ARULKUMAR ARK

225229103

Lab4. Image corpus creation and binary classification using DNN

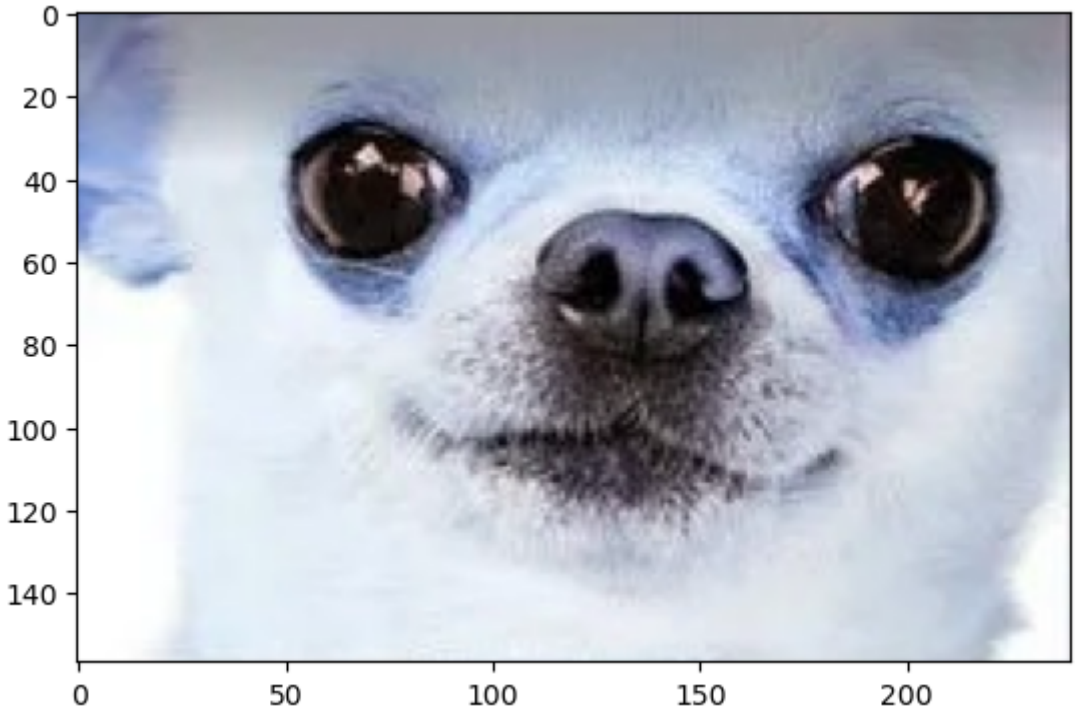
In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
```

Step-1: Dataset Creation

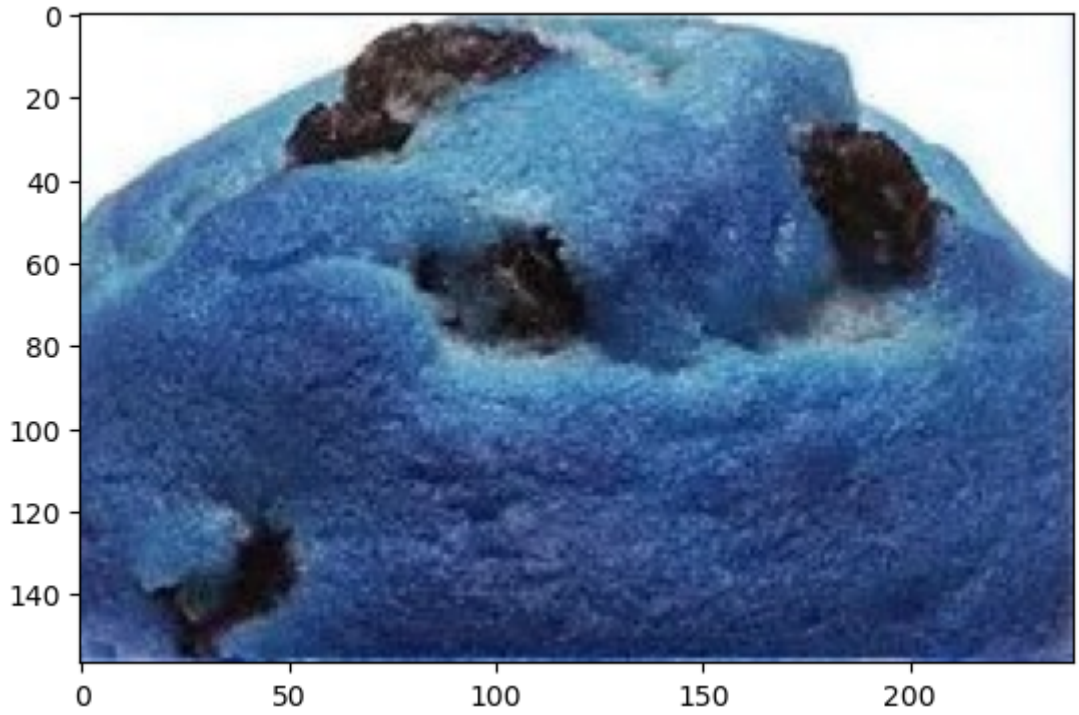
In [2]:

