source code descriptions

2016025141 고세진

2017029870 신호중

구현에 사용한 오픈소스

- Simhash, pytorch, torchvision, sklearn, matplotlib, pandas, numpy
- torchvision.resnet50을 제외한 모든 모델은 pytorch를 이용하여 직접 구현했습니다.

파일 구성

압축 파일은 다음과 같이 구성되어 있습니다.

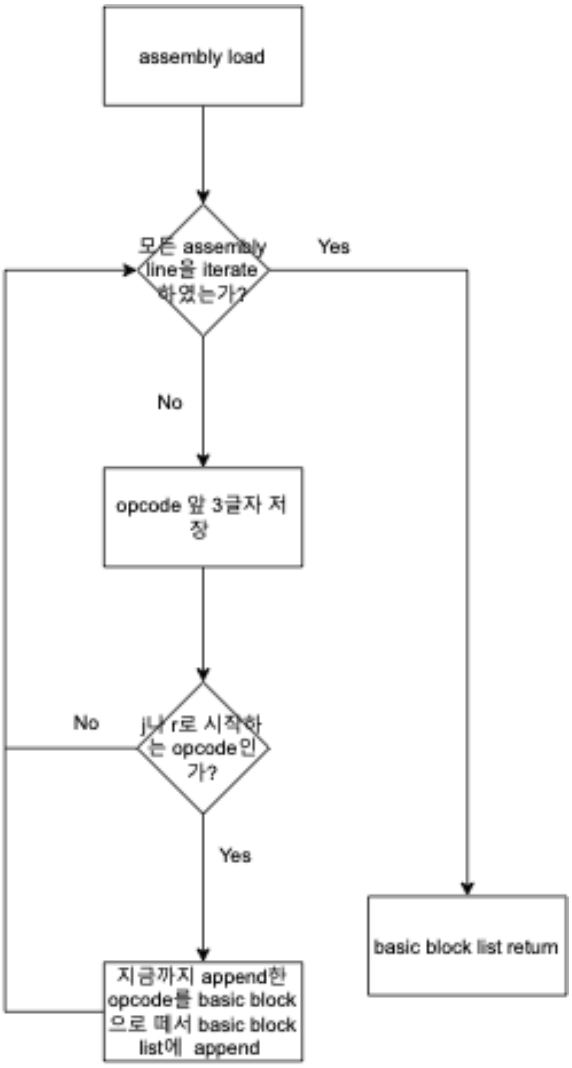
1. basic_block_parser.ipynb
2. image_generator.ipynb
3. cnn_diff_size_imgs_files (폴더)
    - CNN_128, CNN_256, CNN_512.ipynb
4. cnn_256_layer_added_files (폴더)
    - CNN_256_linear_added, CNN_256_conv_added, CNN_256_resnet
5. CNN_PBL_256_linear_layer_1_level_added_Wandb.ipynb

1. basic_block_parser.ipynb

- 어셈블리 파일을 basic block 파일로 바꿔 저장합니다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b4029bbe-30d9-49f0-8bc8-11c5b8d1f6c5/basic_block_parser_flowchart.pdf

```
basic_blocks = []
with open(path, 'r') as file:
    lines = file.readlines()
    block = []
    for line in lines:
        line = line[:-1] # 개행문자 삭제
        block.append(line[:3])
        if line.startswith('j') or line.startswith('r'):
            basic_blocks.append(block)
            block = []
```



