ML Model for Hotspot Detection – Project Summary

Michael Qu

GLOBALFOUNDRIES®

Agenda

- Background
- ML Model Methodology
- ML Model for Site-based Data
- ML Model for Grid-based Data
- Prototype Model
- Extension from Prototype Model
- Conclusions
- Demo on Codes
- Q & A

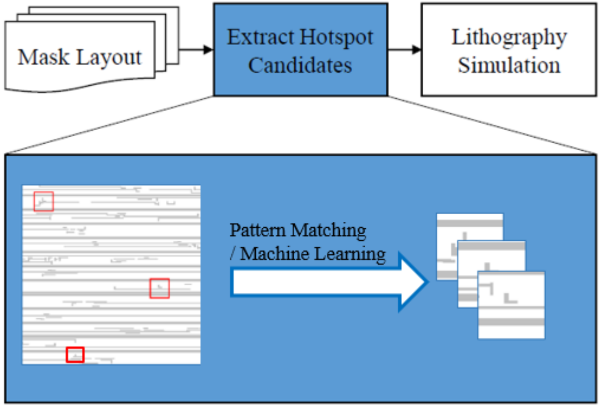
Background

Background

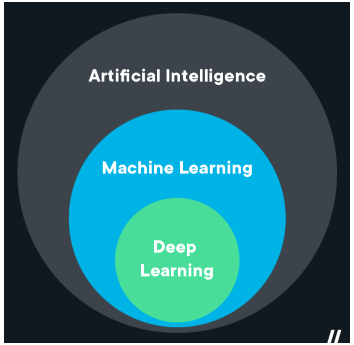
Hotspot: Problematic patterns or “design weak points”. Simply adopting design rule checking (DRC) and resolution enhancement technologies (RET) such as OPC cannot guarantee to avoid the occurrence of hotspot, so additional hotspot detection is required to increase the quality of the printed patterns and to improve OPC recipes.

Hotspot Detection Methods:

1. **Lithography Simulation**: Conventional, most accurate, but time-consuming to obtain the full chip characteristics.
2. **Pattern Matching**: To discover hotspots by comparing the pattern topology with the patterns registered in the hotspot library. Fast, but highly reliant on the hotspot library, which has weak generality for unknown hotspot patterns.
3. **Machine Learning:** To predict hotspots on new patterns using a machine learning (ML) model trained from a known database. Feature extraction can be automatically done by deep neural networks (DNN) and the model can generalize well for unknown hotspot patterns.



VLSI physical varication flow

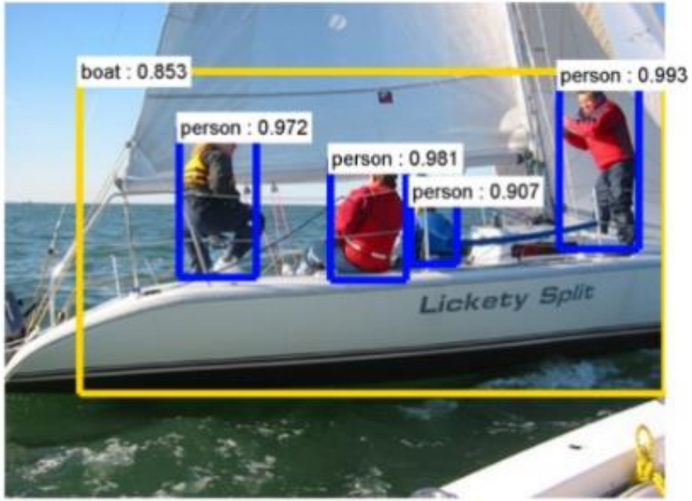


*Note: the “machine learning” in this presentation actually refers to “deep learning”, which is a subset of machine learning, based on deep neural networks.

ML Model Methodology

Terminology

Image Classification
(what?)



Object Detection
(what + where?)

(Binary/Multiclass)

Prototype Model

Extended Model

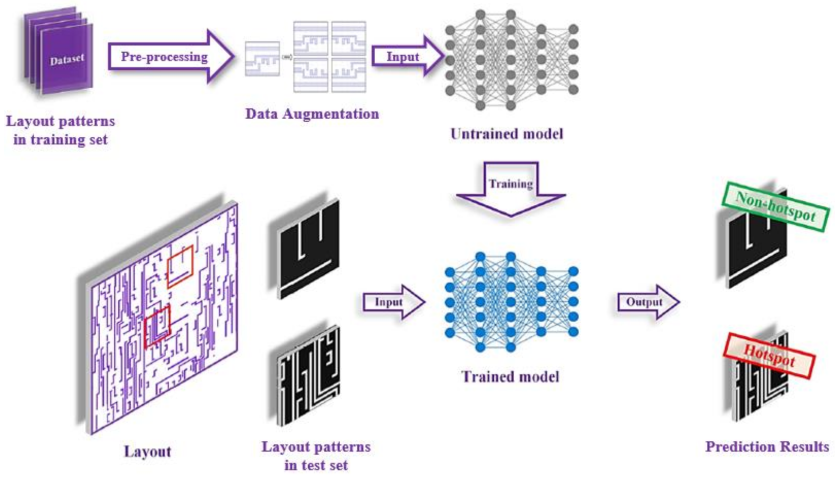
Machine Learning Model Workflow

- The simplest binary classification model (HS vs NHS) is taken as an example.

- Pre-processing: The original data is first split into training set and validation set. Data augmentation is required for training set to make the data invariant to zoom, shift and flipping, and also to reduce overfitting.

- Model training & evaluation: Supervised learning is carried out by using data in training set with known HS/NHS results, and calibrated by data in validation set.

- Model testing: Data in test set will be extracted from a new chip layout. The trained ML model needs to predict HS/NHS on data in test set without knowing the result in advance.



*The data can be in site-based or grid-based. In this figure grid-based data (image) is shown as an example.

Model Training

- Loss (or Cost) Function
  - Function that compares the ground truth and our model prediction to measure our model error, or "how bad our model is doing".
  - When training, we aim to minimize this loss.
  - For regression problem: Mean Square Error (MSE) or Mean Absolute Error (MAE)
  - For classification problem: Cross-Entropy

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)$$

  - We often plot loss function over time (epoch) to evaluate the model's learning progress and convergence
- Optimizer
  - Algorithm to find the best parameters to minimize loss function
  - The most frequently used optimizer is gradient descent method (and its variations, e.g. BGD, SGD, MGD, Adam…)

