```
import time
start = time.time()
```

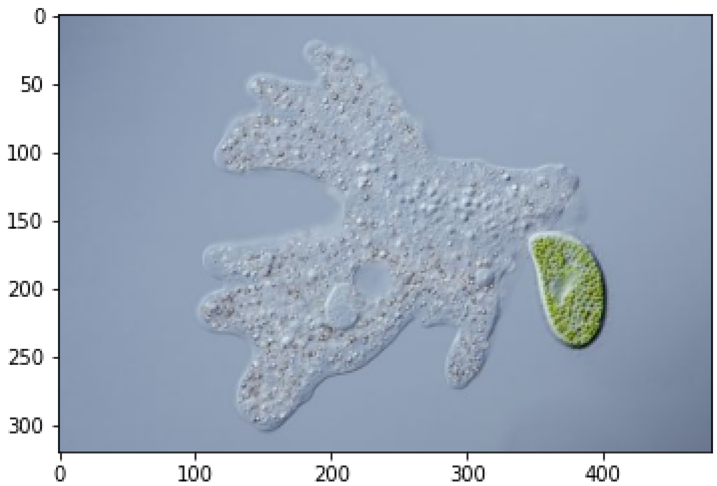
```
!pip install split-folders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

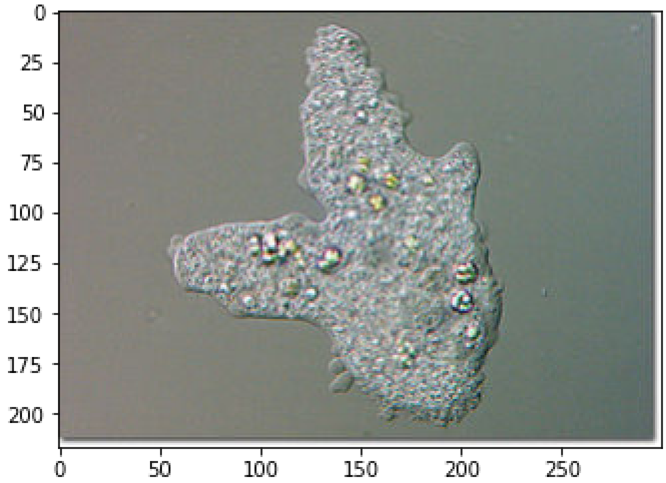
```
import os
import cv2
import splitfolders
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import keras.metrics
from keras.models import Sequential
from keras.layers import Flatten, MaxPooling2D, Conv2D, Dense, Conv3D
from keras import backend as K
```

```
# input_folder = 'Micro_Organism/'
# splitfolders.ratio(input_folder, output="Micro_Organism/SplitImages",
#                      seed=42, ratio=(.7, .2, .1),
#                      group_prefix=None)
# # For split data  || 70% train data, 20% test data and 10% val data
```

```
img_array = cv2.imread('Micro_Organism/SplitImages/train/Amoeba/Image_1.jpg')
img_array = cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
plt.imshow(img_array)
plt.show()
# For test color
```



```
data = 'Micro_Organism/SplitImages/train'
categories = ['Amoeba','Euglena','Hydra','Paramecium','Rod_bacteria','Spherical_bacteria',
# For test our create_data function will be work
for ct in categories:
  path = os.path.join(data,ct)
  for img in os.listdir(path):
    img_array = cv2.imread(os.path.join(path,img))
    img_array = cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
    plt.imshow(img_array)
    plt.show()
    break
  break
```

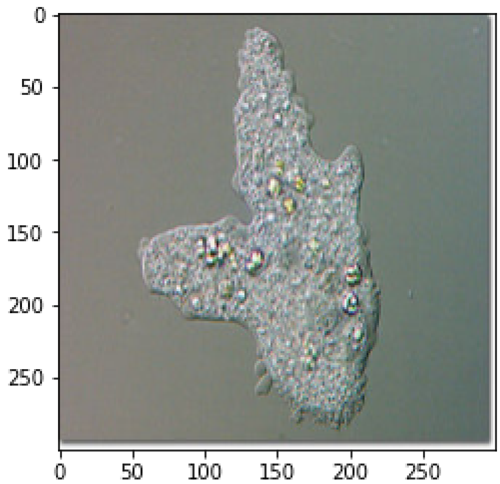


