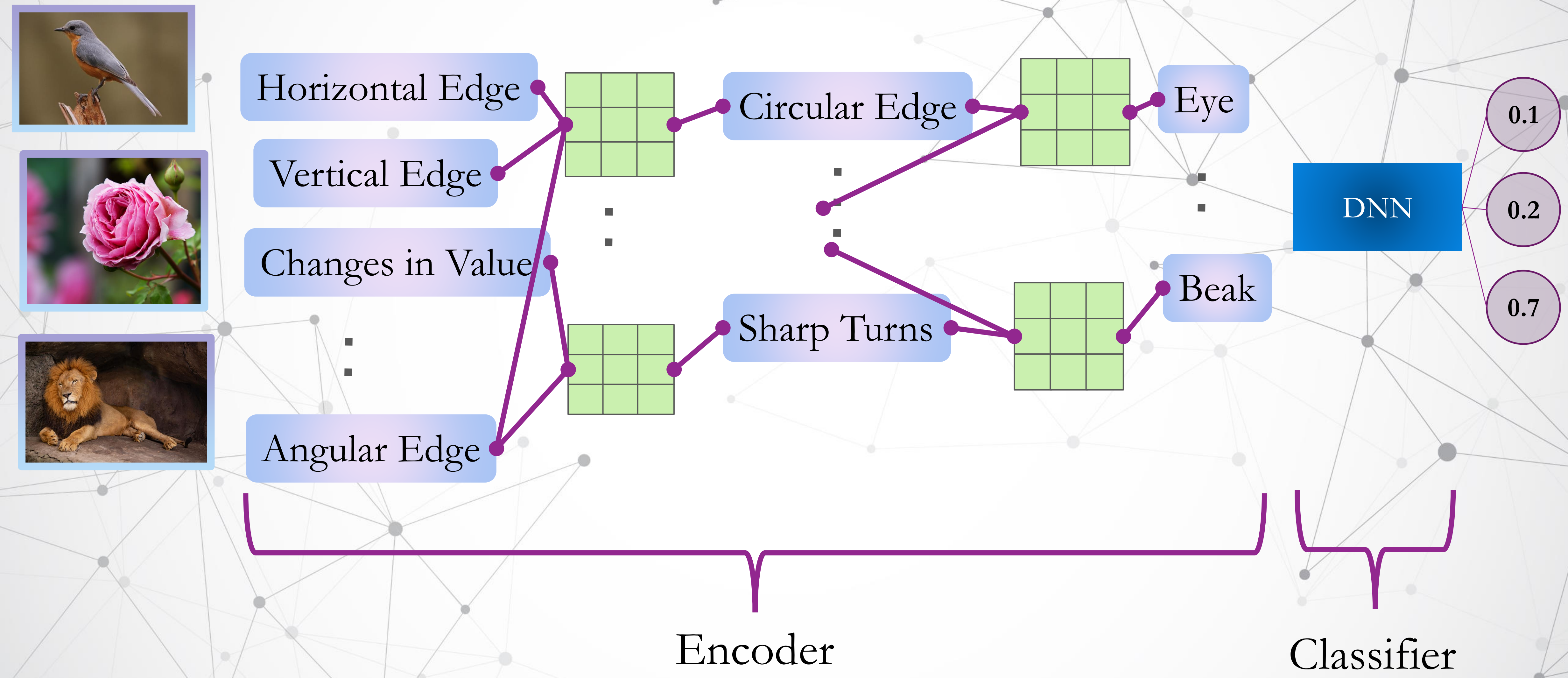
MLDL-I

Machine Learning and Deep Learning - I

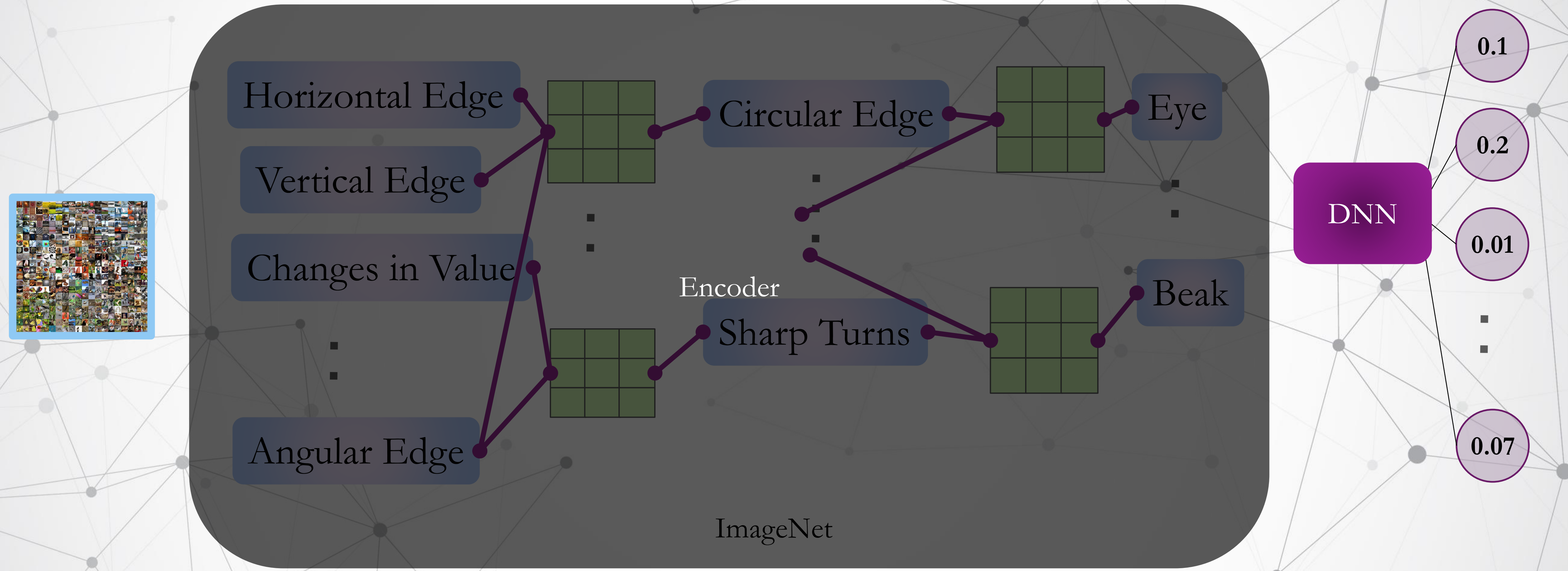



Horizontal Edge
Vertical Edge
Changes in Value
Angular Edge
Circular Edge
Sharp Turns
Eye
Beak
DNN
0.1
0.2
0.7
Encoder
Classifier

ImageNet

- 1.2 M Training Data*
- 50K Validation Data*
- 100K Test Data*
- 1000 Classes*

*indicates the 2010 Competition



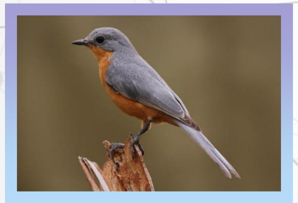
Horizontal Edge
Vertical Edge
Changes in Value
Angular Edge
Circular Edge
Sharp Turns
Eye
Beak
Encoder
ImageNet
DNN
0.1
0.2
0.01
0.07

Transfer Learning

Encoder

DNN

Output



DNN

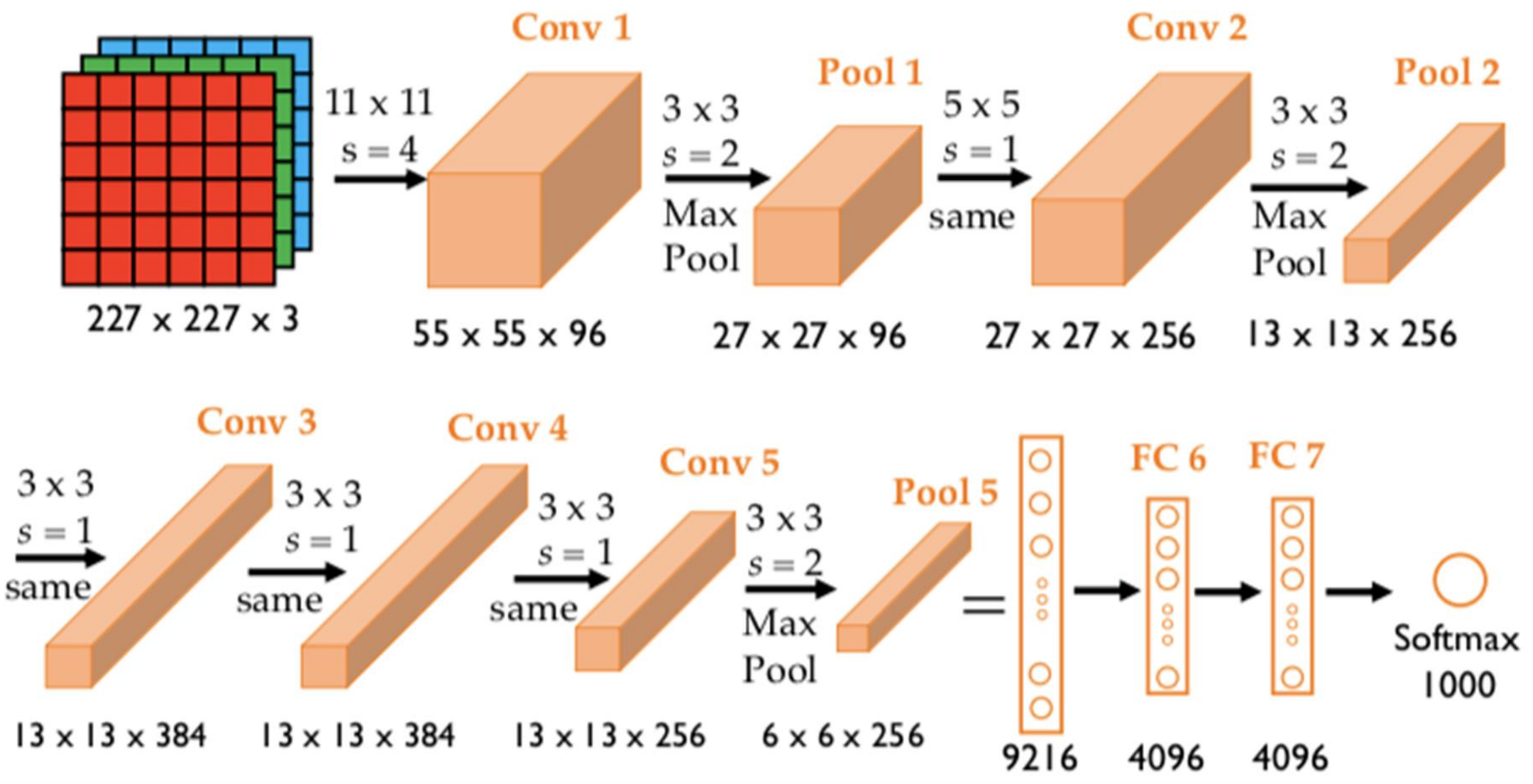
Output

```
inp = Input(shape=input_shape)
base_model = tf.keras.applications.VGG16(weights='imagenet',
                                         include_top=False, # drop classifer head
                                         input_shape=input_shape)

x = base_model(inp)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
out = Dense(num_classes, activation='sigmoid')(x)
```

CNN Architectures

AlexNet

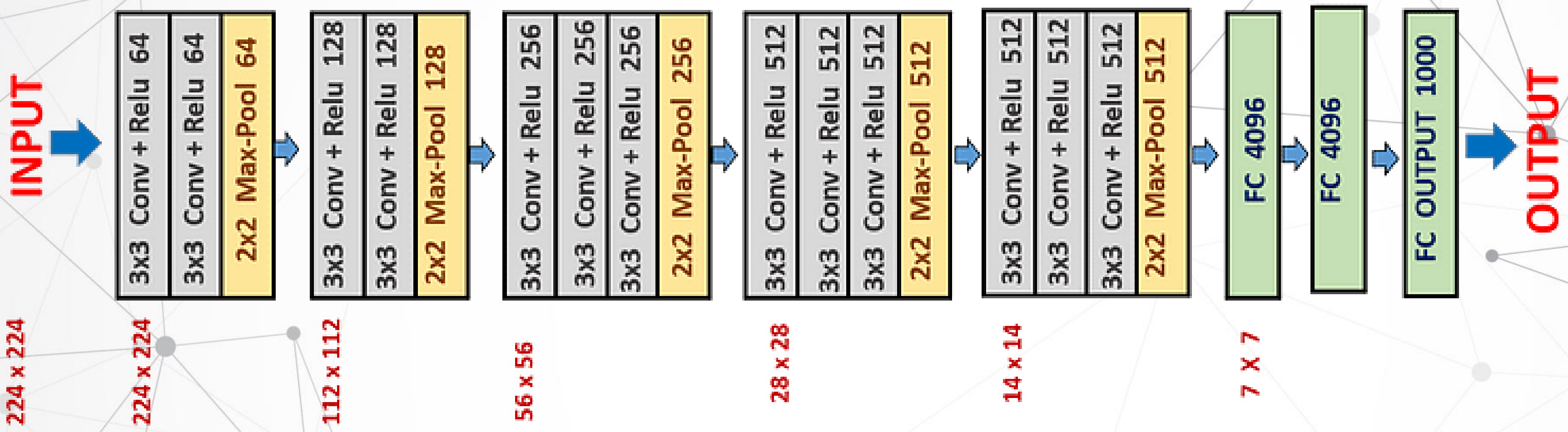


- Used ReLU
- Around 60 M Param
- Used 2 GTX 580
- (VRAM 6 GB Total)
- Overlapping Pooling
- Used Dropout

2012 ImageNet Classification with Deep Convolutional Neural Networks

[Link]

VGG



- Simplified Architecture
- 3x3 Conv ,2x2 MaxPool
- Resolution down, Channel up

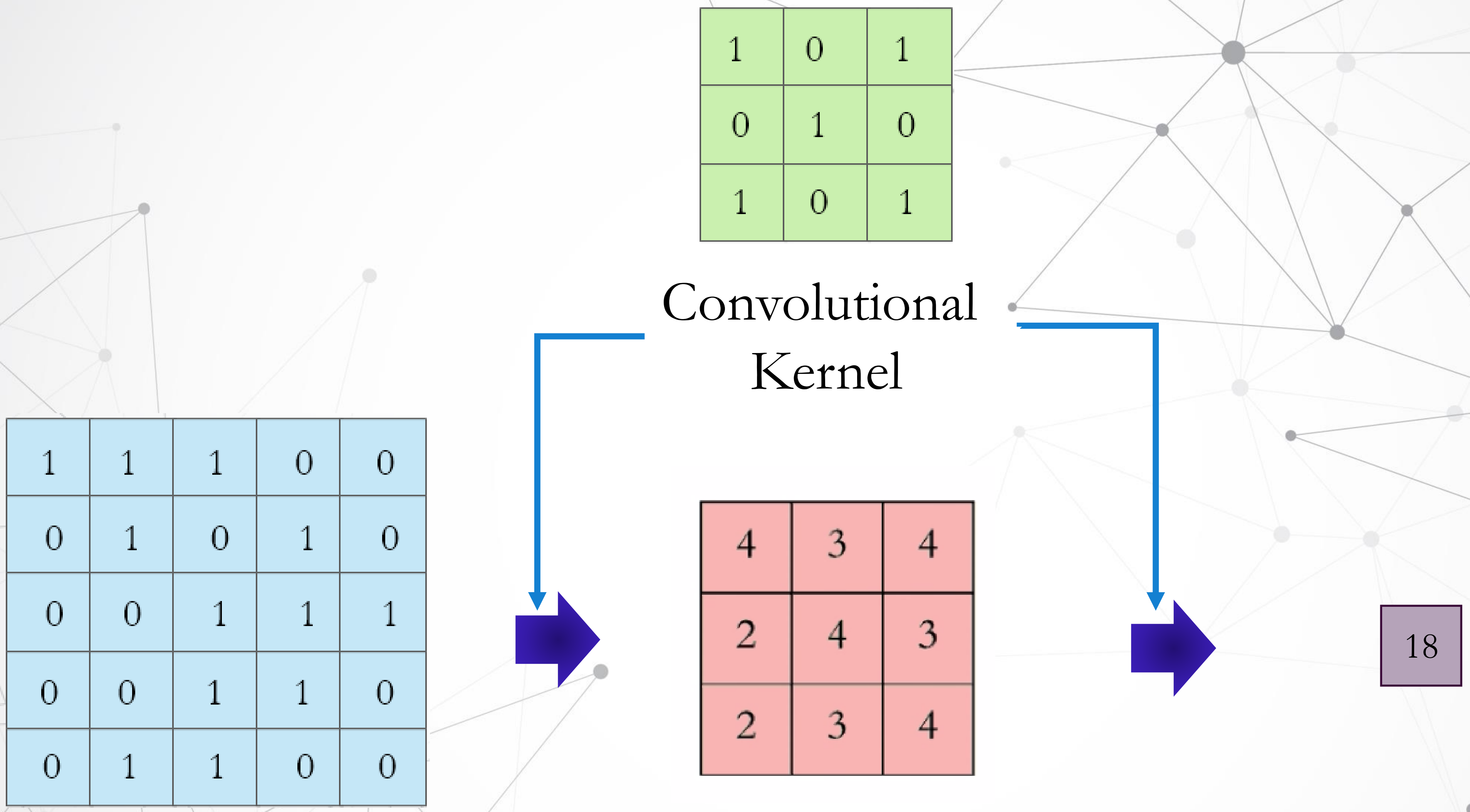
2015 Very deep convolutional networks for large-scale image recognition

Link

2	1	1	0	0
0	1	0	1	0
3	0	-1	1	1
0	0	1	1	0
1	1	1	0	0

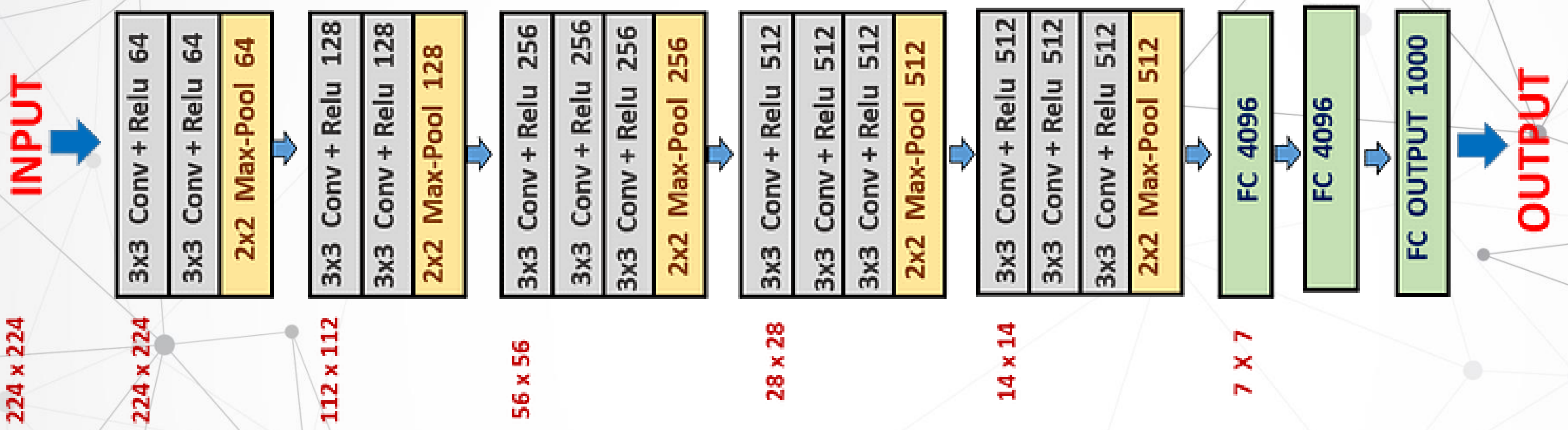
Convolutional
Kernel

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



1 0 1
0 1 0
1 0 1
Convolutional
Kernel
1 1 1 0 0
0 1 0 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
4 3 4
2 4 3
2 3 4
18

VGG



2015 Very deep convolutional networks for large-scale image recognition

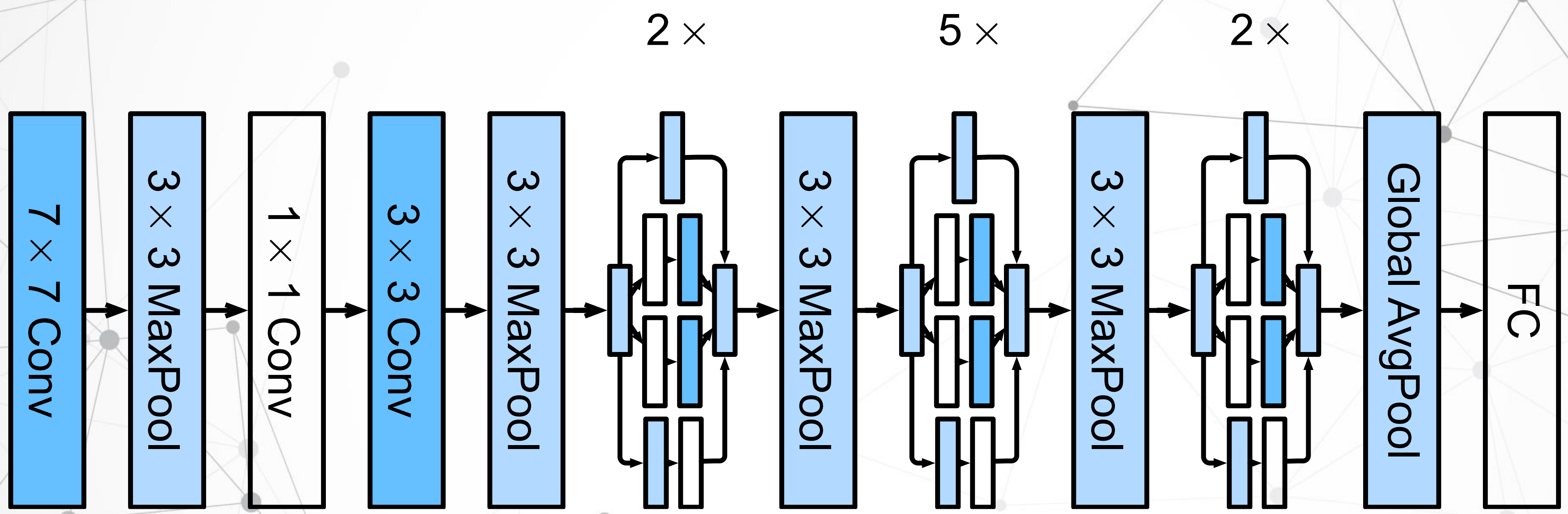
Link

such layers have a 7×7 effective receptive field. So what have we gained by using, for instance, a stack of three 3×3 conv. layers instead of a single 7×7 layer? First, we incorporate three non-linear rectification layers instead of a single one, which makes the decision function more discriminative. Second, we decrease the number of parameters: assuming that both the input and the output of a three-layer 3×3 convolution stack has C channels, the stack is parametrised by $3\left(3^2C^2\right) = 27C^2$ weights; at the same time, a single 7×7 conv. layer would require $7^2C^2 = 49C^2$ parameters, i.e. 81% more. This can be seen as imposing a regularisation on the 7×7 conv. filters, forcing them to have a decomposition through the 3×3 filters (with non-linearity injected in between).