In [107]:
```
opt = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.99)
cnn.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Model Evaluation

Confusion Matrix

		Prediction	
		Positive (HS)	Negative (NHS)
Ground Truth	Positive (HS)	TP (hit)	FN (miss)
	Negative (NHS)	FP (false alarm)	TN

Evaluation Metrics:

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

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

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

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

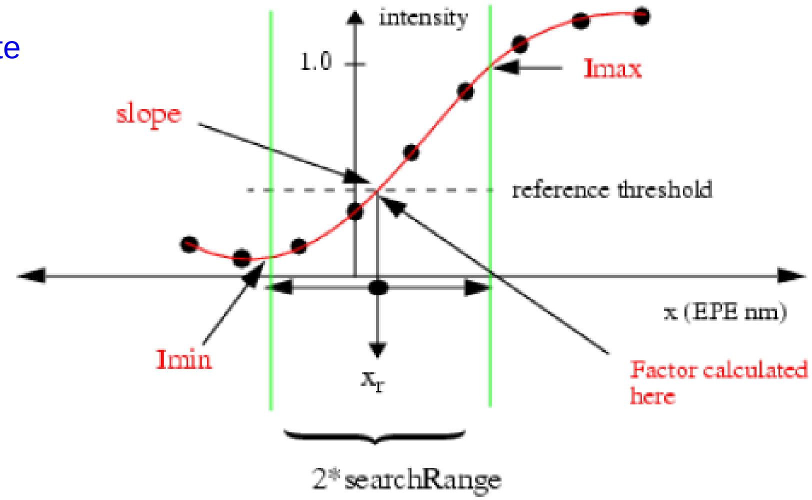
- Higher accuracy means more examples (HS + NHS) are predicted correctly.
- Higher precision means the defected points are more confidently identified as HSs.
- Higher recall means the detector missed fewer real HSs.
- The precision and recall have a trade-off relationship so F1 score is introduced to evaluate the overall impact.

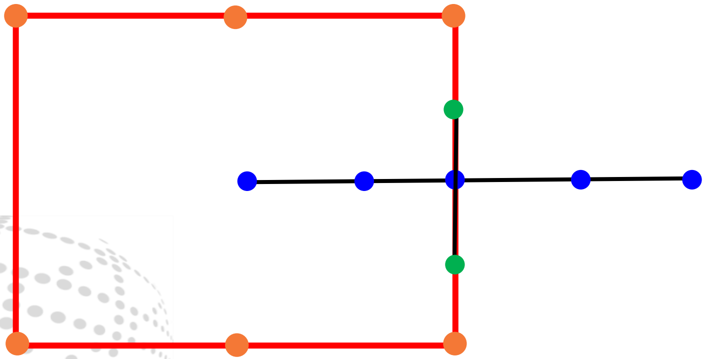
ML Model for Site-based Data

VT5 model

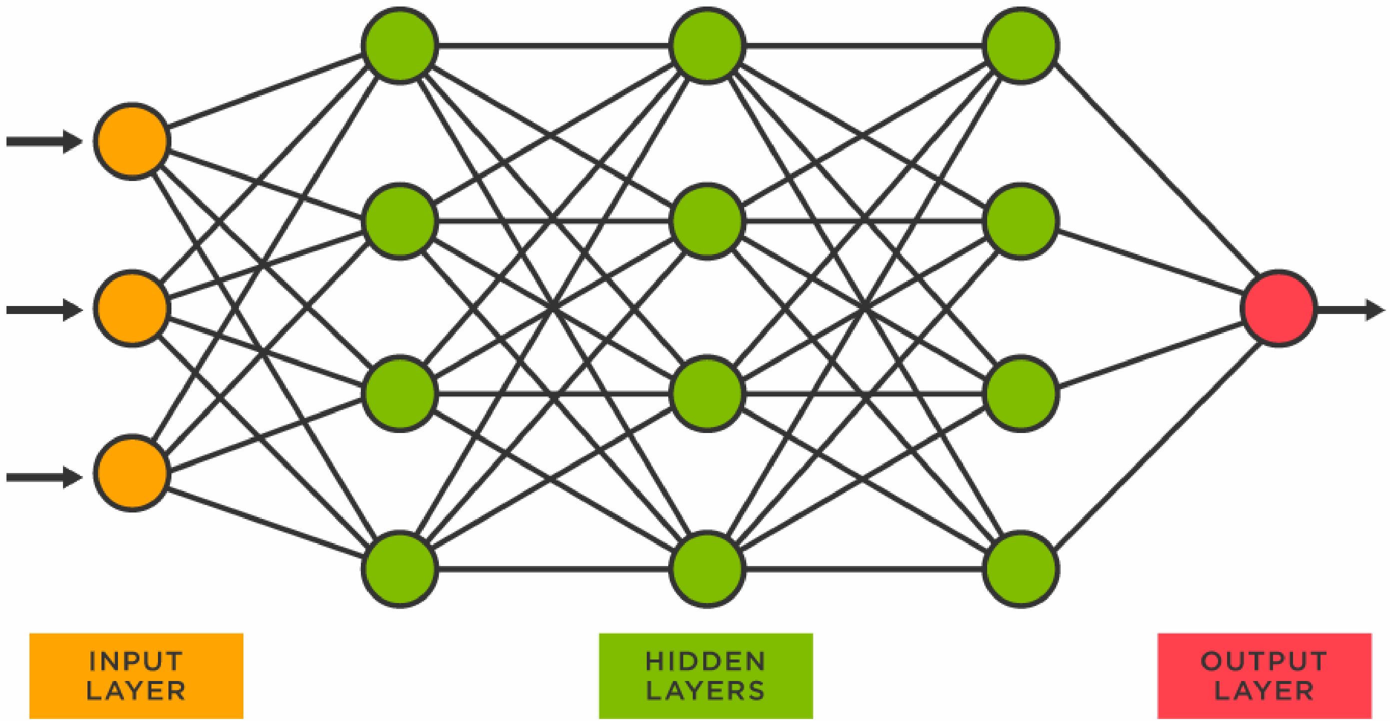
- Definition: VT5 is a Variable Threshold resist model that comprises 5 main parameters to fit the resist development behavior
  - Imax&Imin: maximum and minimum aerial image intensity on the fragment site within a window of size=2*searchRange (default searchRange=0.5*lambda/NA)
    - SearchRange is centered around reference threshold
  - Slope: aerial intensity slope @ reference Threshold
  - Curvature (factor): 2nd derivative of intensity
    - Calculated @ site perpendicular to fragment site
    - Represented the 2D behavior of that geometry
  - Convolutional kernels

Figure 11-1. Calculating Image Parameters for VT5 models





Proposed Model: Artificial Neural Network (ANN)



Data Format: Clip-by-Clip (1)

- Each input Xi is a matrix, including all the fragments on the i-th 20*20 μm clip -> ~200 fragments per clip
- Each output Yi has the same number of rows to label the defect information of that fragment
- Yi may not be accurate due to ~50nm tolerance of available bounding box

Input (Xi)							Output (Yi)	
Extracted from Post-OPC Layout Database							Correlated with UPDM DB	
(xA, yA)	(xB, yB)	Orientation	IP1	IP2	…	IP15	Bridging	Pinching
…	…	3	…	…	…	…	1	0
…	…	2	…	…	…	…	0	0
…	…	1	…	…	…	…	0	0
…	…	4	…	…	…	…	0	1
…	…	…	…	…	…	…	…	…
…	…	2	…	…	…	…	0	0

Data Format: Clip-by-Clip (2)

- Each input Xi is a matrix, including all the fragments on the i-th 20*20 μm clip -> ~200 fragments per clip
- Each output Yi includes the defect type and coordinate of all the defects in this clip
- The tolerance of available bounding box (~50nm) can be neglected in the clip dimension

Input (Xi)						
Extracted from Post-OPC Layout Database						
(xA, yA)	(xB, yB)	Orientation	IP1	IP2	…	IP15
…	…	3	…	…	…	…
…	…	2	…	…	…	…
…	…	1	…	…	…	…
…	…	4	…	…	…	…
…	…	…	…	…	…	…
…	…	2	…	…	…	…

Output (Yi)		
Correlated with UPDM DB		
Defect Type	x	y
3	23	34
2	69	16
…	…	…

Data Format: Fragment-by-Fragment (1)

- Each input Xi is an array, including the information of the i-th fragment
- Each output Yi labels the defect information of Xi fragment
- Yi may not be accurate due to ~50nm tolerance of available bounding box

Input (Xi)						
Extracted from Post-OPC Layout Database						
(xA, yA)	(xB, yB)	Orientation	IP1	IP2	…	IP15
…	…	3	…	…	…	…

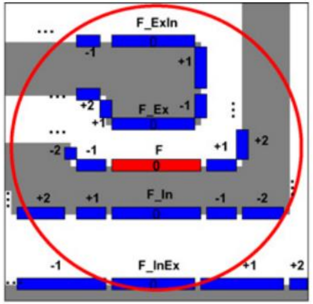
Output (Yi)	
Correlated with UPDM DB	
Bridging	Pinching
1	0

Data Format: Fragment-by-Fragment (2)

- Each input Xi is an array, including the information of the i-th fragment
- Each output Yi labels the defect information of Xi fragment
- Yi may not be accurate due to ~50nm tolerance of available bounding box
- If more special information is required, this method can be used

Input (Xi)									
Extracted from Post-OPC Layout Database					Calculated additionally (if necessary)				
(xA, yA)	(xB, yB)	Orientation	IP1	…	Hotspot signature				Signature of neighbors
…	…	3	…	…	F_corn	F_ext	F_int	F_misc	[F1, F2, F3, …, Fn]

Output (Yi)	
Correlated with UPDM DB	
Bridging	Pinching
1	0



Fi = [F_corn_i, F_ext_i, F_int_i, F_misc_i], i = 1, 2, 3, …, n (all the fragments within the effective region)

Hotspot Signature Measurements

- Objective: to scan the layout and extracting useful information/data for each fragment in the layout.
- Implementation: establishing a table-structure database that can be indexed in constant time

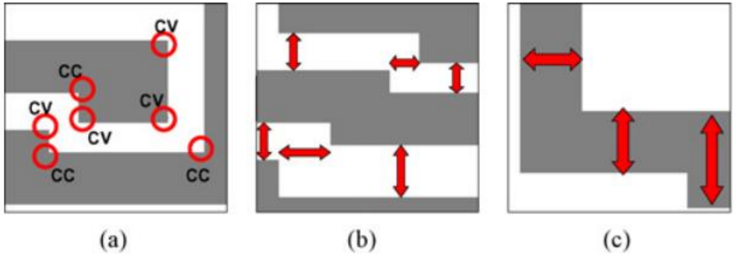


Fig. 4. Three major types of hotspot feature measurements. (a) Corner information. (b) External length. (c) Internal length.

Operators	Operation Description (Features to Measure)