2. image_generator.ipynb

- Simhash 라이브러리를 사용했습니다.

```
def make_image(basic_blocks):
  xyrgb_list = []

  for block in basic_blocks:
    xyrgb_hash = Simhash("".join(block), f=40).value
    x, y, r, g, b = hash_str_to_x_y_r_g_b_128(xyrgb_hash)
    xyrgb_list.append([x, y, r, g, b])
  return bg(xyrgb_list)

def bg(xyrgb_list, bg="black"):
  # for 128
  window = 2**7    # for 512 window = 2**9  # for 256 window = 2**8
  img = Image.new("RGB", (window, window), bg)
  draw = ImageDraw.Draw(img)
  for line in xyrgb_list:
    x = line[0]
    y = line[1]
    r = line[2]
    g = line[3]
    b = line[4]
    r_, g_, b_ = img.getpixel((x,y))
    draw.point((x,y), fill=(r + r_, g + g_, b + b_))
  return img

def hash_str_to_x_y_r_g_b(s):  # xyrgb_hash = Simhash("".join(block), f=40).value
  x = (s & 0xFF00000000) >> 32
  y = (s & 0x00FF000000) >> 24
```


```
  r = (s & 0x0000FF0000) >> 16
  g = (s & 0x000000FF00) >> 8
  b =  s & 0x00000000FF
  return x, y, r, g, b

def hash_str_to_x_y_r_g_b_128(s): # xyrgb_hash = Simhash("".join(block), f=40).value
  x = (s & 0b11111110000000000000000000000000000000000000000) >> 33
  y = (s & 0b00000001111111100000000000000000000000000000000) >> 26
  r = (s & 0b00000000000000011111111100000000000000000000000) >> 18
  g = (s & 0b00000000000000000000000011111111100000000000000) >> 10
  b = (s & 0b00000000000000000000000000000001111111100000000) >> 2
  return x, y, r, g, b
.. etc
```

3. cnn_diff_size_imgs_files폴더

CNN_128, CNN_256, CNN_512.ipynb 파일이 들어있습니다.

동일한 depth의 레이어를 지닌 모델로, dataset path와 hyper parameter만 상이하고 모든 코드가 같습니다.

basic block으로 만든 image 로드하고 트레인, test 하는 과정을 설명하겠습니다.

1. 먼저 ZipFile 라이브러리를 이용해 benign, malign image set의 압축파일을 해제하고 dataset을 담기 위해 pytorch의 Dataset을 상속한 CustomDataset 클래스를 정의합니다.
    - 데이터셋은 이미지 객체, malice 여부, 파일 이름으로 구성됩니다.

```
benign_path = '/content/drive/MyDrive/Malware Analysis/kisa_dataset/benign_image.zip'
malign_path = '/content/drive/MyDrive/Malware Analysis/kisa_dataset/malign_image.zip'
benign_zip = ZipFile(benign_path,'r')
malign_zip = ZipFile(malign_path,'r')

#benign
benign_info_list = benign_zip.infolist()
benign_name_list = benign_zip.namelist()
malign_info_list = malign_zip.infolist()
malign_name_list = malign_zip.namelist()

benign_set = []
malign_set = []


for i in range(len(benign_info_list)):
  _file = benign_zip.open(benign_info_list[i])
  img = Image.open(_file)
  img = convertPILImageToCV2(img)
  name = benign_name_list[i]
  benign_set.append((img, 0, name))

###
etc...
###

class CustomDataset(Dataset):
  def __init__(self, imgs, malices, names):
        super().__init__()
        self.imgs = imgs
        self.malices = malices
        self.names = names

  def __len__(self):
      return len(self.imgs)
```

```
  def __getitem__(self,index):
      return self.imgs[index], self.malices[index], self.names[index]
```

2. 로드한 이미지를 tensor로 변환하고 normalize한 뒤 데이터셋을 생성합니다.

```
train_transform = transforms.Compose([transforms.ToPILImage(),
                                      transforms.ToTensor(),
                                      transforms.Normalize([train_meanR,train_meanG,train_meanB],[train_stdR, train_stdG, train_stdB])
                                      ])
test_transform = transforms.Compose([transforms.ToPILImage(),
                                      transforms.ToTensor(),
                                     transforms.Normalize([test_meanR, test_meanG, test_meanB],[test_stdR, test_stdG, test_stdB])
                                      ])

train_imgs, train_malices, train_names = [],[], []
test_imgs, test_malices, test_names = [],[],[]

for img,mal,name in raw_train_set:
  train_imgs.append(train_transform(img))
  train_malices.append(mal)
  train_names.append(name)
for img,mal,name in raw_test_set:
  test_imgs.append(test_transform(img))
  test_malices.append(mal)
  test_names.append(name)

train_set = CustomDataset(train_imgs, train_malices, train_names)
test_set = CustomDataset(test_imgs, test_malices, test_names)
```