Second, we observe that the classification error decreases with the increased ConvNet depth: from 11 layers in A to 19 layers in E. Notably, in spite of the same depth, the configuration C (which contains three 1×1 conv. layers), performs worse than the configuration D, which uses 3×3 conv. layers throughout the network. This indicates that while the additional non-linearity does help (C is better than B), it is also important to capture spatial context by using conv. filters with non-trivial receptive fields (D is better than C). The error rate of our architecture saturates when the depth reaches 19 layers, but even deeper models might be beneficial for larger datasets. We also compared the net B with a shallow net with five 5×5 conv. layers, which was derived from B by replacing each pair of 3×3 conv. layers with a single 5×5 conv. layer (which has the same receptive field as explained in Sect. 2.3). The top-1 error of the shallow net was measured to be 7% higher than that of B (on a center crop), which confirms that a deep net with small filters outperforms a shallow net with larger filters.

2015 Very deep convolutional networks for large-scale image recognition

GoogLeNet

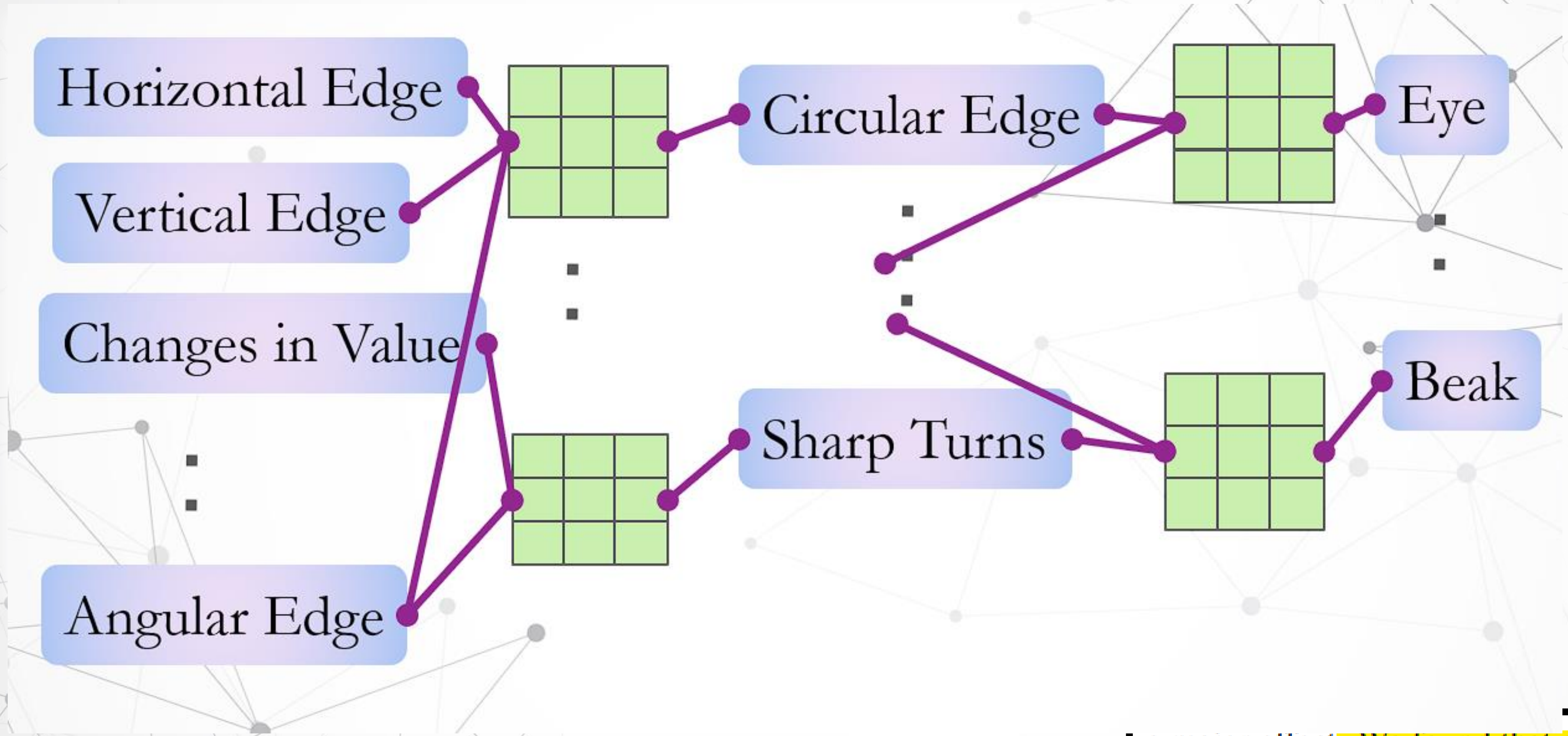


- 22 layers
- (27 Including Pooliing)

2015 Going Deeper with Convolutions

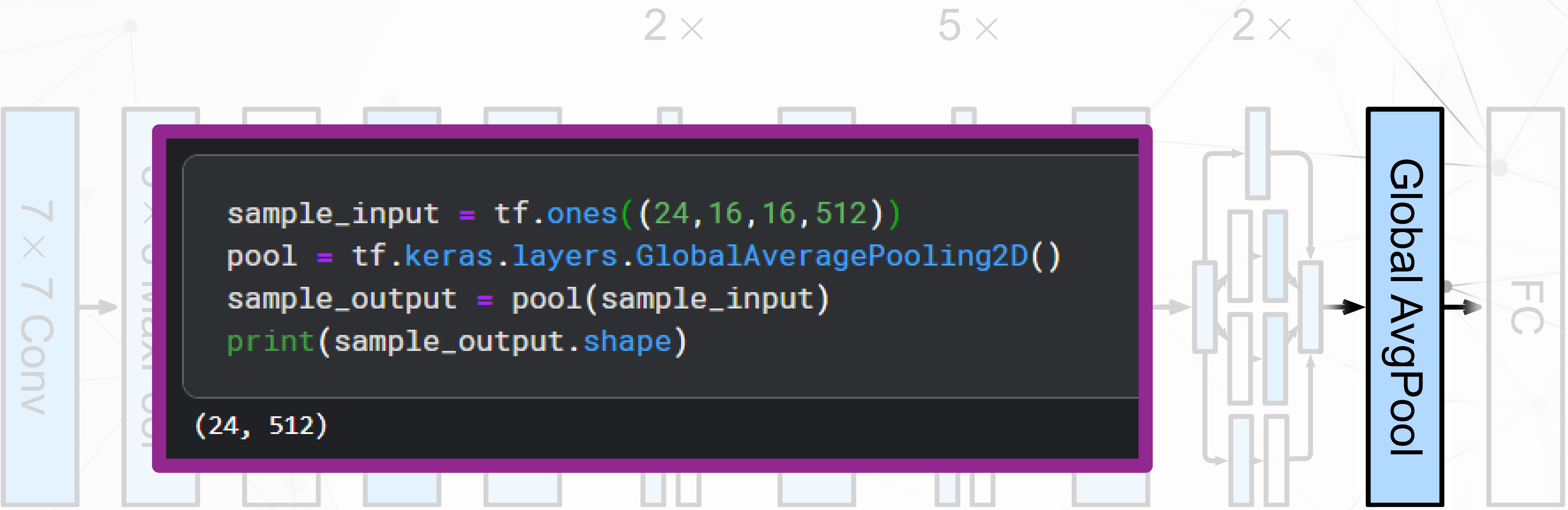
Link

Global Average Pooling



a major effect. We found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%, however the use of dropout remained essential even after removing the fully connected layers.

Global Average Pooling

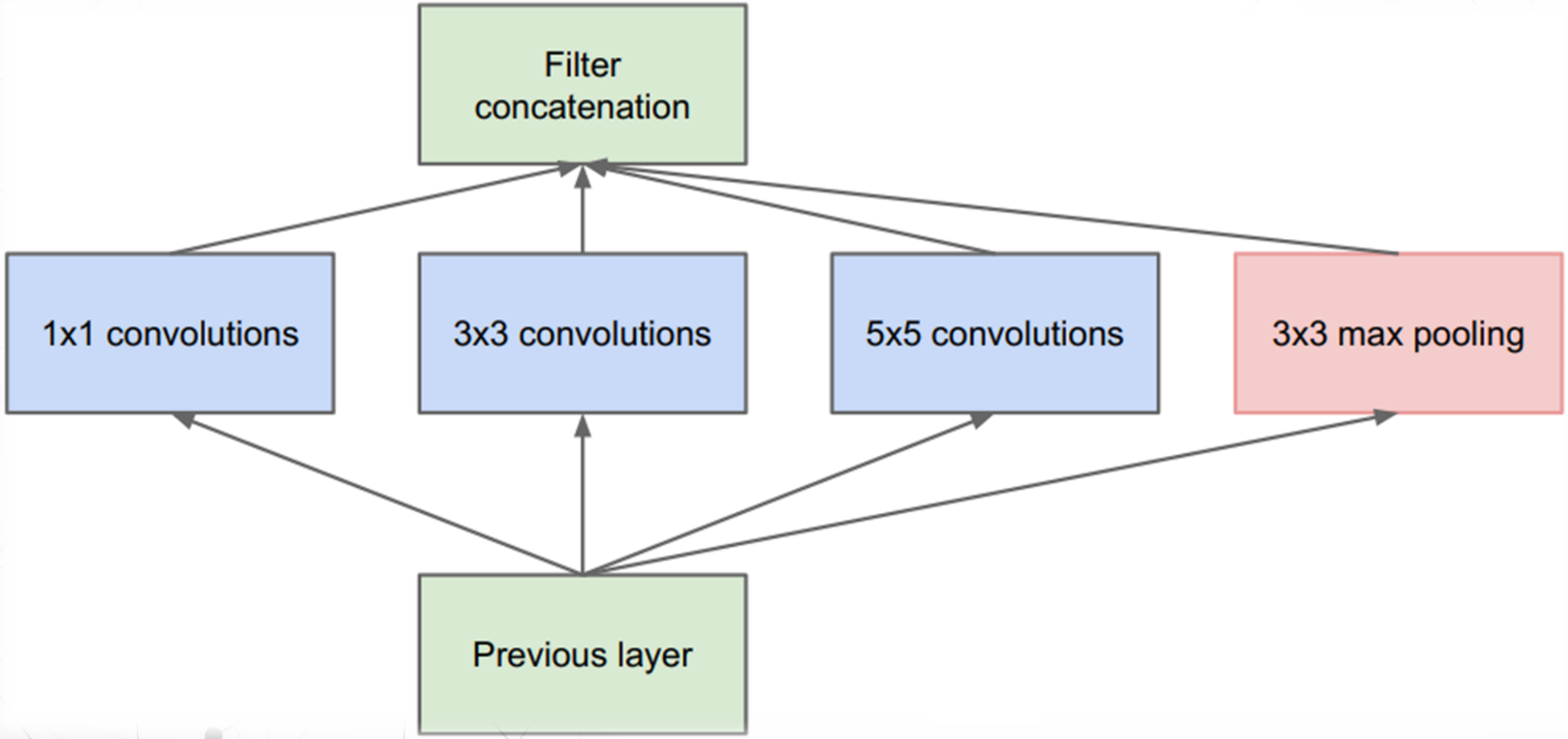



```python
sample_input = tf.ones((24,16,16,512))
pool = tf.keras.layers.GlobalAveragePooling2D()
sample_output = pool(sample_input)
print(sample_output.shape)
```

```
(24, 512)
```

> a major effect. We found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%, however the use of dropout remained essential even after removing the fully connected layers.

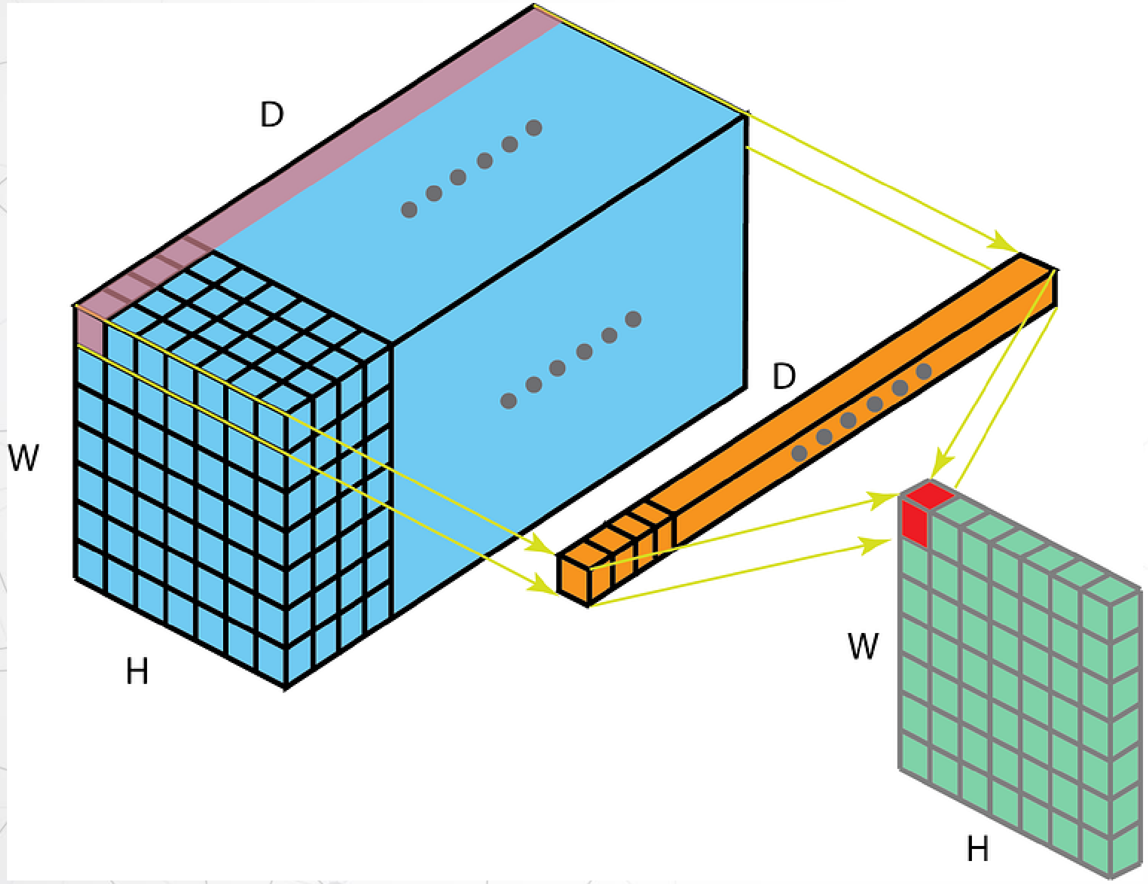
Inception

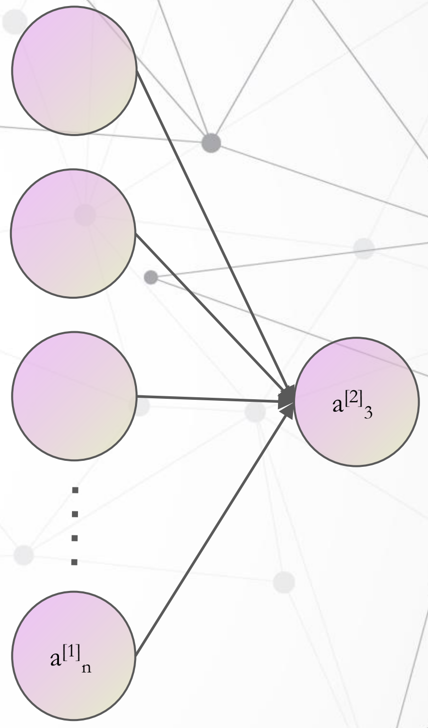


2015 Going Deeper with Convolutions

Link

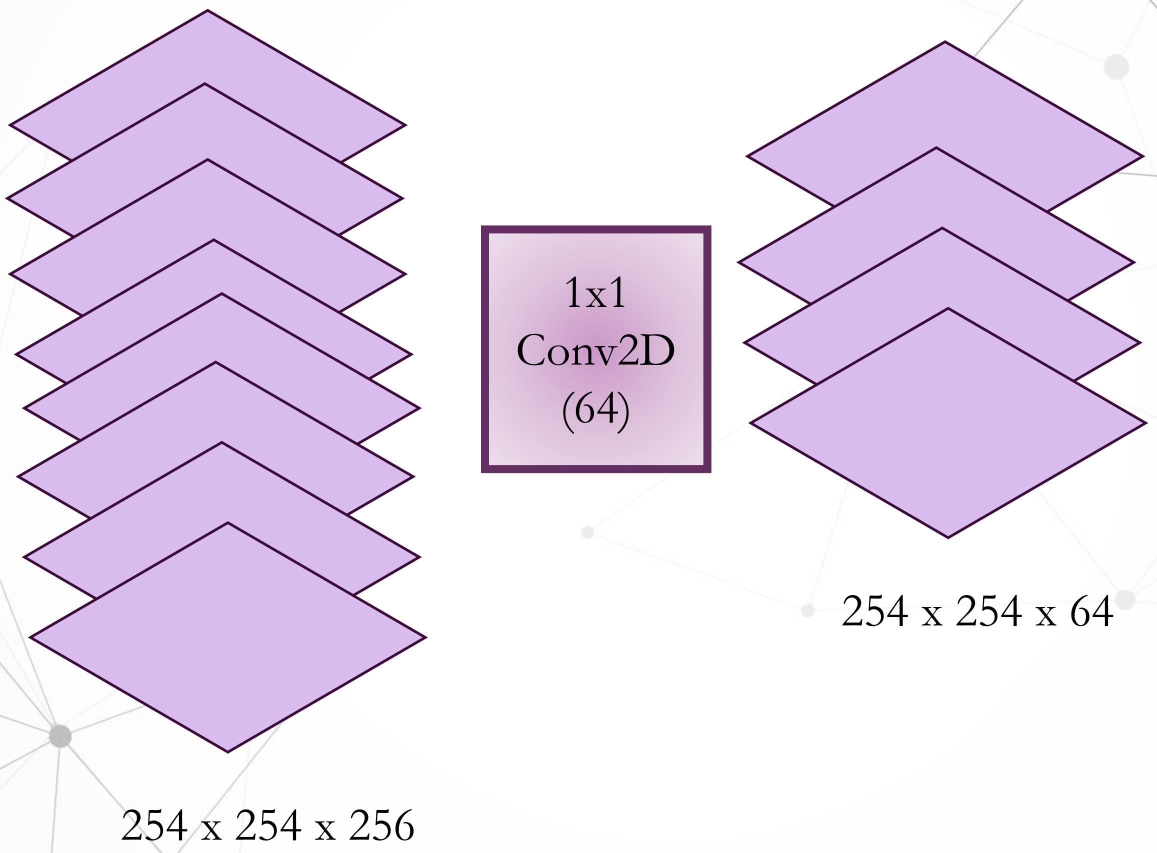
1x1 Convolutions





[Link]

