



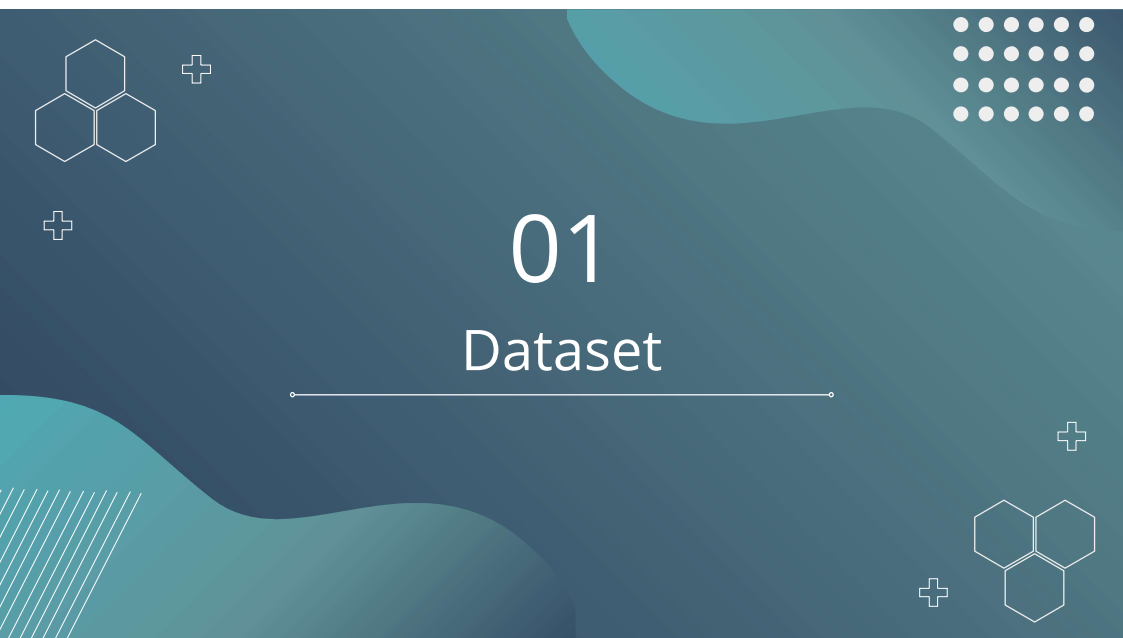
Final Project

B10830009 陳俊嘉
B10830030 吳傑生

TOPIC

Image-Based malware classification using ensemble of CNN architectures (IMCEC)

Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q. (2020).
Image-Based malware classification using ensemble of CNN
architectures (IMCEC). *Comput. Secur.*, 92, 101748.



01 Dataset

Dataset

- Malimg
- Contains 9339 malware images belonging to 25 different malware families.
- Preprocessing:
 - Resize the image to (224, 224)
 - separating images into training (80%) , validation (10%) and test (10%) sets

02 Implementation

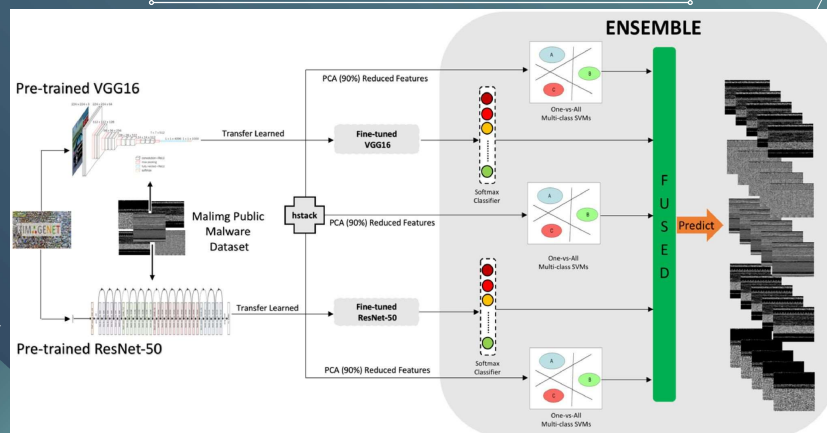
Implementation

IMECE

- CNN architectures previously trained (initialized) using natural images.
 - ImageNet
- Posterior probabilities from softmax classifiers were combined to obtain the families of unknown malware samples