3. hyper parameter를 설정하고 모델을 정의합니다.

```
class CNN(nn.Module):
    def __init__(self):
      super(CNN, self).__init__()
      self.image_size = 256

      self.layer1 = nn.Sequential(
          nn.Conv2d(in_channels=3, out_channels=29, kernel_size=5),
          nn.BatchNorm2d(29),
          nn.ReLU(inplace=True),
          nn.MaxPool2d(2),
          nn.Dropout(0.4)
      )
  ... etc ...
      self.layer4 = nn.Sequential(
          nn.Linear(in_features=183, out_features=2, bias=True),
      )

    def forward(self, x):
      x = self.layer1(x)
      x = self.layer2(x)
      x = x.view(-1, 39 * self.image_size * self.image_size)
      x = self.layer3(x)
      x = self.layer4(x)

      return x


num_epochs = 35
batch_size = 128
learning_rate = 0.0005387
weight_decay = 0.01594

train_loader = DataLoader(dataset = train_set, batch_size = batch_size, shuffle=True, num_workers=0)
test_loader = DataLoader(dataset = test_set, batch_size = batch_size, shuffle=False, num_workers=0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr = learning_rate, weight_decay=weight_decay)
```

4. training을 진행하고 accuracy, f1 score를 측정합니다.

- f1_score는 sklearn 라이브러리를 이용해 측정했습니다.

```
model.eval()  # it-disables-dropout
predict = []
with torch.no_grad():
```

```
    correct = 0
    total = 0
    for images, labels,_ in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predict.extend(predicted.detach().cpu())
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Test Accuracy of the model: {} %'.format(100 * correct / total))


f1 = f1_score(test_set.malices, predict)
...
```

5. 테스트 셋에서 True Benign, True Malign, False Benign, False malign 에 대해 각각 정보손실률(1 - 이미지의 픽셀 수 /block) * 100 %를 측정합니다.

4. cnn_256_layer_added_files (폴더)

CNN_256_linear_added, CNN_256_conv_added. CNN_256_resnet.ipynb

- 모두 256x256 image를 input으로 사용하고, 모델의 아키텍처와 그에 따른 hyper parameter만 다릅니다.
- 각각 리니어 레이어 1개, conv 레이어와 리니어 레이어 1개가 추가되었고, 마지막 파일은 오픈소스 api로 resnet50 모델을 가져와서 사용했습니다.

5. CNN_PBL_256_linear_layer_1_level_added_Wandb.ipynb

- wandb 통해 시뮬레이션하는 스크립트 및 config가 담긴 소스파일입니다.
- wandb로 모델의 hyper parameter 조합을 바꿔가면서 실행하며 성능을 기록합니다.
- (cnn_128, cnn_256, cnn_512, cnn_256_linear_added, cnn_256_conv_added, cnn_256_resnet)데에터셋과 모델만 변경하여 등 모든 경우에 대해 최적의 hyper parameter 조합을 구하여 3번, 4번 폴더에 하드코딩 하는 방법으로 실험을 진행할 수 있었습니다.

```
sweep_config = {
  "name" : "seed_fixed_max_f1_acc_sweep_linear_layer_1_level_added",
  "method" : "random",
  "metric" : {
      "goal" : "minimize",
      "name" : "valid_loss"
  },
  "parameters" : {
    "dropout" : {
      "values" : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
    },
    "dropout2" : {
      "values" : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
   },
    "dropout3" : {
      "values" : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
    },
    "layer_1_out_channels" : {
        "distribution" : "int_uniform",
        "min" : 10,
        "max" : 30,
    },
    ...
    "learning_rate" : {
        "distribution" : "log_uniform_values",
        "min": 1e-4,
        "max": 1e-1,
    },
    "weight_decay" : {
        "distribution" : "log_uniform_values",
        "min": 1e-4,
        "max": 1e-1,
    }
  },
  "early_terminate": {
      "type": "hyperband",
      "eta" : 3,
```

```
hyperparameter_defaults = dict(
    dataset = "MALWARE",
    gpu = "colab",
    dropout = 0.3,
    layer_1_out_channels = 10,
    layer_2_out_channels = 20,
    layer_3_out_features = 300,
    layer_4_out_features = 150,
    batch_size = 256,
    learning_rate = 0.001,
    weight_decay = 1e-5,
    kernel_size = 5,
    )

wandb.init(config=hyperparameter_defaults, project="Malware_anal
config = wandb.config
```

```
      "min_iter" : 3,
  }
}

sweep_id = wandb.sweep(sweep_config, project="Malware_analysis")
```