1x1 Convolutions



(256x1)x64 = 16,384

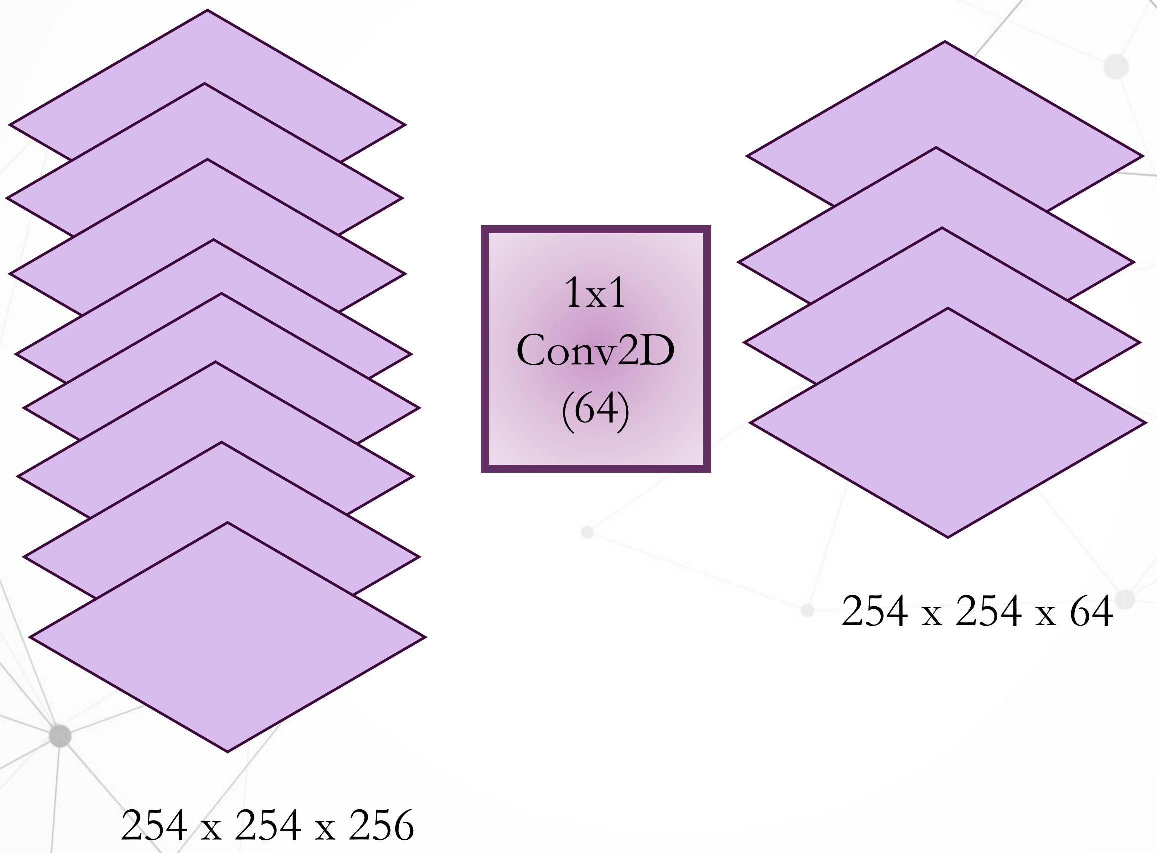
1x1 Convolutions

Artificial intelligence (AI) is the intelligence of machines or softwares, as opposed to the intelligence of human beings or animals. AI applications include advanced web search engines (e.g., Google Search), recommendation s
Netflix), understanding human speech (such as S
Waymo), generative or creative tools (ChatGPT a
level in strategic games (such as chess and Go).

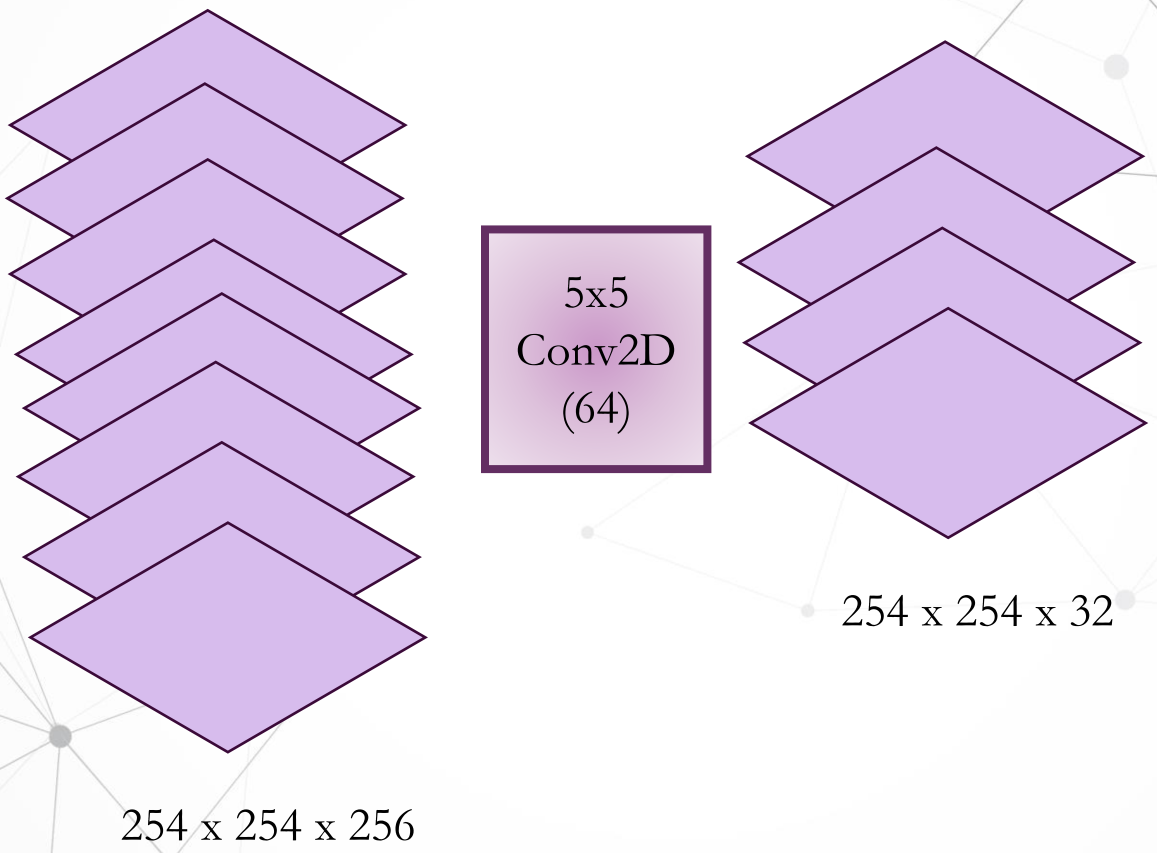
Artificial intelligence was founded as an academi
has experienced several waves of optimism, follo
funding (known as an "AI winter"),followed by n
funding. AI research has tried and discarded mar
the brain, modeling human problem solving, for
imitating animal behavior. In the first decades of the 21st century, highly mathematical and statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

AI is the intelligence of machines/software, used in various applications like search engines, recommendation systems, speech recognition, self-driving cars, creative tools, and strategic game-playing. It has experienced cycles of optimism, disappointment, and renewed funding since its inception in 1956. Different approaches have been explored, with machine learning being dominant in recent years, solving complex problems in various domains

1x1 Convolutions

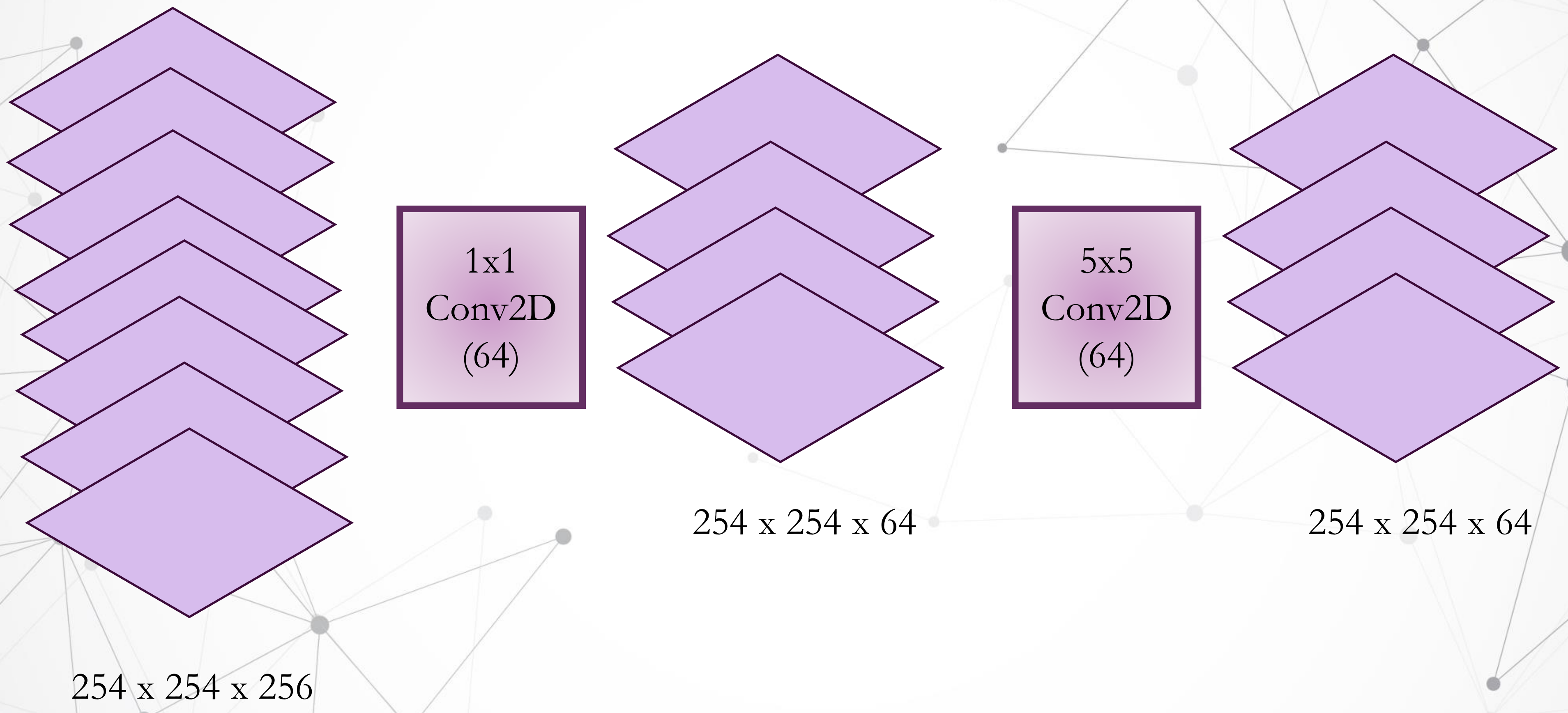


(256x1)x64 = 16,384



(5x5x256) x 64 = 409,600

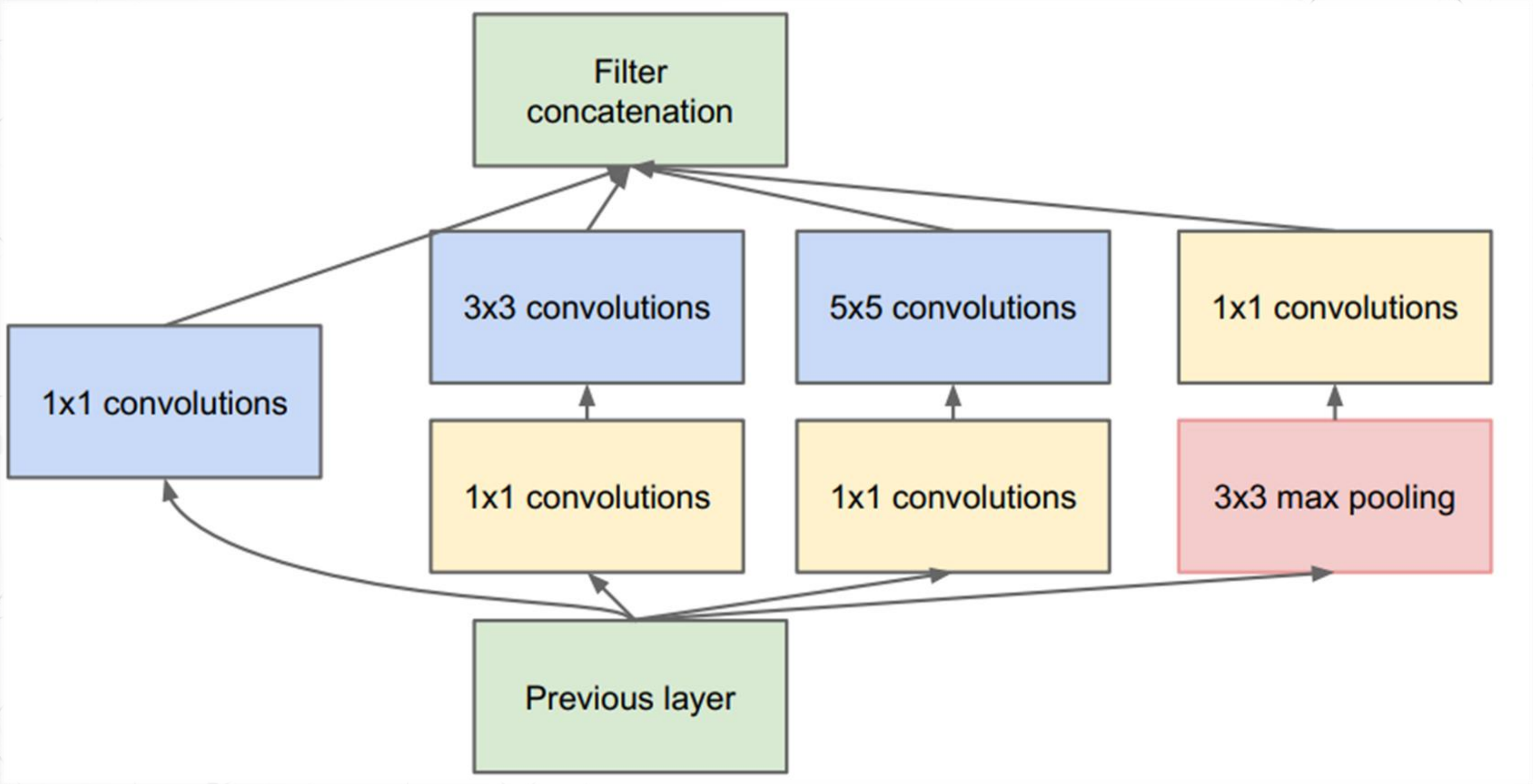
Bottleneck



$(256x1)x64 + (5x5x64) \times 64 = 118{,}784$

$(5x5x256) \times 64 = 409{,}600$

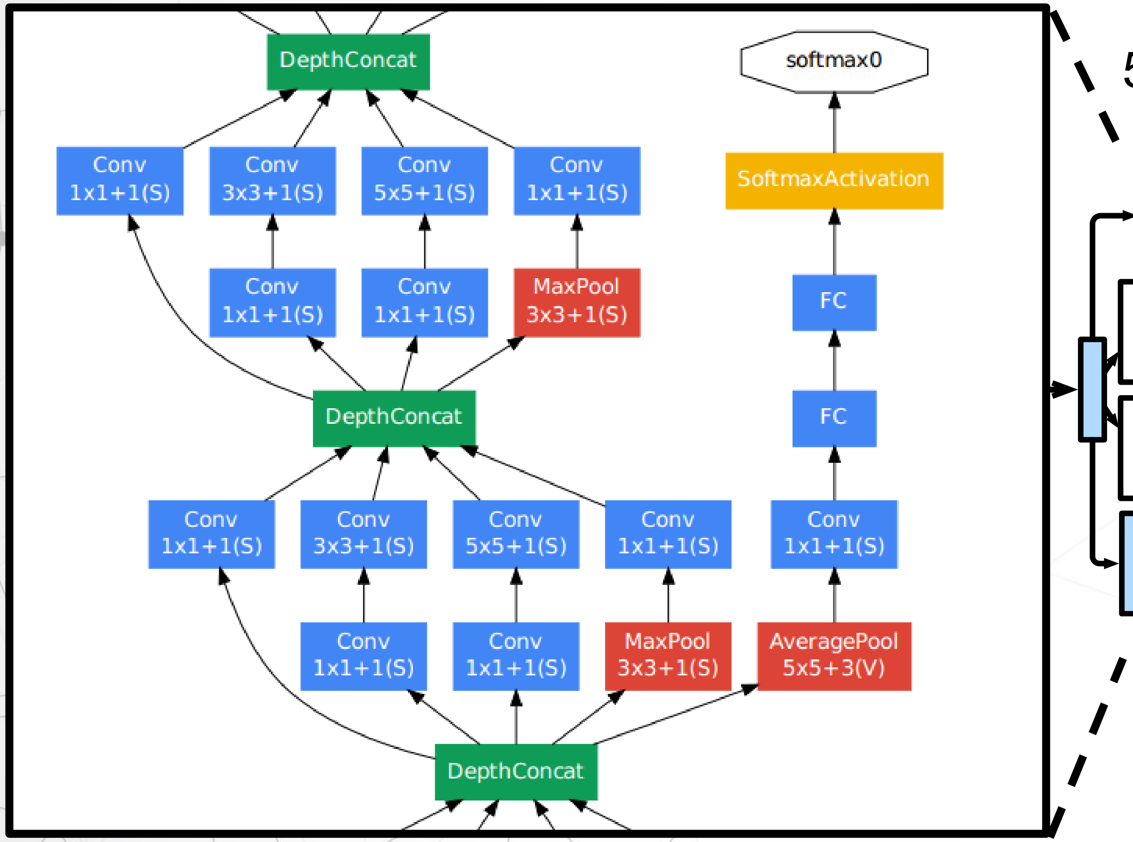
GoogLeNet

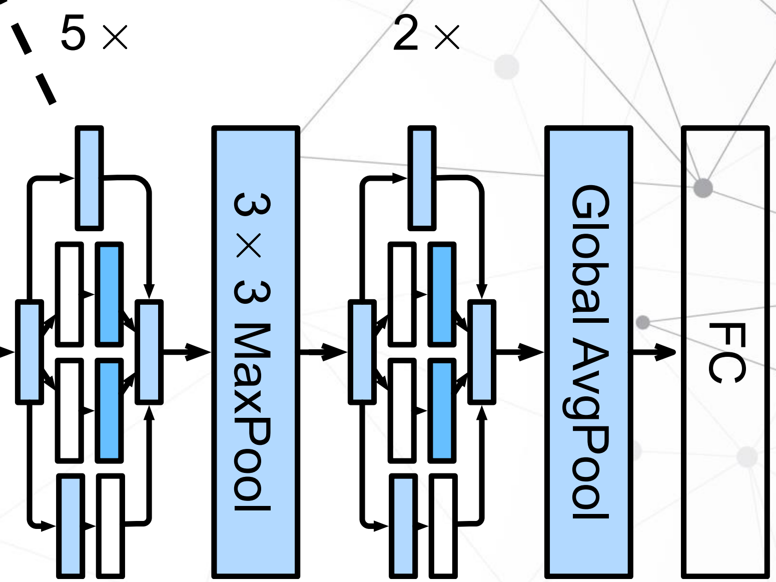


2015 Going Deeper with Convolutions

[Link]