Implementation

IMECE

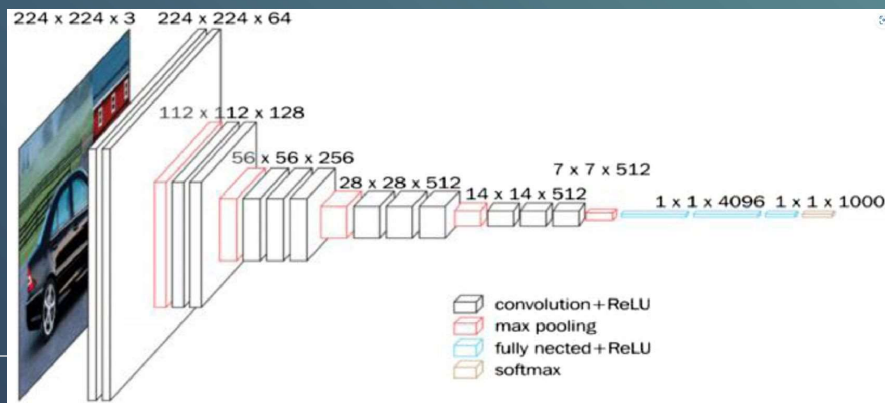


VGG-16

- well-known CNN
- standard neural network architecture
 - 5 convolutional layer blocks
 - 3×3 kernel with 1 padding and 1 stride
 - 3 fully-connected layers
 - Used at each block end for decreasing spatial dimensions
 - 2×2 kernel filter with no padding and 2 strides
 - two fully-connected layers containing 4096 ReLU activated units
 - 1000 fully-connected softmax layer

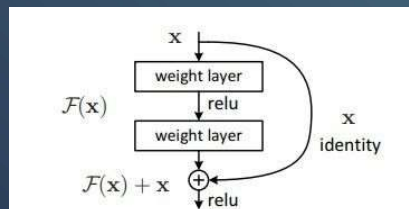
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

VGG-16



Residual networks (ResNet-50)

- Skip convolutional layer blocks with shortcut connections
 - Solve vanishing gradient problem
 - Additional parameter
- Skip connection & convolution layer
 - Element-wise addition
 - Reduce learning difficulty
- Basic blocks, called bottleneck blocks
 - Break 3x3 kernel into a set of (1x1,3x3,1x1) kernel



He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Residual networks (ResNet-50)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112x112	7x7, 64, stride 2				
		3x3 max pool, stride 2				
conv2_x	56x56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28x28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14x14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7x7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1x1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

CNN fine-tuning

- Previously trained on ImageNet
 - Dataset consisting of 1000 classes
 - Customized to our problem by using a fully-connected layer containing 25 classes
- Initial weights of pretrained CNN
 - Fine-tuned techniques was applied to optimize via back-propagation

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

CNN fine-tuning

```
VGG(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=25, bias=True)
  )
)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

```
(0): Linear(in_features=25088, out_features=4096, bias=True)
(1): ReLU(inplace=True)
(2): Dropout(p=0.5, inplace=False)
(3): Linear(in_features=4096, out_features=4096, bias=True)
(4): ReLU(inplace=True)
(5): Dropout(p=0.5, inplace=False)
(6): Linear(in_features=4096, out_features=25, bias=True)
)
```

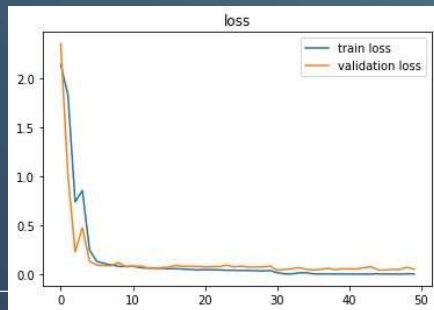
CNN fine-tuning