```
print(img_array.shape)
```

    (217, 300, 3)

```
IMG_Size = 300
```

```
new_array = cv2.resize(img_array, (IMG_Size,IMG_Size))
plt.imshow(new_array)
plt.show()
```



```
training_data = []
```

```
val_data = []
test_data = []
def create_data(my_data_path,my_data):
    for ct in categories:
      path = os.path.join(my_data_path,ct)
      class_num = categories.index(ct)
      for img in os.listdir(path):
        datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./225)
        generator = datagen.flow_from_directory(my_data_path,shuffle=True)
        try:
          img_array = cv2.imread(os.path.join(path,img))
          # img_array = img_array / 255
          img_array = cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
          new_array = cv2.resize(img_array, (IMG_Size,IMG_Size))
          my_data.append([new_array,class_num])
        except Exception as e:
          pass



train_data_path = 'Micro_Organism/SplitImages/train'
test_data_path = 'Micro_Organism/SplitImages/test'
val_data_path = 'Micro_Organism/SplitImages/val'
create_data(train_data_path,training_data)
create_data(val_data_path,val_data)
create_data(test_data_path,test_data)
```

```
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
```

```
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
Found 84 images belonging to 8 classes.
```

```
x = []
y = []


for features, label in training_data:
  x.append(features)
  y.append(label)


x_test = []
y_test = []

for features, label in test_data:
  x_test.append(features)
  y_test.append(label)


x_val = []
y_val = []

for features, label in val_data:
  x_val.append(features)
  y_val.append(label)


y = np.array(y)
y_test = np.array(y_test)
y_val = np.array(y_val)


x = np.array(x).reshape(-1, IMG_Size, IMG_Size, 3) # If you are using grayscale you shoulc
```

```
x_test = np.array(x_test).reshape(-1, IMG_Size, IMG_Size, 3)


x_val = np.array(x_val).reshape(-1, IMG_Size, IMG_Size, 3)


# Model Creating

model = Sequential()

model.add(Conv2D(64, (3,3), input_shape=(300, 300, 3) , activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(8, activation='softmax'))

opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay=1e-6)

# Compile model
model.compile(
    loss='sparse_categorical_crossentropy', # binary_crossentropy, sparse_categorical_cros
    optimizer=opt,
    run_eagerly=True,
    metrics=['accuracy']
)

hist = model.fit(x, y, validation_data=(x_val,y_val), verbose=2, epochs=50)
```

```
18/18 - 270s - loss: 1.3517 - accuracy: 0.5173 - val_loss: 4.7741 - val_accuracy:
Epoch 23/50
18/18 - 271s - loss: 1.2315 - accuracy: 0.6157 - val_loss: 3.3280 - val_accuracy:
Epoch 24/50
18/18 - 269s - loss: 0.9811 - accuracy: 0.6557 - val_loss: 5.6831 - val_accuracy:
Epoch 25/50
18/18 - 270s - loss: 1.1211 - accuracy: 0.6175 - val_loss: 3.2535 - val_accuracy:
Epoch 26/50
18/18 - 271s - loss: 1.7081 - accuracy: 0.3333 - val_loss: 3.3708 - val_accuracy:
Epoch 27/50
18/18 - 270s - loss: 1.6793 - accuracy: 0.3588 - val_loss: 3.7205 - val_accuracy:
Epoch 28/50
18/18 - 270s - loss: 1.5829 - accuracy: 0.4080 - val_loss: 5.1372 - val_accuracy:
Epoch 29/50
18/18 - 271s - loss: 1.4381 - accuracy: 0.4663 - val_loss: 6.0289 - val_accuracy:
Epoch 30/50
18/18 - 270s - loss: 1.1921 - accuracy: 0.5865 - val_loss: 13.4602 - val_accuracy
Epoch 31/50
18/18 - 269s - loss: 0.8616 - accuracy: 0.7377 - val_loss: 10.2362 - val_accuracy
Epoch 32/50
18/18 - 274s - loss: 0.8652 - accuracy: 0.7195 - val_loss: 15.6919 - val_accuracy
Epoch 33/50
18/18 - 269s - loss: 0.6662 - accuracy: 0.8033 - val_loss: 16.3625 - val_accuracy
Epoch 34/50
18/18 - 268s - loss: 0.5142 - accuracy: 0.8434 - val_loss: 13.8769 - val_accuracy
Epoch 35/50
```

