```
import tensorflow
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential,load_model,Model
from keras.layers import Conv2D,MaxPool2D,AveragePooling2D,Dense,Flatten,ZeroPadding2D,BatchNormalization,Activation,Add,Input,Dropout,(
from keras.optimizers import SGD
from keras.initializers import glorot_uniform
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
```

```
base_model_tf=ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),classes=7)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernel
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
#Model building
base_model_tf.trainable=False

pt=Input(shape=(224,224,3),dtype=tensorflow.float32) # Change the input dtype to float32
#func=tensorflow.cast(pt,tensorflow.float32)  # Remove this line, no longer needed
x=preprocess_input(pt) #This function used to zero-center each color channel wrt Imagenet dataset
model_resnet=base_model_tf(x,training=False) #This function used to zero-center each color channel wrt Imagenet dataset
model_resnet=GlobalAveragePooling2D()(model_resnet)
model_resnet=Dense(128,activation='relu')(model_resnet)
model_resnet=Dense(64,activation='relu')(model_resnet)
model_resnet=Dense(6,activation='softmax')(model_resnet)
```

```
model_main=Model(inputs=pt,outputs=model_resnet)
model_main.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-
get_item (GetItem)	(None, 224, 224)	0	input_layer_1[0][0]
get_item_1 (GetItem)	(None, 224, 224)	0	input_layer_1[0][0]
get_item_2 (GetItem)	(None, 224, 224)	0	input_layer_1[0][0]
stack (Stack)	(None, 224, 224, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 224, 224, 3)	0	stack[0][0]
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712	add[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	resnet50[0][0]
dense (Dense)	(None, 128)	262,272	global_average_poolin…
dense_1 (Dense)	(None, 64)	8,256	dense[0][0]
dense_2 (Dense)	(None, 6)	390	dense_1[0][0]

Total params: 23,858,630 (91.01 MB)

```
#Image augmentation
train_datagen= ImageDataGenerator(shear_range=0.2,zoom_range=0.2,horizontal_flip=False,vertical_flip=False
                                  ,fill_mode='nearest',width_shift_range=0.2,height_shift_range=0.2)

val_datagen=ImageDataGenerator()
```

```
path_train='/content/drive/MyDrive/Colab Notebooks/Dataset/Train'

path_valid='/content/drive/MyDrive/Colab Notebooks/Dataset/Vaild'

train= train_datagen.flow_from_directory(directory=path_train,batch_size=64,target_size=(224,224),
                                         color_mode='rgb',class_mode='categorical',seed=42)

valid=val_datagen.flow_from_directory(directory=path_valid,batch_size=64,target_size=(224,224),color_mode='rgb',class_mode='categorical
```

```
Found 438 images belonging to 6 classes.
Found 60 images belonging to 6 classes.
```

```
class_dict=train.class_indices
print(class_dict)
```

```
{'Anthrax': 0, 'Brucellosis': 1, 'CLA': 2, 'Endo parasite': 3, 'FMD': 4, 'Healthy': 5}
```

Double-click (or enter) to edit

```
li = list(class_dict.keys())
print(li)
```

```
['Anthrax', 'Brucellosis', 'CLA', 'Endo parasite', 'FMD', 'Healthy']
```

```
#CallBacks
es=EarlyStopping(monitor='val_accuracy',verbose=1,patience=7,mode='auto')
mc=ModelCheckpoint(filepath='/content/drive/MyDrive/Colab Notebooks/best_model.keras', # Provide a filename ending with '.keras'
                   monitor='val_accuracy',verbose=1,save_best_only=True)
lr=ReduceLROnPlateau(monitor='val_accuracy',verbose=1,patience=6,min_lr=0.001)
```