```
datadir ="D:\\notebooks\PDL"
categories =['dogs']

for category in categories:
    path = os.path.join(datadir, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))
        plt.imshow(img_array)
        plt.show()
        break
    break
```



In [3]:

```
datadir ="D:\\notebooks\PDL"
categories =['muffins']

for category in categories:
    path = os.path.join(datadir, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))
        plt.imshow(img_array)
        plt.show()
        break
    break
```



2. Pre-processing

In [4]:

```
data = []


for category in categories:
    path = os.path.join(datadir, category)
    class_num = categories.index(category)

    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        num_array = cv2.resize(img_array,(500, 500))

        data.append([num_array, class_num])
```

In [5]:

```
X = []
y = []

for features,label in data:
    X.append(features)
    y.append(label)

X = np.asarray(X).reshape(-1,500,500,1)
y = np.asarray(y)
```

3. Dataset Preparation

In [6]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4
```

4. Model Creation

In [7]:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(8, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [8]:

```
model.compile(loss='mse',optimizer='RMSprop',metrics=['binary_accuracy'])
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100)
```

```
Epoch 9/100
1/1 [==============================] - 0s 356ms/step - loss: 0.9975 - b
inary_accuracy: 2.4571e-05 - val_loss: 0.9985 - val_binary_accuracy: 2.
6667e-06
Epoch 10/100
1/1 [==============================] - 0s 390ms/step - loss: 0.9974 - b
inary_accuracy: 2.4571e-05 - val_loss: 0.9985 - val_binary_accuracy: 2.
6667e-06
Epoch 11/100
1/1 [==============================] - 0s 337ms/step - loss: 0.9974 - b
inary_accuracy: 2.4571e-05 - val_loss: 0.9985 - val_binary_accuracy: 2.
6667e-06
Epoch 12/100
1/1 [==============================] - 0s 334ms/step - loss: 0.9974 - b
inary_accuracy: 2.4571e-05 - val_loss: 0.9984 - val_binary_accuracy: 2.
6667e-06
Epoch 13/100
1/1 [==============================] - 0s 396ms/step - loss: 0.9973 - b
inary_accuracy: 2.4571e-05 - val_loss: 0.9984 - val_binary_accuracy: 2.
6667e-06
```

In [9]:

```
model.evaluate(X_train,y_train)
print(model.summary())
```

```
1/1 [==============================] - 0s 161ms/step - loss: 0.9924 - bina
ry_accuracy: 1.1029e-04
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 16

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 25 (100.00 Byte)
Trainable params: 25 (100.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

4. Performance Analysis

In [12]:

```
def training_data(heigth,width):
    data = []
    for category in categories:
        path = os.path.join(datadir,category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
            num_array=cv2.resize(img_array,(heigth,width))
            data.append([num_array,class_num])

    X = []
    y = []
    for features,label in data:
        X.append(features)
        y.append(label)
    X = np.asarray(X).reshape(-1,heigth,width,1)
    y = np.asarray(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_sta
    return X_train, X_test, y_train, y_test


def model_train(model):
    model.compile(loss='mse',optimizer='RMSprop',metrics=['binary_accuracy'])
    model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=1)
    return model.evaluate(X_test,y_test)
```

In [13]:

```
training_data(500,500)

model1 = Sequential()
model1.add(Dense(8, input_dim=1, activation='relu'))
model1.add(Dense(16,  activation='relu'))
model1.add(Dense(32,  activation='relu'))
model1.add(Dense(64,  activation='relu'))
model1.add(Dense(128, activation='relu'))
model1.add(Dense(256, activation='relu'))
model1.add(Dense(512, activation='relu'))
model1.add(Dense(1028,activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

model_train(model1)
```

```
1/1 [==============================] - 179s 179s/step - loss: 0.0799 - bin
ary_accuracy: 1.0000 - val_loss: 1.7196e-05 - val_binary_accuracy: 1.0000
1/1 [==============================] - 11s 11s/step - loss: 1.7196e-05 - b
inary_accuracy: 1.0000
```

Out[13]:

```
[1.71963529282948e-05, 1.0]
```

In [14]:

```
training_data(100,100)

model2 = Sequential()
model2.add(Dense(8, input_dim=1, activation='relu'))
model2.add(Dense(16,  activation='relu'))
model2.add(Dense(32,  activation='relu'))
model2.add(Dense(64,  activation='relu'))
model2.add(Dense(128, activation='relu'))
model2.add(Dense(256, activation='relu'))
model2.add(Dense(512, activation='relu'))
model2.add(Dense(1028,activation='relu'))
model2.add(Dense(1, activation='sigmoid'))

model_train(model2)
```

```
1/1 [==============================] - 218s 218s/step - loss: 0.2210 - bin
ary_accuracy: 1.0000 - val_loss: 6.1008e-04 - val_binary_accuracy: 1.0000
1/1 [==============================] - 13s 13s/step - loss: 6.1008e-04 - b
inary_accuracy: 1.0000
```

Out[14]:

```
[0.0006100849132053554, 1.0]
```

In [15]:

```
training_data(50,50)

model3 = Sequential()
model3.add(Dense(8, input_dim=1, activation='relu'))
model3.add(Dense(16,  activation='relu'))
model3.add(Dense(32,  activation='relu'))
model3.add(Dense(64,  activation='relu'))
model3.add(Dense(128, activation='relu'))
model3.add(Dense(256, activation='relu'))
model3.add(Dense(512, activation='relu'))
model3.add(Dense(1028,activation='relu'))
model3.add(Dense(1, activation='sigmoid'))

model_train(model3)
```

```
WARNING:tensorflow:5 out of the last 104 calls to <function Model.make_tra
in_function.<locals>.train_function at 0x000001BE03A5A680> triggered tf.fu
nction retracing. Tracing is expensive and the excessive number of tracing
s could be due to (1) creating @tf.function repeatedly in a loop, (2) pass
ing tensors with different shapes, (3) passing Python objects instead of t
ensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has reduce_retracing=True option that can avoid unnecess
ary retracing. For (3), please refer to https://www.tensorflow.org/guide/f
unction#controlling_retracing (https://www.tensorflow.org/guide/function#c
ontrolling_retracing) and https://www.tensorflow.org/api_docs/python/tf/fu
nction (https://www.tensorflow.org/api_docs/python/tf/function) for  more
details.
1/1 [==============================] - ETA: 0s - loss: 0.5989 - binary_acc
uracy: 2.4571e-05WARNING:tensorflow:5 out of the last 107 calls to <functi
on Model.make_test_function.<locals>.test_function at 0x000001BE03A5B7F0>
triggered tf.function retracing. Tracing is expensive and the excessive nu
mber of tracings could be due to (1) creating @tf.function repeatedly in a
loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can
avoid unnecessary retracing. For (3), please refer to https://www.tensorfl
ow.org/guide/function#controlling_retracing (https://www.tensorflow.org/gu
ide/function#controlling_retracing) and https://www.tensorflow.org/api_doc
s/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/functi
on) for  more details.
1/1 [==============================] - 193s 193s/step - loss: 0.5989 - bin
ary_accuracy: 2.4571e-05 - val_loss: 3.1122e-05 - val_binary_accuracy: 1.0
000
1/1 [==============================] - 12s 12s/step - loss: 3.1122e-05 - b
inary_accuracy: 1.0000
```

Out[15]:

```
[3.112168997176923e-05, 1.0]
```

In [16]:

