Convolutional Neural Networks

Basic Concepts

CNN (Convolutional Neural Networks): Neural networks specialized for applications in image and video recognition, recommender systems and natural language processing.

Three types of layers in a Convolutional Neural Network:

1. Convolutional Layers.
2. Pooling Layers.
3. Fully-Connected Layers.

Activation and Pooling Layers

Activation : Transformation to the output using Activation functional like ReLU

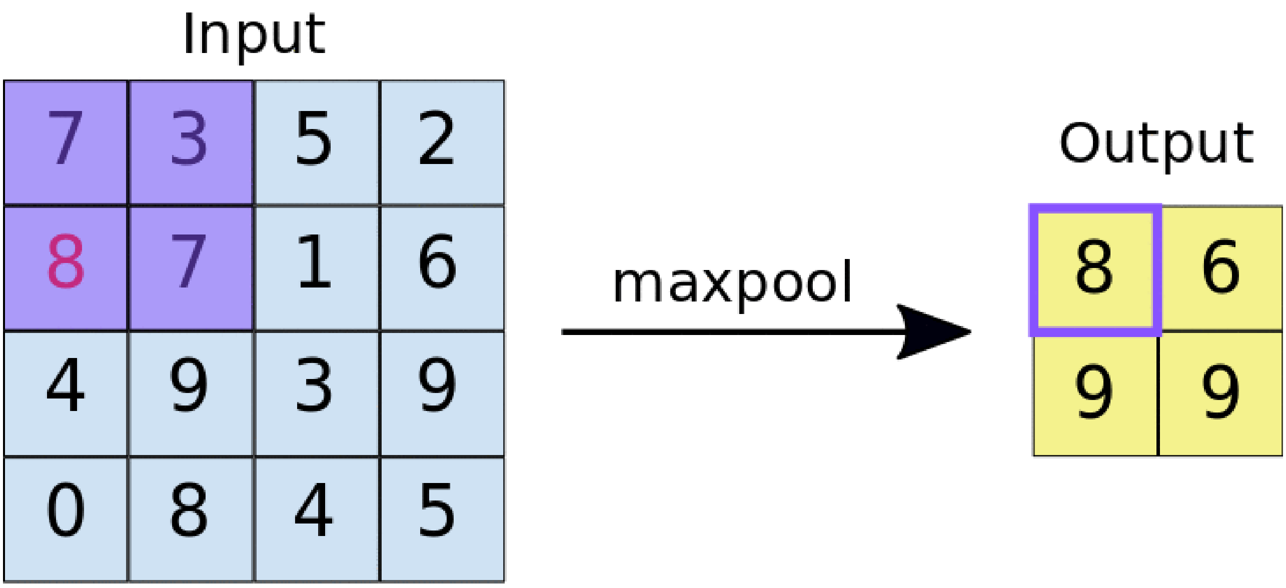
Pooling : The feature map dimensionality is reduced using pooling

Max Pooling and Min Pooling

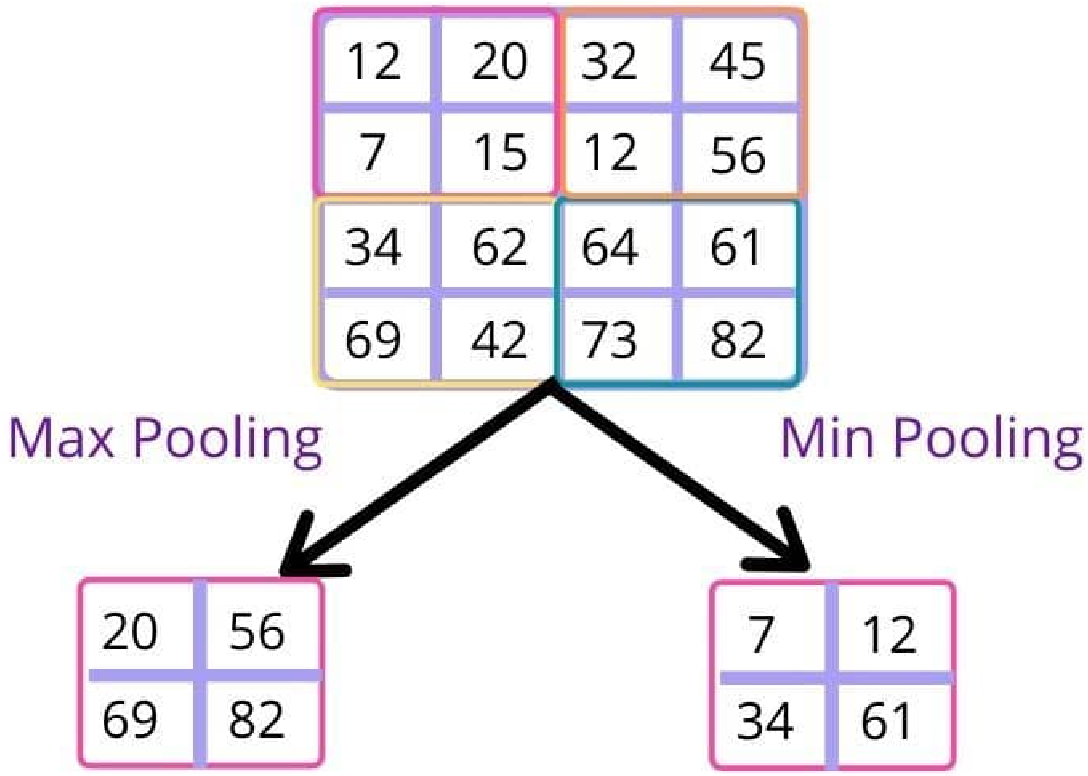
Pooling is done on the newluy created feature map. It's used to **down-sample** the feature map.

