Computer vision assignment

Student name: Stephen Pangga

Student number: 629860

Imports

In [ ]:
```
# %pip install keras
# %pip install tensorflow
# %pip install image_dataset_loader

import keras
import os
from keras.models import *
from keras.layers import *
from keras.datasets import cifar10
from keras.optimizers import *
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
from keras.utils import *
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19

from image_dataset_loader import load


# train_data_dir = os.path.dirname("output_path_no_Plain_220x380v3/train/")
# validation_data_dir = os.path.dirname("output_path_no_Plain_220x380v3/val/")
# test_data_dir = os.path.dirname("output_path_no_Plain_220x380v3/test/")


train_data_dir = os.path.dirname("output_path/train/")
validation_data_dir = os.path.dirname("output_path/val/")
test_data_dir = os.path.dirname("output_path/test/")
```

In [ ]:
```
img_width, img_height = 224, 224
batch_size = 16

datagenerate_train = ImageDataGenerator(rescale=1.0/255, samplewise_center=True)

train_generator = datagenerate_train.flow_from_directory(train_data_dir,
                                                 target_size=(img_width,img_height),
                                                  batch_size=batch_size,
                                                  #subset="training",
                                                  class_mode='categorical',
                                                  shuffle = True)

datagenerate_validation = ImageDataGenerator(rescale=1.0/255, samplewise_center=Tru

validation_generator = datagenerate_validation.flow_from_directory(validation_data_
                                                       target_size=(img_width,img_
                                                       batch_size=batch_size,
                                                       #subset="validation",
                                                       class_mode='categorical',
                                                       shuffle = True)
```

```
print(train_generator)
print(validation_generator)
# print(test_generator)
```

```
Found 816 images belonging to 7 classes.
Found 101 images belonging to 7 classes.
<keras.preprocessing.image.DirectoryIterator object at 0x00000246A0968160>
<keras.preprocessing.image.DirectoryIterator object at 0x00000246F90F85B0>
```

The Model

In [ ]:

```
def define_VGGmodel():

    model = VGG16(include_top=False, input_shape=(img_width, img_height, 3))

    for layer in model.layers:
        layer.trainable = False

    flat1 = Flatten()(model.layers[-1].output)

    class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)

    output = Dense(7, activation='softmax')(class1)

    model = Model(inputs=model.inputs, outputs=output)

    # opt = SGD(lr=0.001, momentum=0.9)

    model.compile(optimizer="Adam", loss='categorical_crossentropy', metrics=['cate

    return model

model = define_VGGmodel()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 17,926,983
Trainable params: 3,212,295
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:
```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 5,
                                        restore_best_weights = True)
```

In [ ]:
```
history = model.fit(train_generator, steps_per_epoch=len(train_generator), validat
 validation_steps=len(validation_generator), epochs=25 , verbose=1, shuffle=False,
```