```
training_data(25,25)
model4 = Sequential()
model4.add(Dense(8, input_dim=1, activation='relu'))
model4.add(Dense(16,  activation='relu'))
model4.add(Dense(32,  activation='relu'))
model4.add(Dense(64,  activation='relu'))
model4.add(Dense(128, activation='relu'))
model4.add(Dense(256, activation='relu'))
model4.add(Dense(512, activation='relu'))
model4.add(Dense(1028,activation='relu'))
model4.add(Dense(1, activation='sigmoid'))

model_train(model4)
```

```
WARNING:tensorflow:6 out of the last 105 calls to <function Model.make_tra
in_function.<locals>.train_function at 0x000001BE065D93F0> triggered tf.fu
nction retracing. Tracing is expensive and the excessive number of tracing
s could be due to (1) creating @tf.function repeatedly in a loop, (2) pass
ing tensors with different shapes, (3) passing Python objects instead of t
ensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has reduce_retracing=True option that can avoid unnecess
ary retracing. For (3), please refer to https://www.tensorflow.org/guide/f
unction#controlling_retracing (https://www.tensorflow.org/guide/function#c
ontrolling_retracing) and https://www.tensorflow.org/api_docs/python/tf/fu
nction (https://www.tensorflow.org/api_docs/python/tf/function) for  more
details.
1/1 [==============================] - ETA: 0s - loss: 0.3861 - binary_acc
uracy: 2.4571e-05WARNING:tensorflow:6 out of the last 109 calls to <functi
on Model.make_test_function.<locals>.test_function at 0x000001BE065DB400>
triggered tf.function retracing. Tracing is expensive and the excessive nu
mber of tracings could be due to (1) creating @tf.function repeatedly in a
loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can
avoid unnecessary retracing. For (3), please refer to https://www.tensorfl
ow.org/guide/function#controlling_retracing (https://www.tensorflow.org/gu
ide/function#controlling_retracing) and https://www.tensorflow.org/api_doc
s/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/functi
on) for  more details.
1/1 [==============================] - 228s 228s/step - loss: 0.3861 - bin
ary_accuracy: 2.4571e-05 - val_loss: 0.0019 - val_binary_accuracy: 1.0000
1/1 [==============================] - 17s 17s/step - loss: 0.0019 - binar
y_accuracy: 1.0000
```

Out[16]:

```
[0.0019273854559287429, 1.0]
```

In [17]:

```
training_data(10,10)

model5 = Sequential()
model5.add(Dense(8, input_dim=1, activation='relu'))
model5.add(Dense(16,  activation='relu'))
model5.add(Dense(32,  activation='relu'))
model5.add(Dense(64,  activation='relu'))
model5.add(Dense(128, activation='relu'))
model5.add(Dense(256, activation='relu'))
model5.add(Dense(512, activation='relu'))
model5.add(Dense(1028,activation='relu'))
model5.add(Dense(1, activation='sigmoid'))

