Static Malware Detection

I. Introduction

We propose a simple effective method for visualizing and classifying malware using image processing techniques. Malware can be visualized as image, and there are significant visual similarities in image texture for malware that belonging to the same family because it is common to reuse the code to create new malware variants. Therefore, we can classify malware family by CNN model. After we monitor a malware and predict which model it is, we can protect our computer faster.

II. Set Environment

A. Download VMware (https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html.html)

B. Download Ubuntu ISO file (https://docs.ossii.com.tw/books/ubuntu-server-2004/page/ubuntu-server-2004-iso)

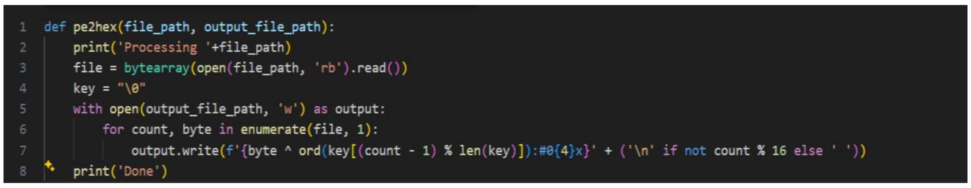
III. Change Image

A. Malware Binary Code

1. Download data from boozallen/MOTIF (https://github.com/boozallen/MOTIF)
2. Decompress the file.

B. Binary to 8 bits Vector

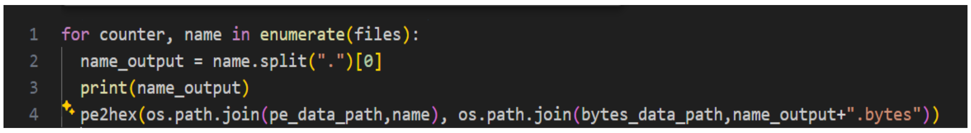
1. This is a function that converts PE files into hexadecimal files.



```python
1 def pe2hex(file_path, output_file_path):
2     print('Processing '+file_path)
3     file = bytearray(open(file_path, 'rb').read())
4     key = "\0"
5     with open(output_file_path, 'w') as output:
6         for count, byte in enumerate(file, 1):
7             output.write(f'{byte ^ ord(key[(count - 1) % len(key)]):#0{4}x}' + ('\n' if not count % 16 else ' '))
8     print('Done')
```

   a. Line 3: Open and read the PE file in binary mode, then convert it into a byte array.
   b. Line 6: Iterate through each byte and convert it into its hexadecimal representation.
   c. Line 7: Write each byte to the output file and start a new line every 16 bytes.

2. Iterate through each file to prepare for conversion.

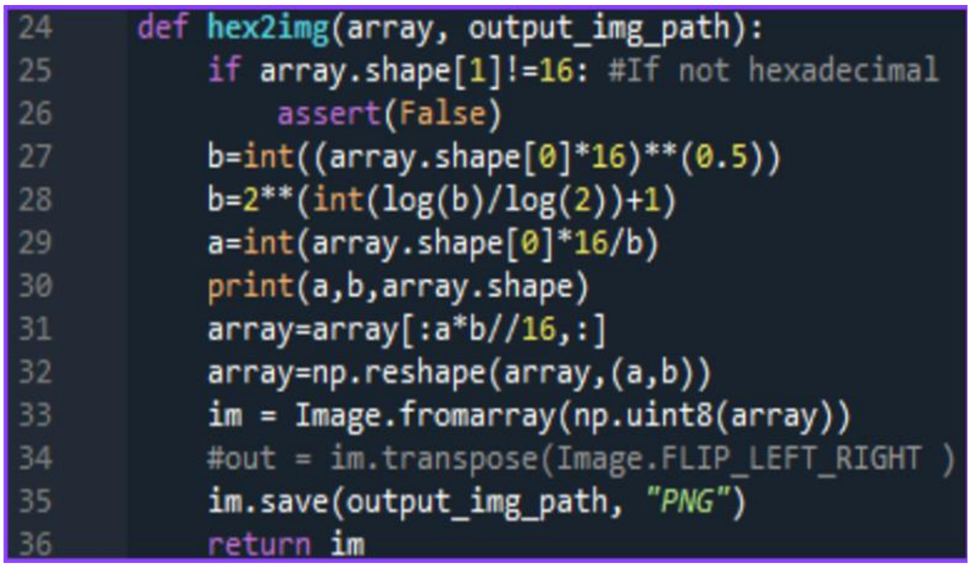


```
1  for counter, name in enumerate(files):
2      name_output = name.split(".")[0]
3      print(name_output)
4      pe2hex(os.path.join(pe_data_path,name), os.path.join(bytes_data_path,name_output+".bytes"))
```

   a. Line 2: Remove the file extension from the filename for generating the output filename.
   b. Line 4: Call the preceding function to convert the PE file into a hexadecimal file.