$f_{corn}(\cdot)$	extracts information of convex and concave corners touching Frag
$f_{ext}(\cdot)$	returns the distance(s) between Frag and the fragments facing Frag on the external side
$f_{int}(\cdot)$	returns the distance(s) between Frag and the fragment facing Frag on the internal side
$f_{misc}(\cdot)$	requests extra information regarding Frag, such as fragment orientation (x or y-axis) and the length of Frag

Fragmentation-Based Context Characterization

- Objective: to generate a 1-D vector to include the above hotspot signature measurements of all the fragments within the effective radius, which is invariant to rotation and mirroring
- In the example below, r is set to be 2, so two neighbors on each side of F_In, F_Ex, F_ExIn, F_InEx are investigated (this can be done using BFS-based algorithm)

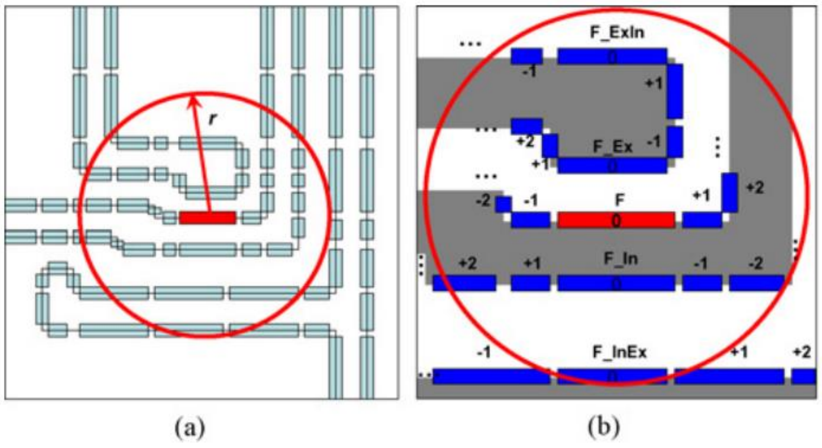


Fig. 5. Fragmentation-based hotspot signature extraction. (a) Effective radius centered at each fragment. (b) Fragmentation-based context characterization.

Notation	Explanation
F (F0)	Current fragment of interest (detection anchor point)
r	radius of interest (effective radius)
F_In	Fragment(s) facing F internally
F_Ex	Fragment(s) facing F externally
F_ExIn	Fragment(s) facing F_Ex internally
F_InEx	Fragment(s) facing F_In externally
F_+i	ith neighbor traced from F clockwise
F_-i	-ith neighbor traced from F counter-clockwise

Fragmentation-Based Context Characterization

- Objective: to generate a 1-D vector to include the above hotspot signature measurements of all the fragments within the effective radius, which is invariant to rotation and mirroring
- In the example below:
- Step 1: look for F_In, F_Ex, F_ExIn, F_InEx (if they exist)
- Step 2: r is set to be 2, look for two neighbors on each side of F_In, F_Ex, F_ExIn, F_InEx (this can be done using BFS-based algorithm)
- Step 3: Extract hotspot signature measurement for all these fragments and calculate vector V_F
- Step 4: Add V_F to our model input for each training example

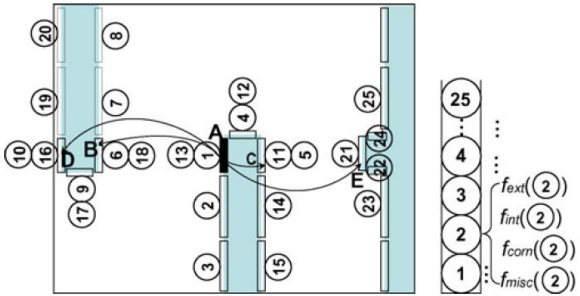


Fig. 6. Illustrative example of the vector generation process for fragment A (black color), where r is set to cover a depth of two neighboring fragments.

Next, we present the characterized context of fragment F in the format of a 1-D data vector defined as

$$V_F = \coprod_{i}^{\widetilde{F}_i \in \delta_r^F} \{ f_{ext}(\widetilde{F}_i) \oplus f_{int}(\widetilde{F}_i) \oplus f_{corn}(\widetilde{F}_i) \oplus f_{misc}(\widetilde{F}_i) \} \quad (6)$$

$$\widetilde{F} = [F, F\_Ex, F\_In, F\_ExIn, F\_InEx...] \quad (7)$$

where F is an integer identification (ID) number representing a certain fragment in the layout, and δ_r^F is the effective region

False Alarm Issue

- Problem: difficult for ML model to understand root cause feature effectively in the limited HS examples (small changes in geometry may avoid HS) -> large amount of false alarms
- Solution: Synthetic patterns

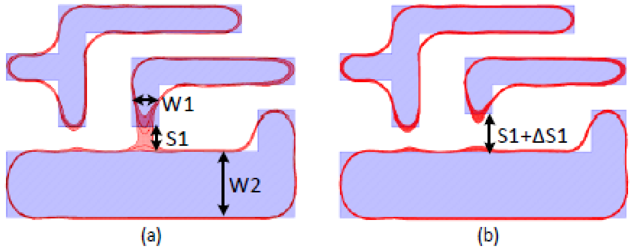
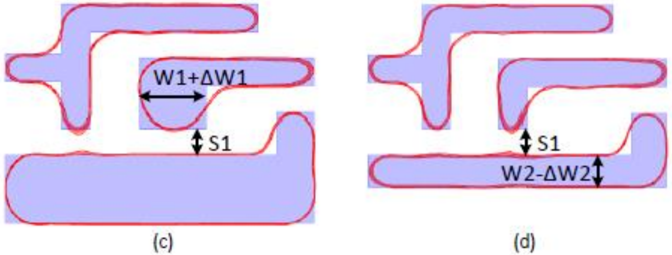


Fig. 2. (a) A hotspot pattern, (b-d) variants of pattern (a) which are non-hot



spots

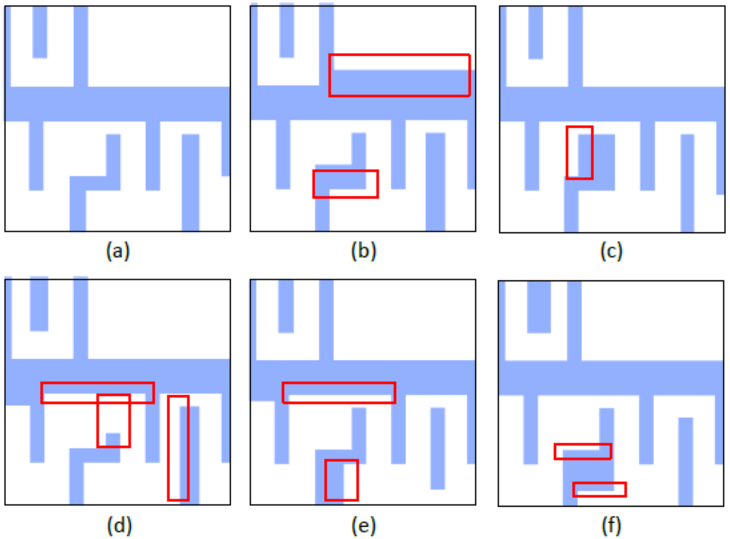


Fig. 4. (a) A hotspot pattern, (b-f) Synthetic patterns generated from pattern (a). Red markers indicate the subtle differences from pattern (a)

Synthetic Pattern Generation

- Objective: to let ML model effectively learn root cause features
- Implemented in training phase only

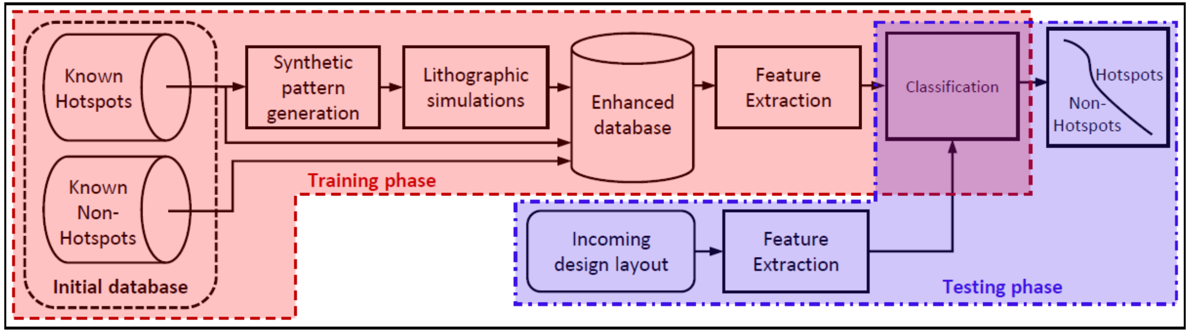


Fig. 3. The proposed machine learning-based Hotspot detection flow

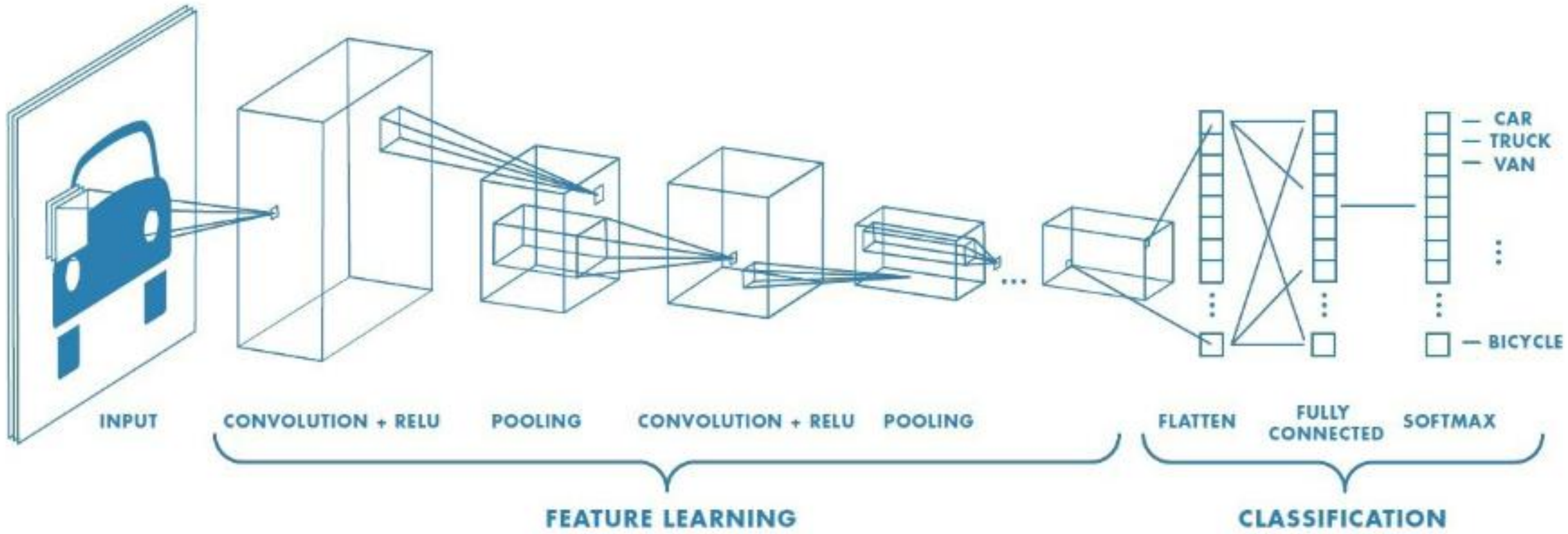
```
def GenerateSyntheticPatterns(KnownHotspot):
    Input: A Known Hotspot, Synthetic pattern count, distance
           PDF, Edge PDF
    Result: Synthetic variants of the Hotspot
1   for i in range(SynPatCount):
2       HotspotPolys = All polygons in the original hotspot pattern
3       for polygon in HotspotPolys:
            /* Sample the no. of edges to be
               varied                                   */
4           EdgeCount = Sample from Edge PDF
5           for j in range(EdgeCount):
6               while EdgeAttempts ≤ MaxEdges:
                    /* Randomly select an edge          */
7                   edge = GetRandomEdge(polygon)
8                   while DistAttempts < FixedCount:
9                       dist = Sample from distance PDF
10                      polygon = polygon.MoveEdge(edge,
                          dist)
                        /* Perform checks to avoid
                           simple DRC errors            */
11                      MinimalDRC(ModifiedPattern)