```
18/18 - 268s - loss: 0.5178 - accuracy: 0.8798 - val_loss: 16.6655 - val_accuracy
Epoch 36/50
18/18 - 270s - loss: 0.4280 - accuracy: 0.8980 - val_loss: 20.6374 - val_accuracy
Epoch 37/50
18/18 - 269s - loss: 1.0605 - accuracy: 0.6175 - val_loss: 9.8888 - val_accuracy:
Epoch 38/50
18/18 - 269s - loss: 0.8386 - accuracy: 0.7814 - val_loss: 21.3229 - val_accuracy
Epoch 39/50
18/18 - 267s - loss: 0.2840 - accuracy: 0.9271 - val_loss: 17.0428 - val_accuracy
Epoch 40/50
18/18 - 268s - loss: 0.3828 - accuracy: 0.9071 - val_loss: 17.1464 - val_accuracy
Epoch 41/50
18/18 - 268s - loss: 0.3859 - accuracy: 0.9271 - val_loss: 25.5359 - val_accuracy
Epoch 42/50
18/18 - 268s - loss: 0.1441 - accuracy: 0.9617 - val_loss: 27.4308 - val_accuracy
Epoch 43/50
18/18 - 268s - loss: 0.4057 - accuracy: 0.9199 - val_loss: 23.5285 - val_accuracy
Epoch 44/50
18/18 - 269s - loss: 0.5910 - accuracy: 0.8288 - val_loss: 19.2959 - val_accuracy
Epoch 45/50
18/18 - 269s - loss: 0.4310 - accuracy: 0.8780 - val_loss: 23.3523 - val_accuracy
Epoch 46/50
18/18 - 270s - loss: 0.2538 - accuracy: 0.9417 - val_loss: 25.7329 - val_accuracy
Epoch 47/50
18/18 - 268s - loss: 0.6392 - accuracy: 0.9089 - val_loss: 17.5868 - val_accuracy
Epoch 48/50
18/18 - 267s - loss: 0.5945 - accuracy: 0.8597 - val_loss: 20.8036 - val_accuracy
Epoch 49/50
18/18 - 268s - loss: 0.5669 - accuracy: 0.8962 - val_loss: 18.1568 - val_accuracy
Epoch 50/50
18/18 - 267s - loss: 0.8197 - accuracy: 0.6995 - val_loss: 24.0877 - val_accuracy
```

```
hist.history??
```

```
# hist.history ?? ---> Use -??- for help
```

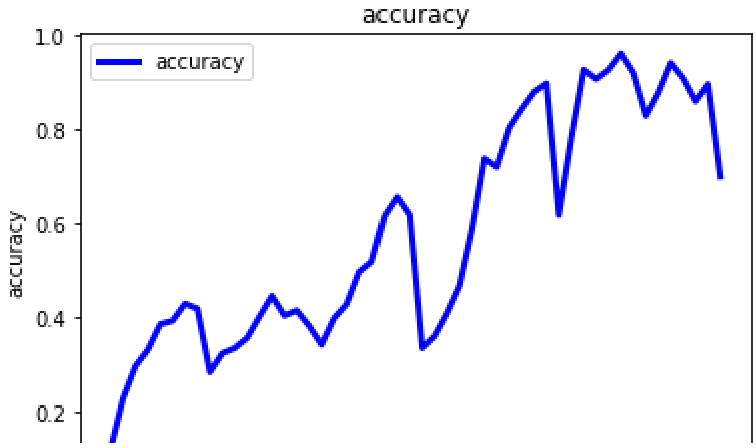
```
acc = hist.history['accuracy']
max(acc)
```