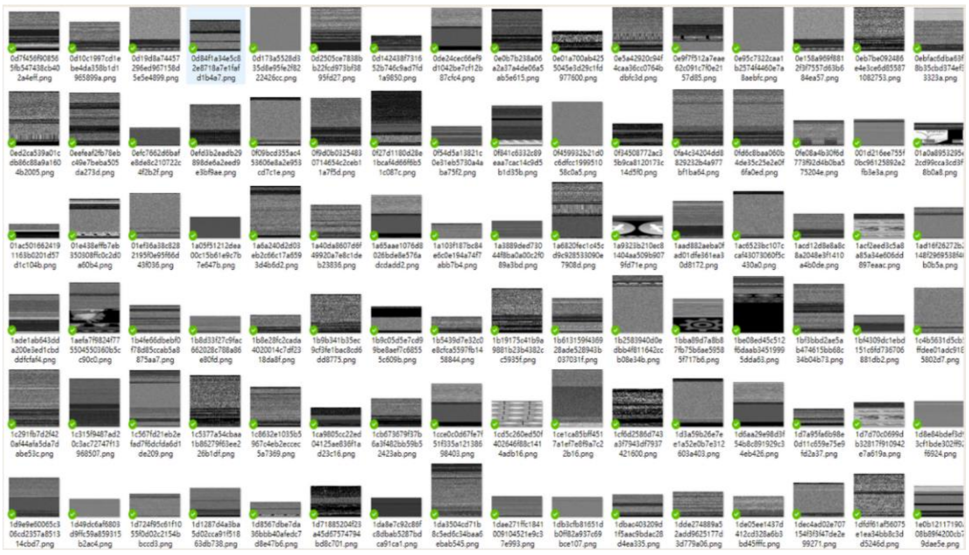
C. 8 bits Vector to Grayscale Image

1. Convert a one-dimensional hexadecimal data array into an image.



```
24  def hex2img(array, output_img_path):
25      if array.shape[1]!=16: #If not hexadecimal
26          assert(False)
27      b=int((array.shape[0]*16)**(0.5))
28      b=2**(int(log(b)/log(2))+1)
29      a=int(array.shape[0]*16/b)
30      print(a,b,array.shape)
31      array=array[:a*b//16,:]
32      array=np.reshape(array,(a,b))
33      im = Image.fromarray(np.uint8(array))
34      #out = im.transpose(Image.FLIP_LEFT_RIGHT )
35      im.save(output_img_path, "PNG")
36      return im
```

   a. Line 27: Calculate the image width by first multiplying the length of the array by 16 (the number of elements per row) to get the total number of pixels, and then take the square root to estimate the side length.
   b. Line 28: Adjust the side length to the next power of 2 to ensure the dimensions are suitable for image conversion.
   c. Line 29: Calculate the other dimension of the image based on the total number of bits and the adjusted side length 'b'.
   d. Line 31: Trim or reshape the array to fit the new dimensions ('a' and 'b'), ensuring it's suitable for image generation.

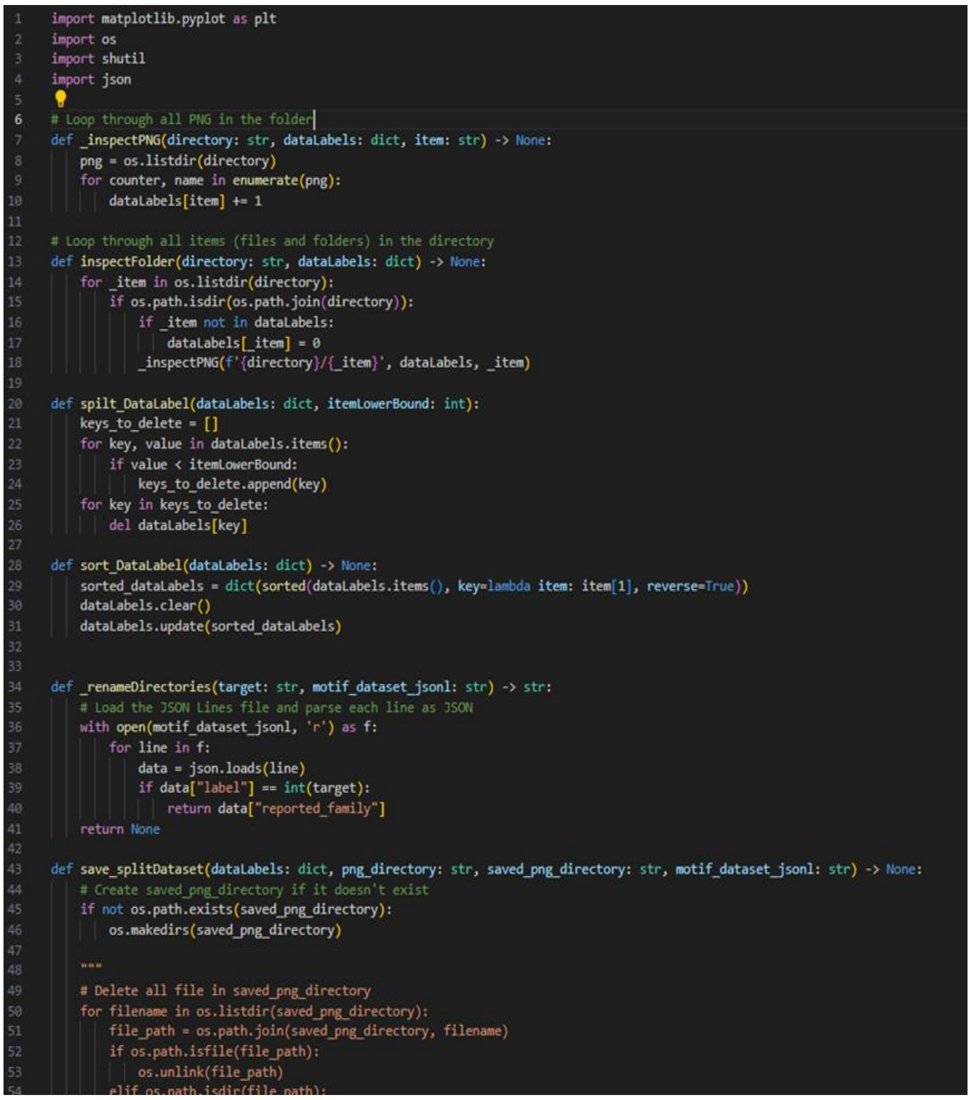


Reference:https://github.com/cridin1/malware-classification-CNN/blob/main/utils/data_conversion.ipynb