```
model_main.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

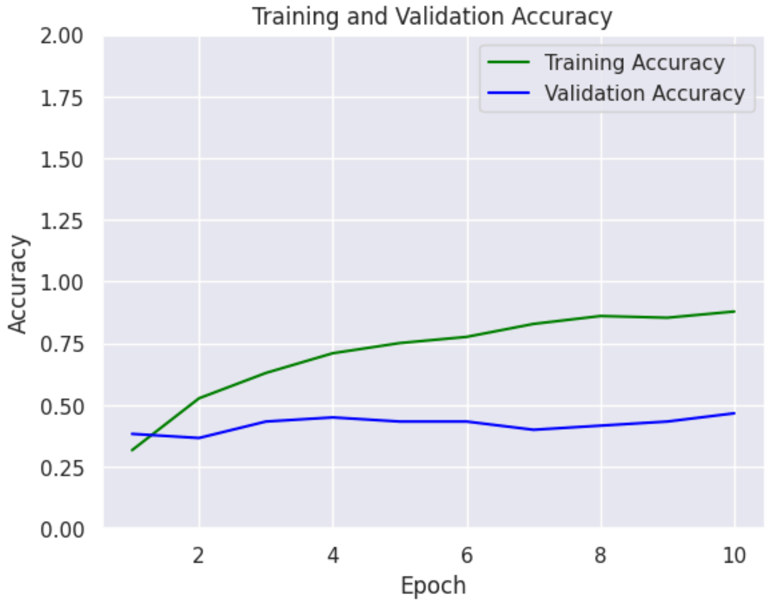
```
#Training
history = model_main.fit(train,validation_data=valid,epochs=10,steps_per_epoch=65,verbose=1,callbacks=[mc,es,lr])
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
 7/65 ━━━━━━━━━━━━━━━━━━━━ 1:05 1s/step - accuracy: 0.2460 - loss: 1.8208/usr/lib/python3.10/contextlib.py:153: UserWarning: Your in
  self.gen.throw(typ, value, traceback)

Epoch 1: val_accuracy improved from -inf to 0.38333, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.keras
65/65 ━━━━━━━━━━━━━━━━━━━━ 153s 601ms/step - accuracy: 0.3158 - loss: 1.7009 - val_accuracy: 0.3833 - val_loss: 1.5870 - learning_ra
Epoch 2/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 143ms/step - accuracy: 0.5909 - loss: 1.1847
Epoch 2: val_accuracy improved from 0.38333 to 0.48333, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.keras
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 33ms/step - accuracy: 0.5648 - loss: 1.2108 - val_accuracy: 0.4833 - val_loss: 1.4145 - learning_rate
Epoch 3/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 146ms/step - accuracy: 0.6471 - loss: 0.9561
Epoch 3: val_accuracy did not improve from 0.48333
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.6462 - loss: 0.9546 - val_accuracy: 0.4667 - val_loss: 1.4697 - learning_rate
Epoch 4/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 143ms/step - accuracy: 0.6803 - loss: 0.8391
Epoch 4: val_accuracy did not improve from 0.48333
65/65 ━━━━━━━━━━━━━━━━━━━━ 7s 22ms/step - accuracy: 0.7109 - loss: 0.8059 - val_accuracy: 0.4333 - val_loss: 1.4344 - learning_rate
Epoch 5/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 140ms/step - accuracy: 0.7279 - loss: 0.6905
Epoch 5: val_accuracy did not improve from 0.48333
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - accuracy: 0.7303 - loss: 0.7073 - val_accuracy: 0.4833 - val_loss: 1.5023 - learning_rate
Epoch 6/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 151ms/step - accuracy: 0.8013 - loss: 0.6169
Epoch 6: val_accuracy improved from 0.48333 to 0.51667, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.keras
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 40ms/step - accuracy: 0.8054 - loss: 0.5842 - val_accuracy: 0.5167 - val_loss: 1.4778 - learning_rate
Epoch 7/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 142ms/step - accuracy: 0.8252 - loss: 0.5646
Epoch 7: val_accuracy did not improve from 0.51667
65/65 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.8182 - loss: 0.5419 - val_accuracy: 0.4500 - val_loss: 1.5285 - learning_rate
Epoch 8/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 142ms/step - accuracy: 0.8880 - loss: 0.4041
Epoch 8: val_accuracy did not improve from 0.51667
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - accuracy: 0.8800 - loss: 0.4128 - val_accuracy: 0.4333 - val_loss: 1.4933 - learning_rate
Epoch 9/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 142ms/step - accuracy: 0.9023 - loss: 0.3541
Epoch 9: val_accuracy did not improve from 0.51667
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.8958 - loss: 0.3608 - val_accuracy: 0.4833 - val_loss: 1.5753 - learning_rate
Epoch 10/10
 7/65 ━━━━━━━━━━━━━━━━━━━━ 8s 144ms/step - accuracy: 0.8991 - loss: 0.3329
Epoch 10: val_accuracy improved from 0.51667 to 0.55000, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.keras
65/65 ━━━━━━━━━━━━━━━━━━━━ 11s 33ms/step - accuracy: 0.9036 - loss: 0.3220 - val_accuracy: 0.5500 - val_loss: 1.4970 - learning_rate
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
print(acc)
print(val_acc)
print(loss)
print(val_loss )
print(epochs)
```