```
0.9617486596107483
```

```
def visualization(name,h,color):
  t = h.history[name]
  my_max = max(t)
  my_min = min(t)
  print(f'Name : {name} max : {my_max} min : {my_min}')
  plt.plot(t,color=color,linewidth=3.0)
  plt.title(name)
  plt.ylabel(name)
  plt.xlabel('Epoch')
  plt.legend([name],loc='upper left')
  plt.show()
```

```
visualization('accuracy',hist,'Blue')
visualization('loss',hist,'Red')
visualization('val_accuracy',hist,'Green')
visualization('val_loss',hist,'Black')
```

```
Name : accuracy max : 0.9617486596107483 min : 0.11839708685874939
```




```
res = model.evaluate(x_test, y_test)
print("test loss, test acc:", res)
```

```
3/3 [==============================] - 10s 3s/step - loss: 39.1115 - accuracy: 0.166
test loss, test acc: [39.111534118652344, 0.1666666716337204]
```

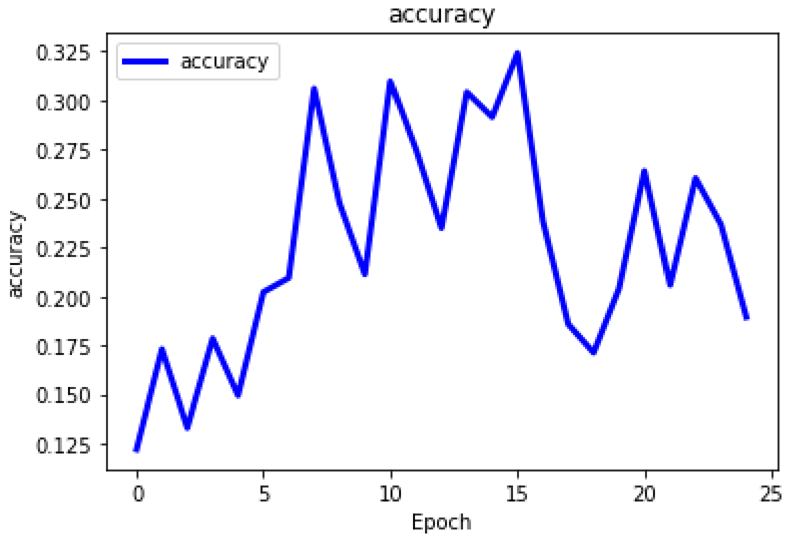
```
model1 = Sequential()

model1.add(Flatten(input_shape=(300, 300, 3)))

model1.add(Dense(units=256,activation='relu'))

model1.add(Dense(units=256,activation='relu'))

model1.add(Dense(units=8,activation='softmax'))

model1.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'