GoogLeNet



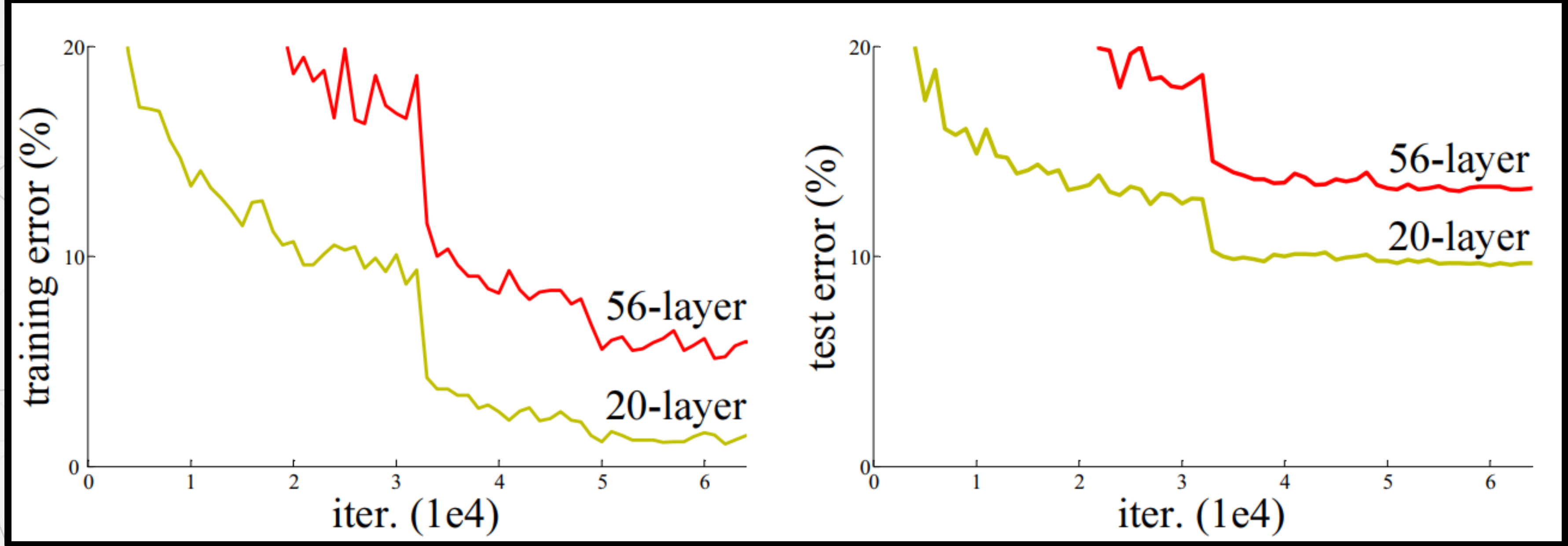


Given relatively large depth of the network, the ability to propagate gradients back through all the layers in an effective manner was a concern. The strong performance of shallower networks on this task suggests that the features produced by the layers in the middle of the network should be very discriminative. By adding auxiliary classifiers connected to these intermediate layers, discrimination in the lower stages in the classifier was expected. This was thought to combat the vanishing gradient problem while

2015 Going Deeper with Convolutions

Link

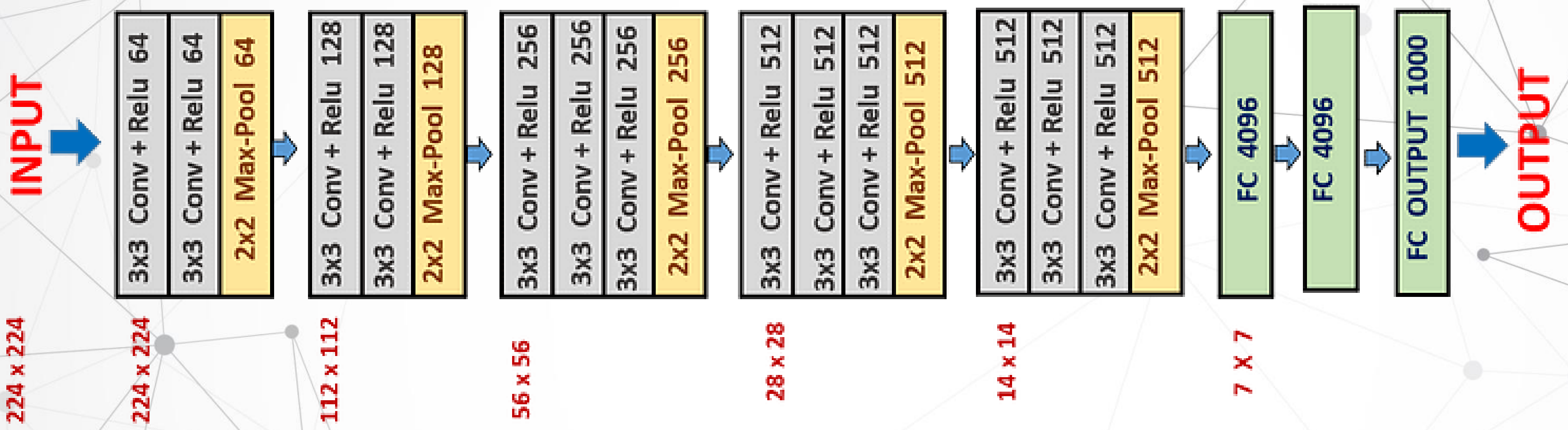
ResNet



When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [10, 41] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

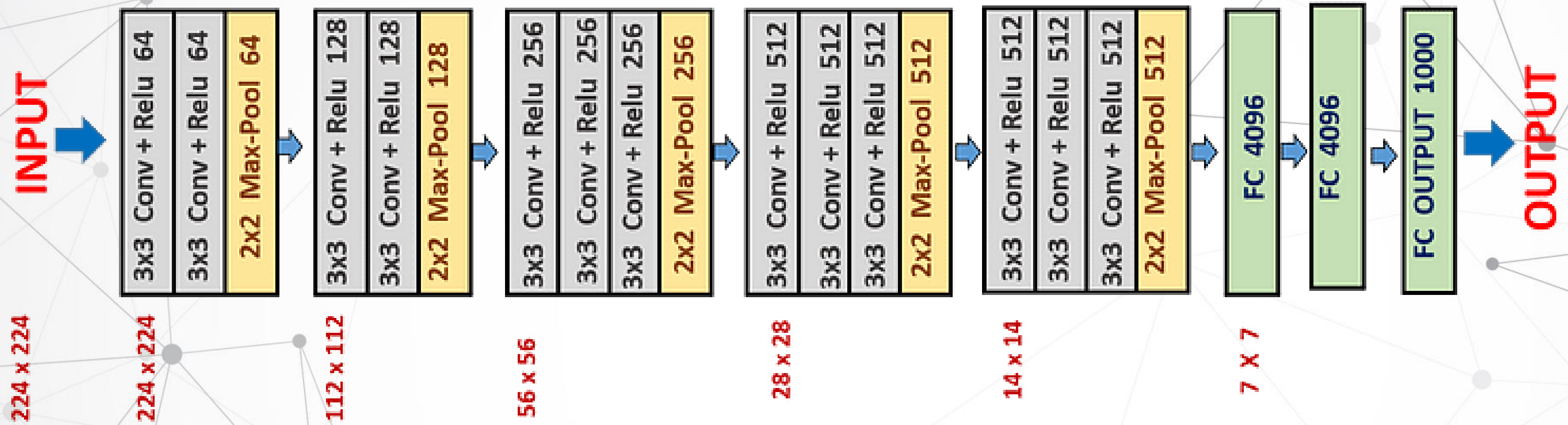
2016 Deep Residual Learning for Image Recognition

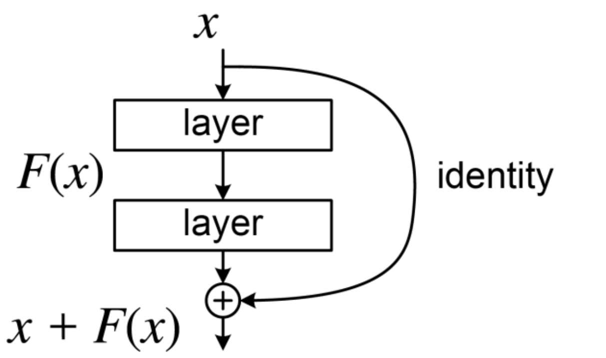
ResNet



2016 Deep Residual Learning for Image Recognition

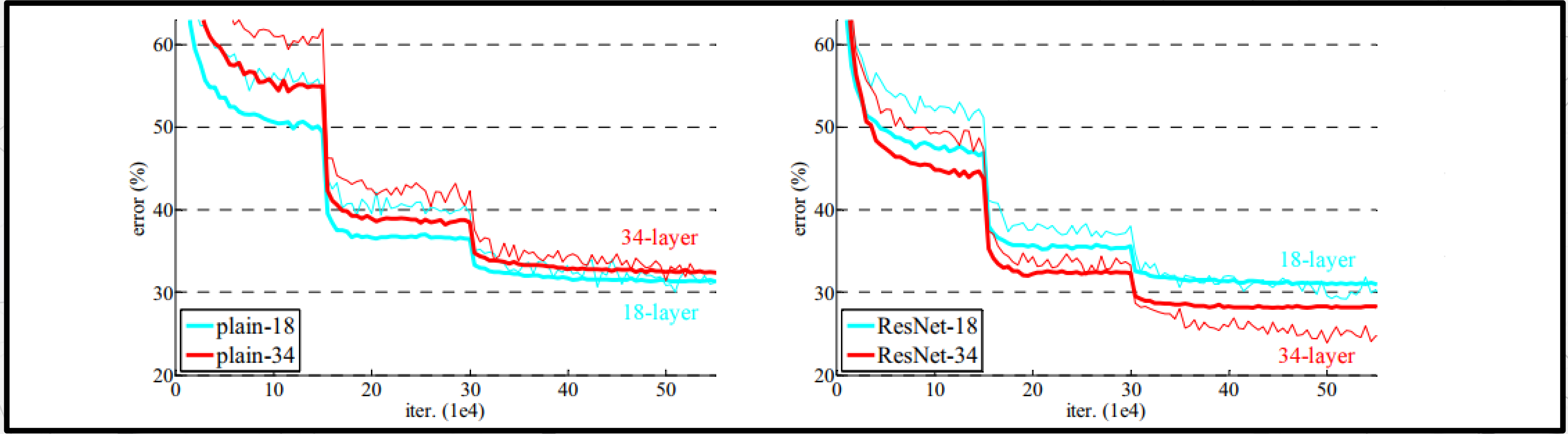
ResNet





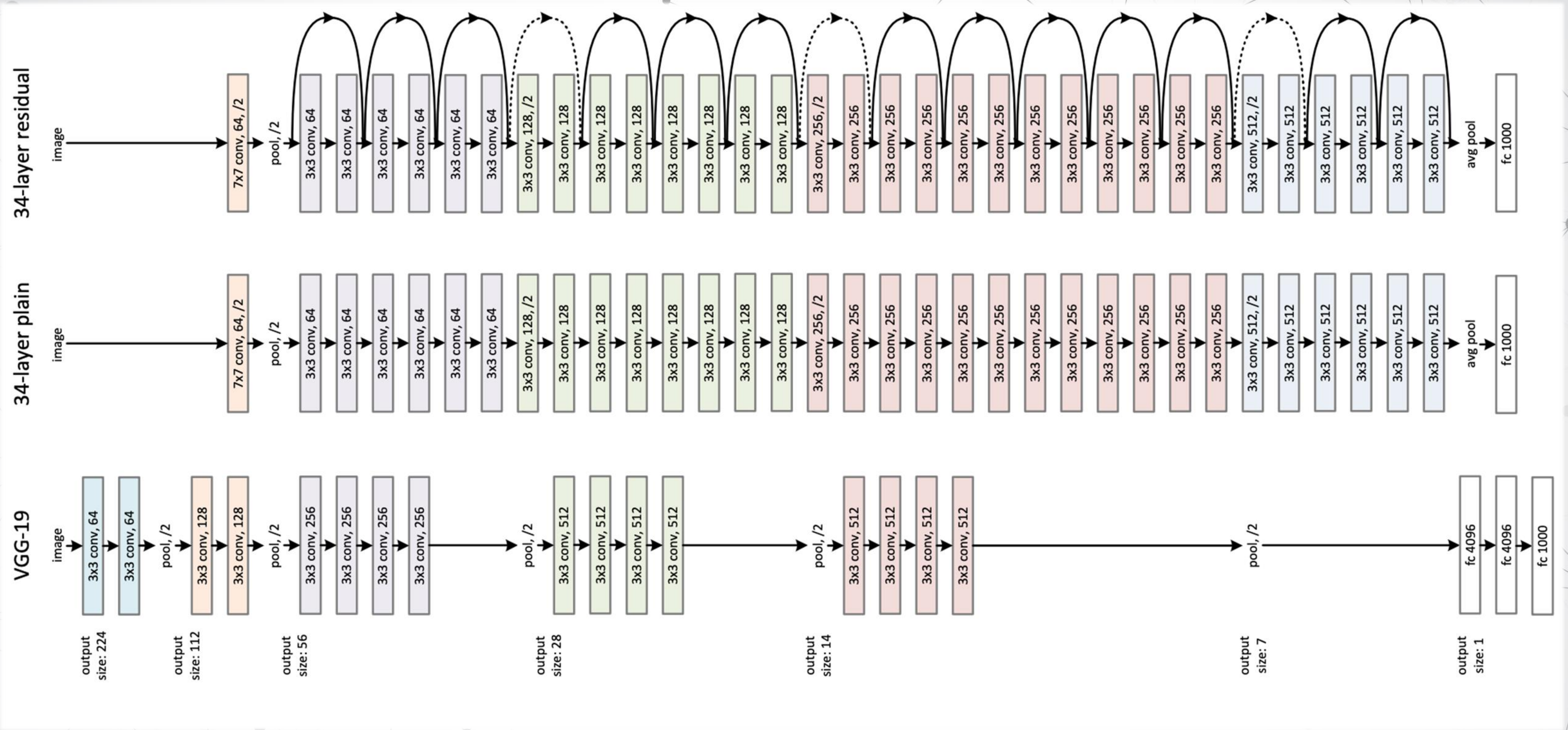
2016 Deep Residual Learning for Image Recognition

ResNet



2016

Deep Residual Learning for Image Recognition



2016 Deep Residual Learning for Image Recognition

2012 ImageNet Classification with Deep Convolutional Neural Networks

2015 Very deep convolutional networks for large-scale image recognition

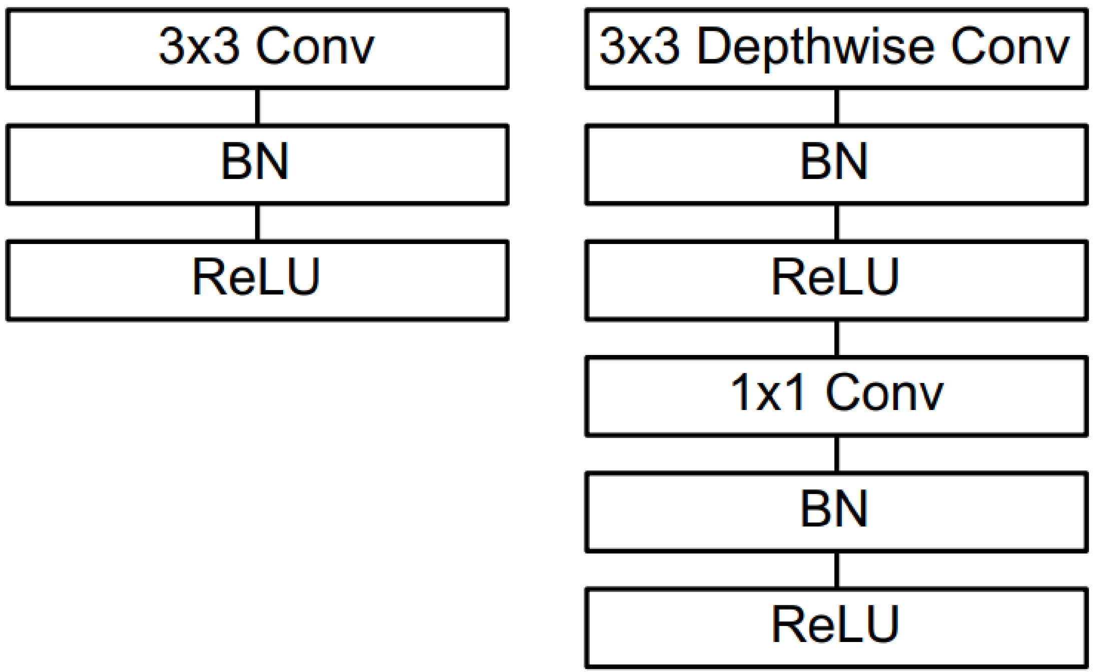
2015 Going Deeper with Convolutions

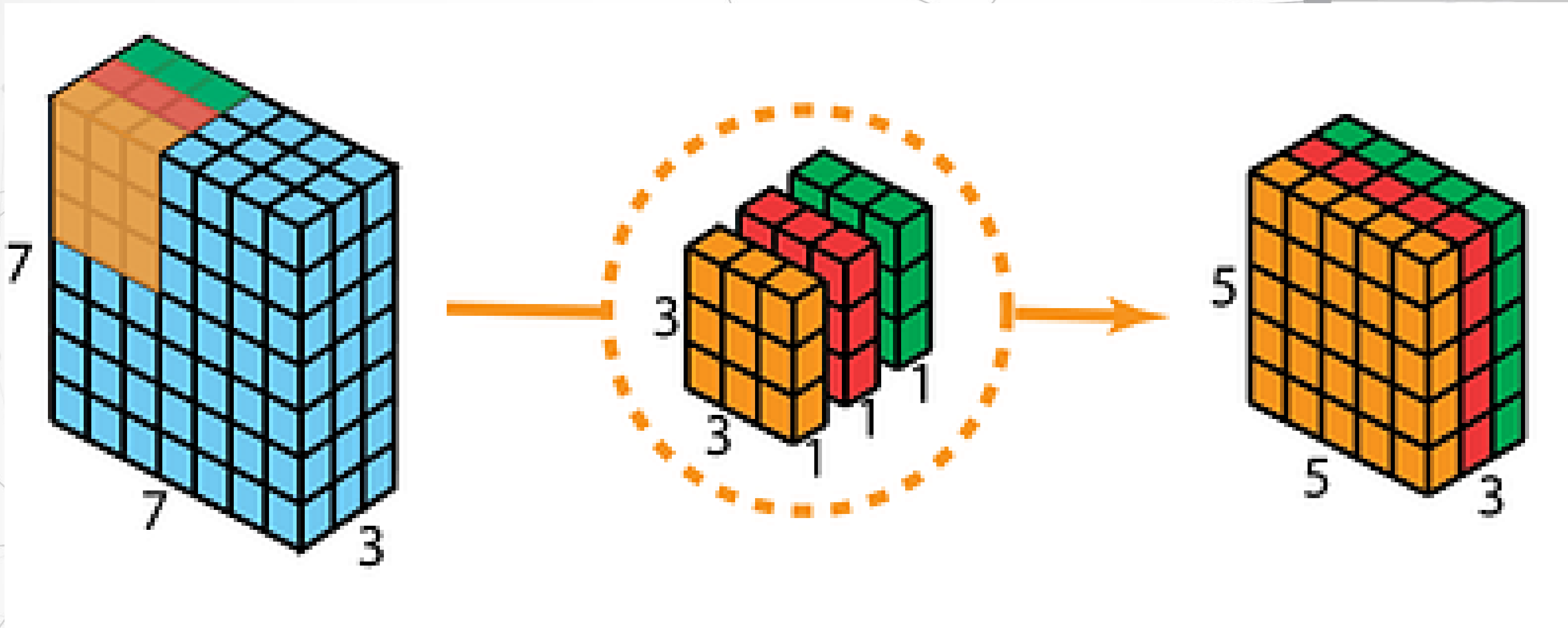
2016 Deep Residual Learning for Image Recognition