There are two different types of pooling:

- Max pooling : in each stride the **max** value within the window is pooled into an output matrix.



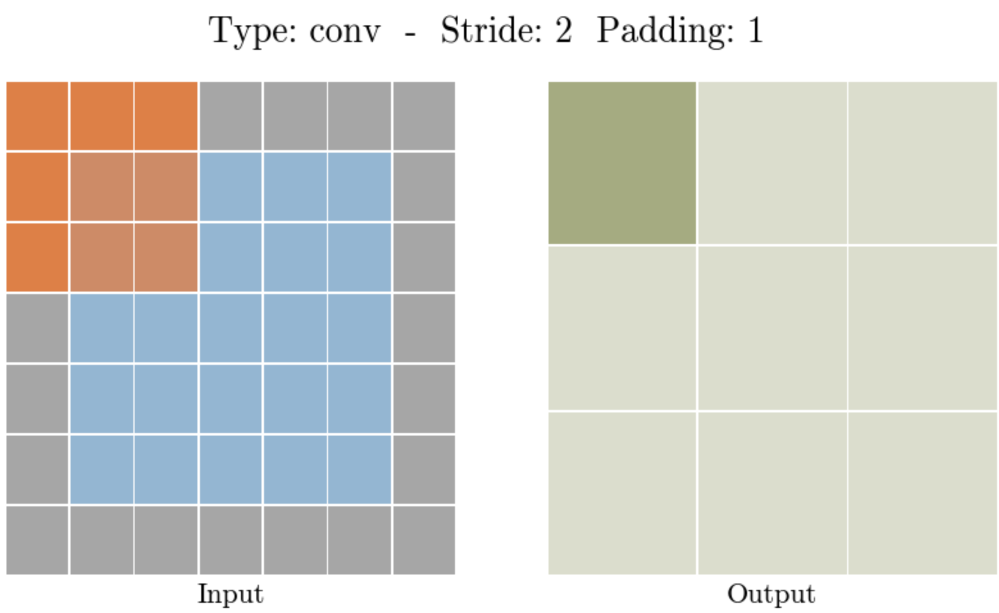
- Min pooling : in each stride the **min** value within the window is pooled into an output matrix.



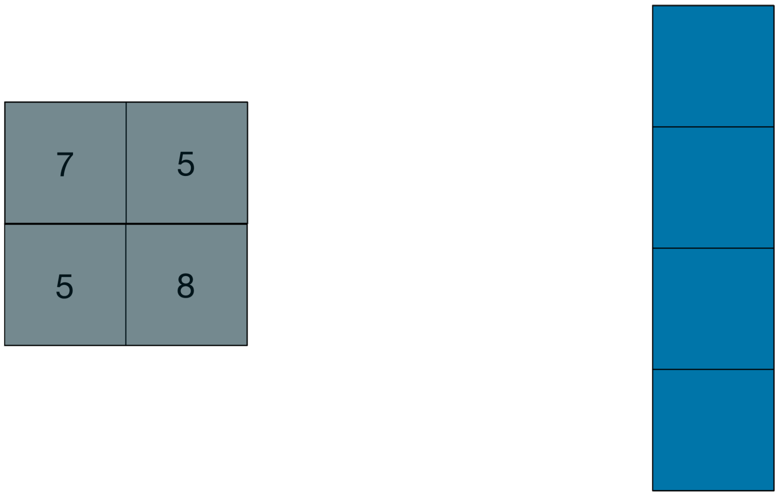
Stride Padding and Flattening Concepts of CNN

Stride : The amount of movement between the filter over the input image. The default is 1 x 1 which means the kernal will jump one unit to the right. Increasing the stride value will make the process of creating a feature map faster and decrease the size of it.

During convolution we tend to lose pixels on the perimeter of the image. To solve this problem we **pad the image** with zeros to allow more space for the kernal to cover the image.



After many convolutions and pooling operations, the 3D image is converted to a feature vector. This process is called **Flattening**



Building a CNN Model

Using a public dataset from kaggle [Flower Recognition Dataset (https://www.kaggle.com/datasets/alxmamaev/flowers-recognition)](https://www.kaggle.com/datasets/alxmamaev/flowers-recognition)

In [2]:
```
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from tensorflow.keras.utils import img_to_array, img_to_array , load_img
import matplotlib.pyplot as plt
```

2. Split data into training and testing

In [5]:
```
def split_data(files_dir, data_dir, class_name, train_ratio=0.0, val_ratio=0.0, test_
    from os import path, listdir
    if not path.exists(files_dir) or not path.exists(data_dir):
        print(f"Error @ split_data: one of the specified directories paths '{files_d
    elif (train_ratio + val_ratio + test_ratio != 1):
        print(f"Error @ split_data: the ratios of splitted data don't add up to 1")
    else:
        total_num_files = len(listdir(files_dir))
        train_num_files = round(total_num_files*train_ratio)
        val_num_files = round(total_num_files*val_ratio)
        test_num_files = round(total_num_files*test_ratio)

        if (train_num_files + val_num_files + test_num_files) > total_num_files:
            print("Error @ split_data: the total number of files in each split of da
                  "files, this could be the result of rounding up the number of files
                  f"Total number of files: {total_num_files}\n",
                  f"Number of files in train split: {train_num_files}\n",
                  f"Number of files in validation split: {val_num_files}\n",
                  f"Number of files in test split: {test_num_files}\n",
                  "Try changing your choice of ratios in splits, or number of files i
        else:
            #ensure there exist the needed folders
            create_dir(at=data_dir, name="train")
            create_dir(at=data_dir, name="validation")
            create_dir(at=data_dir, name="test")
            create_dir(at=path.join(data_dir, "train"), name=class_name)
            create_dir(at=path.join(data_dir, "validation"), name=class_name)
            create_dir(at=path.join(data_dir, "test"), name=class_name)
            all_files_path = []
            for file in listdir(files_dir):
                file_path = path.join(files_dir, file)
                assert path.exists(file_path)
                if path.isfile(file_path):
                    all_files_path.append(path.join(files_dir, file))
            from shutil import copy
            if (train_num_files + val_num_files + test_num_files) < total_num_files:
                train_num_files += 1
            if shuffle:
                from random import shuffle
                shuffle(all_files_path)
            i = 0
            while i < train_num_files:
                copy(src=all_files_path[i], dst=path.join(data_dir, "train", class_n
                i += 1
            print(f"successfully copied [{train_num_files}/{total_num_files}] traini
            j = 0
            while j < val_num_files:
                copy(src=all_files_path[i + j], dst=path.join(data_dir, "validation"
                j += 1
            print(f"successfully copied [{val_num_files}/{total_num_files}] validati
            k = 0
            while k <test_num_files:
                copy(src=all_files_path[j + k + i], dst=path.join(data_dir, "test",
                k += 1
            print(f"successfully copied [{test_num_files}/{total_num_files}] testing
```