hist1 = model1.fit(x,y,validation_data=(x_val,y_val), epochs=25, verbose=2)
```

```
Epoch 1/25
18/18 - 8s - loss: 47035.4883 - accuracy: 0.1220 - val_loss: 22439.0703 - val_accura
Epoch 2/25
18/18 - 7s - loss: 13048.3955 - accuracy: 0.1730 - val_loss: 14125.7324 - val_accura
Epoch 3/25
18/18 - 7s - loss: 11693.5977 - accuracy: 0.1330 - val_loss: 8886.3838 - val_accurac
Epoch 4/25
18/18 - 7s - loss: 8943.6787 - accuracy: 0.1785 - val_loss: 8381.9004 - val_accuracy
Epoch 5/25
18/18 - 7s - loss: 5284.0732 - accuracy: 0.1494 - val_loss: 6361.8896 - val_accuracy
Epoch 6/25
18/18 - 7s - loss: 5010.2197 - accuracy: 0.2022 - val_loss: 2756.6897 - val_accuracy
Epoch 7/25
18/18 - 7s - loss: 1720.3639 - accuracy: 0.2095 - val_loss: 2017.0404 - val_accuracy
Epoch 8/25
18/18 - 7s - loss: 1009.8175 - accuracy: 0.3060 - val_loss: 1139.5677 - val_accuracy
Epoch 9/25
18/18 - 7s - loss: 949.7007 - accuracy: 0.2477 - val_loss: 1023.9896 - val_accuracy:
Epoch 10/25
18/18 - 7s - loss: 1326.0703 - accuracy: 0.2113 - val_loss: 1477.3208 - val_accuracy
```

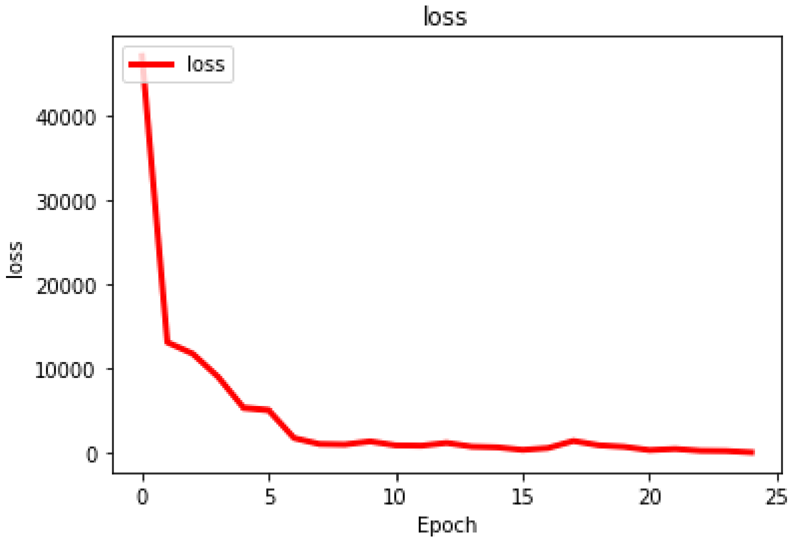
```
Epoch 11/25
18/18 - 7s - loss: 852.6691 - accuracy: 0.3097 - val_loss: 1620.2799 - val_accuracy:
Epoch 12/25
18/18 - 7s - loss: 809.6503 - accuracy: 0.2750 - val_loss: 1113.9762 - val_accuracy:
Epoch 13/25
18/18 - 7s - loss: 1142.2897 - accuracy: 0.2350 - val_loss: 912.8844 - val_accuracy:
Epoch 14/25
18/18 - 7s - loss: 698.2715 - accuracy: 0.3042 - val_loss: 803.0130 - val_accuracy:
Epoch 15/25
18/18 - 7s - loss: 617.8412 - accuracy: 0.2914 - val_loss: 601.8291 - val_accuracy:
Epoch 16/25
18/18 - 7s - loss: 313.0997 - accuracy: 0.3242 - val_loss: 799.9505 - val_accuracy:
Epoch 17/25
18/18 - 7s - loss: 530.2900 - accuracy: 0.2386 - val_loss: 2620.4473 - val_accuracy:
Epoch 18/25
18/18 - 7s - loss: 1377.9227 - accuracy: 0.1858 - val_loss: 622.8046 - val_accuracy:
Epoch 19/25
18/18 - 7s - loss: 861.8452 - accuracy: 0.1712 - val_loss: 922.1077 - val_accuracy:
Epoch 20/25
18/18 - 7s - loss: 680.6086 - accuracy: 0.2040 - val_loss: 423.5898 - val_accuracy:
Epoch 21/25
18/18 - 7s - loss: 286.3119 - accuracy: 0.2641 - val_loss: 574.8794 - val_accuracy:
Epoch 22/25
18/18 - 7s - loss: 424.3523 - accuracy: 0.2058 - val_loss: 276.9922 - val_accuracy:
Epoch 23/25
18/18 - 7s - loss: 208.8260 - accuracy: 0.2605 - val_loss: 351.6682 - val_accuracy:
Epoch 24/25
18/18 - 7s - loss: 180.1535 - accuracy: 0.2368 - val_loss: 165.5076 - val_accuracy:
Epoch 25/25
18/18 - 7s - loss: 22.3051 - accuracy: 0.1894 - val_loss: 4.3324 - val_accuracy: 0.2
```