MobileNet

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

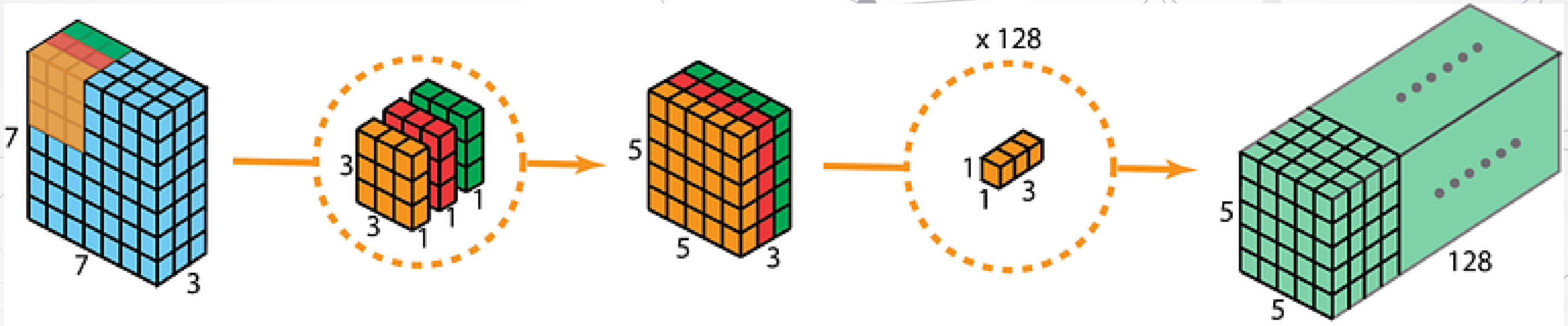





```
sample_input = tf.ones((24,16,16,512))
layer = tf.keras.layers.DepthwiseConv2D(
        kernel_size = (3,3),
        strides=(1, 1),
        padding="same")
sample_output = layer(sample_input)
print(sample_output.shape)
```

```
(24, 16, 16, 512)
```

Link

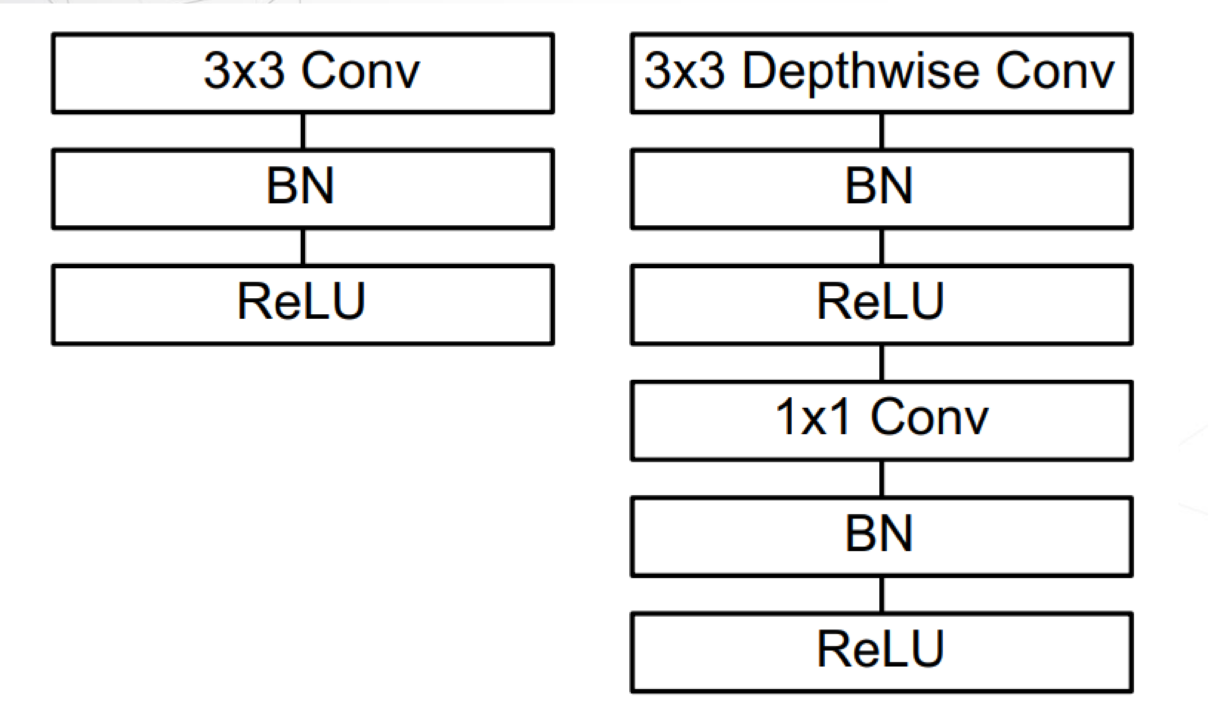


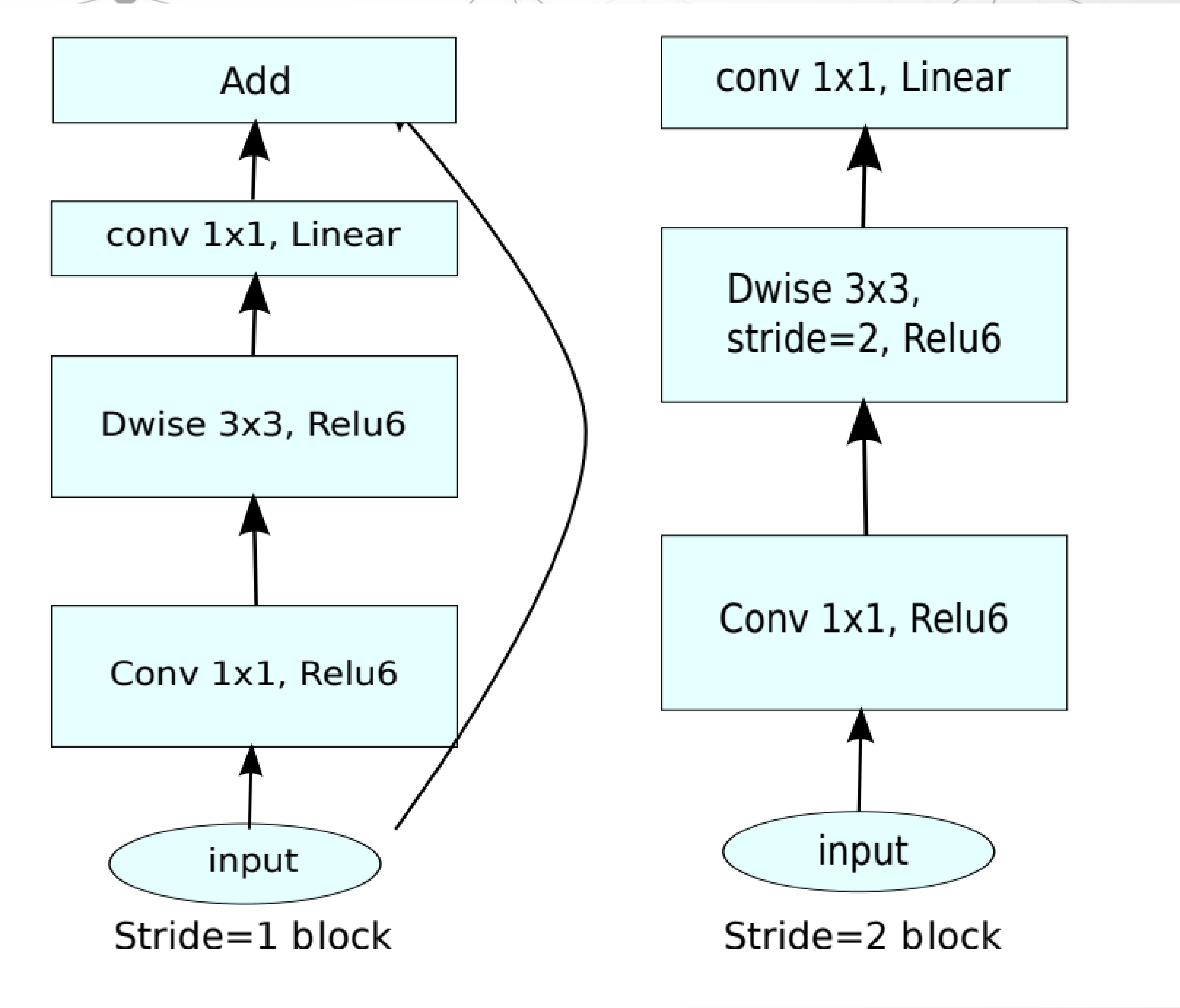

```
sample_input = tf.ones((24,16,16,512))
layer1 = tf.keras.layers.DepthwiseConv2D(
        kernel_size = (3,3),
        strides=(1, 1),
        padding="same")
interm_output = layer1(sample_input)
print("Interm Output shape: ",interm_output.shape)
layer2 = tf.keras.layers.Conv2D(filters = 512,
                                kernel_size = (1,1),
                                padding='same')
final_output = layer2(interm_output)
print("Final Output shape: ",final_output.shape)
```

```
Interm Output shape:  (24, 16, 16, 512)
Final Output shape:  (24, 16, 16, 512)
```

Link

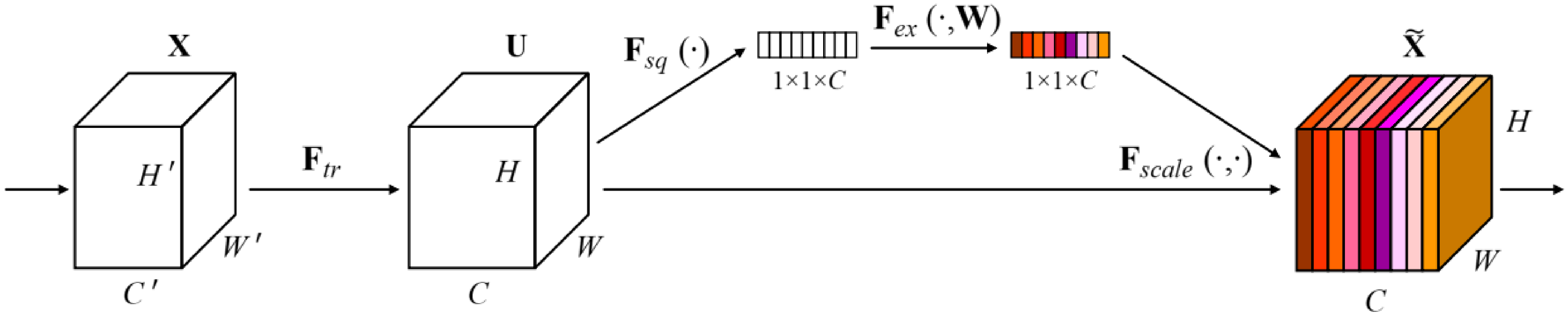
MobileNetV2





2018 MobileNetV2: Inverted Residuals and Linear Bottlenecks

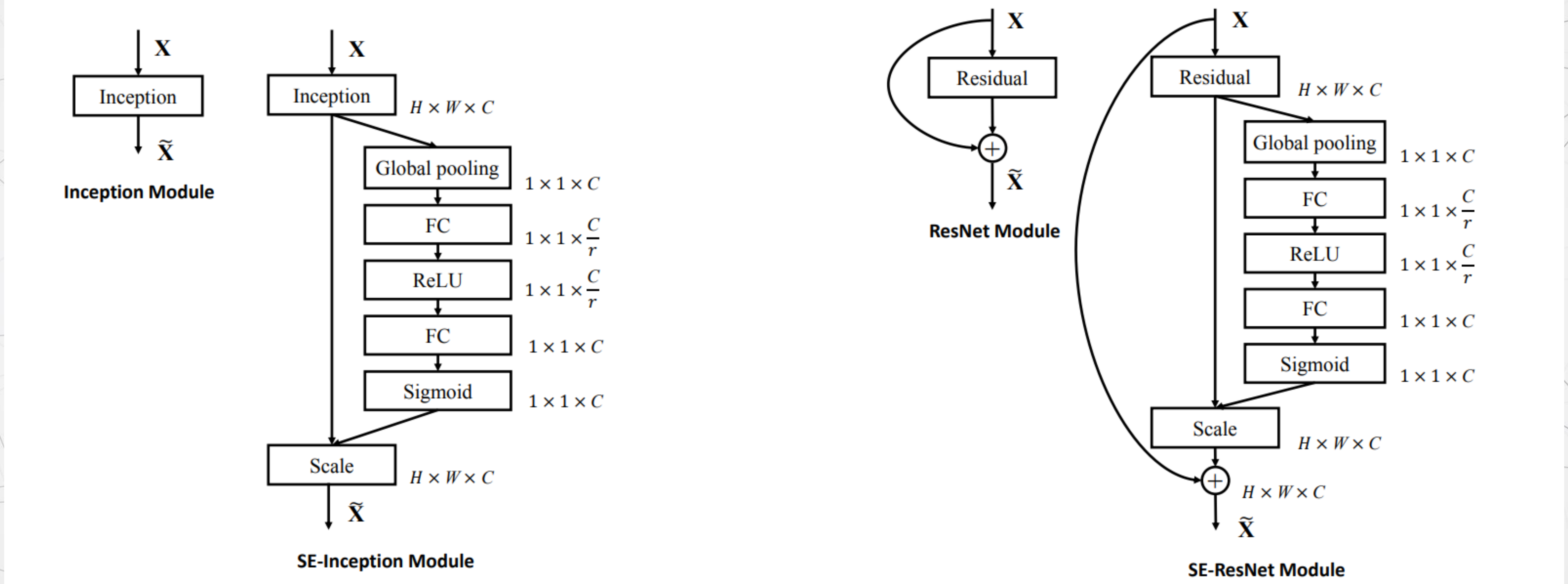
SENet




```
sample_input = tf.ones((24,16,16,512))
se = tf.keras.layers.GlobalAveragePooling2D()(sample_input)
se = tf.keras.layers.Dense(512, activation='sigmoid', use_bias=False)(se)
sample_output = tf.keras.layers.multiply([sample_input, se])
print(sample_output.shape)
```

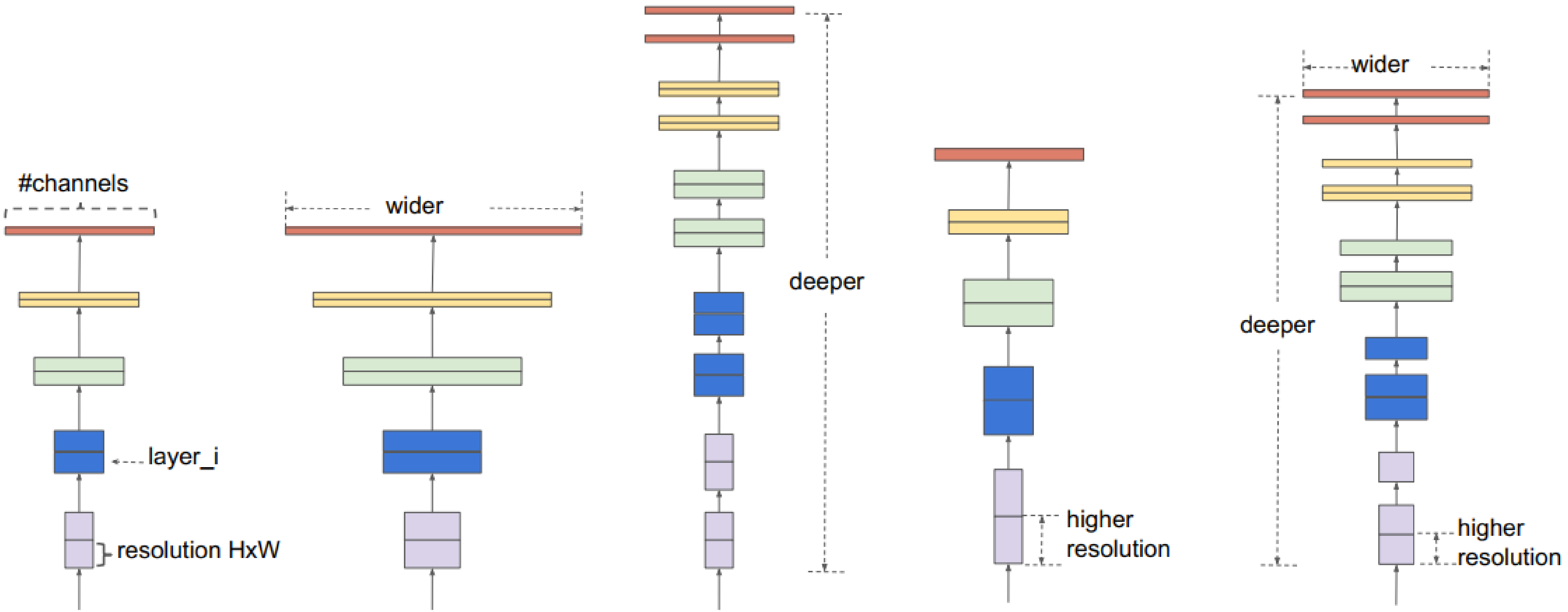
```
(24, 16, 16, 512)
```

SENet



2018 Squeeze-and-Excitation Networks

EfficientNet



(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling (e) compound scaling

2019

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

EfficientNet

In this paper, we propose a new **compound scaling method**, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

2019

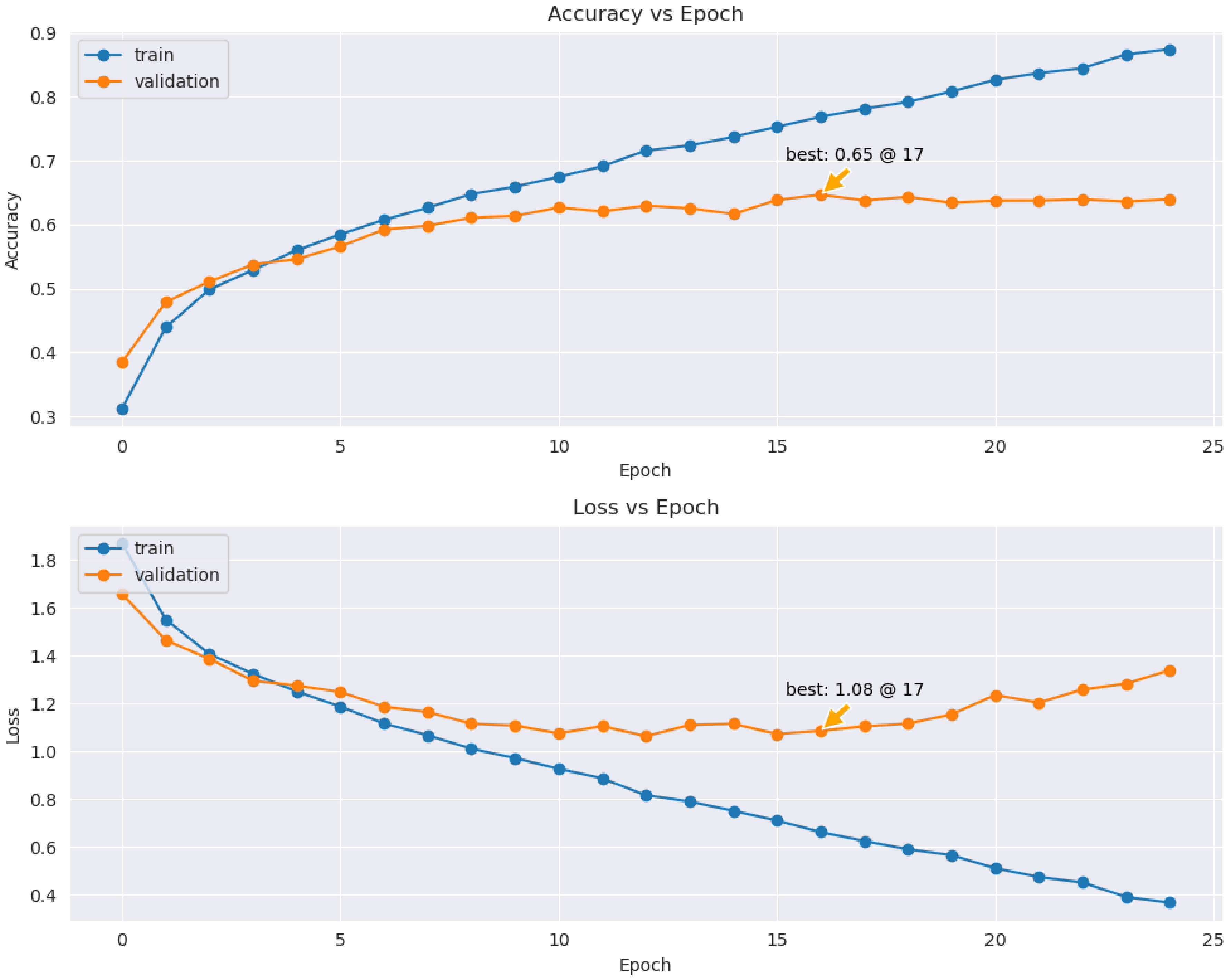
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Convolutional Kernel Shapes