```
Epoch 1/25
51/51 [==============================] - 83s 2s/step - loss: 2.0130 - categorical_
accuracy: 0.3811 - val_loss: 1.2446 - val_categorical_accuracy: 0.5545
Epoch 2/25
51/51 [==============================] - 82s 2s/step - loss: 0.8218 - categorical_
accuracy: 0.7414 - val_loss: 0.8679 - val_categorical_accuracy: 0.6238
Epoch 3/25
51/51 [==============================] - 75s 1s/step - loss: 0.4796 - categorical_
accuracy: 0.8934 - val_loss: 0.5692 - val_categorical_accuracy: 0.8317
Epoch 4/25
51/51 [==============================] - 82s 2s/step - loss: 0.2860 - categorical_
accuracy: 0.9559 - val_loss: 0.5903 - val_categorical_accuracy: 0.7822
Epoch 5/25
51/51 [==============================] - 81s 2s/step - loss: 0.2206 - categorical_
accuracy: 0.9706 - val_loss: 0.4360 - val_categorical_accuracy: 0.8911
Epoch 6/25
51/51 [==============================] - 80s 2s/step - loss: 0.1335 - categorical_
accuracy: 0.9877 - val_loss: 0.3606 - val_categorical_accuracy: 0.9010
Epoch 7/25
51/51 [==============================] - 80s 2s/step - loss: 0.0900 - categorical_
accuracy: 0.9975 - val_loss: 0.4320 - val_categorical_accuracy: 0.8515
Epoch 8/25
51/51 [==============================] - 81s 2s/step - loss: 0.0849 - categorical_
accuracy: 0.9951 - val_loss: 0.3602 - val_categorical_accuracy: 0.8812
Epoch 9/25
51/51 [==============================] - 83s 2s/step - loss: 0.0753 - categorical_
accuracy: 0.9963 - val_loss: 0.2816 - val_categorical_accuracy: 0.9109
Epoch 10/25
51/51 [==============================] - 79s 2s/step - loss: 0.0401 - categorical_
accuracy: 1.0000 - val_loss: 0.2737 - val_categorical_accuracy: 0.9010
Epoch 11/25
51/51 [==============================] - 80s 2s/step - loss: 0.0309 - categorical_
accuracy: 1.0000 - val_loss: 0.2734 - val_categorical_accuracy: 0.8911
Epoch 12/25
51/51 [==============================] - 80s 2s/step - loss: 0.0267 - categorical_
accuracy: 1.0000 - val_loss: 0.2580 - val_categorical_accuracy: 0.9109
Epoch 13/25
51/51 [==============================] - 81s 2s/step - loss: 0.0208 - categorical_
accuracy: 1.0000 - val_loss: 0.2787 - val_categorical_accuracy: 0.9010
Epoch 14/25
51/51 [==============================] - 81s 2s/step - loss: 0.0179 - categorical_
accuracy: 1.0000 - val_loss: 0.2540 - val_categorical_accuracy: 0.8911
Epoch 15/25
51/51 [==============================] - 81s 2s/step - loss: 0.0154 - categorical_
accuracy: 1.0000 - val_loss: 0.2406 - val_categorical_accuracy: 0.9109
Epoch 16/25
51/51 [==============================] - 81s 2s/step - loss: 0.0132 - categorical_
accuracy: 1.0000 - val_loss: 0.2253 - val_categorical_accuracy: 0.9208
Epoch 17/25
51/51 [==============================] - 80s 2s/step - loss: 0.0124 - categorical_
accuracy: 1.0000 - val_loss: 0.2253 - val_categorical_accuracy: 0.9208
Epoch 18/25
51/51 [==============================] - 80s 2s/step - loss: 0.0115 - categorical_
accuracy: 1.0000 - val_loss: 0.2324 - val_categorical_accuracy: 0.9010
Epoch 19/25
51/51 [==============================] - 85s 2s/step - loss: 0.0092 - categorical_
accuracy: 1.0000 - val_loss: 0.2204 - val_categorical_accuracy: 0.9109
Epoch 20/25