```
visualization('accuracy',hist1,'Blue')
visualization('loss',hist1,'Red')
visualization('val_accuracy',hist1,'Green')
visualization('val_loss',hist1,'Black')
```

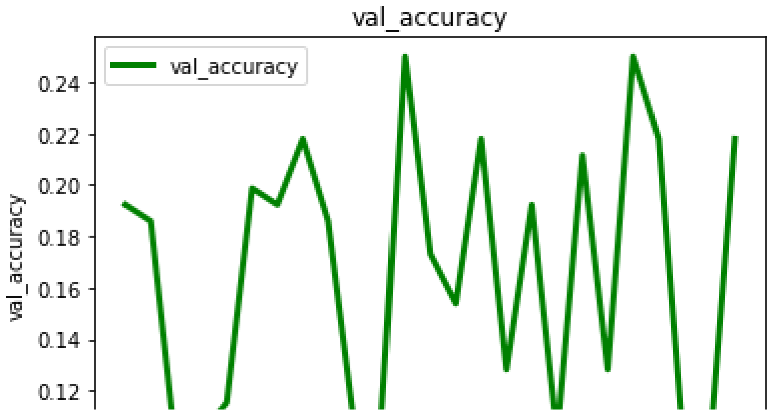
```
Name : accuracy max : 0.32422587275505066 min : 0.12204007059335709
```




```
Name : loss max : 47035.48828125 min : 22.30510902404785
```




```
Name : val_accuracy max : 0.25 min : 0.09615384787321091
```




```
res = model1.evaluate(x_test, y_test)
print("test loss, test acc:", res)
```

```
3/3 [==============================] - 0s 105ms/step - loss: 2.1047 - accuracy: 0.21
test loss, test acc: [2.1047019958496094, 0.2142857164144516]
```


```
newpath = r'Micro_Organism/Model'
if not os.path.exists(newpath):
    os.makedirs(newpath)
```

```
import pickle
# To save model  || You can use tensorflow model.save
pickle_out = open('Micro_Organism/Model/model.pickle','wb')
pickle.dump(model,pickle_out)
pickle_out.close()
pickle_out1 = open('Micro_Organism/Model/model1.pickle','wb')
pickle.dump(model1,pickle_out1)
pickle_out1.close()
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 64)      1792      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 149, 149, 64)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 147, 147, 128)     73856     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 73, 73, 128)      0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 71, 71, 128)       147584    
                                                                 
 flatten (Flatten)           (None, 645248)            0         
                                                                 
 dense (Dense)               (None, 128)               82591872  
                                                                 
 dense_1 (Dense)             (None, 8)                 1032      
                                                                 
=================================================================
Total params: 82,816,136
Trainable params: 82,816,136
Non-trainable params: 0
_________________________________________________________________
```

```
model1.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_8 (Flatten)         (None, 270000)            0         
                                                                 
 dense_23 (Dense)            (None, 256)               69120256  
                                                                 
 dense_24 (Dense)            (None, 256)               65792     
                                                                 
 dense_25 (Dense)            (None, 8)                 2056      
                                                                 
=================================================================
```

```
Total params: 69,188,104
Trainable params: 69,188,104
Non-trainable params: 0
_________________________________________________________________
```

```
end = time.time()
print((end - start)/60)
```

```
315.27953653732936
```

```
# For load model
pickle_in = open('Micro_Organism/Model/model.pickle','rb')
model = pickle.load(pickle_in)
```

✓ 3 sn. tamamlanma zamanı: 04:02