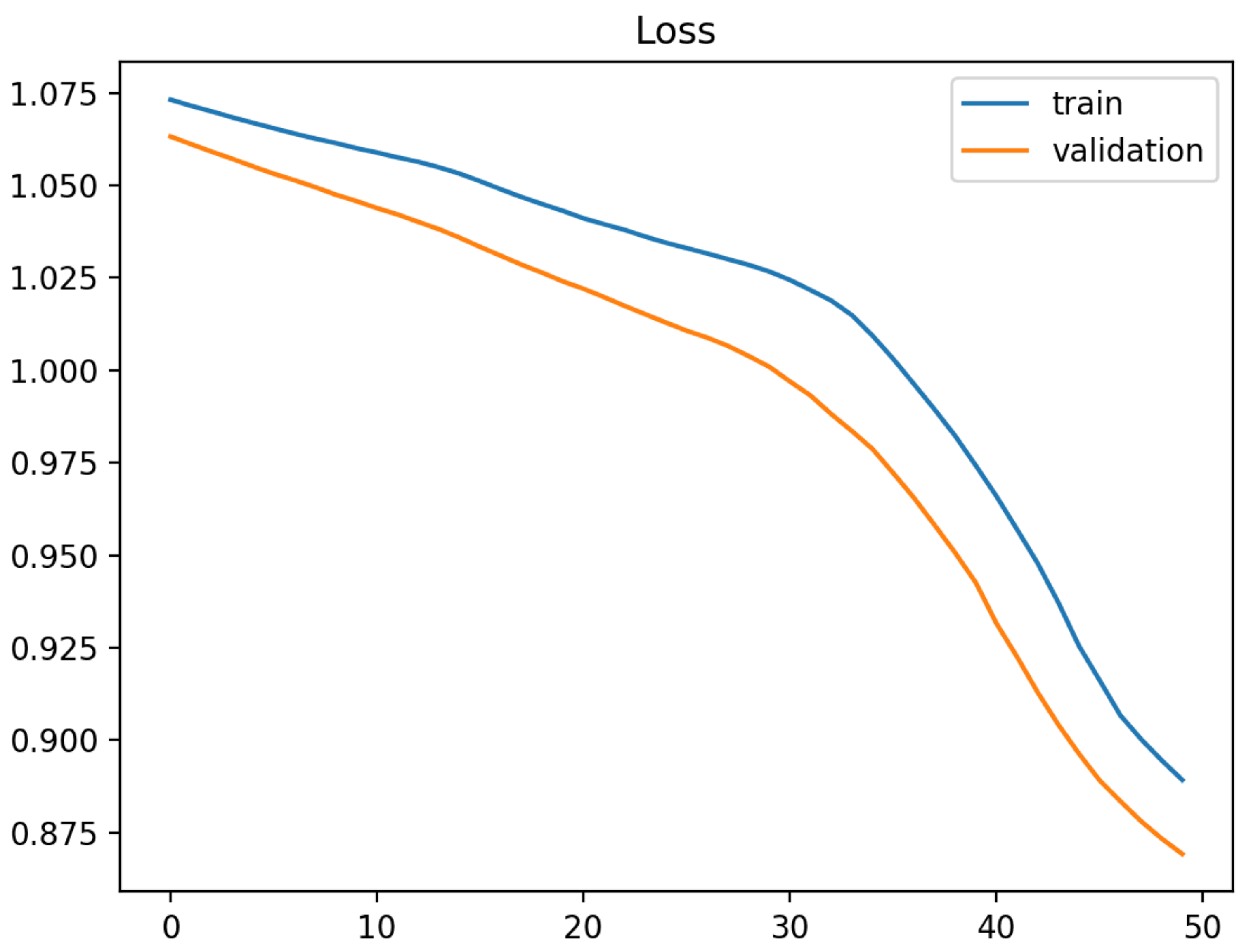
?	?	?
?	?	?
?	?	?

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

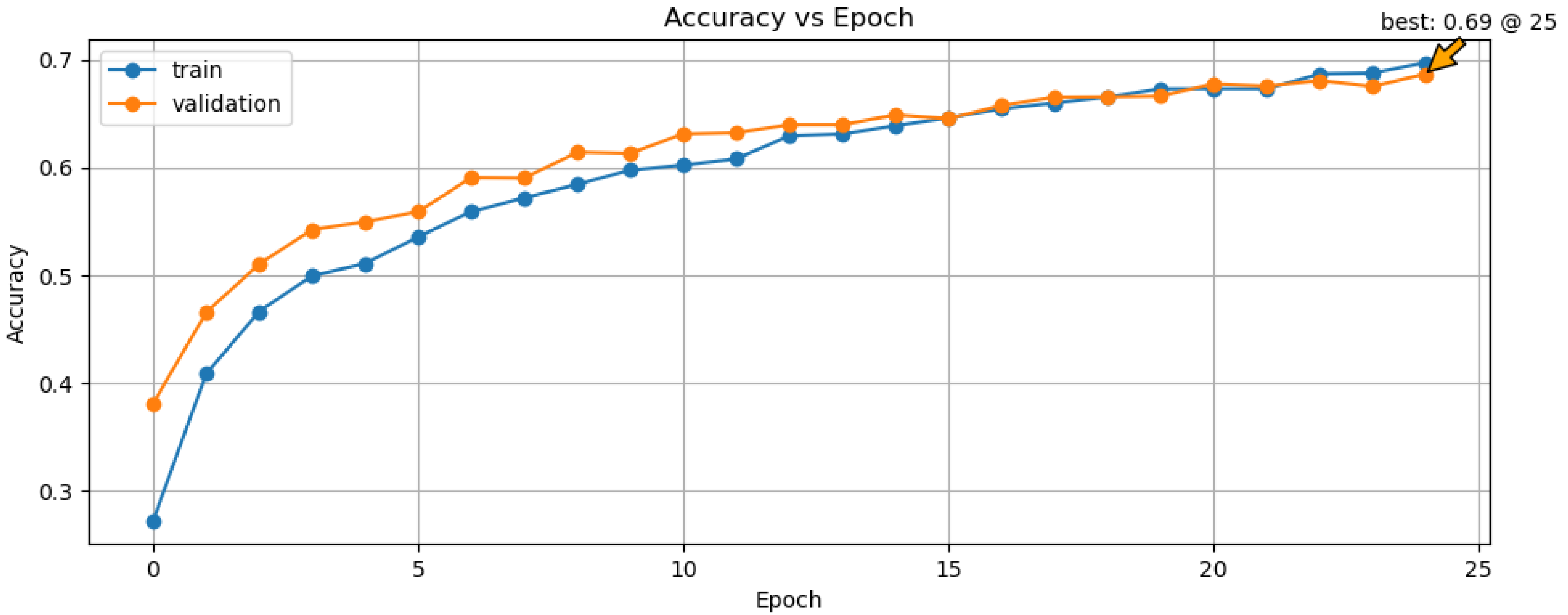
Underfitting vs Overfitting

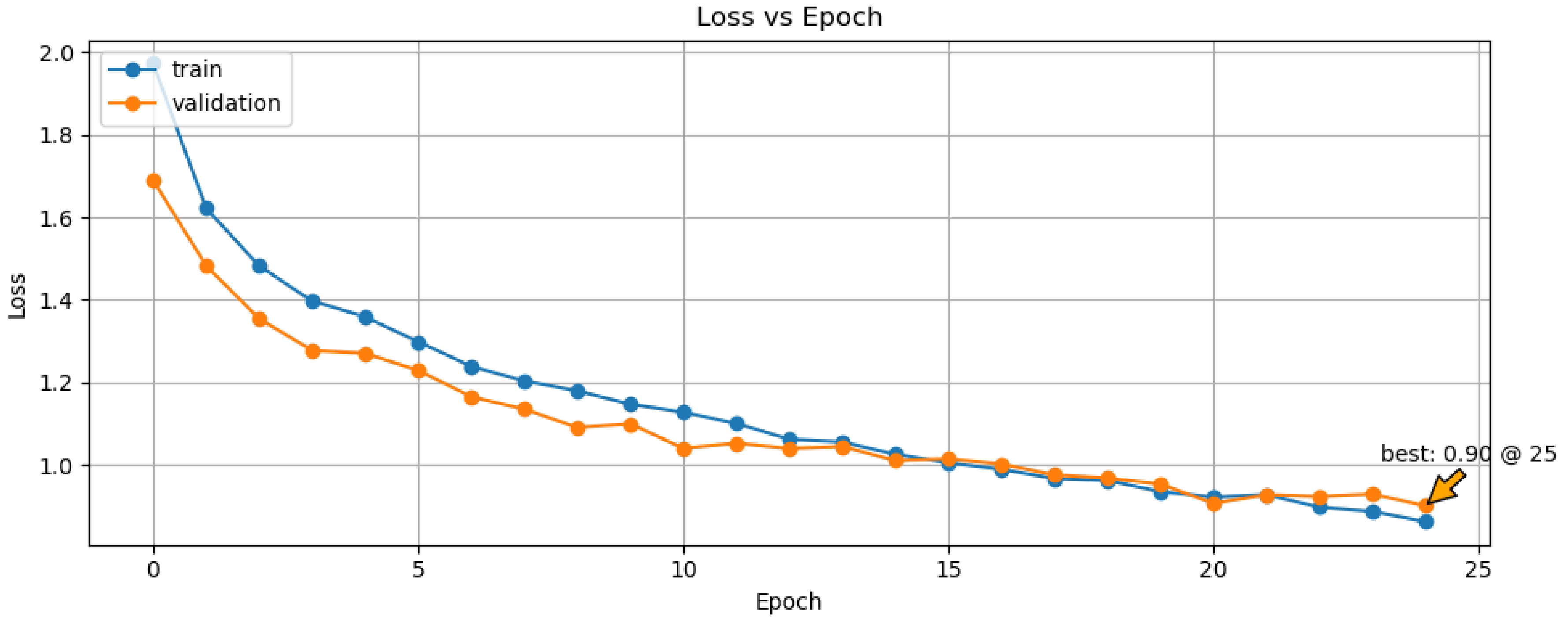


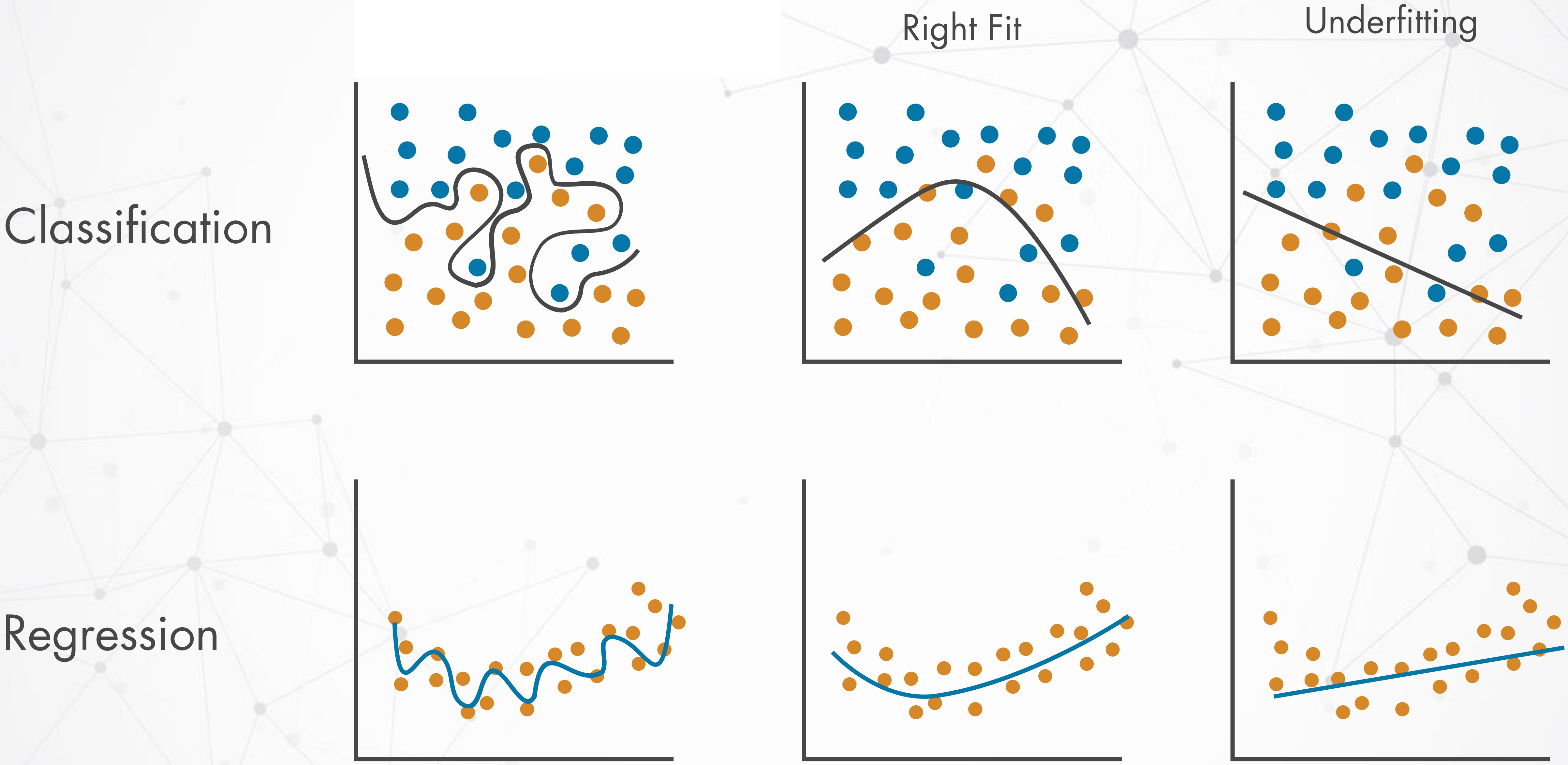
Underfitting vs Overfitting



Underfitting vs Overfitting







Link

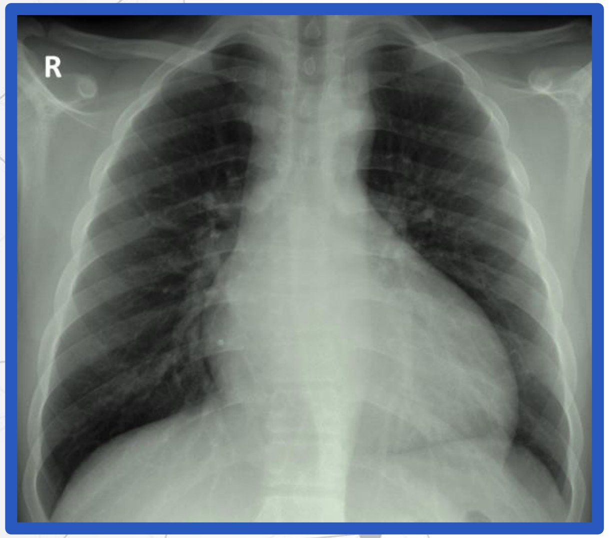
How to address Underfitting?

- Increase Model Depth
- Increase Training Epoch/ Increase Learning Rate
- Reduce Data

How to address Overfitting?

- Increase Training Data
  - Use augmentation
- Reduce Model Depth / Change Architecture
- Use Regularization / Dropout