IV. Train Data

A. Exclusive of malware family with few data

1. This code implements a script for processing and managing an image dataset, specifically targeting PNG images contained within a specific directory. It accomplishes this task by counting the number of images in the directory, filtering, sorting, renaming, and saving the processed dataset.



```python
1   import matplotlib.pyplot as plt
2   import os
3   import shutil
4   import json
5
6   # Loop through all PNG in the folder
7   def _inspectPNG(directory: str, dataLabels: dict, item: str) -> None:
8       png = os.listdir(directory)
9       for counter, name in enumerate(png):
10          dataLabels[item] += 1
11
12  # Loop through all items (files and folders) in the directory
13  def inspectFolder(directory: str, dataLabels: dict) -> None:
14      for _item in os.listdir(directory):
15          if os.path.isdir(os.path.join(directory)):
16              if _item not in dataLabels:
17                  dataLabels[_item] = 0
18              _inspectPNG(f'{directory}/{_item}', dataLabels, _item)
19
20  def spilt_DataLabel(dataLabels: dict, itemLowerBound: int):
21      keys_to_delete = []
22      for key, value in dataLabels.items():
23          if value < itemLowerBound:
24              keys_to_delete.append(key)
25      for key in keys_to_delete:
26          del dataLabels[key]
27
28  def sort_DataLabel(dataLabels: dict) -> None:
29      sorted_dataLabels = dict(sorted(dataLabels.items(), key=lambda item: item[1], reverse=True))
30      dataLabels.clear()
31      dataLabels.update(sorted_dataLabels)
32
33
34  def _renameDirectories(target: str, motif_dataset_jsonl: str) -> str:
35      # Load the JSON Lines file and parse each line as JSON
36      with open(motif_dataset_jsonl, 'r') as f:
37          for line in f:
38              data = json.loads(line)
39              if data["label"] == int(target):
40                  return data["reported_family"]
41      return None
42
43  def save_splitDataset(dataLabels: dict, png_directory: str, saved_png_directory: str, motif_dataset_jsonl: str) -> None:
44      # Create saved_png_directory if it doesn't exist
45      if not os.path.exists(saved_png_directory):
46          os.makedirs(saved_png_directory)
47
48      """
49      # Delete all file in saved_png_directory
50      for filename in os.listdir(saved_png_directory):
51          file_path = os.path.join(saved_png_directory, filename)
52          if os.path.isfile(file_path):
53              os.unlink(file_path)
54          elif os.path.isdir(file_path):
```

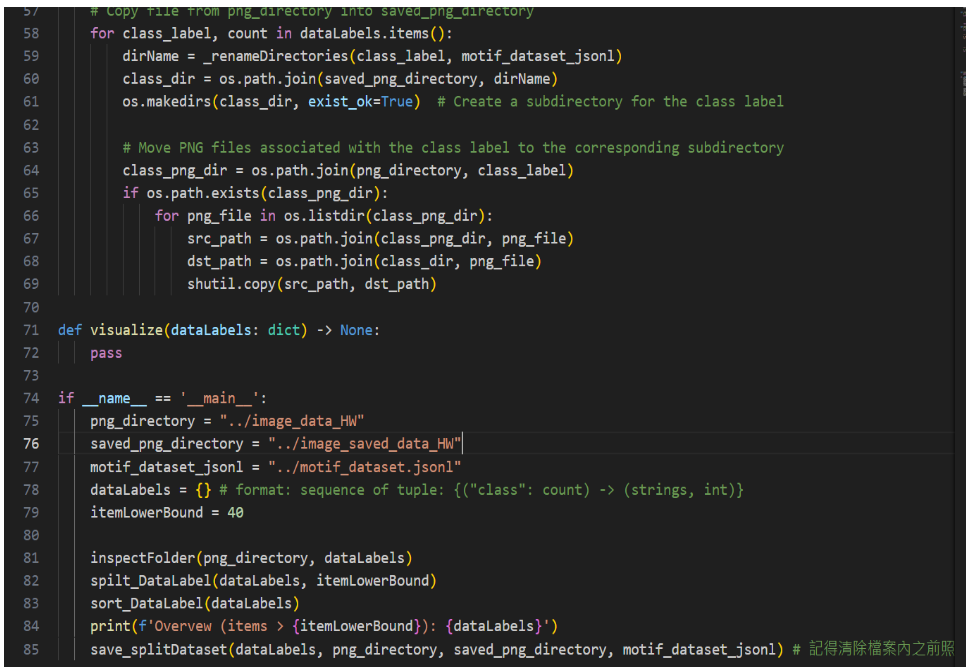
a. Line 17: If it is a directory and not in dataLabels, then add it and set the initial count to 0.

b. Line 26: Delete items with a count below the threshold.

c. Line 31: Update the dictionary to the sorted dictionary.

d. Line 39: Check if the label matches the target.

2. This code snippet is a high-level outline for a script designed to process and manipulate a dataset, specifically targeting image data and associated labels. The script organizes, filters, and saves the processed data according to specified criteria.



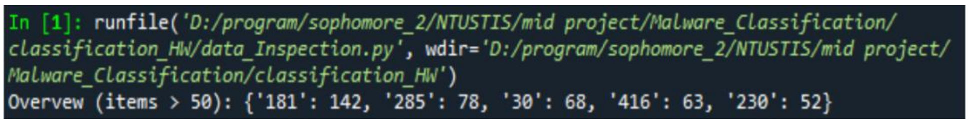
```
57          # Copy file from png_directory into saved_png_directory
58          for class_label, count in dataLabels.items():
59              dirName = _renameDirectories(class_label, motif_dataset_jsonl)
60              class_dir = os.path.join(saved_png_directory, dirName)
61              os.makedirs(class_dir, exist_ok=True)  # Create a subdirectory for the class label
62
63              # Move PNG files associated with the class label to the corresponding subdirectory
64              class_png_dir = os.path.join(png_directory, class_label)
65              if os.path.exists(class_png_dir):
66                  for png_file in os.listdir(class_png_dir):
67                      src_path = os.path.join(class_png_dir, png_file)
68                      dst_path = os.path.join(class_dir, png_file)
69                      shutil.copy(src_path, dst_path)
70
71  def visualize(dataLabels: dict) -> None:
72      pass
73
74  if __name__ == '__main__':
75      png_directory = "../image_data_HW"
76      saved_png_directory = "../image_saved_data_HW"
77      motif_dataset_jsonl = "../motif_dataset.jsonl"
78      dataLabels = {} # format: sequence of tuple: {("class": count) -> (strings, int)}
79      itemLowerBound = 40
80
81      inspectFolder(png_directory, dataLabels)
82      spilt_DataLabel(dataLabels, itemLowerBound)
83      sort_DataLabel(dataLabels)
84      print(f'Overvew (items > {itemLowerBound}): {dataLabels}')
85      save_splitDataset(dataLabels, png_directory, saved_png_directory, motif_dataset_jsonl) # 記得清除檔案內之前照
```