model_train(model5)
```

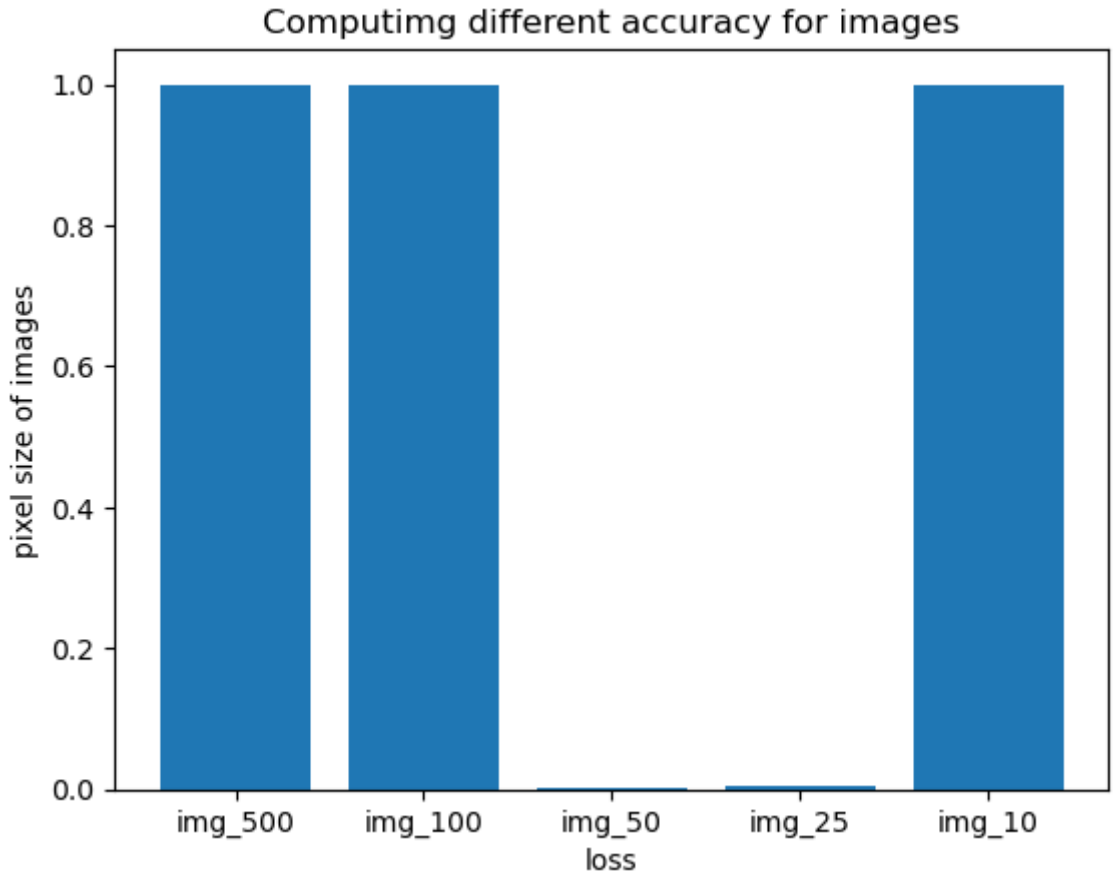
```
1/1 [==============================] - 208s 208s/step - loss: 0.0451 - bin
ary_accuracy: 1.0000 - val_loss: 5.7587e-05 - val_binary_accuracy: 1.0000
1/1 [==============================] - 12s 12s/step - loss: 5.7587e-05 - b
inary_accuracy: 1.0000
```

Out[17]:

```
[5.758739644079469e-05, 1.0]
```

In [27]:

```
bar={'img_500':1.0,'img_100':1.0,'img_50':0.0027,'img_25':0.0031,'img_10':1.0}
courses = list(bar.keys())
values = list(bar.values())

plt.bar(courses, values)

plt.xlabel("loss")
plt.ylabel("pixel size of images")
plt.title("Computimg different accuracy for images")
plt.show()
```



b.Different neural network

In [19]:

```
model2l =Sequential()
model2l.add(Dense(32,input_dim=1,activation='relu'))
model2l.add(Dense(32,input_dim=1,activation='relu'))
model2l.add(Dense(1,activation='sigmoid'))

model_train(model2l)
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.9169 - binary_
accuracy: 2.4571e-05 - val_loss: 0.0259 - val_binary_accuracy: 1.0000
1/1 [==============================] - 0s 159ms/step - loss: 0.0259 - bina
ry_accuracy: 1.0000
```

Out[19]:

```
[0.025924989953637123, 1.0]
```

In [21]:

```
model3l =Sequential()
model3l.add(Dense(32,input_dim=1,activation='relu'))
model3l.add(Dense(32,input_dim=1,activation='relu'))
model3l.add(Dense(32,input_dim=1,activation='relu'))
model3l.add(Dense(1,activation='sigmoid'))

model_train(model3l)
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.0014 - binary_
accuracy: 1.0000 - val_loss: 4.8188e-04 - val_binary_accuracy: 1.0000
1/1 [==============================] - 0s 202ms/step - loss: 4.8188e-04 -
binary_accuracy: 1.0000
```

Out[21]:

```
[0.00048188105574809015, 1.0]
```

In [23]:

```
model4l =Sequential()
model4l.add(Dense(32,input_dim=1,activation='relu'))
model4l.add(Dense(32,input_dim=1,activation='relu'))
model4l.add(Dense(32,input_dim=1,activation='relu'))
model4l.add(Dense(32,input_dim=1,activation='relu'))
model4l.add(Dense(1,activation='sigmoid'))

model_train(model4l)
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.9606 - binary_
accuracy: 2.4571e-05 - val_loss: 0.9380 - val_binary_accuracy: 2.6667e-06
1/1 [==============================] - 0s 252ms/step - loss: 0.9380 - bina
ry_accuracy: 2.6667e-06
```

Out[23]:

```
[0.9379870295524597, 2.6666666599339806e-06]
```

In [24]:

```
model5l =Sequential()
model5l.add(Dense(32,input_dim=1,activation='relu'))
model5l.add(Dense(32,input_dim=1,activation='relu'))
model5l.add(Dense(32,input_dim=1,activation='relu'))
model5l.add(Dense(32,input_dim=1,activation='relu'))
model5l.add(Dense(32,input_dim=1,activation='relu'))
model5l.add(Dense(1,activation='sigmoid'))

model_train(model5l)
```

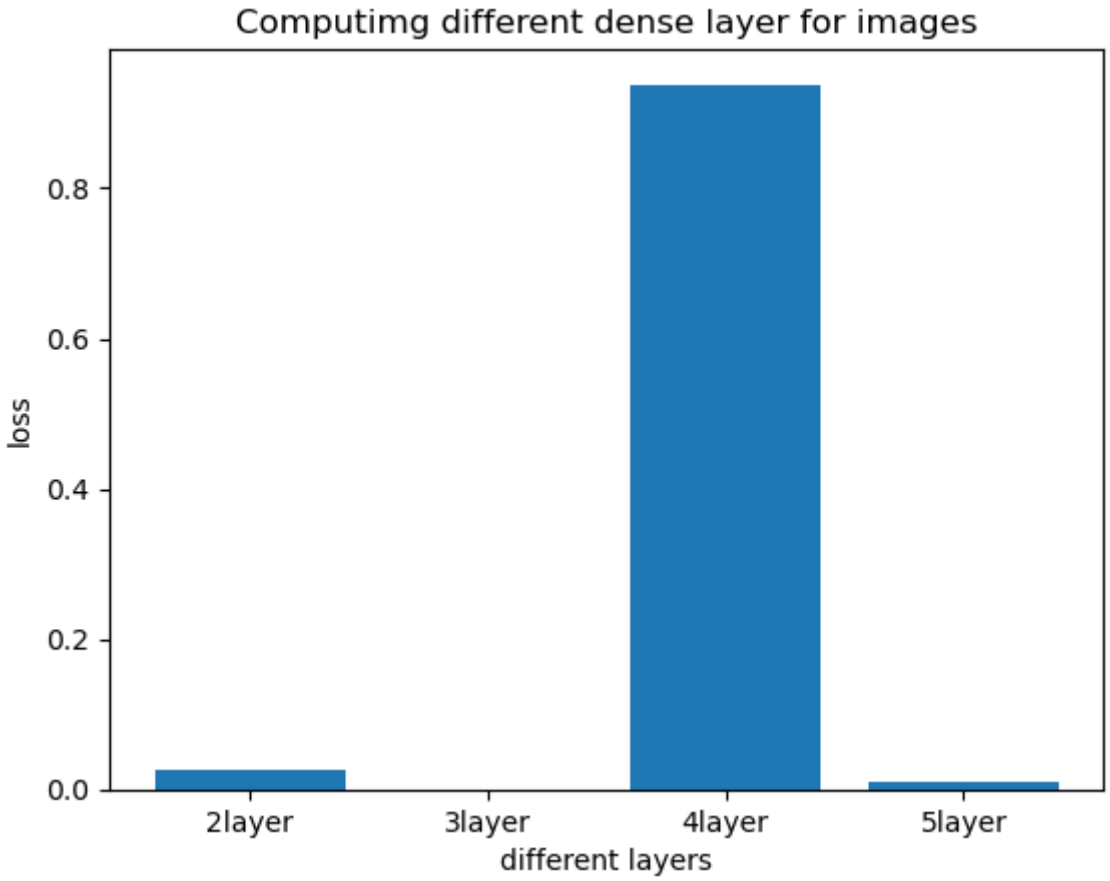
```
1/1 [==============================] - 3s 3s/step - loss: 0.5986 - binary_
accuracy: 2.4571e-05 - val_loss: 0.0100 - val_binary_accuracy: 1.0000
1/1 [==============================] - 0s 294ms/step - loss: 0.0100 - bina
ry_accuracy: 1.0000
```

Out[24]:

```
[0.01001710258424282, 1.0]
```

In [26]:

```
# creating the barset
bar={'2layer':0.025924,'3layer':0.00043,'4layer':0.9379,'5layer':0.01001}
courses = list(bar.keys())
values = list(bar.values())
# creating the bar plot
plt.bar(courses, values)

plt.xlabel("different layers")
plt.ylabel("loss")
plt.title("Computimg different dense layer for images")
plt.show()
```



In [ ]:

```

```