Begin by creating the ipynb file in the same parent folder where the downloaded data set is kept. The CNN model should have the following layers: ● Input layer ● Convolutional layer 1 with 32 filters of kernel size[5,5] ● Pooling layer 1 with pool size[2,2] and stride 2 ● Convolutional layer 2 with 64 filters of kernel size[5,5] ● Pooling layer 2 with pool size[2,2] and stride 2 ● Dense layer whose output size is fixed in the hyper parameter: fc_size=32 ● Dropout layer with dropout probability 0.4 Predict the class by doing a softmax on the output of the dropout layers. This should be followed by training and evaluation: ● For the training step, define the loss function and minimize it ● For the evaluation step, calculate the accuracy Run the program for 100, 200, and 300 iterations, respectively. Follow this by a report on the final accuracy and loss on the evaluation data.

In [1]:

```
#import all the require labraries
```

In [2]:

```
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import models, layers
```

In [3]:

```
dataflow = ImageDataGenerator(rescale = 1.0 / 255.0 )
```

In [4]:

```
#import training data from file
```

In [5]:

```
train = dataflow.flow_from_directory('../input/dog-cat-classification/data/train', class_mo
```

In [6]:

```
#import test data from file
```

In [7]:

```
test = dataflow.flow_from_directory('../input/dog-cat-classification/data/test', class_mode
```

In [8]:

```
#build model according to
#Input layer ● Convolutional layer 1 with 32 filters of kernel size[5,5]
#Pooling layer 1 with pool size[2,2] and stride
#Convolutional layer 2 with 64 filters of kernel size[5,5]
#Pooling layer 2 with pool size[2,2] and stride 2
#Dense layer whose output size is fixed in the hyper parameter: fc_size=32
#Dropout layer with dropout probability 0.4 Predict the class by doing a softmax on the out
#
```

In [9]:

```
model = models.Sequential()
model.add( layers.Conv2D( 32, (5, 5 ), activation = 'relu', padding = 'same', input_shape =
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (5, 5 ), activation = 'relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.4))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation = 'relu'))
model.add(layers.Dense(2, activation = 'softmax'))
```

In [10]:

```
#For the training step, define the loss function and minimize it
```

In [11]:

```
sgd_opt = tf.keras.optimizers.SGD(lr = 0.001)
```

In [12]:

```
#compiling model
```

In [13]:

```
history=model.compile( optimizer = sgd_opt, loss = 'binary_crossentropy', metrics =  ['accu
```

In [14]:

```
#model train for 100 epochs
```

In [15]:

```
history =model.fit(train, validation_data = test, epochs =100)
```

In [16]:

```
test_loss, test_accuracy = model.evaluate(test)
```

In [17]:

```
test_loss
```

In [18]:

```
from matplotlib import pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc = 'upper right')
plt.show()
```

In [19]:

```
history =model.fit(train, validation_data = test, epochs =200)
```

In [20]:

```
test_loss, test_accuracy = model.evaluate(test)
```

In [21]:

```
test_loss
```

In [22]:

```
from matplotlib import pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc = 'upper right')
plt.show()
```

In [23]:

```
history =model.fit(train, validation_data = test, epochs =300)
```

In [24]:

```
from matplotlib import pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc = 'upper right')
plt.show()
```

In [25]:

```
test_loss, test_accuracy = model.evaluate(test)
```

In [26]:

```
test_loss
```