In [6]:
```
split_data('dataset/flowers', 'dataset/flowers', ['daisy' , 'dandelion' , 'rose' , '
```

...

Define Bseline CNN Model

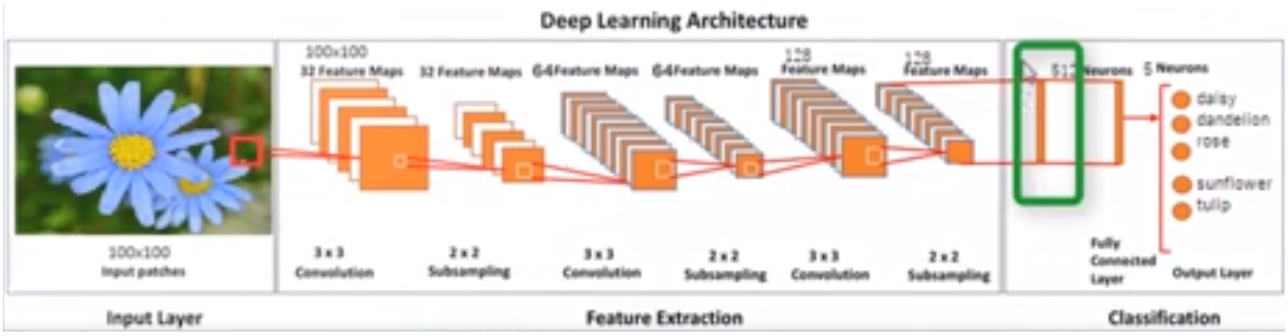
Convolutional Neural Network will contain the following layers:

`Convolution2D` : To make the convolutional network that deals with the images.

`Podling2D` : To add the pooling layers.

`Flatten` : Converts the pooled feature map to a single column to pass to the fully connected

`Dense` : Add the fully connected layer to the neural network:



In [8]:
```
from tensorflow import keras
from tensorflow.keras import layers
```

In [9]:
```
train_datagen = ImageDataGenerator(rescale = 1)
test_datagen = ImageDataGenerator(rescale = 1)
# 1 means there's no scaling
```

In [10]:
```
model = keras.Sequential()
```

Define `Conv2D` layer

In [11]:
```
model.add(layers.Conv2D(32 , (3,3) , activation = 'relu' , input_shape = (100,100,3)
# Cov2D (number_of_filters , (kernal_size) , activation_function , input_shape 'only
# number of filters increase with every conv layer
```

Define `MaxPool2D` layer

In [12]:
```
model.add(layers.MaxPool2D((2,2)))
```

In [13]:
```
model.add(layers.Conv2D(64 , (3,3) , activation = 'relu'))
```

In [14]:
```
model.add(layers.MaxPool2D((2,2)))
```

In [15]:
```
model.add(layers.Conv2D(128 , (3,3) , activation = 'relu'))
```

The start of the calssification

In [16]:
```
model.add(layers.MaxPool2D((2,2)))
```

In [17]:
```
model.add(layers.Flatten())
```

In [18]:
```
model.add(layers.Dense(512 , activation ='relu'))
```

In [19]:
```
model.add(layers.Dense(5 , activation = 'softmax'))
# 5 represent the number of classes because it's the last layer
```

Training and Visualization

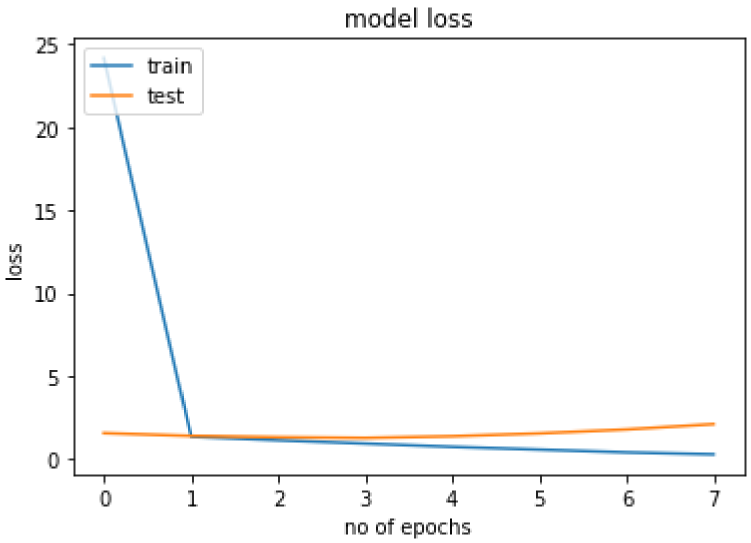
In [20]:
```
training_iterator = train_datagen.flow_from_directory('dataset/flowers/train' , batc
testing_iterator = test_datagen.flow_from_directory('dataset/flowers/test' , batch_s
```

```
Found 3117 images belonging to 5 classes.
Found 1200 images belonging to 5 classes.
```

