```
import sys
import os
from math import log
import numpy as np
import scipy as sp
from PIL import Image
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator


path_root = "malimg_dataset/train"


batches = ImageDataGenerator().flow_from_directory(directory=path_root, target_size=(64,64), batch_size=10000)
```

```
     Found 7459 images belonging to 25 classes.
```

```
imgs, labels = next(batches)
imgs.shape
labels.shape

# plots images with labels within jupyter notebook
def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = 10 # len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(0,50):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(list(batches.class_indices.keys())[np.argmax(titles[i])], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')

plots(imgs, titles = labels)
classes = batches.class_indices.keys()
perc = (sum(labels)/labels.shape[0])*100
plt.xticks(rotation='vertical')
plt.bar(classes,perc)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(imgs/255.,labels, test_size=0.3)
X_train.shape
X_test.shape
import keras
from keras.models import Sequential, Input, Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import batch_normalization
num_classes = 25
def malware_model():
    Malware_model = Sequential()
    Malware_model.add(Conv2D(30, kernel_size=(3, 3),
                     activation='relu',
```

```
                        input_shape=(64,64,3)))

    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Dropout(0.25))
    Malware_model.add(Flatten())
    Malware_model.add(Dense(128, activation='relu'))
    Malware_model.add(Dropout(0.5))
    Malware_model.add(Dense(50, activation='relu'))
    Malware_model.add(Dense(num_classes, activation='softmax'))
    Malware_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'])
    return Malware_model
Malware_model = malware_model()
Malware_model.summary()
y_train.shape
y_train_new = np.argmax(y_train, axis=1)
y_train_new
Malware_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=22)
print('Final CNN accuracy: ', scores[1])
import numpy as np
import pandas as pd
y_pred = Malware_model.predict(X_test[[0]])
y_pred
y_test2 = np.argmax(y_test)
print(y_test2)
from tensorflow.keras.models import save_model
from tensorflow.keras.models import load_model
save_model(Malware_model,'malware.h5')
model2=load_model('./malware.h5')
model2.evaluate(X_test,y_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-20b44b02c805> in <cell line: 1>()
----> 1 imgs, labels = next(batches)
      2 imgs.shape
      3 labels.shape
      4
      5 # plots images with labels within jupyter notebook

NameError: name 'batches' is not defined
```

SEARCH STACK OVERFLOW

```
cd drive/MyDrive/malware_detection/
```

```
/content/drive/MyDrive/malware_detection
```

```
import csv
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load and preprocess the data
```

```
# Load and preprocess the data
data = []
with open("Import.csv") as f:
    reader = csv.reader(f)
    next(reader)  # Skip the header row
    for row in reader:
        evidence = [int(cell) for cell in row[:-1]]
        label = int(row[-1])
        data.append((evidence, label))

# Separate data into features (evidence) and labels
evidence = [row[0] for row in data]
labels = [row[1] for row in data]

# Convert to NumPy arrays and scale the features
evidence = np.array(evidence, dtype=float)
scaler = StandardScaler()
evidence = scaler.fit_transform(evidence)
labels = np.array(labels)

# Split the data into training and testing sets
X_training, X_testing, y_training, y_testing = train_test_split(evidence, labels, test_size=0.25)

# Create a neural network model
model = tf.keras.models.Sequential()

# Add a hidden layer with ReLU activation
model.add(tf.keras.layers.Dense(int(evidence.shape[1]), input_shape=(evidence.shape[1],), activation="relu"))
model.add(tf.keras.layers.Dense(120, activation="relu"))
model.add(tf.keras.layers.Dense(120, activation="sigmoid"))
model.add(tf.keras.layers.Dense(120, activation="sigmoid"))
# Add the output layer with softmax activation for multi-class classification
num_classes = len(np.unique(labels))
model.add(tf.keras.layers.Dense(num_classes, activation="sigmoid"))

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Train the neural network