51/51 [==============================] - 89s 2s/step - loss: 0.0083 - categorical_
accuracy: 1.0000 - val_loss: 0.2336 - val_categorical_accuracy: 0.8911
Epoch 21/25
51/51 [==============================] - 81s 2s/step - loss: 0.0072 - categorical_
accuracy: 1.0000 - val_loss: 0.2134 - val_categorical_accuracy: 0.9010
Epoch 22/25
```

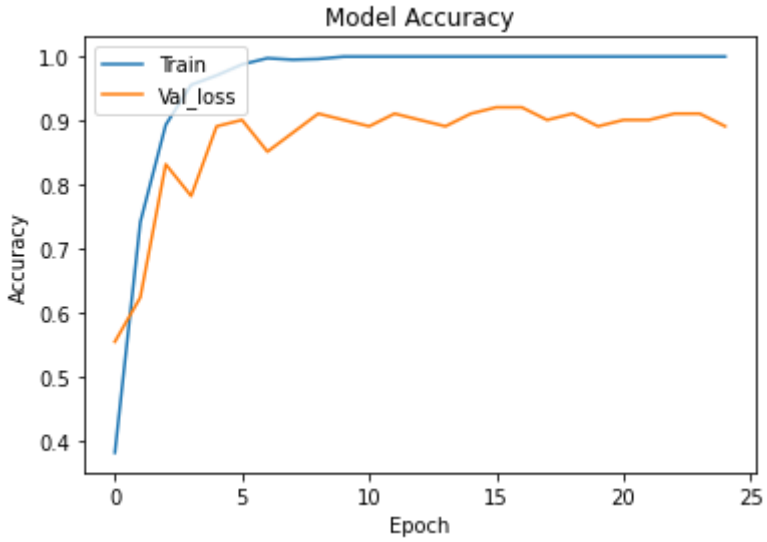
```
51/51 [==============================] - 78s 2s/step - loss: 0.0067 - categorical_
accuracy: 1.0000 - val_loss: 0.2289 - val_categorical_accuracy: 0.9010
Epoch 23/25
51/51 [==============================] - 77s 2s/step - loss: 0.0061 - categorical_
accuracy: 1.0000 - val_loss: 0.2094 - val_categorical_accuracy: 0.9109
Epoch 24/25
51/51 [==============================] - 78s 2s/step - loss: 0.0055 - categorical_
accuracy: 1.0000 - val_loss: 0.2110 - val_categorical_accuracy: 0.9109
Epoch 25/25
51/51 [==============================] - 77s 2s/step - loss: 0.0051 - categorical_
accuracy: 1.0000 - val_loss: 0.2282 - val_categorical_accuracy: 0.8911
```

In [ ]:
```
print(history.history)
```

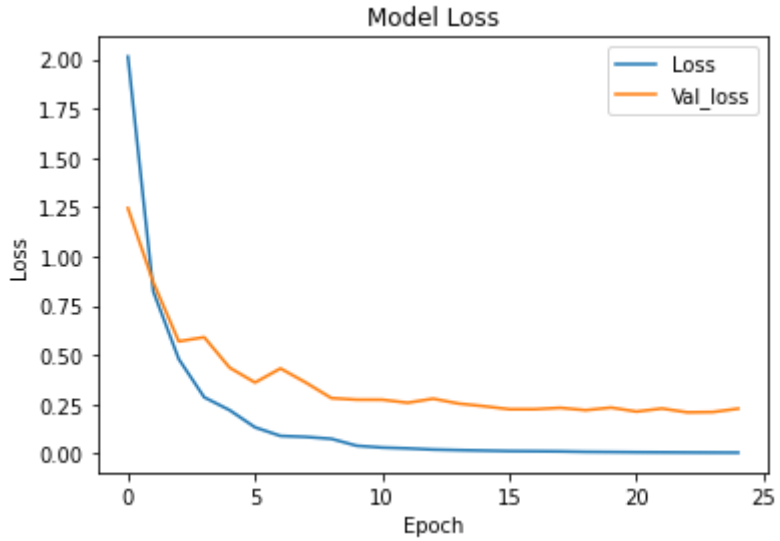
```
{'loss': [2.012996196746826, 0.8217641711235046, 0.47962120175361633, 0.2860125303
2684326, 0.2206156849861145, 0.13351689279079437, 0.08996354043483734, 0.084902554
7504425, 0.07527531683444977, 0.04009522125124931, 0.03093503974378109, 0.02673844
8068499565, 0.02081669308245182, 0.017879987135529518, 0.015433376654982567, 0.013
170113787055016, 0.012445982545614243, 0.011478454805910587, 0.009217247366905212,
0.0083229411393404, 0.007217416074126959, 0.006735136266797781, 0.0060526682063937
19, 0.005545631516724825, 0.005145586561411619], 'categorical_accuracy': [0.381127
4468898773, 0.7414215803146362, 0.8933823704719543, 0.9558823704719543, 0.97058820
7244873, 0.9877451062202454, 0.9975489974021912, 0.9950980544090271, 0.99632352590
56091, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0], 'val_loss': [1.2445828914642334, 0.8678803443908691, 0.5691992044448853, 0.5
90302586555481, 0.4359528124332428, 0.3605560064315796, 0.43199849128723145, 0.360
18794775009155, 0.28156766295433044, 0.2736772298812866, 0.2734403908252716, 0.257
9731047153473, 0.2786790132522583, 0.2539757192134857, 0.2405688762664795, 0.22528
916597366333, 0.22528526186943054, 0.2323930710554123, 0.220425084233284, 0.233551
0402917862, 0.21336442232131958, 0.22891883552074432, 0.2094143182039261, 0.210956
409573555, 0.22817252576351166], 'val_categorical_accuracy': [0.5544554591178894,
0.6237623691558838, 0.8316831588745117, 0.7821782231330872, 0.8910890817642212, 0.
9009901285171509, 0.8514851331710815, 0.8811880946159363, 0.9108911156654358, 0.90
09901285171509, 0.8910890817642212, 0.9108911156654358, 0.9009901285171509, 0.8910
890817642212, 0.9108911156654358, 0.9207921028137207, 0.9207921028137207, 0.900990
1285171509, 0.9108911156654358, 0.8910890817642212, 0.9009901285171509, 0.90099012
85171509, 0.9108911156654358, 0.9108911156654358, 0.8910890817642212]}
```

Diagnostic plot

In [ ]:
```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val_loss'], loc='upper left')
plt.show()
```



In [ ]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Loss', 'Val_loss'], loc='upper right')
plt.show()
```