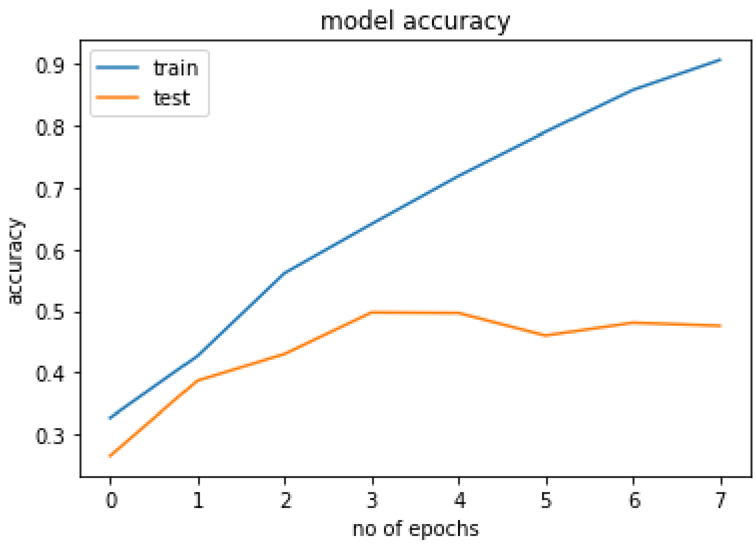
In [21]:
```
model.compile(loss = 'categorical_crossentropy' , metrics = ['accuracy'] , optimizer
history = model.fit(training_iterator , validation_data = testing_iterator , epochs
```

```
Epoch 1/8
49/49 [==============================] - 136s 3s/step - loss: 24.1725 - accuracy:
0.3266 - val_loss: 1.5767 - val_accuracy: 0.2650
Epoch 2/8
49/49 [==============================] - 84s 2s/step - loss: 1.3602 - accuracy: 0.4
267 - val_loss: 1.4084 - val_accuracy: 0.3867
Epoch 3/8
49/49 [==============================] - 51s 1s/step - loss: 1.1455 - accuracy: 0.5
614 - val_loss: 1.3301 - val_accuracy: 0.4300
Epoch 4/8
49/49 [==============================] - 60s 1s/step - loss: 0.9466 - accuracy: 0.6
413 - val_loss: 1.2859 - val_accuracy: 0.4975
Epoch 5/8
49/49 [==============================] - 49s 989ms/step - loss: 0.7482 - accuracy:
0.7193 - val_loss: 1.3876 - val_accuracy: 0.4967
Epoch 6/8
49/49 [==============================] - 50s 1s/step - loss: 0.5819 - accuracy: 0.7
905 - val_loss: 1.5598 - val_accuracy: 0.4600
Epoch 7/8
49/49 [==============================] - 50s 1s/step - loss: 0.4126 - accuracy: 0.8
585 - val_loss: 1.7991 - val_accuracy: 0.4808
Epoch 8/8
49/49 [==============================] - 43s 867ms/step - loss: 0.2959 - accuracy:
0.9073 - val_loss: 2.1199 - val_accuracy: 0.4758
```

In [22]:
```python
import matplotlib.pyplot as plt

#plot Loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [23]:
```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [24]:
```python
# Save the model
model.save('models/flower_baseline_model.h5')
```

In [25]:
```python
#get the class labels
class_labels = testing_iterator.class_indices
print(class_labels)
```

```
{'daisy': 0, 'dandelion': 1, 'rose': 2, 'sunflower': 3, 'tulip': 4}
```

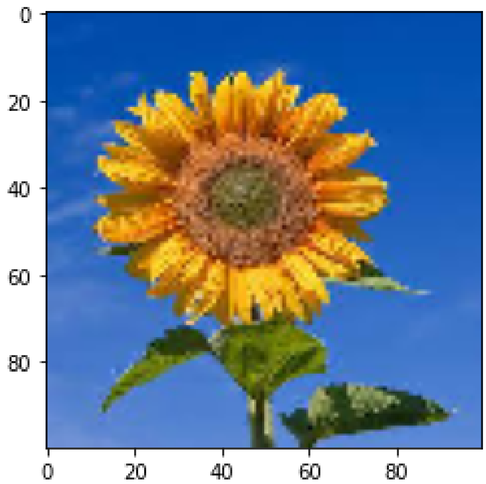
Load Model and make predictions

In [27]:
```python
from tensorflow.keras.models import load_model
```

In [28]:
```python
model = load_model('models/flower_baseline_model.h5')
```

Make predictions

In [29]:
```python
image1 = load_img('images/test1.jpg' , target_size = (100,100))
plt.imshow(image1)
plt.show()
```



In [30]:
```python
# convert the image to array and add one dimenssion
image1 = img_to_array(image1)
image1 = image1.reshape(1,100 , 100 , 3)
```

In [31]:
```python
result = model.predict(image1)
```

```
1/1 [==============================] - 1s 945ms/step
```

In [33]:
```python
print(result)
```

```
[[0.02161606 0.02248923 0.00239826 0.87949467 0.07400182]]
```