model.fit(X_training, y_training, epochs=250, batch_size=64, validation_data=(X_testing, y_testing))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_testing, y_testing, verbose=2)
print(f"Test loss: {loss}, Test accuracy: {accuracy}")

# Save the trained model to a file
model.save("model.h5")
```

```
Epoch 1/250
5/5 [==============================] - 1s 110ms/step - loss: 1.7658 - accuracy: 0.5700 - val_loss: 1.7453 - val_accuracy: 0.5248
Epoch 2/250
5/5 [==============================] - 0s 87ms/step - loss: 1.5917 - accuracy: 0.5933 - val_loss: 1.6726 - val_accuracy: 0.5248
```

```
Epoch 3/250
5/5 [==============================] - 0s 90ms/step - loss: 1.4798 - accuracy: 0.6200 - val_loss: 1.6151 - val_accuracy: 0.5248
Epoch 4/250
5/5 [==============================] - 0s 88ms/step - loss: 1.3919 - accuracy: 0.6200 - val_loss: 1.5723 - val_accuracy: 0.5248
Epoch 5/250
5/5 [==============================] - 0s 98ms/step - loss: 1.3217 - accuracy: 0.6200 - val_loss: 1.5399 - val_accuracy: 0.5248
Epoch 6/250
5/5 [==============================] - 0s 81ms/step - loss: 1.2606 - accuracy: 0.6233 - val_loss: 1.5153 - val_accuracy: 0.5248
Epoch 7/250
5/5 [==============================] - 0s 83ms/step - loss: 1.2107 - accuracy: 0.6300 - val_loss: 1.4959 - val_accuracy: 0.5248
Epoch 8/250
5/5 [==============================] - 0s 81ms/step - loss: 1.1672 - accuracy: 0.6300 - val_loss: 1.4806 - val_accuracy: 0.5248
Epoch 9/250
5/5 [==============================] - 0s 62ms/step - loss: 1.1270 - accuracy: 0.6467 - val_loss: 1.4672 - val_accuracy: 0.5446
Epoch 10/250
5/5 [==============================] - 0s 65ms/step - loss: 1.0909 - accuracy: 0.6667 - val_loss: 1.4562 - val_accuracy: 0.5446
Epoch 11/250
5/5 [==============================] - 0s 69ms/step - loss: 1.0552 - accuracy: 0.6833 - val_loss: 1.4459 - val_accuracy: 0.5446
Epoch 12/250
5/5 [==============================] - 0s 63ms/step - loss: 1.0228 - accuracy: 0.7033 - val_loss: 1.4374 - val_accuracy: 0.5743
Epoch 13/250
5/5 [==============================] - 0s 63ms/step - loss: 0.9914 - accuracy: 0.7400 - val_loss: 1.4312 - val_accuracy: 0.5743
Epoch 14/250
5/5 [==============================] - 0s 61ms/step - loss: 0.9612 - accuracy: 0.7533 - val_loss: 1.4245 - val_accuracy: 0.5743
Epoch 15/250
5/5 [==============================] - 0s 68ms/step - loss: 0.9318 - accuracy: 0.7600 - val_loss: 1.4150 - val_accuracy: 0.5644
Epoch 16/250
5/5 [==============================] - 0s 62ms/step - loss: 0.9044 - accuracy: 0.7600 - val_loss: 1.4068 - val_accuracy: 0.5545
Epoch 17/250
5/5 [==============================] - 0s 63ms/step - loss: 0.8774 - accuracy: 0.7767 - val_loss: 1.3976 - val_accuracy: 0.5545
Epoch 18/250
5/5 [==============================] - 0s 70ms/step - loss: 0.8531 - accuracy: 0.7800 - val_loss: 1.3902 - val_accuracy: 0.5644
Epoch 19/250
5/5 [==============================] - 0s 64ms/step - loss: 0.8277 - accuracy: 0.7800 - val_loss: 1.3820 - val_accuracy: 0.5644
Epoch 20/250
5/5 [==============================] - 0s 67ms/step - loss: 0.8041 - accuracy: 0.7800 - val_loss: 1.3743 - val_accuracy: 0.5644
Epoch 21/250
5/5 [==============================] - 0s 65ms/step - loss: 0.7808 - accuracy: 0.7867 - val_loss: 1.3676 - val_accuracy: 0.5644
Epoch 22/250
5/5 [==============================] - 0s 65ms/step - loss: 0.7593 - accuracy: 0.7867 - val_loss: 1.3614 - val_accuracy: 0.5644
Epoch 23/250
5/5 [==============================] - 0s 68ms/step - loss: 0.7376 - accuracy: 0.7867 - val_loss: 1.3542 - val_accuracy: 0.5644
Epoch 24/250
5/5 [==============================] - 0s 67ms/step - loss: 0.7177 - accuracy: 0.7933 - val_loss: 1.3473 - val_accuracy: 0.5446
Epoch 25/250
5/5 [==============================] - 0s 63ms/step - loss: 0.6985 - accuracy: 0.7967 - val_loss: 1.3401 - val_accuracy: 0.5347
Epoch 26/250
5/5 [==============================] - 0s 62ms/step - loss: 0.6795 - accuracy: 0.8100 - val_loss: 1.3338 - val_accuracy: 0.5446
Epoch 27/250
5/5 [==============================] - 0s 69ms/step - loss: 0.6625 - accuracy: 0.8300 - val_loss: 1.3274 - val_accuracy: 0.5446
Epoch 28/250
5/5 [==============================] - 0s 67ms/step - loss: 0.6471 - accuracy: 0.8333 - val_loss: 1.3214 - val_accuracy: 0.5446
```