In [ ]:
```
print('Validation image - evaluation')
_, acc = model.evaluate(validation_generator, steps=len(validation_generator), verb
print('accuracy score: ', acc*100 )
```

```
Validation image - evaluation
7/7 [==============================] - 9s 1s/step - loss: 0.2282 - categorical_acc
uracy: 0.8911
accuracy score:  89.10890817642212
```

In [ ]:
```
model.save('VGG16-accu-score-92%')

#model=load_model('vgg15-softmax-classifier_train-91.54_val-72.06_test-83.82%')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_c
ompiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_o
p, _jit_compiled_convolution_op while saving (showing 5 of 13). These functions wi
ll not be directly callable after loading.
INFO:tensorflow:Assets written to: VGG16-accu-score-92%\assets
INFO:tensorflow:Assets written to: VGG16-accu-score-92%\assets
```

Testing

In [ ]:
```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn import metrics
# test_data_dir = os.path.dirname("output_path_no_Plain_220x380v1/train-test/")

datagenerate_test = ImageDataGenerator(rescale=1.0/255, samplewise_center=True)

test_generator = datagenerate_test.flow_from_directory(test_data_dir,
                                                       target_size=(img_width,img_height),
                                                       batch_size=batch_size,
                                                       #subset="validation",
                                                       class_mode='categorical',
                                                       shuffle = False)
```

```
Found 103 images belonging to 7 classes.
```

In [ ]:
```
#model=load_model('VGG16-accu-score-92%')
```

In [ ]:
```
print('Test image - evaluation')
_, acc = model.evaluate(test_generator, steps=len(test_generator), verbose=1)
print('accuracy score: ', acc*100 )
```

```
Test image - evaluation
7/7 [==============================] - 10s 1s/step - loss: 0.1904 - categorical_ac
curacy: 0.9126
accuracy score:  91.26213788986206
```

In [ ]:
```
Y_pred = model.predict(test_generator)
y_pred = np.argmax(Y_pred, axis=1)
print('accuracy: ', metrics.accuracy_score(y_pred,test_generator.classes))
```

```
7/7 [==============================] - 9s 1s/step
accuracy:  0.912621359223301
```

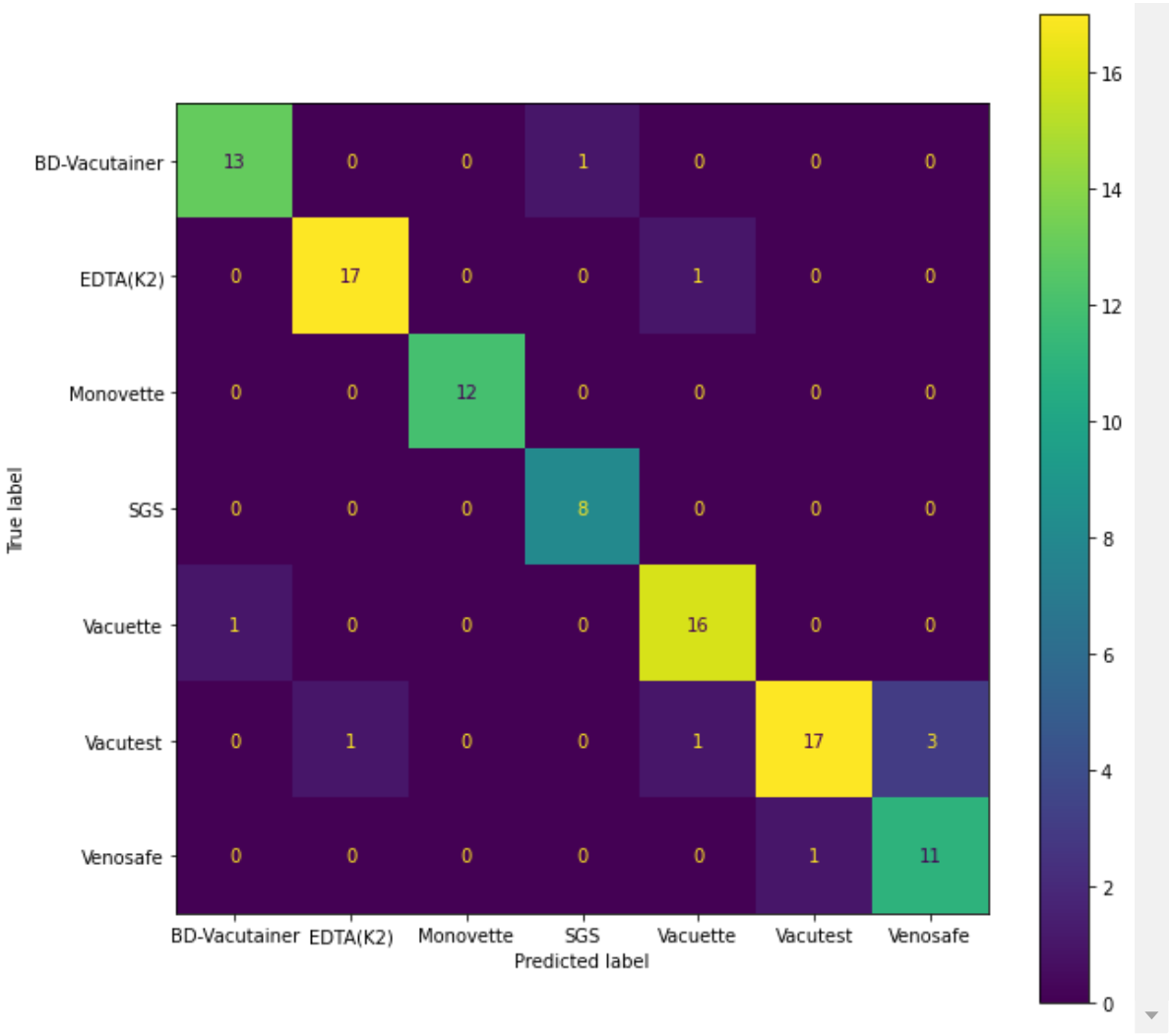
In [ ]:
```
print('Confusion Matrix')
print(confusion_matrix(y_pred, test_generator.classes))
```

```
Confusion Matrix
[[13  0  0  1  0  0  0]
 [ 0 17  0  0  1  0  0]
 [ 0  0 12  0  0  0  0]
 [ 0  0  0  8  0  0  0]
 [ 1  0  0  0 16  0  0]
 [ 0  1  0  0  1 17  3]
 [ 0  0  0  0  0  1 11]]
```