12                      if MinimalDRC == Pass:
13                          go to line 5
14                      else:
15                          polygon = UnmodifiedPolygon
16                          DistAttempts+ = 1
17                          try a different dist value (go to line
                              8)
18                  EdgeAttempts+ = 1
19                  try a different edge (go to line 6)
        /* All polygons with/without updates,
           together form the modified pattern
           */
20      SyntheticPattern = All Polygons (including modifications)
    /* Return patterns with variations              */
21  return SyntheticPatterns
```

Algorithm 1: Synthetic pattern generation

Fast DRC check is required
Need to do necessary ones on a suitable size of domain to achieve desirable performance

ML Model for Grid-based Data

Proposed Model: Convolutional Neural Network (CNN)



Data Format

- For single channel: each input Xi is a 2D matrix, representing the spatial distribution of a particular scalar property (geometry info, image parameter, …) over a regular mesh on a rectangular domain
- For multiple channels: each input Xi is a 3D tensor, representing the spatial distribution of an array of properties (geometry info, image parameter, …), over a regular, consistent mesh on a rectangular domain
- Each output Yi is a scalar (for binary classification, 1 for HS, 0 for NHS; for multiclass classification, more defect types can be defined)

Input (Xi)		
Geometry Info	Image Parameter	…

Geometry Info

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image Parameter

0	0	0	0	0	0	0	0	0	0
0	3	2	9	4	3	1	2	5	0
0	2	5	7	1	2	6	4	3	0
0	4	3	3	9	1	7	7	2	0
0	7	2	6	0	0	0	0	0	0
0	9	1	7	0	0	0	0	0	0
0	8	2	4	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Output (Yi)
0 or 1 (for binary classification)

In the initial stage, IP may be sufficient because:

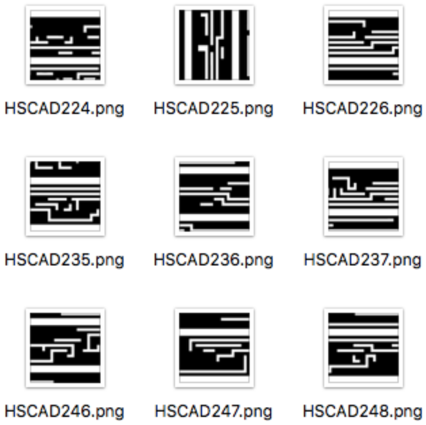
- Logically: IP distribution can include geometric features. We can try using IP only and check the performance.
- Implementation: Grid size for OPC/IP calculation is 10-20nm, but that for geometry output is 0.1nm. Proper mapping may be required to include geometry info. No need to make model too complicated at the initial stage.

Prototype Model

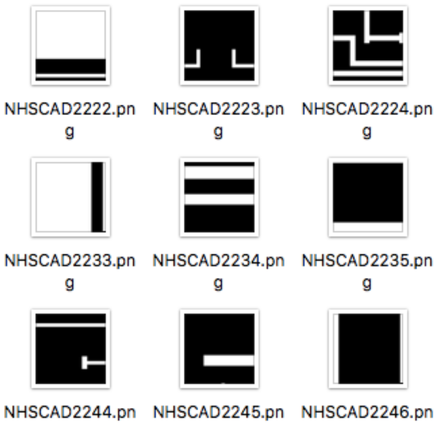
ICCAD 2012 Dataset

- Used by many researchers, even in papers published in 2022
- Five sub datasets with different types of layouts. Each sub-dataset contains training and test set
- Benchmark 1 is obtained from 32 nm process and Benchmark 2-5 are obtained from 28 nm process
- Each image has 1200 x 1200 pixels, representing 1.2 x 1.2 μm^2
- It does not give any layer information or hot spot coordinates
- But it can used as a ref at the initial stage (as a part of the deliverable as well)

Sub-dataset		Training Set			Test Set		
		HS	NHS	Total	HS	NHS	Total
Benchmark 1	32nm	99	340	439	226	3869	4095
Benchmark 2	28nm	174	5285	5459	498	41298	41796
Benchmark 3	28nm	909	4643	5552	1808	46333	48141
Benchmark 4	28nm	95	4452	4547	177	31890	32067
Benchmark 5	28nm	26	2176	2202	41	19327	19368
Total		1303	16896	18199	2750	142717	145467

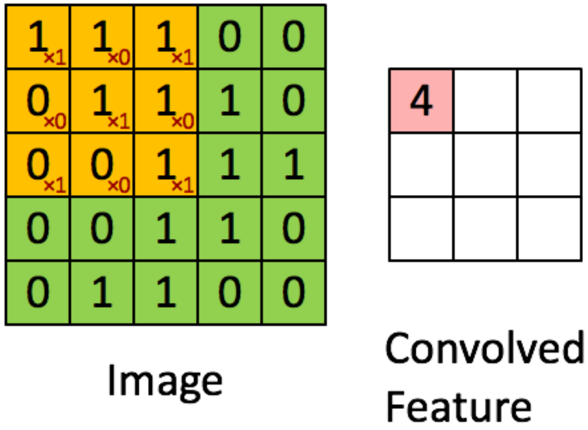


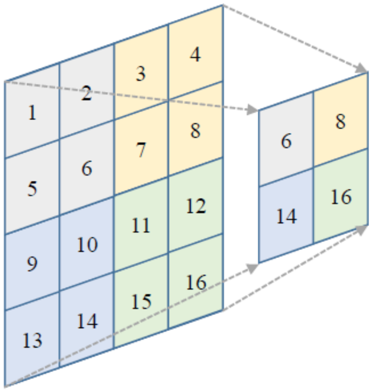
Images with Hotspot

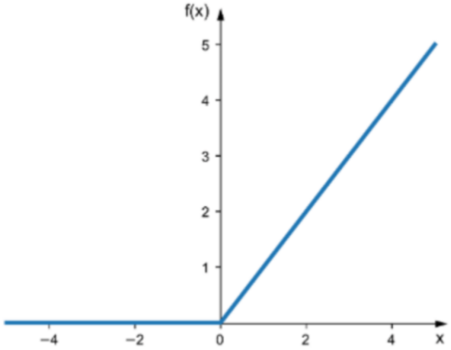


Images without Hotspot

CNN Concepts







Convolution operation:

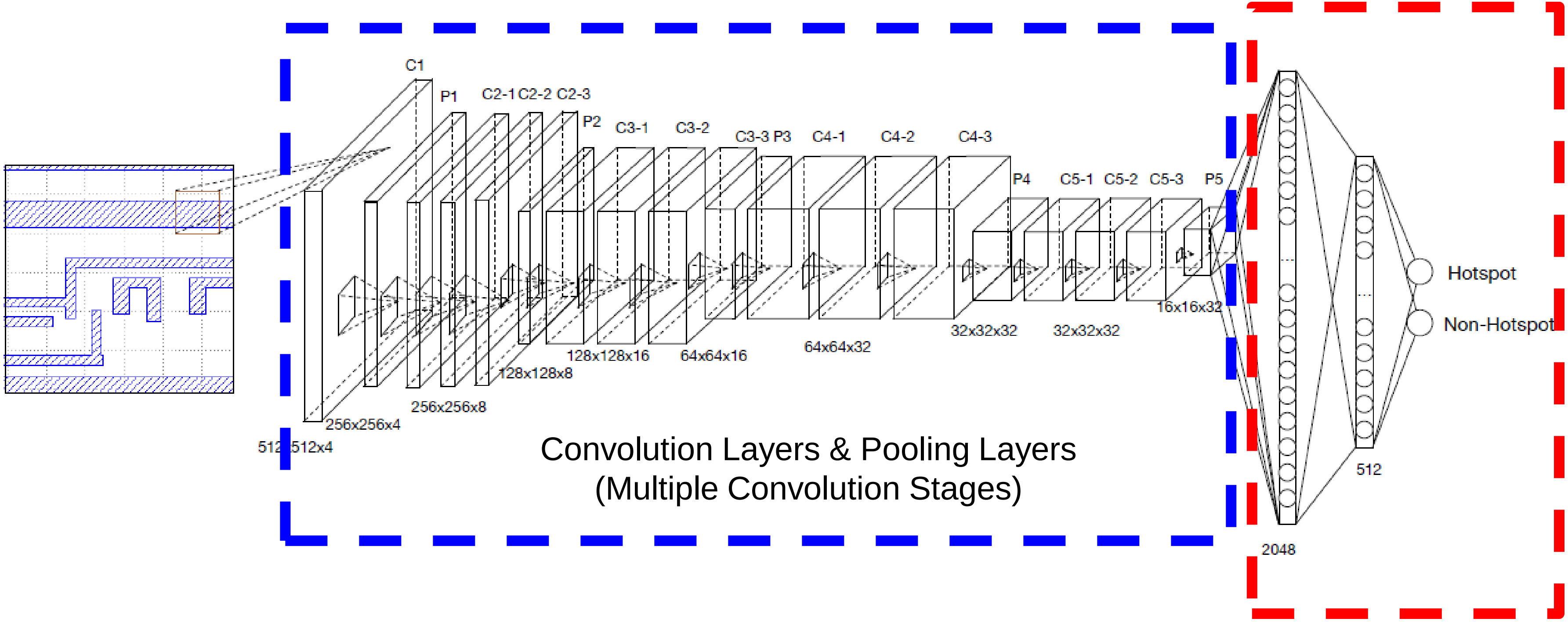
$$\boldsymbol{I}^{(i)} \otimes \boldsymbol{K}_{m\times m}(x,y) = \sum_{i=1}^{c}\sum_{j=1}^{m}\sum_{k=1}^{m} \boldsymbol{I}^{(i)}(x-j, y-k)\boldsymbol{K}(j,k)$$

Max pooling

Activation function:

$$ReLU(x) = \max\{x, 0\}$$

CNN Model Architecture



Convolution Layers & Pooling Layers
(Multiple Convolution Stages)

Fully Connected Layers
(Hidden and Output Layers in ANN)

CNN Model Architecture

		Yang et al (2017)	Liao et al (2022)	Our Model
Convolution Layer	# of Layers	1 + 3 x 4 = 13	2 x 5 = 10	4
	# of Filters	4, 8, 16, 32	64, 128, 256, 512	4, 8, 16, 32
	Kernel Size	2x2 & 3x3	3x3	3x3
Pooling Layer	# of Layers	5	5	4
	Pool Size	2x2	2x2	2x2
Fully Connected (FC) Layer	# of Layers	3	3	3
	# of Neurons	2048, 512, 2	4096, 4096, 2	512, 256, 1

Pre-processing

- Tool: Keras (deep learning API running on Tensorflow)
- Data augmentation: random transformation applied on training set (e.g. zoom, shift, horizontal/vertical flip)
- No random transformation applied on test set
- Overfitting can be effectively reduced by data augmentation

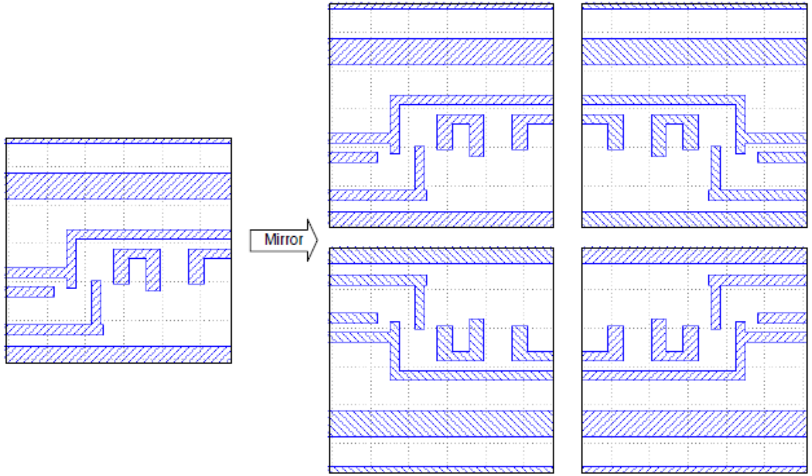


Figure 7. Random Mirror Flipping with X, Y, and XY.

Preprocessing the Training set