a. Line 79: Define a lower bound for item filtering, indicating a minimum count requirement for items to be included in processing

b. Line 82: Call a function to split or filter `dataLabels` based on `itemLowerBound`, modifying `dataLabels` directly.

c. Line 83: Call a function to sort `dataLabels` in a specific order, likely based on the class names or counts.

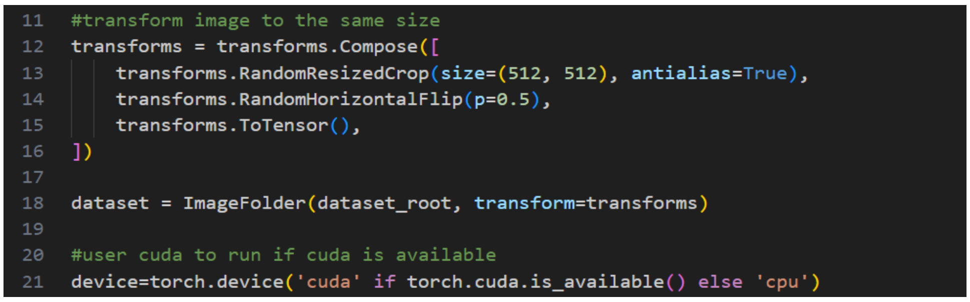
3. Final approach: many classes have fewer than 50 instances.



```
In [1]: runfile('D:/program/sophomore_2/NTUSTIS/mid project/Malware_Classification/classification_HW/data_Inspection.py', wdir='D:/program/sophomore_2/NTUSTIS/mid project/Malware_Classification/classification_HW')
Overvew (items > 50): {'181': 142, '285': 78, '30': 68, '416': 63, '230': 52}
```

B. Convert images to the same size

1. This code snippet is from PyTorch. It sets up an image transformation pipeline and prepares a dataset for training or evaluation on a neural network, leveraging GPU computation if available.



```
11  #transform image to the same size
12  transforms = transforms.Compose([
13      transforms.RandomResizedCrop(size=(512, 512), antialias=True),
14      transforms.RandomHorizontalFlip(p=0.5),
15      transforms.ToTensor(),
16  ])
17
18  dataset = ImageFolder(dataset_root, transform=transforms)
19
20  #user cuda to run if cuda is available
21  device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

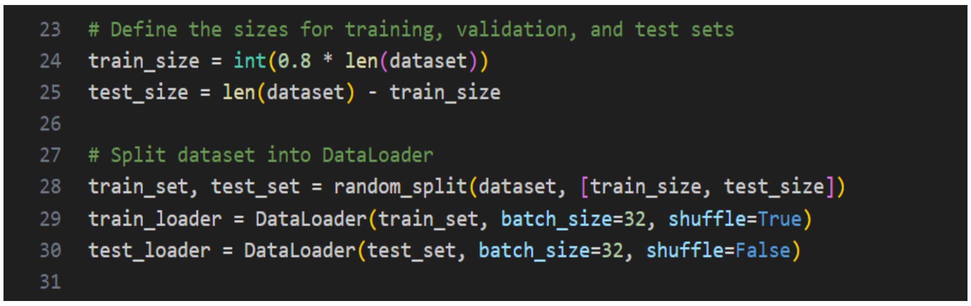
a. Line 13: Randomly crops the images to 512x512 pixels with antialiasing to reduce aliasing effects.

b. Line 14: Randomly flips the images horizontally with a 50% probability.

c. Line 21: Sets the device to 'cuda' if a CUDA-compatible GPU is available, otherwise falls back to the CPU.

C. Split train and test data

1. This block of code is used for splitting a dataset into training and testing sets and then loading those subsets into data loaders for use in training and evaluating a machine learning model.