Covid Classification

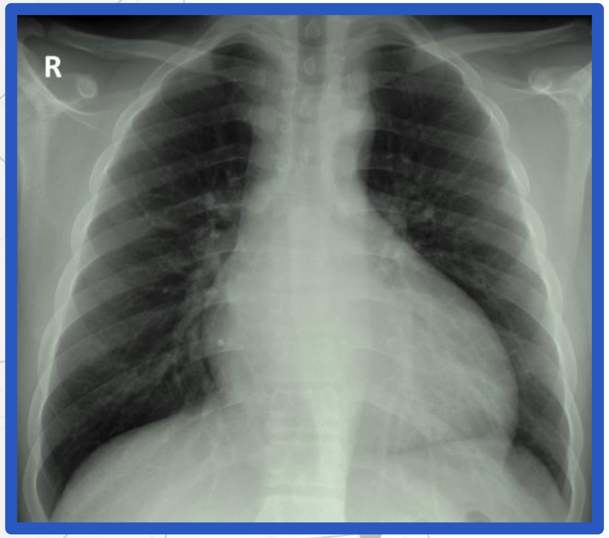


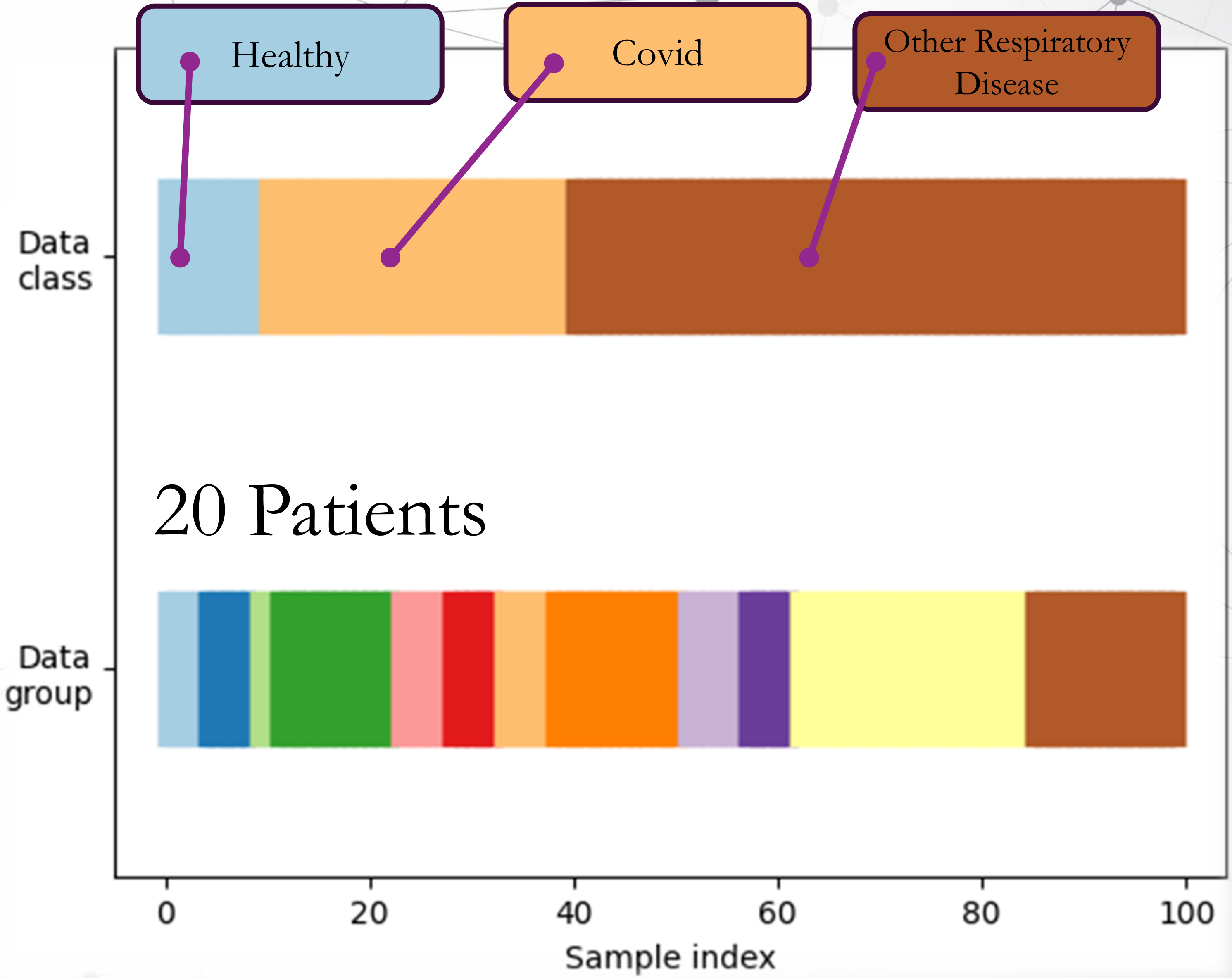
Healthy

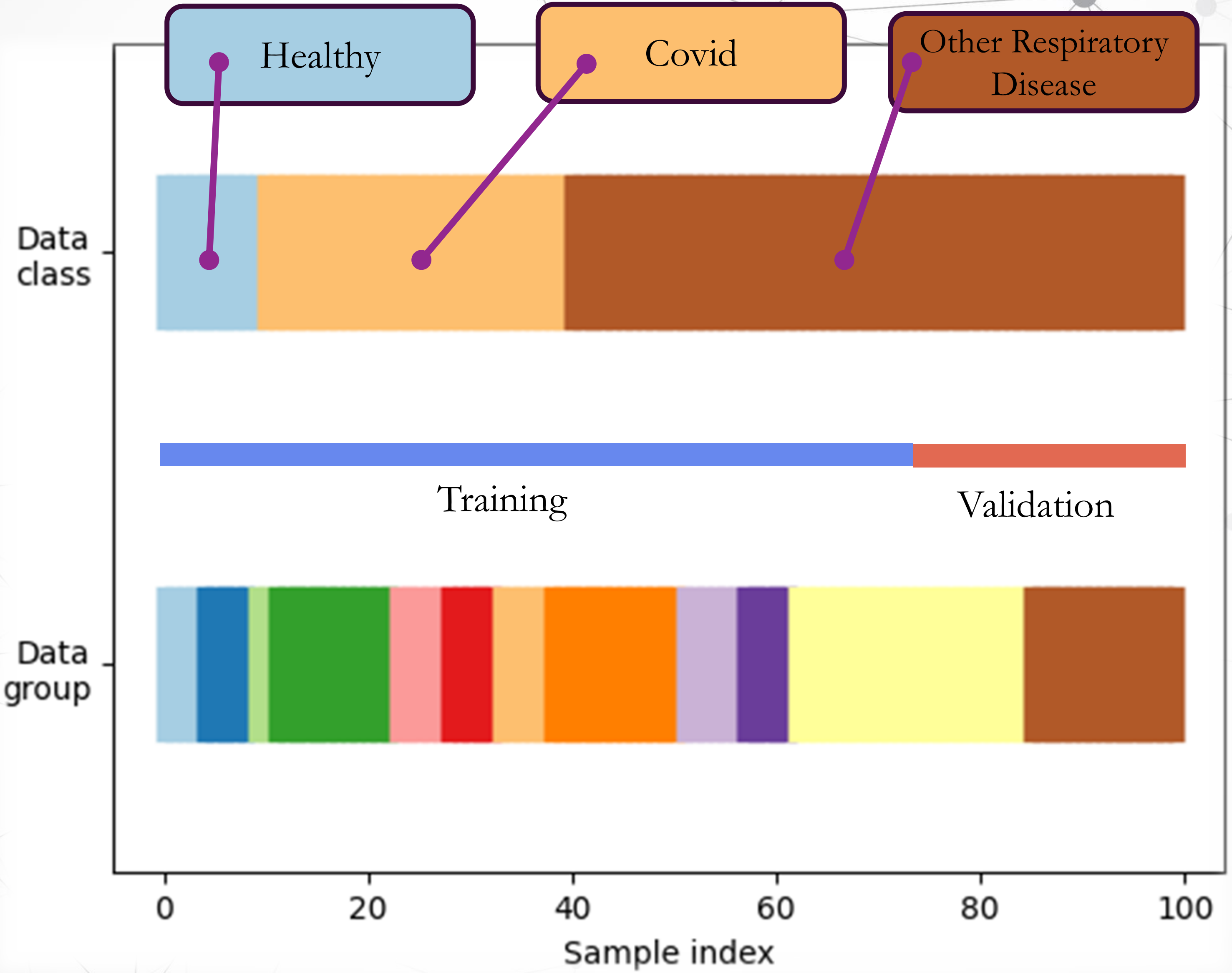
Covid

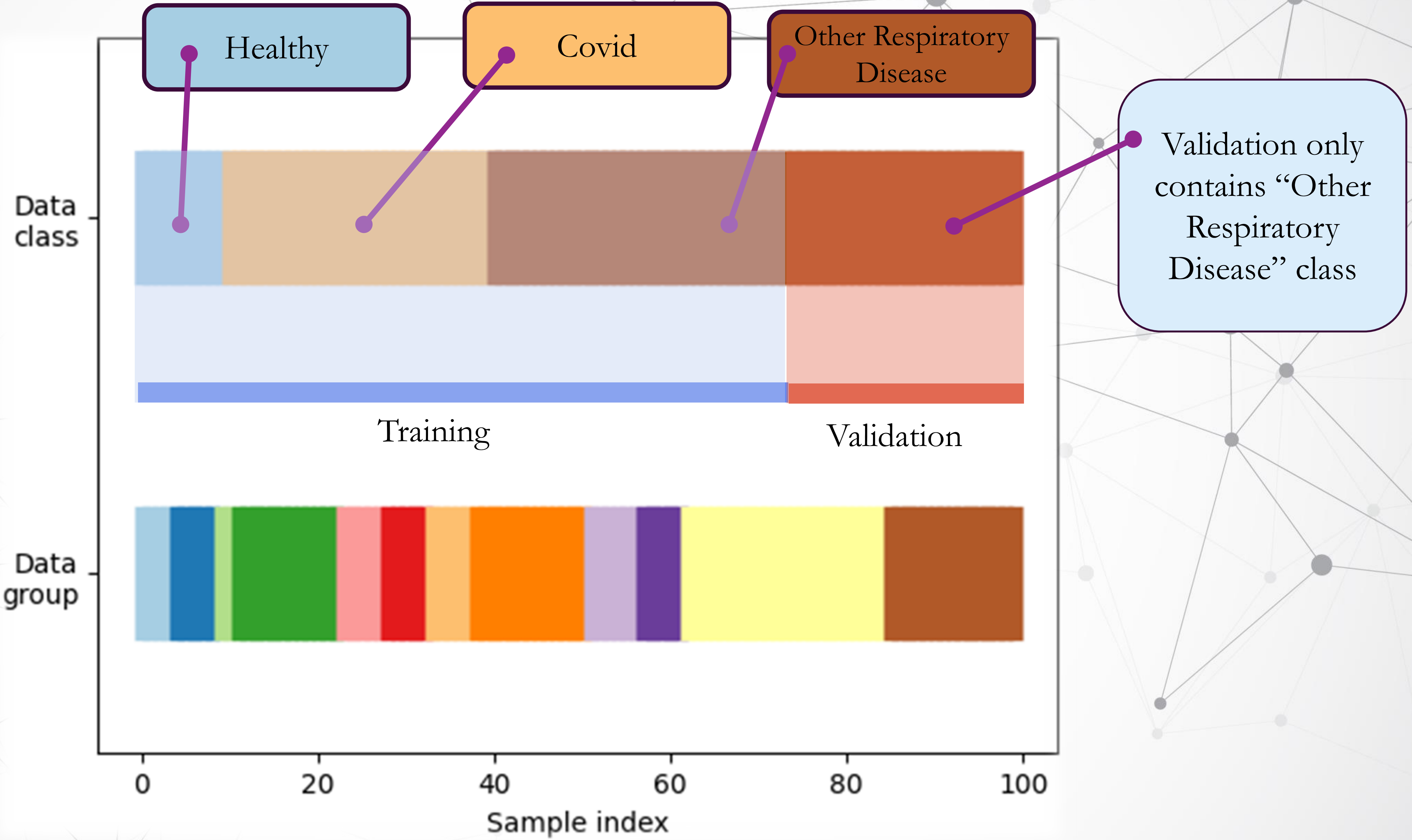
Other Respiratory Disease

20 Patients

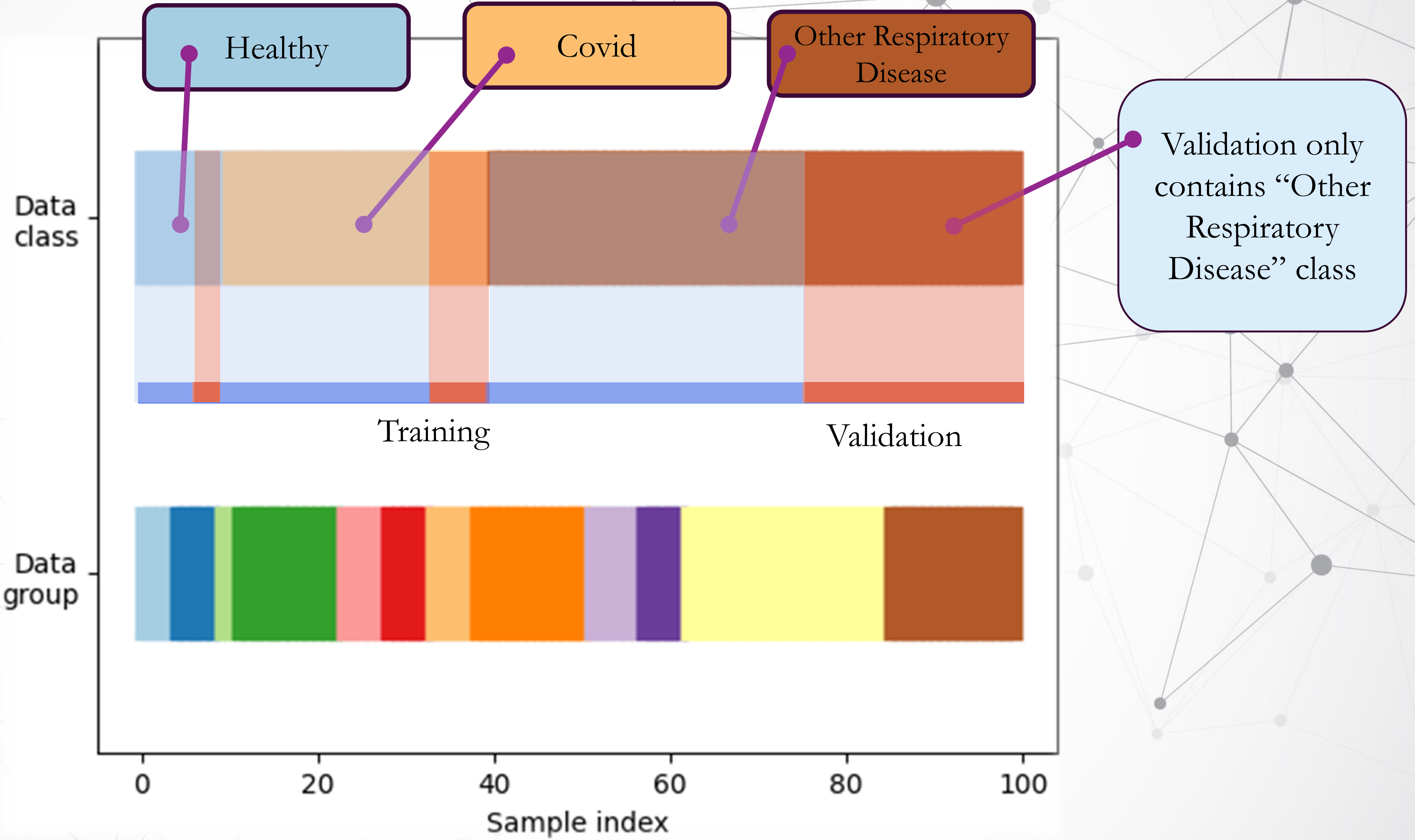

R

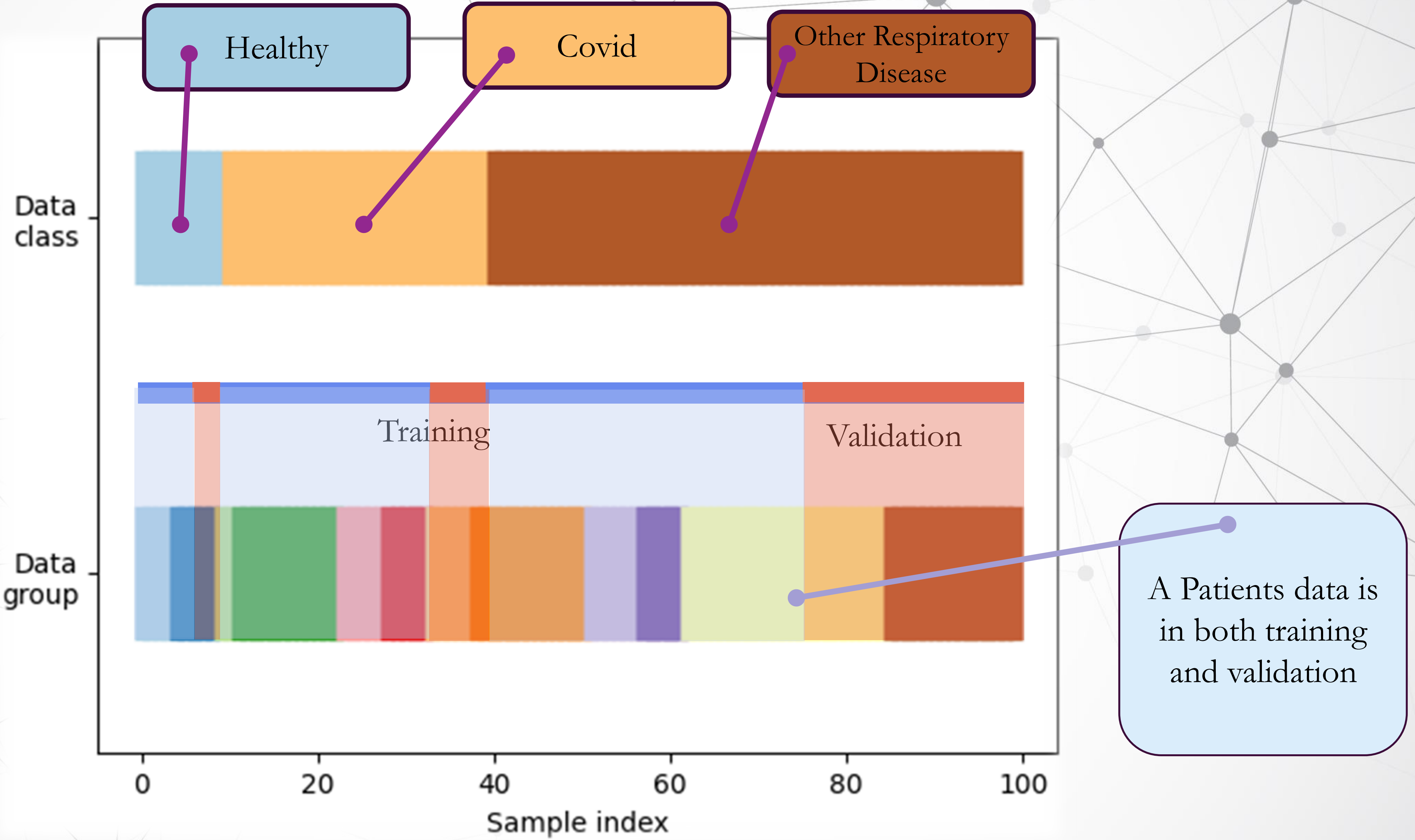

Healthy
Covid
Other Respiratory Disease
Data class
20 Patients
Data group
0
20
40
60
80
100
Sample index

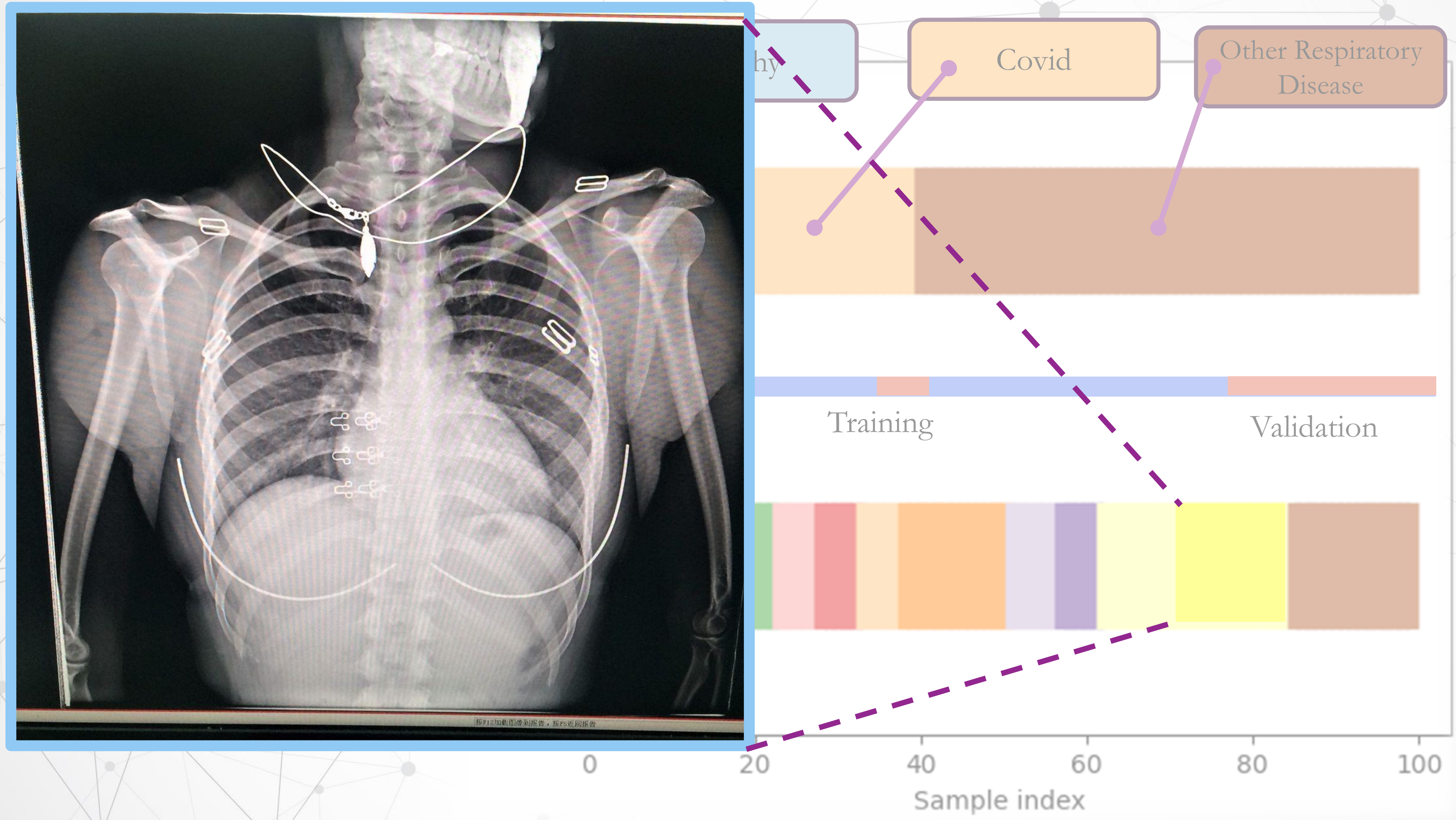

Healthy
Covid
Other Respiratory Disease
Data class
Training
Validation
Data group
0
20
40
60
80
100
Sample index



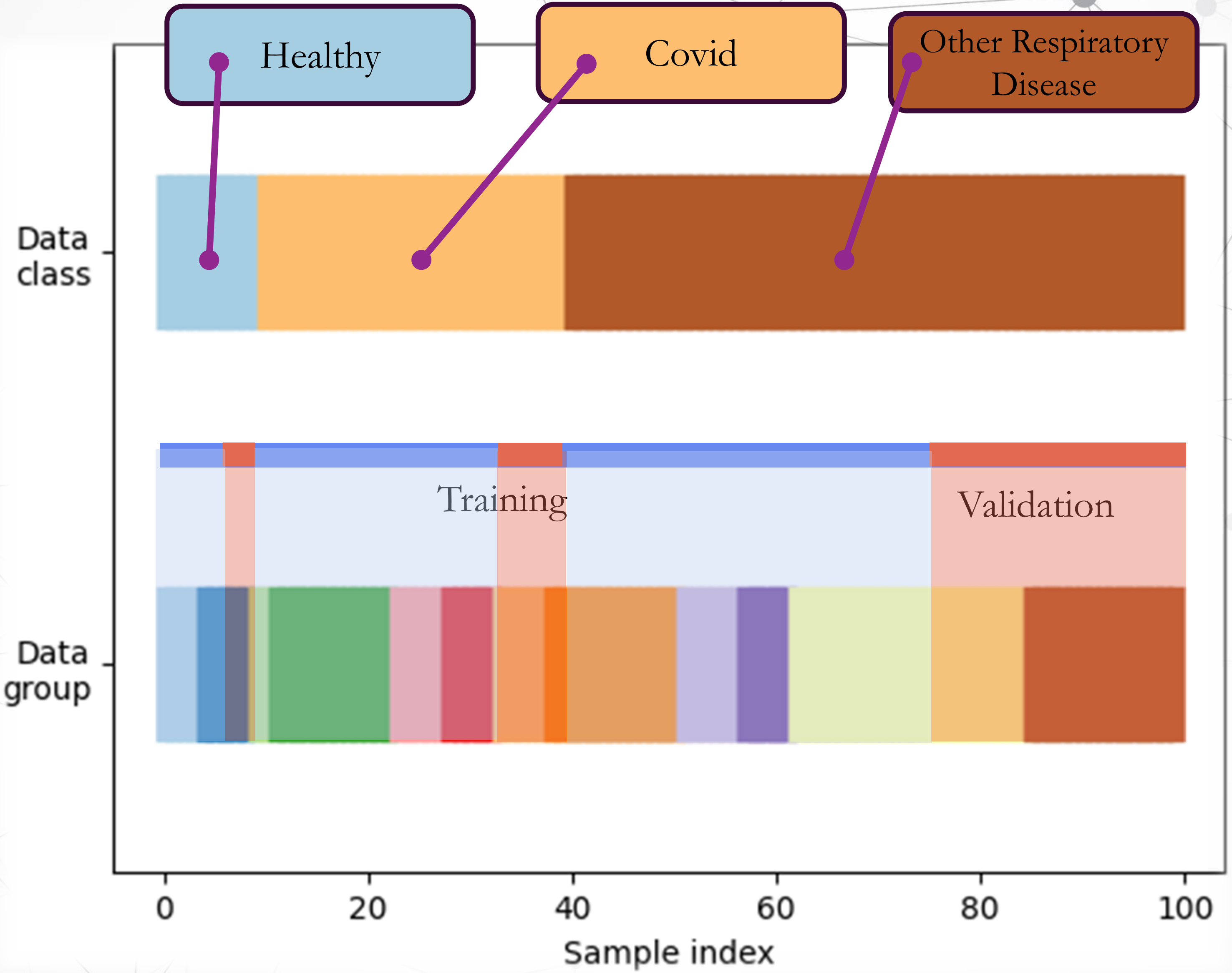
Healthy
Covid
Other Respiratory Disease
Validation only contains "Other Respiratory Disease" class
Data class
Data group
Training
Validation
0
20
40
60
80
100
Sample index