```
In [3]: # https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
SIZE = 64
# Image Parameters in the future can be added to the channels
CHANNELS = 3
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip = True,
                                   vertical_flip = True)
training_set = train_datagen.flow_from_directory(directory = 'iccad1/train',
                                                 target_size = (SIZE, SIZE),
                                                 batch_size = 32,
                                                 classes = {'NHS':0, 'HS':1},
                                                 class_mode = 'binary',
                                                 shuffle = True,
                                                 seed = 42
                                                )

Found 439 images belonging to 2 classes.
```

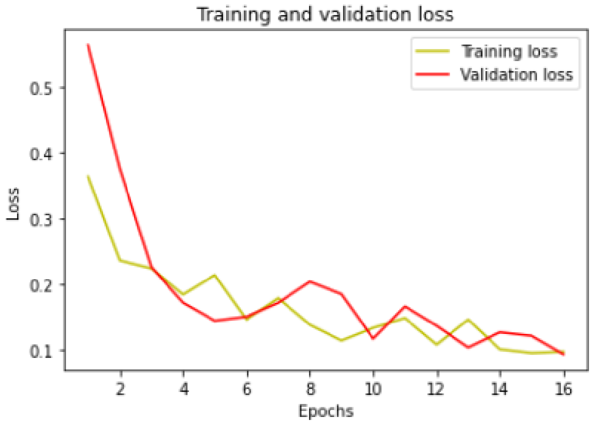
Preprocessing the Test set