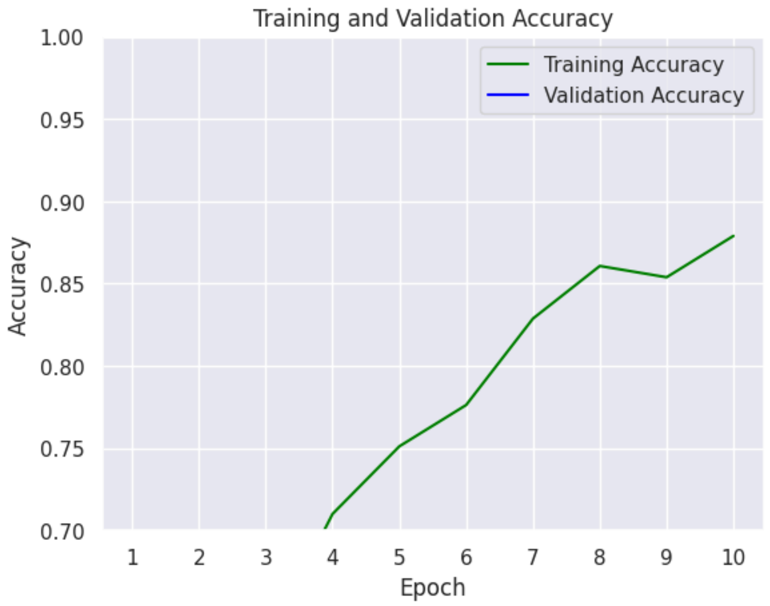
```
[0.3173516094684601, 0.5273972749710083, 0.6301369667053223, 0.7100456357002258, 0.7511415481567383, 0.77625572681427, 0.828767120834
[0.38333332538604736, 0.36666667461395264, 0.4333333373069763, 0.44999998807907104, 0.4333333373069763, 0.4333333373069763, 0.400000
[1.7105236053466797, 1.2071670293807983, 0.979316234588623, 0.824601948261261, 0.7327224612236023, 0.6366028189659119, 0.54018229246
[1.5616118907928467, 1.4973362684249878, 1.569789171218872, 1.5154476165771484, 1.7454307079315186, 1.6852245330810547, 1.65258717537
range(1, 11)
```

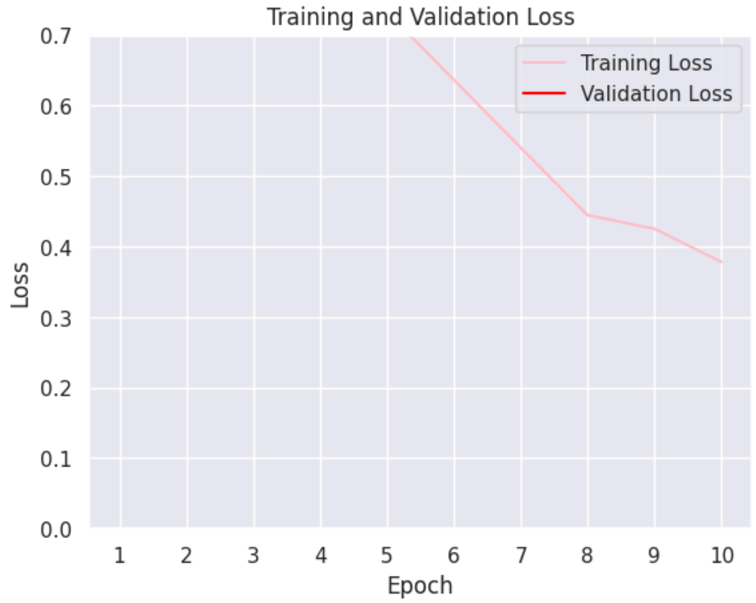
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)

