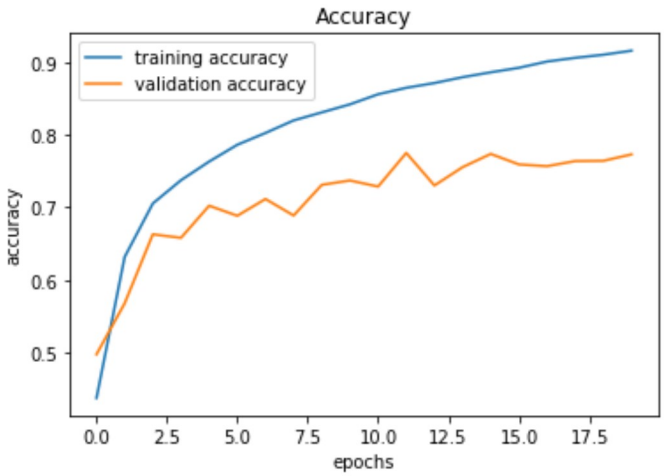
In [14]:
```
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
activation_9 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
activation_10 (Activation)   (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
activation_11 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
activation_12 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 16384)             0
_________________________________________________________________
dropout_5 (Dropout)          (None, 16384)             0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               2097280
_________________________________________________________________
activation_13 (Activation)   (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,192,714
Trainable params: 2,192,266
Non-trainable params: 448
_________________________________________________________________
```

In [15]:
```
history = model3.fit(x_train, y_train,
          batch_size=128,
          epochs=20,
          verbose=1,
          validation_split=0.16)
score = model3.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 42000 samples, validate on 8000 samples
Epoch 1/20
42000/42000 [==============================] - 7s 164us/step - loss: 1.6904 - ac
c: 0.4372 - val_loss: 1.3672 - val_acc: 0.4974
Epoch 2/20
42000/42000 [==============================] - 6s 134us/step - loss: 1.0367 - ac
c: 0.6316 - val_loss: 1.3962 - val_acc: 0.5676
Epoch 3/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.8372 - ac
c: 0.7052 - val_loss: 0.9667 - val_acc: 0.6627
Epoch 4/20
42000/42000 [==============================] - 6s 138us/step - loss: 0.7372 - ac
c: 0.7373 - val_loss: 1.0293 - val_acc: 0.6579
Epoch 5/20
42000/42000 [==============================] - 6s 135us/step - loss: 0.6639 - ac
c: 0.7630 - val_loss: 0.8704 - val_acc: 0.7021
Epoch 6/20
42000/42000 [==============================] - 6s 137us/step - loss: 0.6017 - ac
c: 0.7861 - val_loss: 0.9566 - val_acc: 0.6883
Epoch 7/20
42000/42000 [==============================] - 6s 138us/step - loss: 0.5589 - ac
c: 0.8024 - val_loss: 0.8497 - val_acc: 0.7114
Epoch 8/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.5155 - ac
c: 0.8197 - val_loss: 1.0059 - val_acc: 0.6886
Epoch 9/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.4755 - ac
c: 0.8307 - val_loss: 0.7957 - val_acc: 0.7309
Epoch 10/20
42000/42000 [==============================] - 6s 137us/step - loss: 0.4447 - ac
c: 0.8418 - val_loss: 0.7924 - val_acc: 0.7369
Epoch 11/20
42000/42000 [==============================] - 6s 137us/step - loss: 0.4089 - ac
c: 0.8557 - val_loss: 0.9105 - val_acc: 0.7285
Epoch 12/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.3834 - ac
c: 0.8646 - val_loss: 0.6806 - val_acc: 0.7749
Epoch 13/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.3604 - ac
c: 0.8711 - val_loss: 0.9370 - val_acc: 0.7300
Epoch 14/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.3393 - ac
c: 0.8792 - val_loss: 0.7943 - val_acc: 0.7552
Epoch 15/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.3182 - ac
c: 0.8860 - val_loss: 0.7162 - val_acc: 0.7735
Epoch 16/20
42000/42000 [==============================] - 6s 137us/step - loss: 0.2977 - ac
c: 0.8921 - val_loss: 0.8306 - val_acc: 0.7590
Epoch 17/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.2753 - ac
c: 0.9008 - val_loss: 0.7934 - val_acc: 0.7566
Epoch 18/20
42000/42000 [==============================] - 6s 137us/step - loss: 0.2644 - ac
c: 0.9059 - val_loss: 0.8136 - val_acc: 0.7636
Epoch 19/20
42000/42000 [==============================] - 6s 135us/step - loss: 0.2505 - ac
c: 0.9102 - val_loss: 0.8338 - val_acc: 0.7640
Epoch 20/20
42000/42000 [==============================] - 6s 136us/step - loss: 0.2338 - ac
c: 0.9158 - val_loss: 0.7986 - val_acc: 0.7729
Test loss: 0.8293685411453247
Test accuracy: 0.7715
```

In [16]:
```
plt.plot(history.history['acc'], label='training accuracy')
plt.plot(history.history['val_acc'], label='validation accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



In [0]:
```
```