```
In [4]: test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory(directory = 'iccad1/test',
                                            target_size = (SIZE, SIZE),
                                            batch_size = 32,
                                            classes = {'NHS':0, 'HS':1},
                                            class_mode = 'binary',
                                            shuffle = True,
                                            seed = 42
                                           )

Found 4905 images belonging to 2 classes.
```

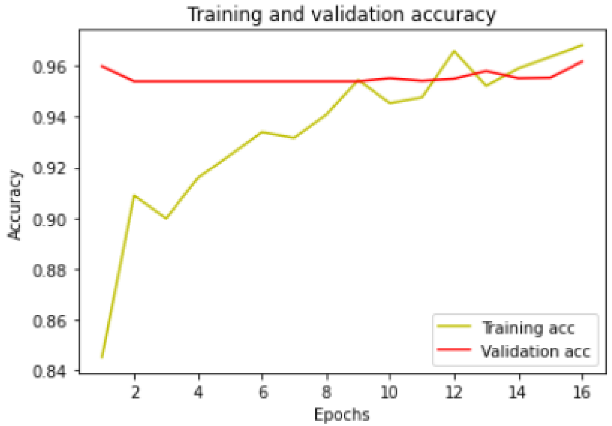
Training

Loss function and accuracy plotted at each epoch (# of complete passes through the training dataset)

In [34]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [35]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Prediction

```
if result[0][0] > 0.5:
  prediction = 'HS'
else:
  prediction = 'NHS'

print(prediction)
```

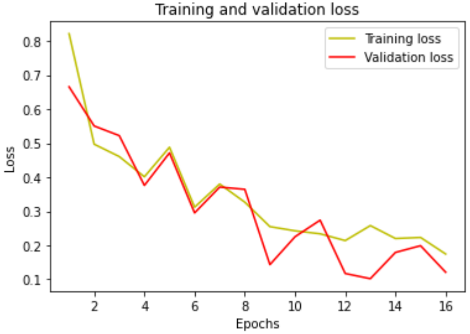
```
In [63]: showPNG('test_HS12')
```

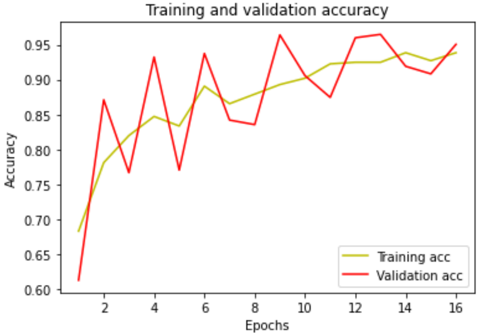
```
In [64]: single_predict('test_HS12')

1/1 [==============================] - 0s 12ms/step
[[0.6891398]]
{'NHS': 0, 'HS': 1}
HS
```

```
In [67]: showPNG('test_NHS108.png9')
```

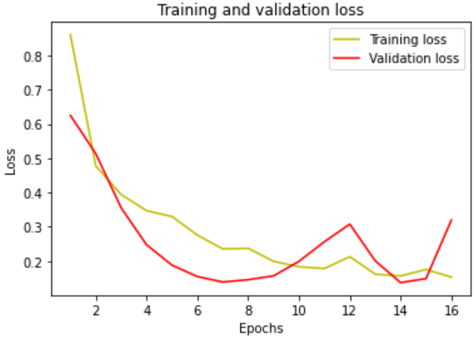
```
In [68]: single_predict('test_NHS108.png9')

1/1 [==============================] - 0s 14ms/step
[[0.00104977]]
{'NHS': 0, 'HS': 1}
NHS
```

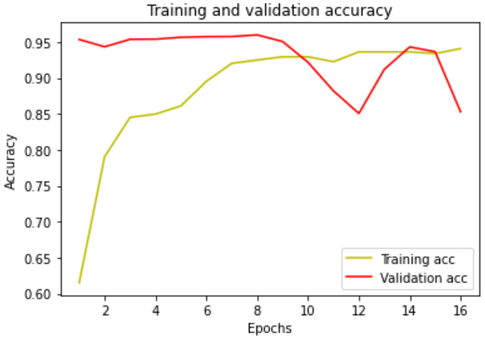
Discussion: No. of Convolution Stages



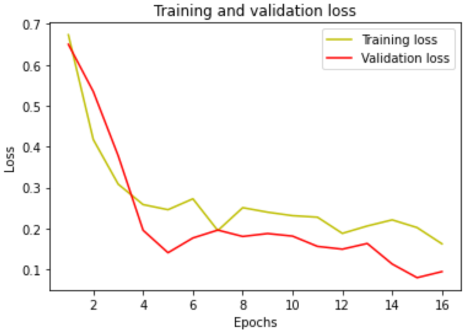


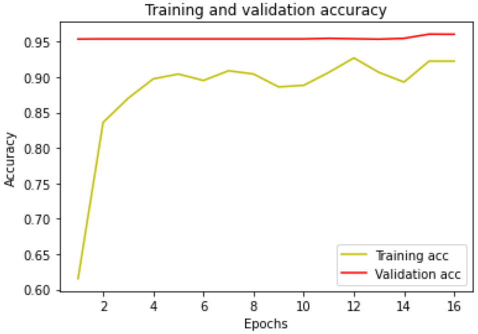
2 Conv-Pool





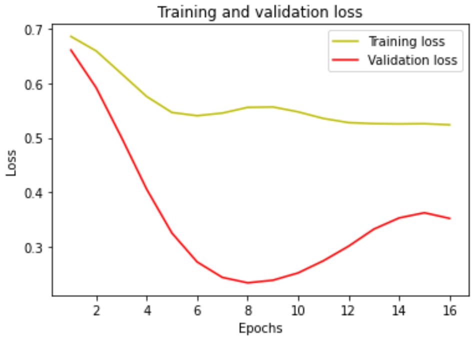
3 Conv-Pool

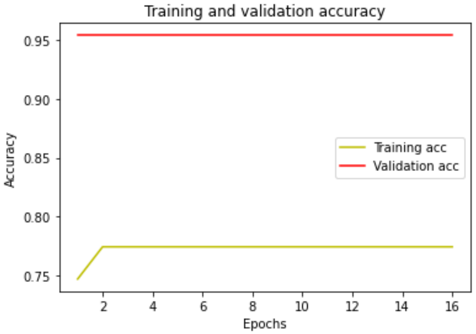




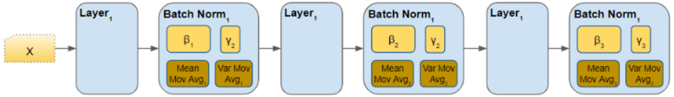
4 Conv-Pool

Discussion: Batch Normalization

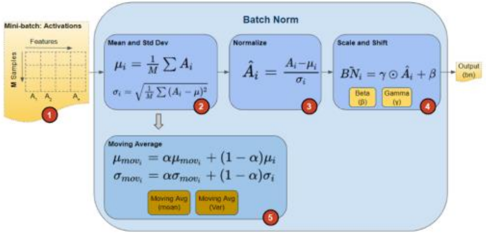




No BN

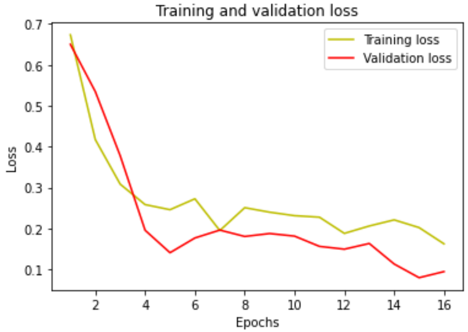


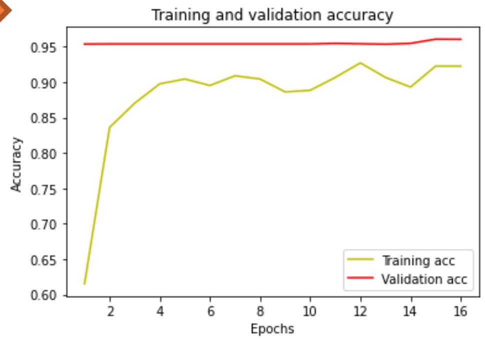
Each Batch Norm layer has its own copy of the parameters (Image by Author)



Calculations performed by Batch Norm layer (Image by Author)

Normalize to zero mean and unit variance





After Applying BN

Discussion: Optimizer

```
In [26]: showPNG('test_NHS108.png9')
```

```
In [27]: single_predict('test_NHS108.png9')

1/1 [==============================] - 0s 16ms/step
[[0.00020799]]
{'NHS': 0, 'HS': 1}
NHS
```

```
In [22]: showPNG('test_HS12')
```

```
In [23]: single_predict('test_HS12')

1/1 [==============================] - 0s 15ms/step
[[0.18223694]]
{'NHS': 0, 'HS': 1}
NHS
```

Adam (adaptive optimization algorithm): sometimes fail to generalize

```
In [67]: showPNG('test_NHS108.png9')
```

```
In [68]: single_predict('test_NHS108.png9')

1/1 [==============================] - 0s 14ms/step
[[0.00104977]]
{'NHS': 0, 'HS': 1}
NHS
```

```
In [63]: showPNG('test_HS12')
```

```
In [64]: single_predict('test_HS12')

1/1 [==============================] - 0s 12ms/step
[[0.6891398]]
{'NHS': 0, 'HS': 1}
HS
```

$$w_i = w_i - \gamma \frac{\partial l}{\partial w_i}.$$

$$v = \mu v - \gamma \frac{\partial l}{\partial w_i},$$

$$w_i = w_i + v,$$

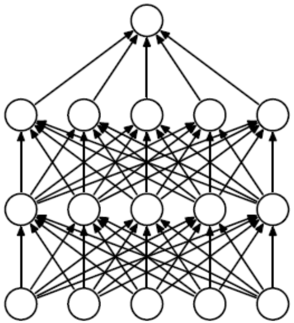
Table 5. Momentum Configuration.

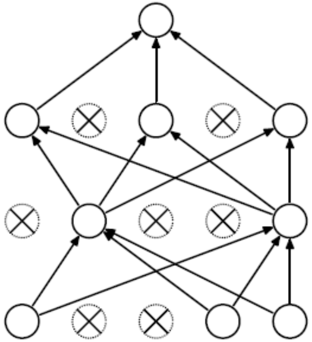
μ	Learning Rate	Validation Loss
0.5	0.001	0.21
0.9	0.001	0.22
0.95	0.001	0.21
0.99	0.001	**0.16**

Use learning rate=0.001, momentum=0.99

SGD (stochastic gradient descent): better generalization performance

Discussion: Dropout


(a) Standard Neural Net


(b) After applying dropout.