In [ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_pred, test_generator.classes)
cmd = ConfusionMatrixDisplay(cm, display_labels=[ 'BD-Vacutainer', 'EDTA(K2)', 'Mo
fig, ax = plt.subplots(figsize=(10,10))
plt.grid(False)
cmd.plot(ax=ax)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x246b68c9880>
```



In [ ]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

print('Accuracy Score :', accuracy_score(y_pred, test_generator.classes))
print ('Report : ')
target_names = ['BD-Vacutainer', 'EDTA(K2)', 'Monovette', 'SGS', 'Vacuette' , 'Vacu
print (classification_report(y_pred, test_generator.classes, target_names=target_na
```

```
Accuracy Score : 0.912621359223301
Report :
              precision    recall  f1-score   support

BD-Vacutainer       0.93      0.93      0.93        14
     EDTA(K2)       0.94      0.94      0.94        18
    Monovette       1.00      1.00      1.00        12
          SGS       0.89      1.00      0.94         8
     Vacuette       0.89      0.94      0.91        17
     Vacutest       0.94      0.77      0.85        22
     Venosafe       0.79      0.92      0.85        12

     accuracy                           0.91       103
    macro avg       0.91      0.93      0.92       103
 weighted avg       0.92      0.91      0.91       103
```

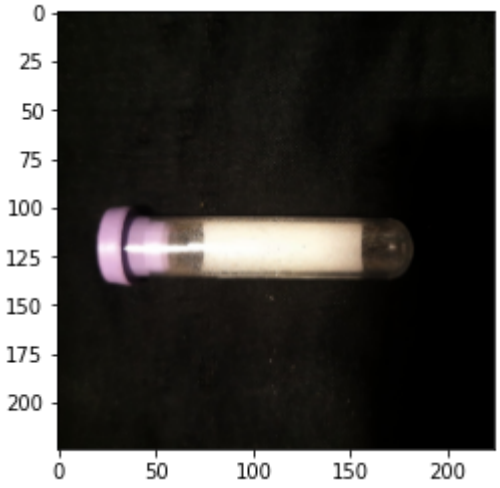
The first accuracy score was 15.53%, this could be due to the image. One of the plausible issues i can think of that cause such a low accuracy is the fact that maybe tht augmented image has not been since by the model.

Update:

There was an issue where the accuracy was giving 15 percent accuracy, that has been now solved. The issue was when model.predict() the train folder it organized the result of the folder and doesn't work with the randomize order of test_generator.classes. To prevent this from happening add "shuffle= False" in the parameter of your imageDataGenerator.

Test an image

In [ ]:
```
from skimage.transform import resize
Image = plt.imread("output_path/test/BD-Vacutainer/BD-Vacutainer1_small_constrast.
RImage = resize(Image, (img_width, img_height, 3))
img = plt.imshow(RImage)
```



In [ ]:
```
predictions = model.predict(np.array( [RImage] ))
predictions

y_pred = np.argmax(predictions, axis=1)

print(y_pred)
# should give 0 as it is vacutainer
```

```
1/1 [==============================] - 0s 192ms/step
[0]
```