```
23  # Define the sizes for training, validation, and test sets
24  train_size = int(0.8 * len(dataset))
25  test_size = len(dataset) - train_size
26
27  # Split dataset into DataLoader
28  train_set, test_set = random_split(dataset, [train_size, test_size])
29  train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
30  test_loader = DataLoader(test_set, batch_size=32, shuffle=False)
31
```

a. Line 24: Calculate the training set size as 80% of the total dataset size.

b. Line 25: Calculate the test set size as the remaining 20% of the dataset.

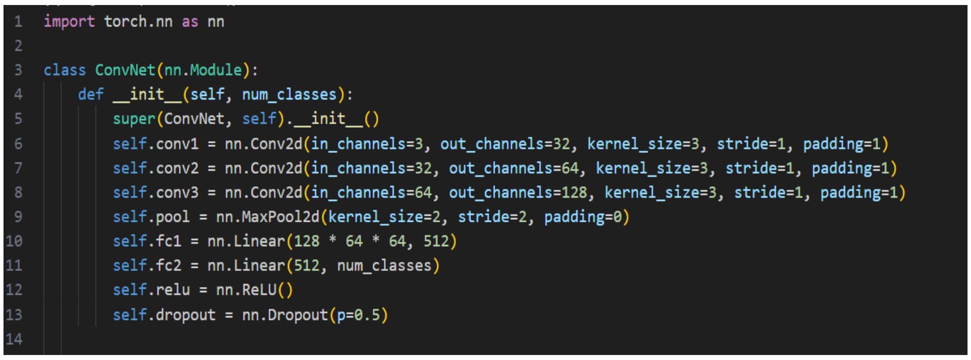
c. Line 28: Split the dataset into training and test sets according to the sizes calculated above.

d. Line 29: Create a DataLoader for the training set, with a batch size of 32 and shuffling enabled to mix the data before each epoch.

e. Line 30: Create a DataLoader for the test set, with a batch size of 32 and shuffling disabled, as shuffling is not necessary for evaluation.

D. Train and test data

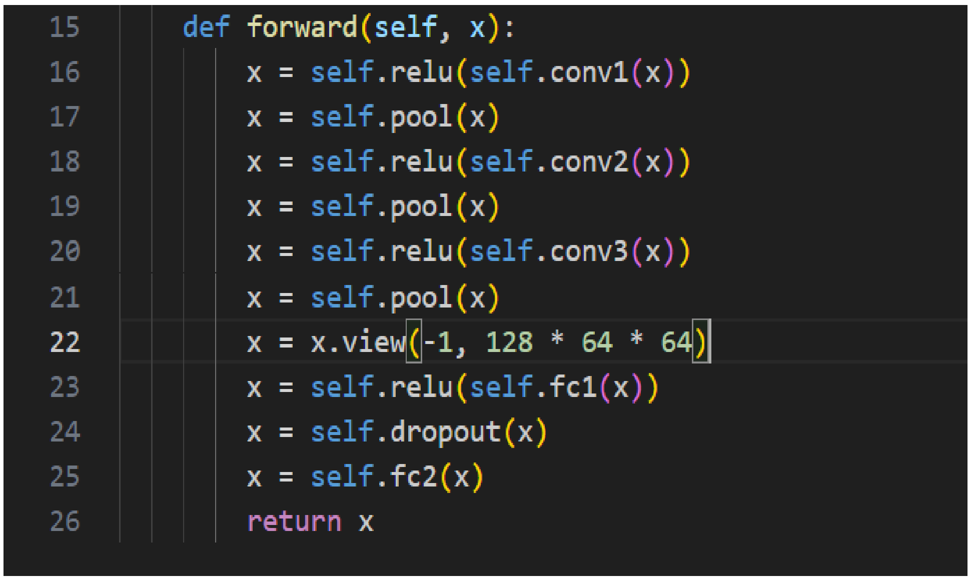
1. This code defines the structure of a convolutional neural network (ConvNet) for image recognition tasks, especially when dealing with datasets with multiple classes.



```python
1   import torch.nn as nn
2
3   class ConvNet(nn.Module):
4       def __init__(self, num_classes):
5           super(ConvNet, self).__init__()
6           self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1)
7           self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1)
8           self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1)
9           self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
10          self.fc1 = nn.Linear(128 * 64 * 64, 512)
11          self.fc2 = nn.Linear(512, num_classes)
12          self.relu = nn.ReLU()
13          self.dropout = nn.Dropout(p=0.5)
14
```

a. Line 6: Defines the first convolutional layer with 3 input channels (assuming the input images are RGB), 32 output channels, a kernel size of 3x3, stride of 1, and padding of 1.
b. Line 9: Defines a max pooling layer with a kernel size of 2x2, stride of 2, and no padding.
c. Line 10: Defines the first fully connected layer with 128*64*64 input features (assuming this is the size of the feature map after convolutions and pooling), and 512 output features.
d. Line 13: Defines a Dropout layer with a dropout rate of 0.5, used to reduce overfitting.

2. Defines the network's forward propagation path.

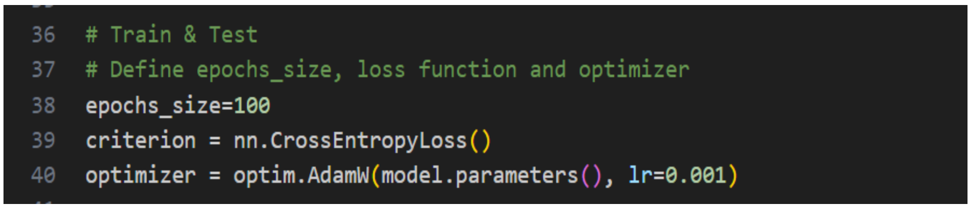


```python
15      def forward(self, x):
16          x = self.relu(self.conv1(x))
17          x = self.pool(x)
18          x = self.relu(self.conv2(x))
19          x = self.pool(x)
20          x = self.relu(self.conv3(x))
21          x = self.pool(x)
22          x = x.view(-1, 128 * 64 * 64)
23          x = self.relu(self.fc1(x))
24          x = self.dropout(x)
25          x = self.fc2(x)
26          return x
```

a. Line 16: Passes through the first convolutional layer, ReLU activation, and max pooling layer.
b. Line 22: Flattens the feature map into a one-dimensional vector for input to the fully connected layers.
c. Line 23: Passes through the first fully connected layer and ReLU

activation.

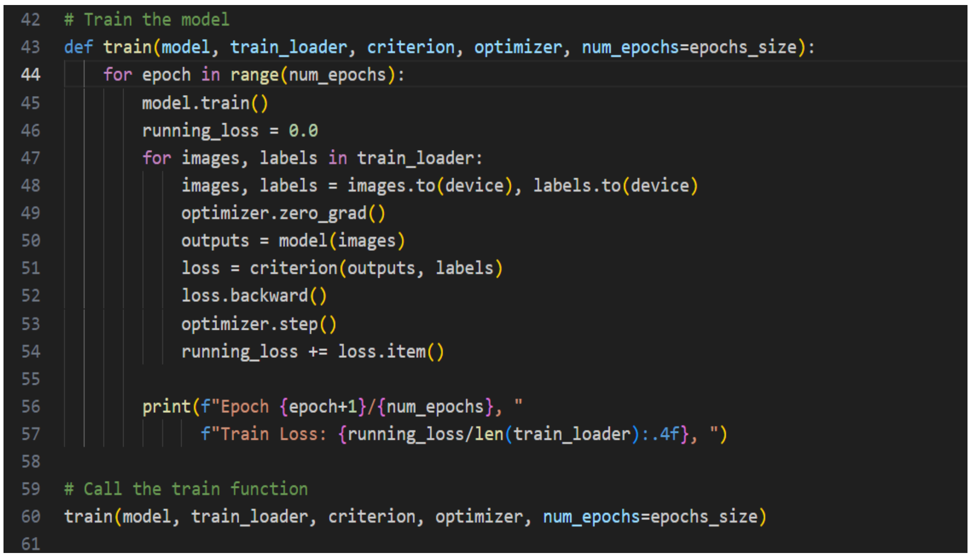
3. Define training parameters: number of epochs, loss function, and optimizer.