```
In [24]: showPNG('test_NHS133.png1')
```

```
In [25]: single_predict('test_NHS133.png1')

1/1 [==============================] - 0s 13ms/step
[[0.5707997]]
{'NHS': 0, 'HS': 1}
HS
```

```
In [65]: showPNG('test_NHS133.png1')
```

```
In [66]: single_predict('test_NHS133.png1')

1/1 [==============================] - 0s 11ms/step
[[0.1025407]]
{'NHS': 0, 'HS': 1}
NHS
```

Without dropout: sometimes fail to generalize

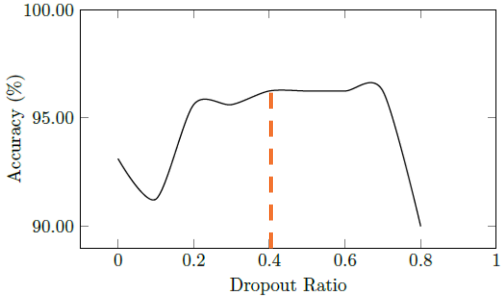


Figure 5. Dropout Ratio Effect.

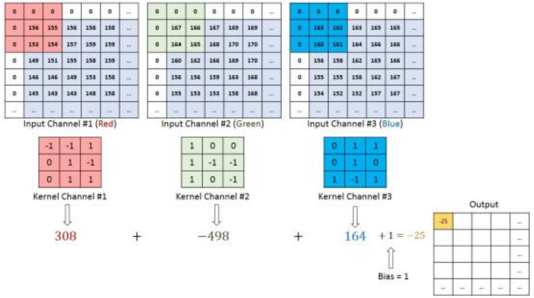
After applying dropout ratio = 0.4 on FC layers: better generalization performance

Extension from Prototype

Multiple Channel Input

- Image parameters can be viewed as multiple channels (layers) defined on the same grids.
- CNN and its training, calibration are the same. Only need to load the data in 3D tensor format (channels = # of image parameters), instead of 2D image (channels = 3 for RGB, 1 for grayscale)

```
In [46]: cnn.add(tf.keras.layers.Conv2D(filters=4, kernel_size=3, activation='relu', input_shape=[SIZE, SIZE, CHANNELS]))
         cnn.add(tf.keras.layers.BatchNormalization())
         cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```



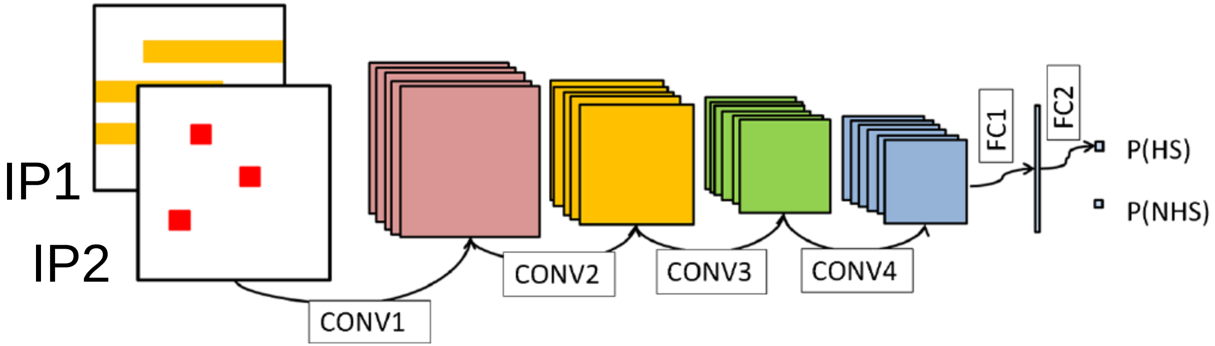
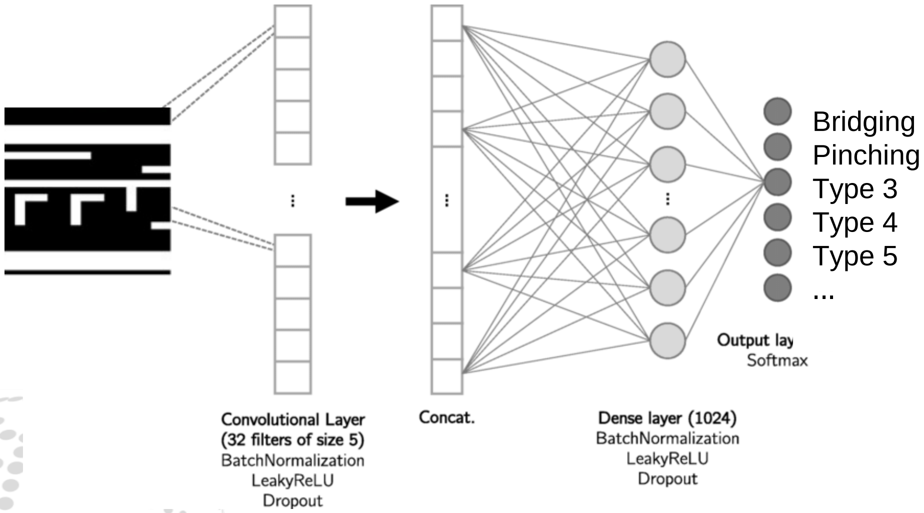


Fig. 17 The example of multiple channel input for CNNs.

Multi-class Classification

- Different failure types (bridging, pinching) can be viewed as multiple class classification problem
- CNN and its training, calibration are the same. Only need to label the data accordingly and change the number of the last FC layer (output layer) to the number of failure types to be predicted by the model




```
In [44]: #cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
         cnn.add(tf.keras.layers.Dense(units=n, activation='softmax'))
```

Defect Localization

- Sliding window
  - Computation can be reduced by fast scan algorithm (conv. the clip in advance)
  - Computation can be speeded up GPU processing
- Clustering
  - DBSCAN scan clustering algorithm: accepts clusters if more than a minimum number of HS (threshold) are detected within a specified range
- Coordinate extraction

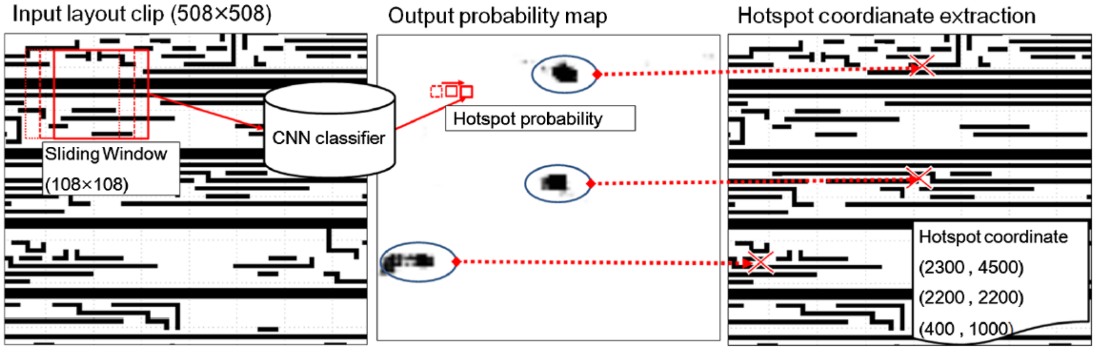


Fig. 1 HS detection using sliding window scan and coordinate extraction.