```
ResNet(
  (conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
)
```

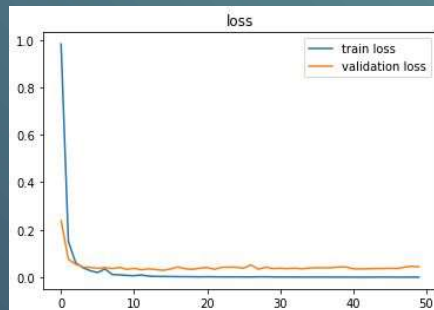
```
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=2048, out_features=25, bias=True)
)
```

CNN fine-tuning

VGG-16

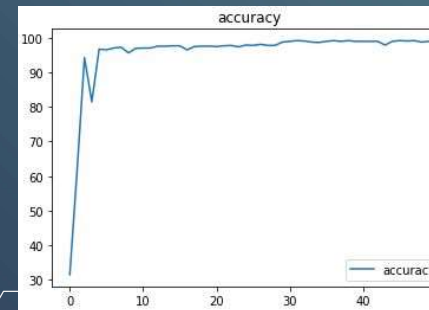


ResNet50

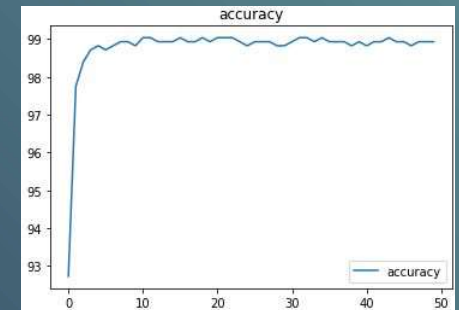


CNN fine-tuning

VGG-16



ResNet50



Classifiers

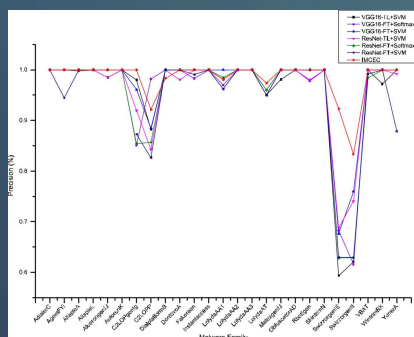
- Our proposed IMCEC design contains the following classifiers
 - Fine-tuned VGG16 employing a Softmax classifier.
 - Fine-tuned ResNet50 employing a Softmax classifier
- obtained posterior probabilities $P_i, k(f)$
 - the i th test image presented a certain family f based on the k th classifier
 - obtained image family f^* by integrating posterior probabilities as

$$f^* = \underset{f}{\operatorname{argmax}} \sum_i P_{i,k}(f)$$

evaluation metrics

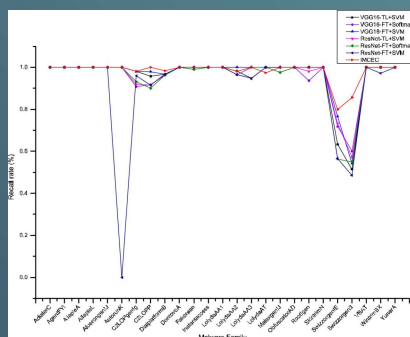
- accuracy, f1-score, recall, precision
- True Positive (TP): The true category is positive, predicted category is positive.
- True Negative (TN): The true category is negative, predicted category is negative.
- False Positive (FP): The true category is negative, predicted category is positive.
- False Negative (FN): The true category is positive, predicted category is negative.

Result



Precision

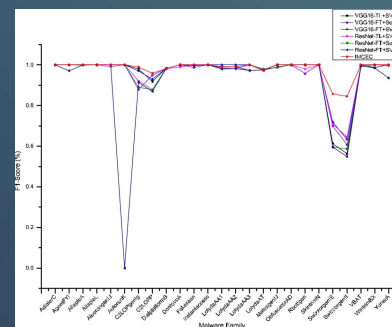
$$\text{Precision} = \frac{TP}{TP+FP}$$



Recall rate

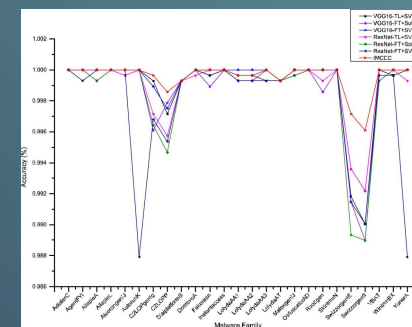
$$\text{Recall} = \frac{TP}{TP+FN}$$

Result



F1 score

$$F1 = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$



Accuracy

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$