```
36  # Train & Test
37  # Define epochs_size, loss function and optimizer
38  epochs_size=100
39  criterion = nn.CrossEntropyLoss()
40  optimizer = optim.AdamW(model.parameters(), lr=0.001)
```

   a. Line 40: Initialize the optimizer as AdamW, a variant of the Adam optimization algorithm, with a learning rate (lr) of 0.001. The optimizer is responsible for updating the model's parameters (weights and biases) based on the gradients of the loss function, aiming to minimize the loss over the training epochs.

4. This code defines a function for training a neural network model and then calls that function to perform the training. It iterates over the dataset for a specified number of epochs, updating the model's weights to minimize the loss function. Here's the explanation with comments.

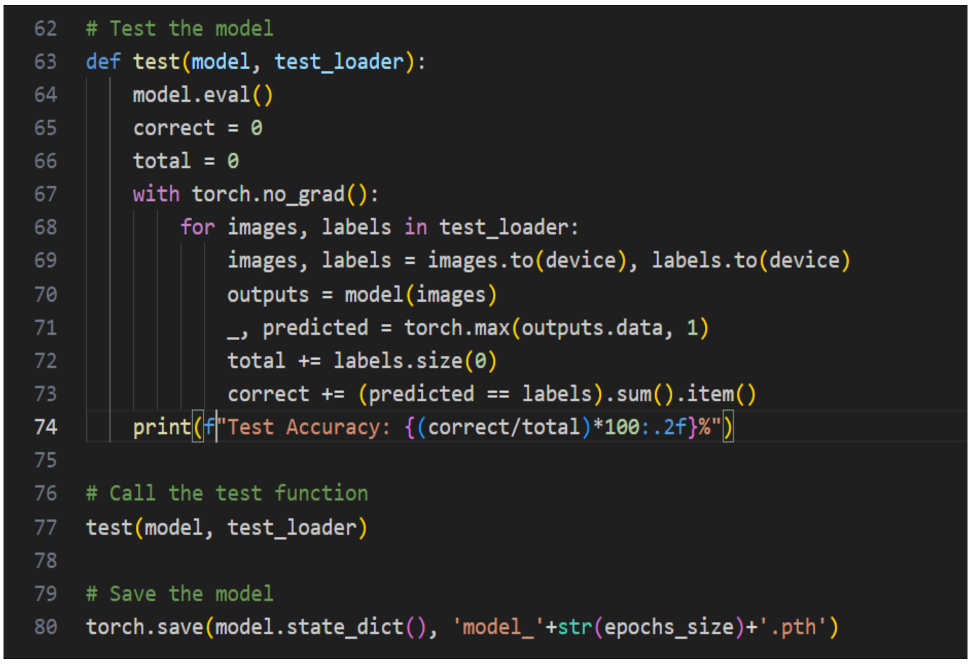


```
42  # Train the model
43  def train(model, train_loader, criterion, optimizer, num_epochs=epochs_size):
44      for epoch in range(num_epochs):
45          model.train()
46          running_loss = 0.0
47          for images, labels in train_loader:
48              images, labels = images.to(device), labels.to(device)
49              optimizer.zero_grad()
50              outputs = model(images)
51              loss = criterion(outputs, labels)
52              loss.backward()
53              optimizer.step()
54              running_loss += loss.item()
55
56          print(f"Epoch {epoch+1}/{num_epochs}, "
57                f"Train Loss: {running_loss/len(train_loader):.4f}, ")
58
59  # Call the train function
60  train(model, train_loader, criterion, optimizer, num_epochs=epochs_size)
61
```

   a. Line 45: Set the model to training mode. This is necessary because certain layers like dropout and batch norm behave differently during training.
   b. Line 51: Calculate the loss between the predicted outputs and the actual labels.

    c. Line 52: Backward pass: compute the gradient of the loss with respect to model parameters.
    d. Line 53: Perform a single optimization step (parameter update).

5. This code snippet is responsible for evaluating the trained model on a test dataset to measure its performance, specifically its accuracy, and then saving the model's learned parameters to a file.