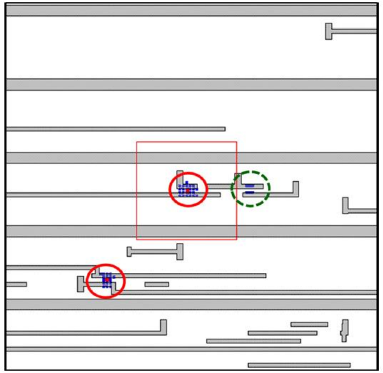
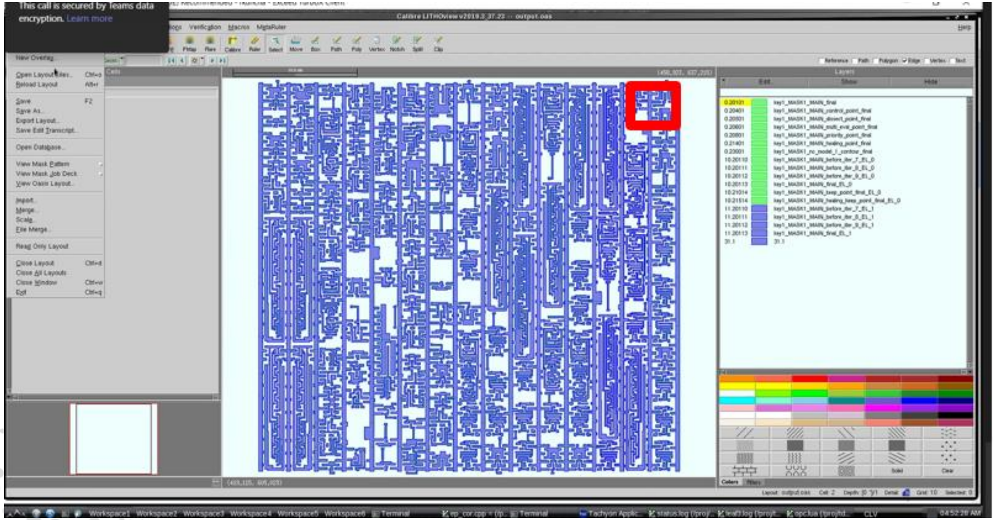
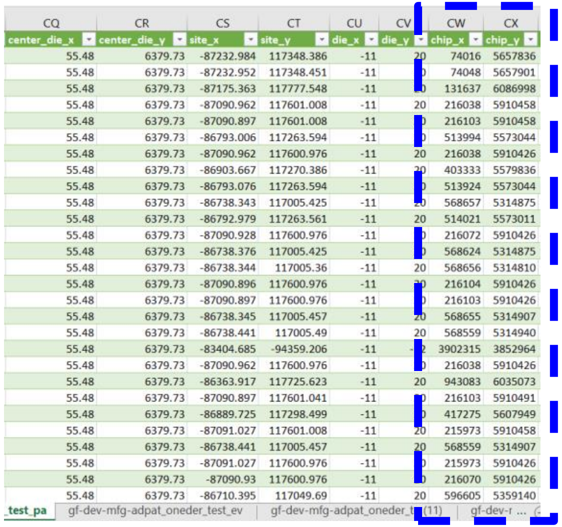


Fig. 8 Potential HS (dots), clustered point (line circle), and not accepted clusters (dotted line circle).

Defect Localization

- Sliding window
  - ➢ Computation can be reduced by fast scan algorithm (conv. the clip in advance)
  - ➢ Computation can be speeded up GPU processing
- Clustering
  - ➢ DBSCAN scan clustering algorithm: accepts clusters if more than a minimum number of HS (threshold) are detected within a specified range
- Coordinate extraction





CQ center_die_x	CR center_die_y	CS site_x	CT site_y	CU die_x	CV die_y	CW chip_x	CX chip_y
55.48	6379.73	-87232.984	117348.386	-11	20	74016	5657836
55.48	6379.73	-87232.952	117348.451	-11	[illegible]	74048	5657901
55.48	6379.73	-87175.363	117777.548	-11	20	131637	6086998
55.48	6379.73	-87090.962	117601.008	-11	20	216038	5910458
55.48	6379.73	-87090.897	117601.008	-11	[illegible]	216103	5910458
55.48	6379.73	-86793.006	117263.594	-11	[illegible]	513994	5573044
55.48	6379.73	-87090.962	117600.976	-11	20	216038	5910426
55.48	6379.73	-86903.667	117270.386	-11	20	403333	5579836
55.48	6379.73	-86793.076	117263.594	-11	[illegible]	513924	5573044
55.48	6379.73	-86738.343	117005.425	-11	20	568657	5314875
55.48	6379.73	-86792.979	117263.561	-11	20	514021	5573011
55.48	6379.73	-87090.928	117600.976	-11	[illegible]	216072	5910426
55.48	6379.73	-86738.376	117005.425	-11	[illegible]	568624	5314875
55.48	6379.73	-86738.344	117005.36	-11	20	568656	5314810
55.48	6379.73	-87090.896	117600.976	-11	[illegible]	216104	5910426
55.48	6379.73	-87090.897	117600.976	-11	[illegible]	216103	5910426
55.48	6379.73	-86738.345	117005.457	-11	20	568655	5314907
55.48	6379.73	-86738.441	117005.49	-11	20	568559	5314940
55.48	6379.73	-83404.685	-94359.206	-11	[illegible]	3902315	3852964
55.48	6379.73	-87090.962	117600.976	-11	[illegible]	216038	5910426
55.48	6379.73	-86363.917	117725.623	-11	20	943083	6035073
55.48	6379.73	-87090.897	117601.041	-11	[illegible]	216103	5910491
55.48	6379.73	-86889.725	117298.499	-11	[illegible]	417275	5607949
55.48	6379.73	-87091.027	117601.008	-11	20	215973	5910458
55.48	6379.73	-86738.441	117005.457	-11	20	568559	5314907
55.48	6379.73	-87091.027	117600.976	-11	[illegible]	215973	5910426
55.48	6379.73	-87090.93	117600.976	-11	[illegible]	216070	5910426
55.48	6379.73	-86710.395	117049.69	-11	20	596605	5359140

Fast Scan Algorithm

- Conducted in the test stage, not in training stage
- Feature map is obtained by convolution operation performed on the entire layout
- Window scan is performed on the shared feature map, instead of on the original layout, so no redundant convolution operation is carried out
- For each window scan, only FC layer calculation is required for hotspot prediction

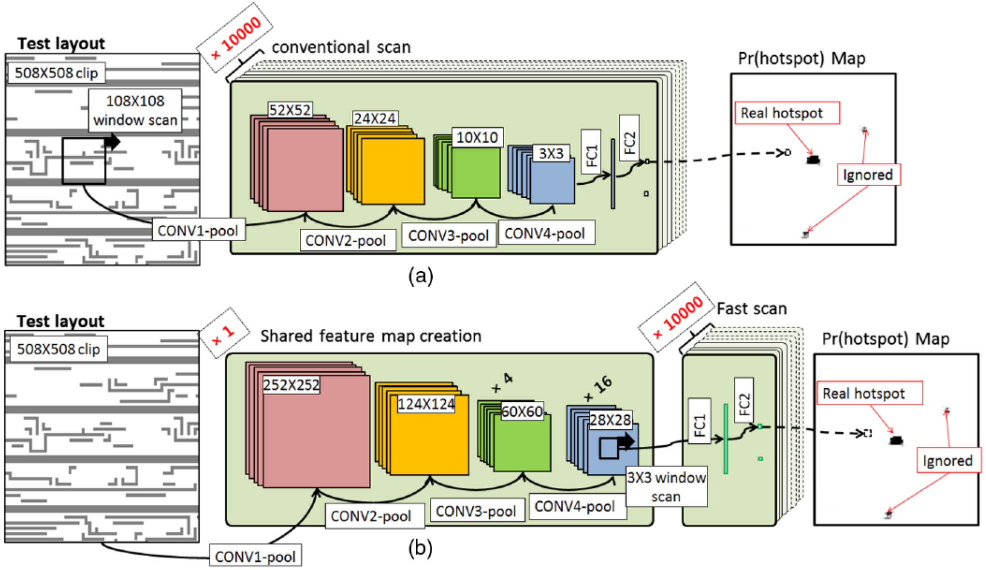


Fig. 6 Our HS detection framework with (a) conventional sliding window scan and (b) fast scan with shared feature map creation.

Conclusions

Conclusions

- ML models are proposed for both site-based data (ANN) and grid-based data (CNN).
- For site-based data, a proper input format is still needed to be determined
- The current prototype model is able to handle binary classification problem (HS/NHS) for grid-based data. It is trained and tested on the open database ICCAD2012 and reached satisfied accuracy with good generalization performance.
- The prototype model can be extended to multiclass classification (to detect different defect types) with multiple channel input (e.g. different image parameters).
- Hotspot localization can be achieved (as object detection problem) after obtaining the prediction of each scan window and clustering the probability map of the whole clip
- To further improve the computation efficiency, the ML model can be deployed on multiple machines and GPU computing can be also a good choice.

Demo on Codes