#accuracy plot
plt.plot(epochs, acc, color='green', label='Training Accuracy')
plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.ylim(0, 2)  # Set y-axis limits starting from 0 (assuming accuracy range is 0 to 1)
plt.legend()

plt.figure()
#loss plot
plt.plot(epochs, loss, color='pink', label='Training Loss')
plt.plot(epochs, val_loss, color='red', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.ylim(0, 2)  # Set y-axis limits starting from 0 (assuming accuracy range is 0 to 1)

plt.legend()

plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)

# Accuracy plot
plt.plot(epochs, acc, color='green', label='Training Accuracy')
plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.xticks(epochs)  # Explicitly set x-axis ticks to show epochs 1, 2, ..., 10
plt.ylim(0.7, 1)  # Set y-axis limits starting from 0.7 to 1 for accuracy
plt.legend()

plt.figure()
# Loss plot
plt.plot(epochs, loss, color='pink', label='Training Loss')
plt.plot(epochs, val_loss, color='red', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks(epochs)  # Explicitly set x-axis ticks to show epochs 1, 2, ..., 10
plt.ylim(0, 0.7)  # Set y-axis limits starting from 0 to 0.7 for loss
plt.legend()
```

```
plt.show()
```






```
import joblib
# Save the model as a pickle in a file
joblib.dump(model_main, '/content/drive/MyDrive/Colab Notebooks/resnet50_1.pkl')

# Load the model from the file
# classifier = joblib.load('vgg16.pkl')
```

```
['/content/drive/MyDrive/Colab Notebooks/resnet50_1.pkl']
```

```
#Saving our model
filepath="/content/drive/MyDrive/Colab Notebooks/Resnet50_2.h5"
model_main.save(filepath)
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
# predicting an image
from keras.preprocessing import image
import numpy as np
image_path = "/content/drive/MyDrive/Colab Notebooks/Dataset/Test/Anthrax/Anthrax 91.jpeg"
new_img = image.load_img(image_path, target_size=(224, 224))
img = image.img_to_array(new_img)
img = np.expand_dims(img, axis=0)
# img = img/255

print("Following is our prediction:")
prediction = model_main.predict(img)
# decode the results into a list of tuples (class, description, probability)
```

```
# (one such list for each sample in the batch)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = li[index]

#ploting image with predicted class name
plt.figure(figsize = (4,4))
plt.imshow(new_img)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
```

Anthrax

```
# predicting an image
from keras.preprocessing import image
import numpy as np
all_preds = []
all_labels = []
imagesname=['Anthrax', 'Brucellosis', 'CLA', 'Endo parasite', 'FMD','Healthy']

for nme in imagesname:
  for i in range(91,101):
    image_path = "/content/drive/MyDrive/Colab Notebooks/Dataset/Test/"+nme+"/"+nme+" "+str(i)+".jpeg"
    new_img = image.load_img(image_path, target_size=(224, 224))
    img = image.img_to_array(new_img)
    img = np.expand_dims(img, axis=0)
    # img = img/255

    print("Following is our prediction:")
    prediction = model_main.predict(img)
    # decode the results into a list of tuples (class, description, probability)
    # (one such list for each sample in the batch)
    d = prediction.flatten()
    j = d.max()
    for index,item in enumerate(d):
        if item == j:
            class_name = li[index]

    all_preds.append(class_name)
    all_labels.append(nme)
    #ploting image with predicted class name
    plt.figure(figsize = (2,2))
    plt.imshow(new_img)
    plt.axis('off')
    plt.title(class_name)
    plt.show()
```

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

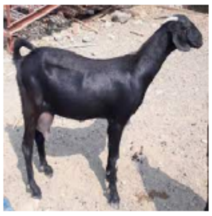
Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```

Healthy

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

Brucellosis



```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Brucellosis

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

CLA

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

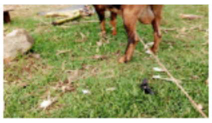
CLA

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

Brucellosis



```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Brucellosis

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

Healthy

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```

Brucellosis

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Brucellosis

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

Anthrax

```
Following is our prediction:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

CLA