Print the highest prediction

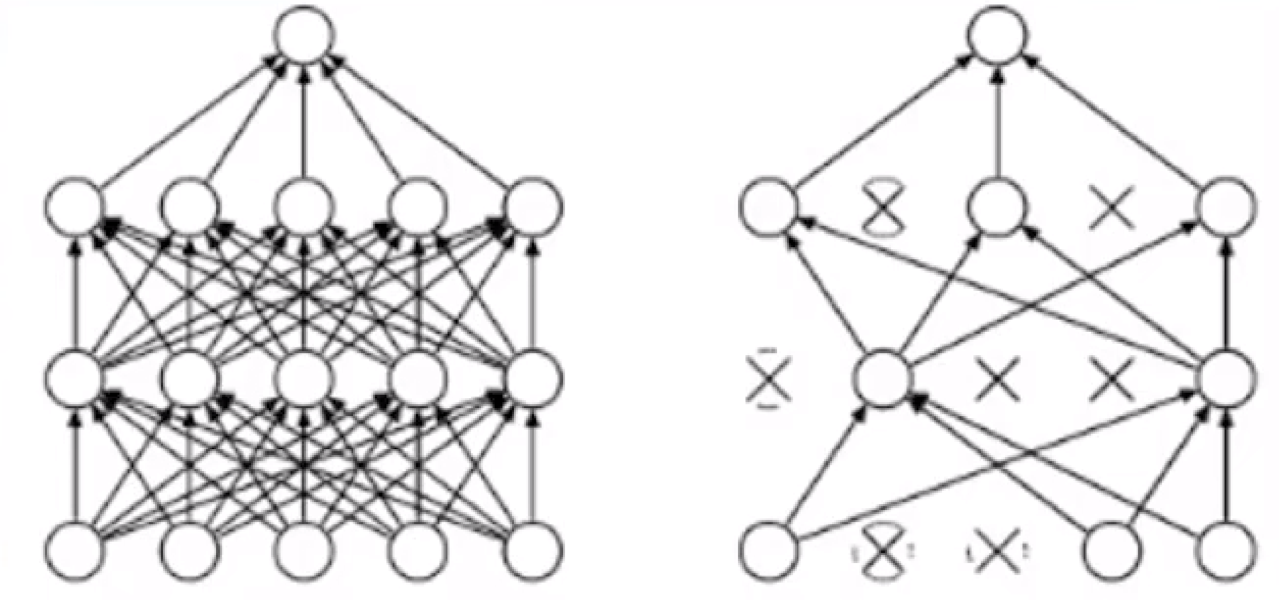
In [34]:
```python
print([label for label in class_labels][np.argmax(result)])
```

```
sunflower
```

Improving Model - Optimization Techniques

1. Dropout Regularization

Dropout is a technique used as a regulaizer to prevent the model from **overfitting**.It works by randomly setting the outgoing hidden neurons to 0 (OFF) randomly at each update of the training phase.



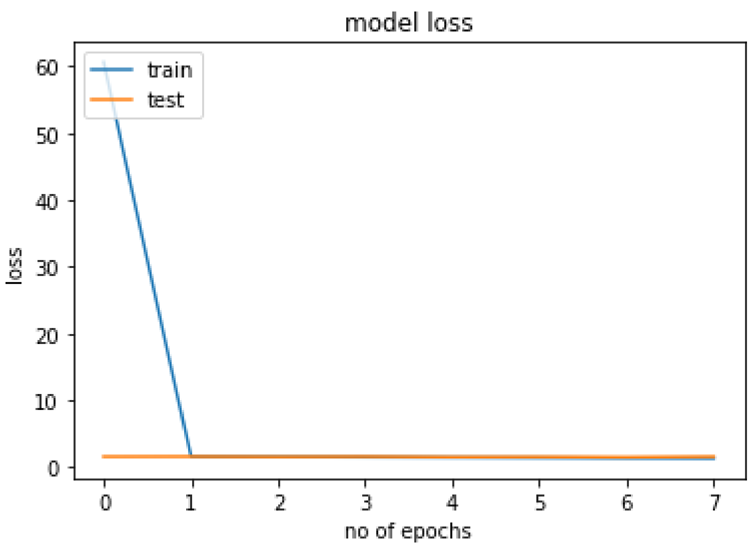
(a) Standard Neural Net (b) After applying dropout.

In [35]:
```
model2 = keras.Sequential()
model2.add(layers.Conv2D(32 , (3,3) , activation = 'relu' , input_shape = (100,100,3
model2.add(layers.MaxPool2D((2,2)))
model2.add(layers.Dropout(0.2))
model2.add(layers.Conv2D(64 , (3,3) , activation = 'relu'))
model2.add(layers.MaxPool2D((2,2)))
model2.add(layers.Dropout(0.2))
model2.add(layers.Conv2D(128 , (3,3) , activation = 'relu'))
model2.add(layers.MaxPool2D((2,2)))
model2.add(layers.Dropout(0.2))
model2.add(layers.Flatten())
model2.add(layers.Dense(512 , activation ='relu'))
model2.add(layers.Dropout(0.3)) #increase the dropout because the number of neurons
model2.add(layers.Dense(5 , activation = 'softmax'))
```