```
62  # Test the model
63  def test(model, test_loader):
64      model.eval()
65      correct = 0
66      total = 0
67      with torch.no_grad():
68          for images, labels in test_loader:
69              images, labels = images.to(device), labels.to(device)
70              outputs = model(images)
71              _, predicted = torch.max(outputs.data, 1)
72              total += labels.size(0)
73              correct += (predicted == labels).sum().item()
74      print(f"Test Accuracy: {(correct/total)*100:.2f}%")
75
76  # Call the test function
77  test(model, test_loader)
78
79  # Save the model
80  torch.save(model.state_dict(), 'model_'+str(epochs_size)+'.pth')
```

    a. Line 64: Switch the model to evaluation mode. This is important for layers like dropout and batch normalization to work in inference mode.
    b. Line 69: Move the images and labels to the device (GPU or CPU).
    c. Line 71: Get the predictions from the maximum value of the output.
    d. Line 73: Update the number of correct predictions.
    e. Line 74: Print the accuracy of the model on the test dataset.

V. Outcome different approach

1. ignore malware family with less data

   →Training loss：4.91、Accuracy：10.8%

2. 10 epoch

   →Training loss：1.99、Accuracy：32.31%

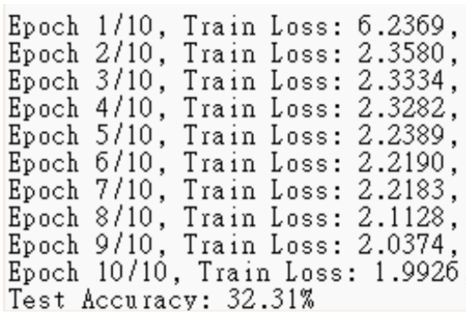
3. 20 epoch

   →Training loss：1.58、Accuracy：47.69%

4. 100 epoch

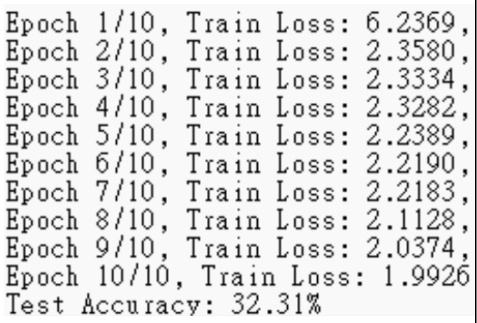
→Training loss：0.99、Accuracy：58.25%

Ignore less data



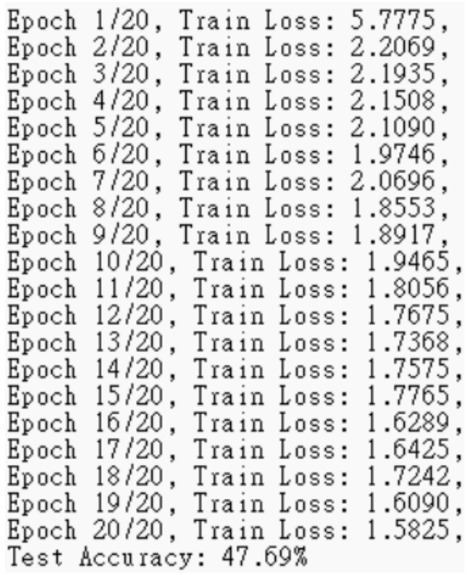
```
Epoch 1/10, Train Loss: 6.2369,
Epoch 2/10, Train Loss: 2.3580,
Epoch 3/10, Train Loss: 2.3334,
Epoch 4/10, Train Loss: 2.3282,
Epoch 5/10, Train Loss: 2.2389,
Epoch 6/10, Train Loss: 2.2190,
Epoch 7/10, Train Loss: 2.2183,
Epoch 8/10, Train Loss: 2.1128,
Epoch 9/10, Train Loss: 2.0374,
Epoch 10/10, Train Loss: 1.9926
Test Accuracy: 32.31%
```

10 epoch



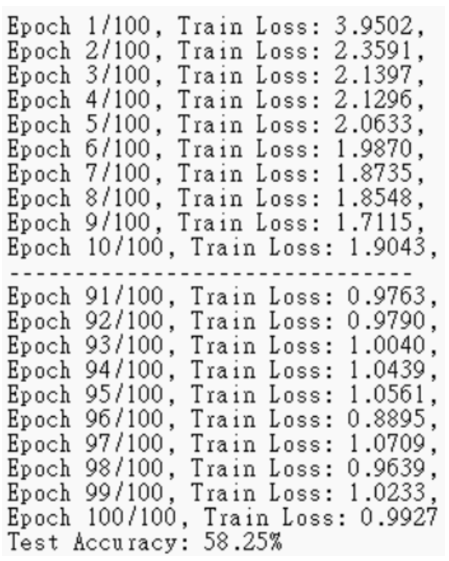
```
Epoch 1/10, Train Loss: 6.2369,
Epoch 2/10, Train Loss: 2.3580,
Epoch 3/10, Train Loss: 2.3334,
Epoch 4/10, Train Loss: 2.3282,
Epoch 5/10, Train Loss: 2.2389,
Epoch 6/10, Train Loss: 2.2190,
Epoch 7/10, Train Loss: 2.2183,
Epoch 8/10, Train Loss: 2.1128,
Epoch 9/10, Train Loss: 2.0374,
Epoch 10/10, Train Loss: 1.9926
Test Accuracy: 32.31%
```

20 epoch



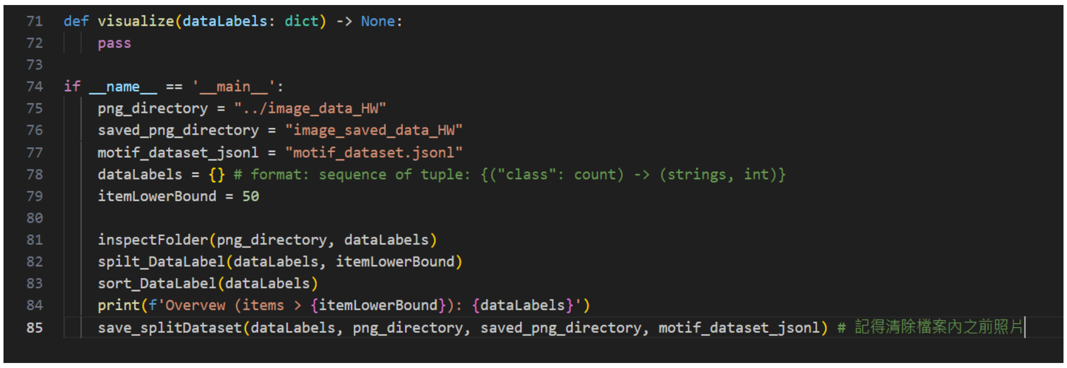
```
Epoch 1/20, Train Loss: 5.7775,
Epoch 2/20, Train Loss: 2.2069,
Epoch 3/20, Train Loss: 2.1935,
Epoch 4/20, Train Loss: 2.1508,
Epoch 5/20, Train Loss: 2.1090,
Epoch 6/20, Train Loss: 1.9746,
Epoch 7/20, Train Loss: 2.0696,
Epoch 8/20, Train Loss: 1.8553,
Epoch 9/20, Train Loss: 1.8917,
Epoch 10/20, Train Loss: 1.9465,
Epoch 11/20, Train Loss: 1.8056,
Epoch 12/20, Train Loss: 1.7675,
Epoch 13/20, Train Loss: 1.7368,
Epoch 14/20, Train Loss: 1.7575,
Epoch 15/20, Train Loss: 1.7765,
Epoch 16/20, Train Loss: 1.6289,
Epoch 17/20, Train Loss: 1.6425,
Epoch 18/20, Train Loss: 1.7242,
Epoch 19/20, Train Loss: 1.6090,
Epoch 20/20, Train Loss: 1.5825,
Test Accuracy: 47.69%
```

100 epoch