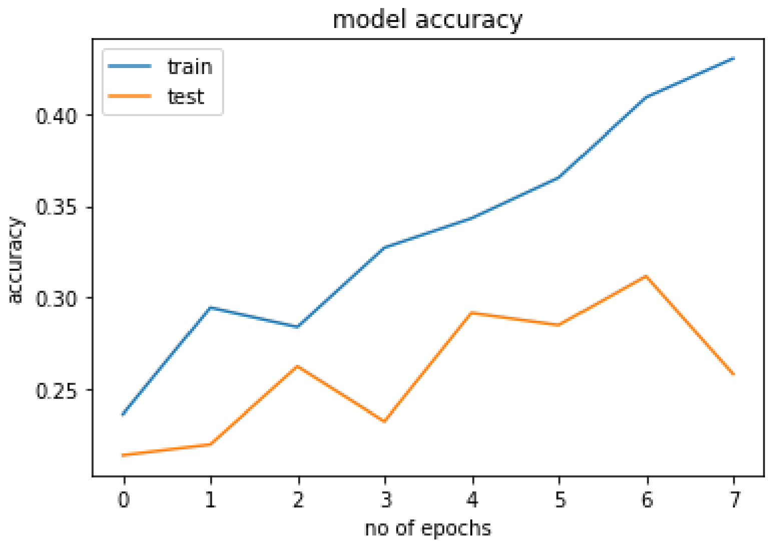
In [36]:
```
model2.compile(loss = 'categorical_crossentropy' , metrics = ['accuracy'] , optimize
history = model2.fit(training_iterator , validation_data = testing_iterator , epochs
```

```
Epoch 1/8
49/49 [==============================] - 46s 903ms/step - loss: 60.6667 - accuracy:
0.2364 - val_loss: 1.5892 - val_accuracy: 0.2142
Epoch 2/8
49/49 [==============================] - 43s 875ms/step - loss: 1.5791 - accuracy:
0.2945 - val_loss: 1.6027 - val_accuracy: 0.2200
Epoch 3/8
49/49 [==============================] - 62s 1s/step - loss: 1.5516 - accuracy: 0.2
839 - val_loss: 1.5622 - val_accuracy: 0.2625
Epoch 4/8
49/49 [==============================] - 53s 1s/step - loss: 1.5147 - accuracy: 0.3
272 - val_loss: 1.5939 - val_accuracy: 0.2325
Epoch 5/8
49/49 [==============================] - 54s 1s/step - loss: 1.4662 - accuracy: 0.3
433 - val_loss: 1.5278 - val_accuracy: 0.2917
Epoch 6/8
49/49 [==============================] - 48s 983ms/step - loss: 1.4181 - accuracy:
0.3654 - val_loss: 1.5497 - val_accuracy: 0.2850
Epoch 7/8
49/49 [==============================] - 51s 1s/step - loss: 1.3641 - accuracy: 0.4
090 - val_loss: 1.5088 - val_accuracy: 0.3117
Epoch 8/8
49/49 [==============================] - 68s 1s/step - loss: 1.3305 - accuracy: 0.4
302 - val_loss: 1.5830 - val_accuracy: 0.2583
```

In [37]:
```
#plot Loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [38]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



Comparing the results of the baseline model and the dropout model shows that the baseline model performs better

2. Padding and Filter Optimization

Padding is to add extra pixels outside the image. And zero padding means every pixel value that you add is zero.

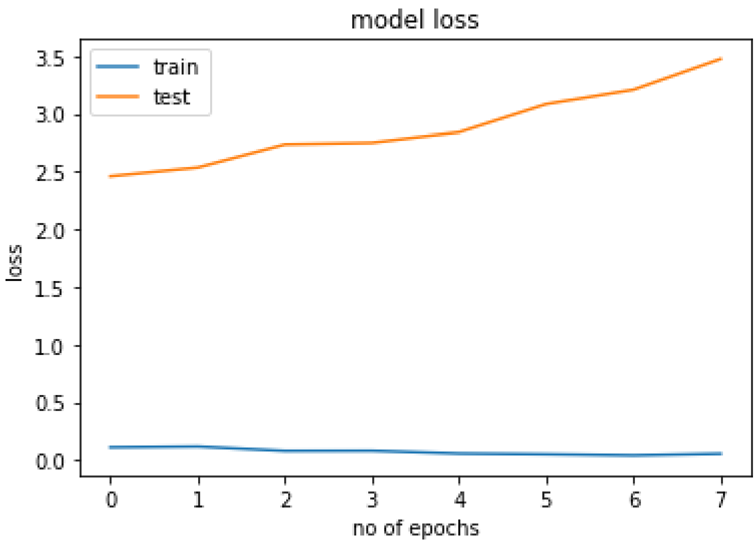
By default padding will be '**valid**', which means apply when required only. If we want to apply it always we have to use '**same**' so that all image pixels are used.

In [41]:
```
model3 = keras.Sequential()
model3.add(layers.Conv2D(32 , (3,3) , activation = 'relu' , input_shape = (100,100,3
model3.add(layers.MaxPool2D((2,2)))

model3.add(layers.Conv2D(64 , (3,3) , activation = 'relu' , padding = 'same'))
model3.add(layers.MaxPool2D((2,2)))

model3.add(layers.Conv2D(128 , (3,3) , activation = 'relu' , padding = 'same'))
model3.add(layers.MaxPool2D((2,2)))

model3.add(layers.Flatten())
model3.add(layers.Dense(512 , activation ='relu'))
model3.add(layers.Dense(5 , activation = 'softmax'))
```

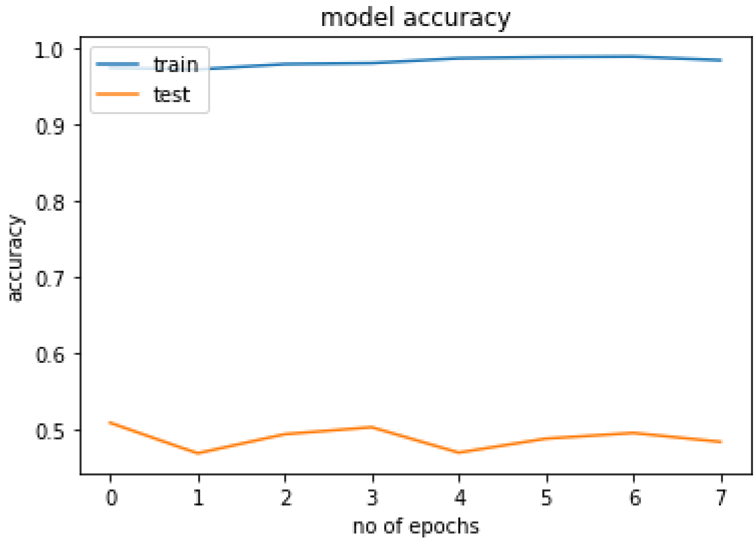
In [42]:
```
model3.compile(loss = 'categorical_crossentropy' , metrics = ['accuracy'] , optimize
history = model.fit(training_iterator , validation_data = testing_iterator , epochs
```

```
Epoch 1/8
49/49 [==============================] - 42s 862ms/step - loss: 0.1121 - accuracy:
0.9747 - val_loss: 2.4604 - val_accuracy: 0.5100
Epoch 2/8
49/49 [==============================] - 53s 1s/step - loss: 0.1187 - accuracy: 0.9
724 - val_loss: 2.5361 - val_accuracy: 0.4700
Epoch 3/8
49/49 [==============================] - 45s 906ms/step - loss: 0.0810 - accuracy:
0.9795 - val_loss: 2.7356 - val_accuracy: 0.4950
Epoch 4/8
49/49 [==============================] - 53s 1s/step - loss: 0.0820 - accuracy: 0.9
808 - val_loss: 2.7496 - val_accuracy: 0.5042
Epoch 5/8
49/49 [==============================] - 59s 1s/step - loss: 0.0580 - accuracy: 0.9
872 - val_loss: 2.8406 - val_accuracy: 0.4708
Epoch 6/8
49/49 [==============================] - 49s 992ms/step - loss: 0.0516 - accuracy:
0.9888 - val_loss: 3.0829 - val_accuracy: 0.4892
Epoch 7/8
49/49 [==============================] - 43s 876ms/step - loss: 0.0423 - accuracy:
0.9894 - val_loss: 3.2076 - val_accuracy: 0.4967
Epoch 8/8
49/49 [==============================] - 43s 881ms/step - loss: 0.0561 - accuracy:
0.9846 - val_loss: 3.4740 - val_accuracy: 0.4850
```

In [43]:
```
#plot Loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [44]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



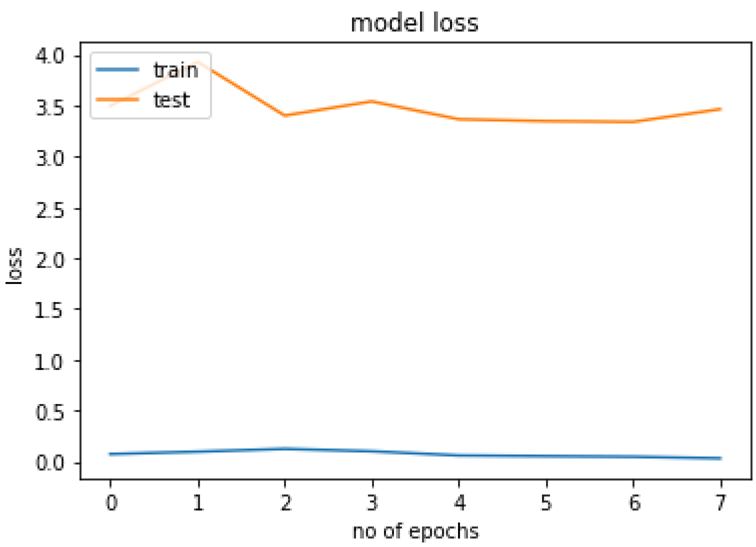
3. Adding more filters

In [45]:
```
model4 = keras.Sequential()
model4.add(layers.Conv2D(64 , (3,3) , activation = 'relu' , input_shape = (100,100,3
model4.add(layers.MaxPool2D((2,2)))

model4.add(layers.Conv2D(128 , (3,3) , activation = 'relu' ))
model4.add(layers.MaxPool2D((2,2)))

model4.add(layers.Conv2D(256 , (3,3) , activation = 'relu' ))
model4.add(layers.MaxPool2D((2,2)))

model4.add(layers.Flatten())
model4.add(layers.Dense(512 , activation ='relu'))
model4.add(layers.Dense(5 , activation = 'softmax'))
```

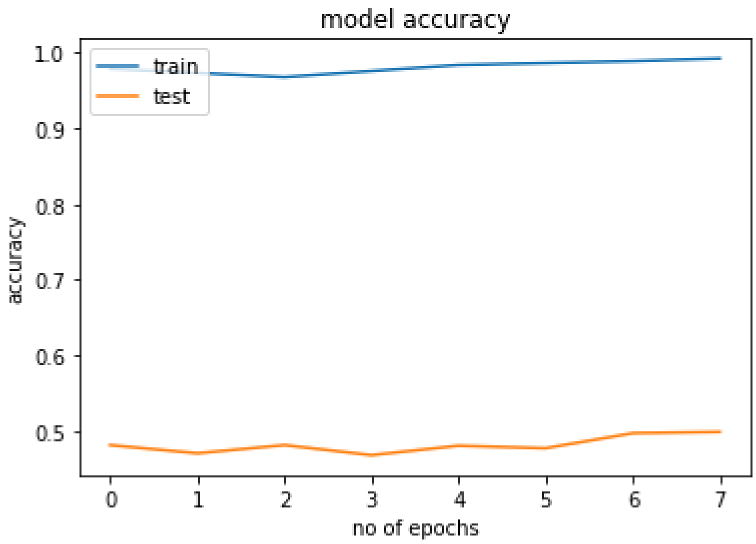
In [46]:
```
model4.compile(loss = 'categorical_crossentropy' , metrics = ['accuracy'] , optimize
history = model.fit(training_iterator , validation_data = testing_iterator , epochs
```

```
Epoch 1/8
49/49 [==============================] - 44s 897ms/step - loss: 0.0752 - accuracy: 
0.9788 - val_loss: 3.4948 - val_accuracy: 0.4808
Epoch 2/8
49/49 [==============================] - 43s 869ms/step - loss: 0.0987 - accuracy: 
0.9734 - val_loss: 3.9269 - val_accuracy: 0.4700
Epoch 3/8
49/49 [==============================] - 44s 895ms/step - loss: 0.1264 - accuracy: 
0.9679 - val_loss: 3.3994 - val_accuracy: 0.4808
Epoch 4/8
49/49 [==============================] - 47s 960ms/step - loss: 0.1025 - accuracy: 
0.9759 - val_loss: 3.5390 - val_accuracy: 0.4675
Epoch 5/8
49/49 [==============================] - 42s 852ms/step - loss: 0.0618 - accuracy: 
0.9840 - val_loss: 3.3631 - val_accuracy: 0.4800
Epoch 6/8
49/49 [==============================] - 49s 998ms/step - loss: 0.0549 - accuracy: 
0.9865 - val_loss: 3.3446 - val_accuracy: 0.4767
Epoch 7/8
49/49 [==============================] - 50s 1s/step - loss: 0.0490 - accuracy: 0.9
891 - val_loss: 3.3371 - val_accuracy: 0.4967
Epoch 8/8
49/49 [==============================] - 43s 876ms/step - loss: 0.0331 - accuracy: 
0.9926 - val_loss: 3.4621 - val_accuracy: 0.4983
```