```
Epoch 1/100, Train Loss: 3.9502,
Epoch 2/100, Train Loss: 2.3591,
Epoch 3/100, Train Loss: 2.1397,
Epoch 4/100, Train Loss: 2.1296,
Epoch 5/100, Train Loss: 2.0633,
Epoch 6/100, Train Loss: 1.9870,
Epoch 7/100, Train Loss: 1.8735,
Epoch 8/100, Train Loss: 1.8548,
Epoch 9/100, Train Loss: 1.7115,
Epoch 10/100, Train Loss: 1.9043,
--------------------------------
Epoch 91/100, Train Loss: 0.9763,
Epoch 92/100, Train Loss: 0.9790,
Epoch 93/100, Train Loss: 1.0040,
Epoch 94/100, Train Loss: 1.0439,
Epoch 95/100, Train Loss: 1.0561,
Epoch 96/100, Train Loss: 0.8895,
Epoch 97/100, Train Loss: 1.0709,
Epoch 98/100, Train Loss: 0.9639,
Epoch 99/100, Train Loss: 1.0233,
Epoch 100/100, Train Loss: 0.9927
Test Accuracy: 58.25%
```

VI. Difficulty

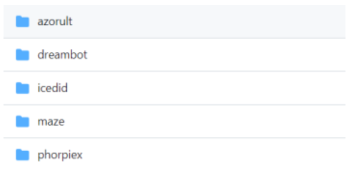


```python
def visualize(dataLabels: dict) -> None:
    pass

if __name__ == '__main__':
    png_directory = "../image_data_HW"
    saved_png_directory = "image_saved_data_HW"
    motif_dataset_jsonl = "motif_dataset.jsonl"
    dataLabels = {} # format: sequence of tuple: {("class": count) -> (strings, int)}
    itemLowerBound = 50

    inspectFolder(png_directory, dataLabels)
    spilt_DataLabel(dataLabels, itemLowerBound)
    sort_DataLabel(dataLabels)
    print(f'Overvew (items > {itemLowerBound}): {dataLabels}')
    save_splitDataset(dataLabels, png_directory, saved_png_directory, motif_dataset_jsonl) # 記得清除檔案內之前照片
```

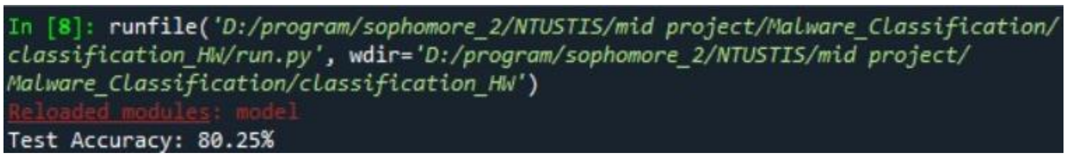
Data set is not big enough retrain 100 epoch 1 time

and change the label from 11 to 5

→Accuracy：**80.25%**



azorult
dreambot
icedid
maze
phorpiex



```
In [8]: runfile('D:/program/sophomore_2/NTUSTIS/mid project/Malware_Classification/
classification_HW/run.py', wdir='D:/program/sophomore_2/NTUSTIS/mid project/
Malware_Classification/classification_HW')
Reloaded modules: model
Test Accuracy: 80.25%
```

VII. Conclusion

A. Difficulty

1. Too much malware families with few data

   This indicates that the dataset contains a large number of categories (malware families) but only a small amount of data for each category. This scarcity of data can make it difficult to train a model effectively because the model may not have enough examples to learn the unique characteristics of each malware family.

2. Dataset is not big enough

   This point reinforces the first by stating the overall dataset is small. A small dataset can lead to various problems, such as overfitting, where the model learns the training data too well, including its noise and outliers, and performs poorly on unseen data.

3. category is many but without enough data.

   This is similar to the first point and emphasizes the problem of having many categories (malware families) in the dataset but not enough data for each. This situation can complicate the training process, as the model might struggle to distinguish between the different categories due to insufficient examples.

B. Observation

1. More families with few data less accuracy

   This observation suggests that the presence of more malware families, each with limited data, leads to lower accuracy in the model's predictions. It's likely because the model cannot accurately learn the features of each family with such sparse data.

2. More epochs may result to better accuracy

   An epoch in machine learning is one complete pass through the entire training dataset. This observation implies that increasing the number of epochs—allowing the model more opportunities to learn from the dataset—could improve its accuracy. However, it's important to note that after a certain point, more epochs can lead to overfitting.

3. More dataset more accuracy

   This straightforward observation suggests that having more data can lead to higher accuracy in the model's predictions. More data provides a richer set of examples for the model to learn from, potentially improving its ability to generalize to new, unseen data.

小組分工表

姚睿銘	code, Presentation, Presentation Speech
王家宏	code, Presentation
謝秉成	code
唐于硯	Report File
張安睿	Report File
林宗賢	Report File