In [47]:
```
#plot Loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [48]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



4. Augmentation Optimization

In [49]:
```
train_datagen = ImageDataGenerator(rescale = 1/255 , horizontal_flip = True , rotati
                                  height_shift_range = 0.2)
test_datagen = ImageDataGenerator(rescale = 1/255 , horizontal_flip = True , rotatio
                                  height_shift_range = 0.2)
```

In [51]:
```
training_iterator = train_datagen.flow_from_directory('dataset/flowers/train' , batc
testing_iterator = test_datagen.flow_from_directory('dataset/flowers/test' , batch_s
```

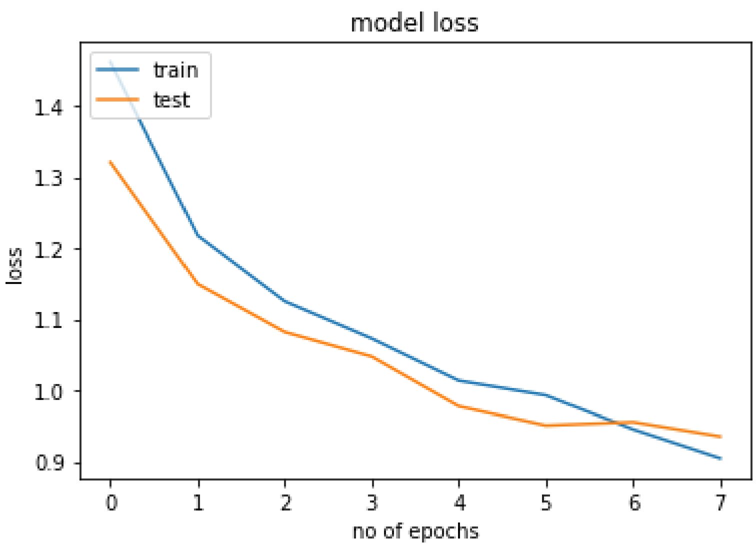
```
Found 3117 images belonging to 5 classes.
Found 1200 images belonging to 5 classes.
```

In [50]:
```
model5 = keras.Sequential()
model5.add(layers.Conv2D(32 , (3,3) , activation = 'relu' , input_shape = (100,100,3
model5.add(layers.MaxPool2D((2,2)))

model5.add(layers.Conv2D(64 , (3,3) , activation = 'relu' ))
model5.add(layers.MaxPool2D((2,2)))

model5.add(layers.Conv2D(128 , (3,3) , activation = 'relu' ))
model5.add(layers.MaxPool2D((2,2)))

model5.add(layers.Flatten())
model5.add(layers.Dense(512 , activation ='relu'))
model5.add(layers.Dense(5 , activation = 'softmax'))
```

In [52]:
```
model5.compile(loss = 'categorical_crossentropy' , metrics = ['accuracy'] , optimize
history = model.fit(training_iterator , validation_data = testing_iterator , epochs
```

```
Epoch 1/8
49/49 [==============================] - 90s 2s/step - loss: 1.4611 - accuracy: 0.3
545 - val_loss: 1.3207 - val_accuracy: 0.4050
Epoch 2/8
49/49 [==============================] - 69s 1s/step - loss: 1.2176 - accuracy: 0.4
678 - val_loss: 1.1495 - val_accuracy: 0.5275
Epoch 3/8
49/49 [==============================] - 76s 2s/step - loss: 1.1252 - accuracy: 0.5
274 - val_loss: 1.0820 - val_accuracy: 0.5592
Epoch 4/8
49/49 [==============================] - 64s 1s/step - loss: 1.0729 - accuracy: 0.5
509 - val_loss: 1.0475 - val_accuracy: 0.5717
Epoch 5/8
49/49 [==============================] - 76s 2s/step - loss: 1.0138 - accuracy: 0.5
922 - val_loss: 0.9779 - val_accuracy: 0.5950
Epoch 6/8
49/49 [==============================] - 63s 1s/step - loss: 0.9933 - accuracy: 0.5
913 - val_loss: 0.9508 - val_accuracy: 0.6217
Epoch 7/8
49/49 [==============================] - 64s 1s/step - loss: 0.9453 - accuracy: 0.6
320 - val_loss: 0.9555 - val_accuracy: 0.6117
Epoch 8/8
49/49 [==============================] - 81s 2s/step - loss: 0.9049 - accuracy: 0.6
468 - val_loss: 0.9354 - val_accuracy: 0.6458
```

In [53]:
```
#plot Loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```



In [54]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('no of epochs')
plt.legend([ 'train', 'test'],loc = 'upper left')
plt.show()
```