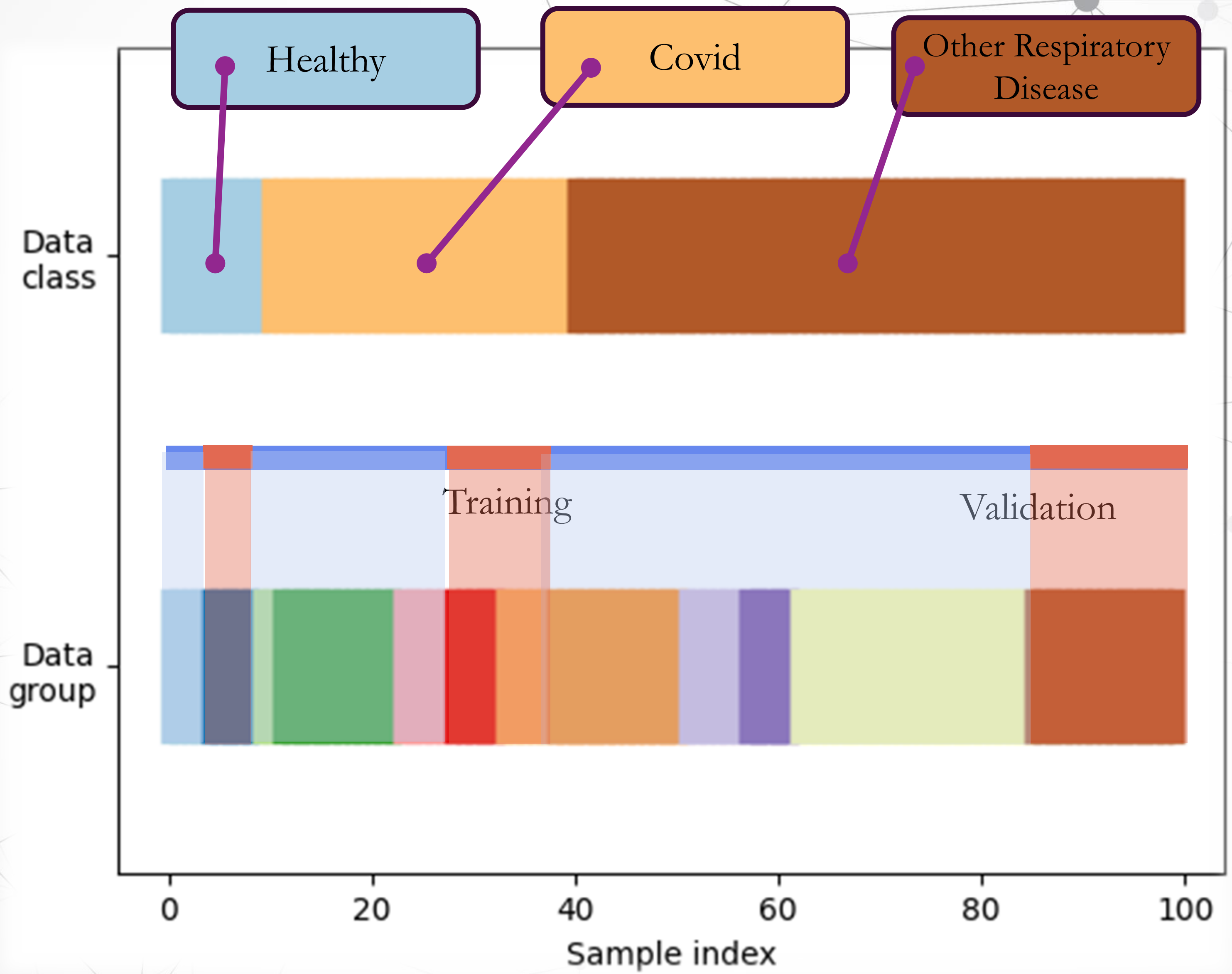

Healthy
Covid
Other Respiratory Disease
Validation only contains "Other Respiratory Disease" class
Data class
Training
Validation
Data group
0
20
40
60
80
100
Sample index

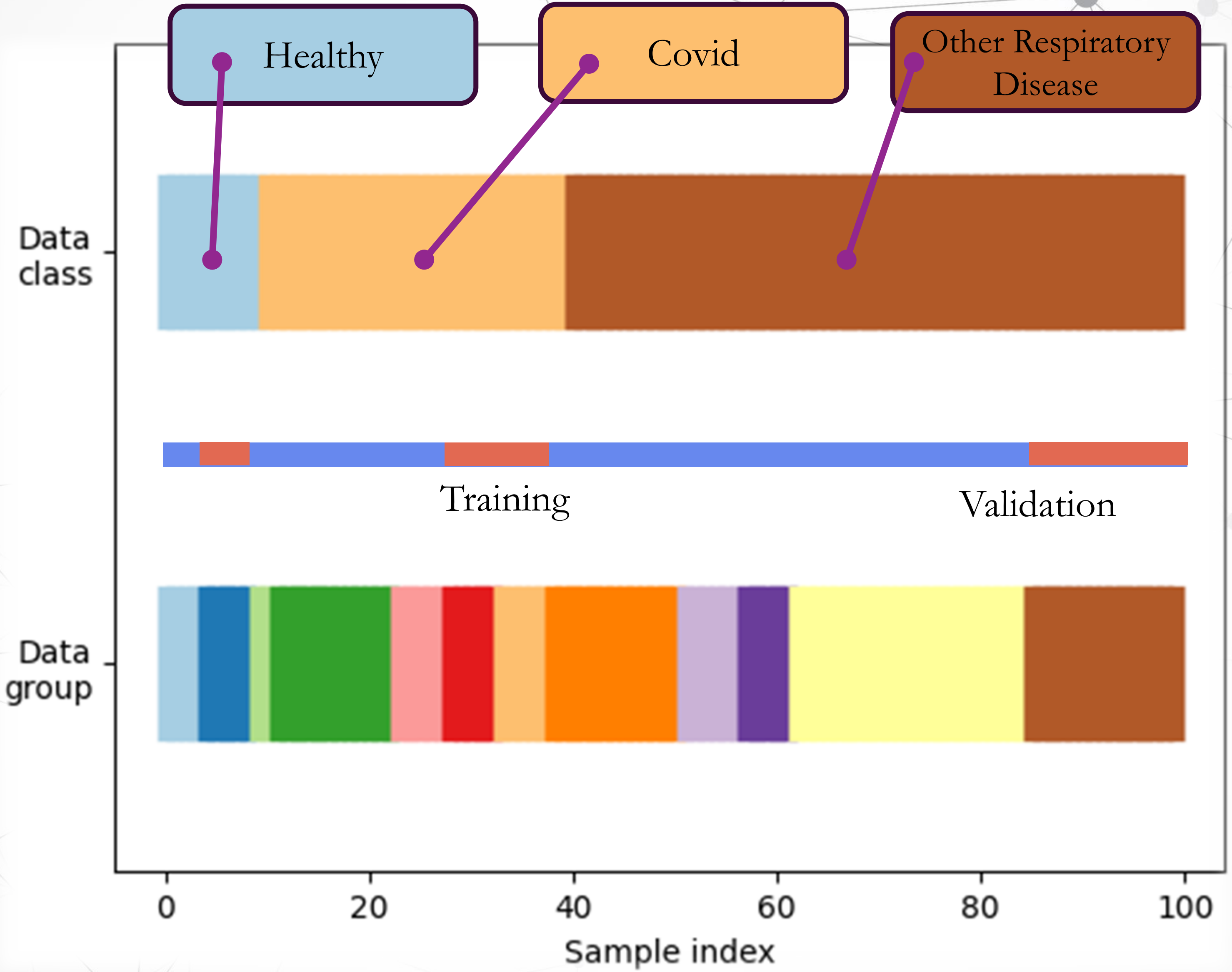

Healthy
Covid
Other Respiratory Disease
Data class
Training
Validation
Data group
A Patients data is in both training and validation
0
20
40
60
80
100
Sample index



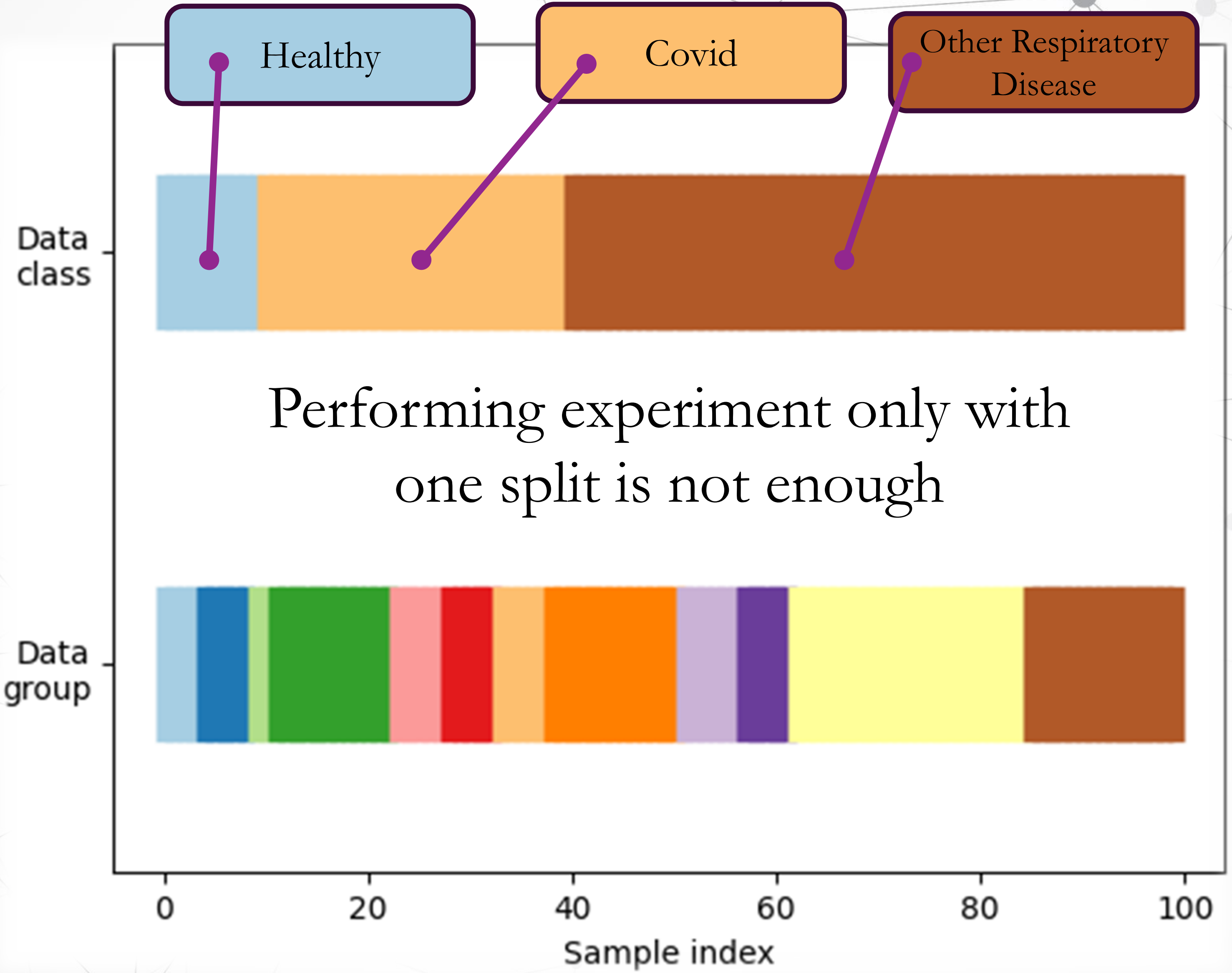
hy
Covid
Other Respiratory Disease
Training
Validation
0
20
40
60
80
100
Sample index

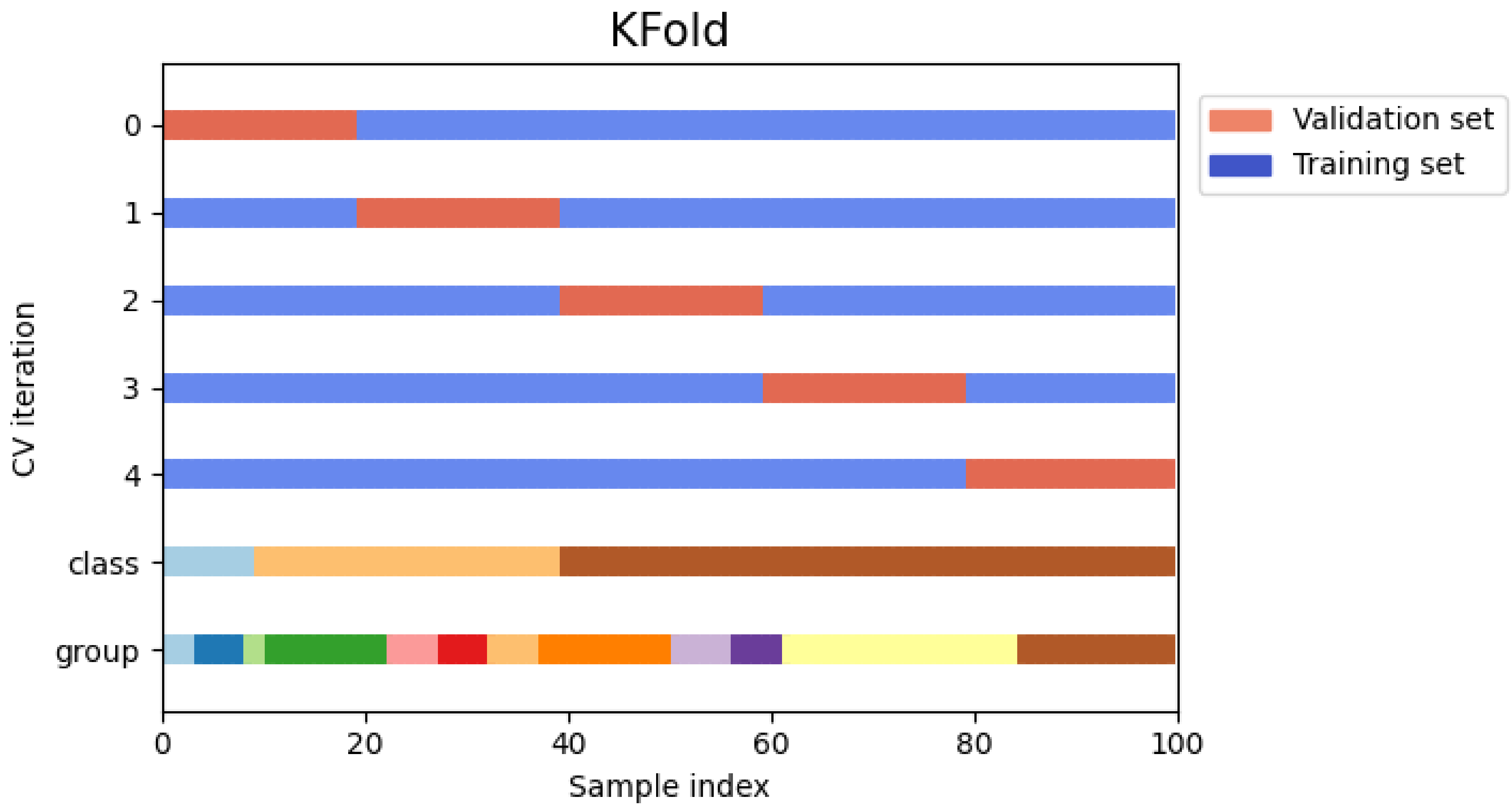

Healthy
Covid
Other Respiratory Disease
Data class
Data group
Training
Validation
0
20
40
60
80
100
Sample index


Healthy
Covid
Other Respiratory Disease
Data class
Training
Validation
Data group
0
20
40
60
80
100
Sample index

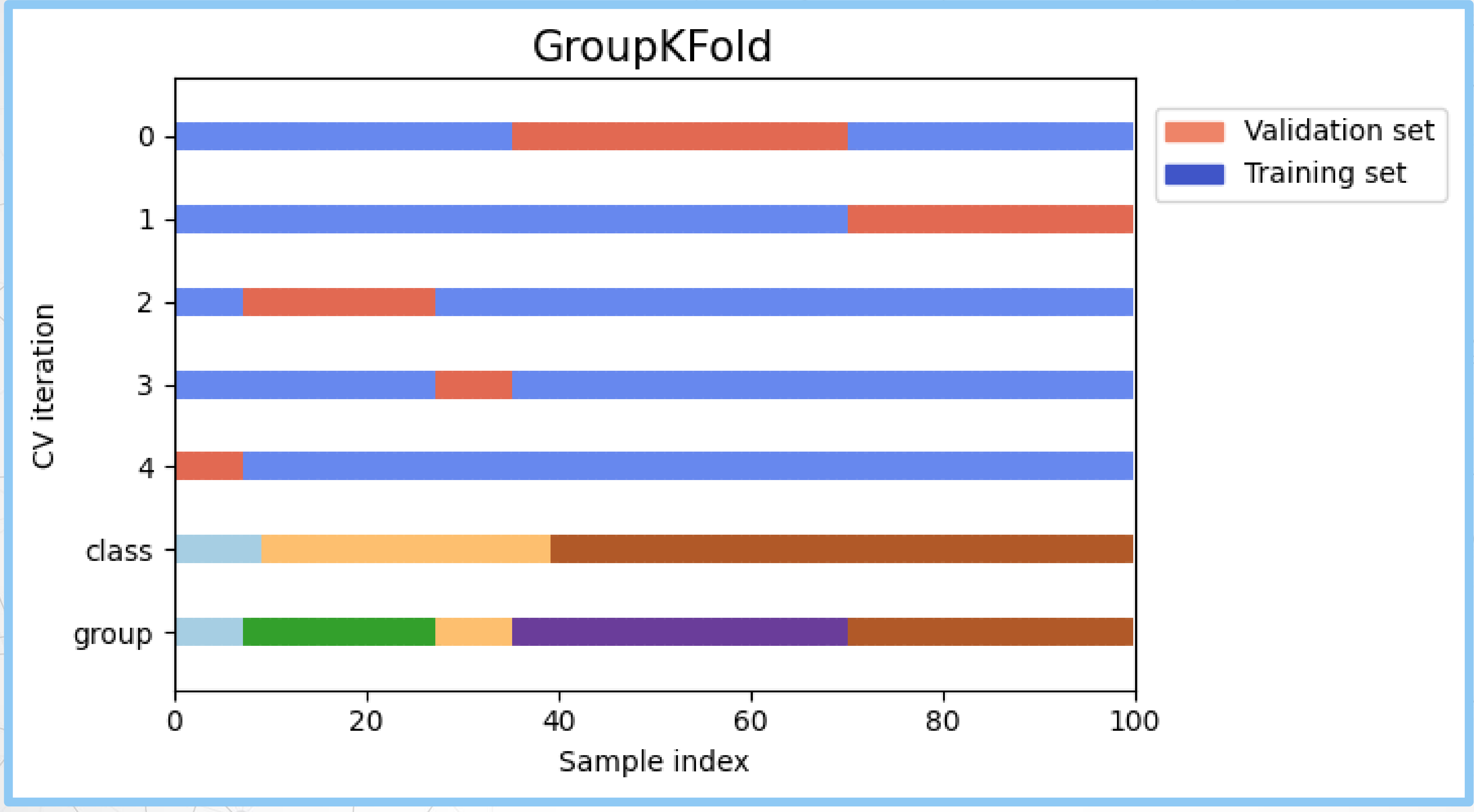


Healthy
Covid
Other Respiratory Disease
Data class
Training
Validation
Data group
0
20
40
60
80
100
Sample index

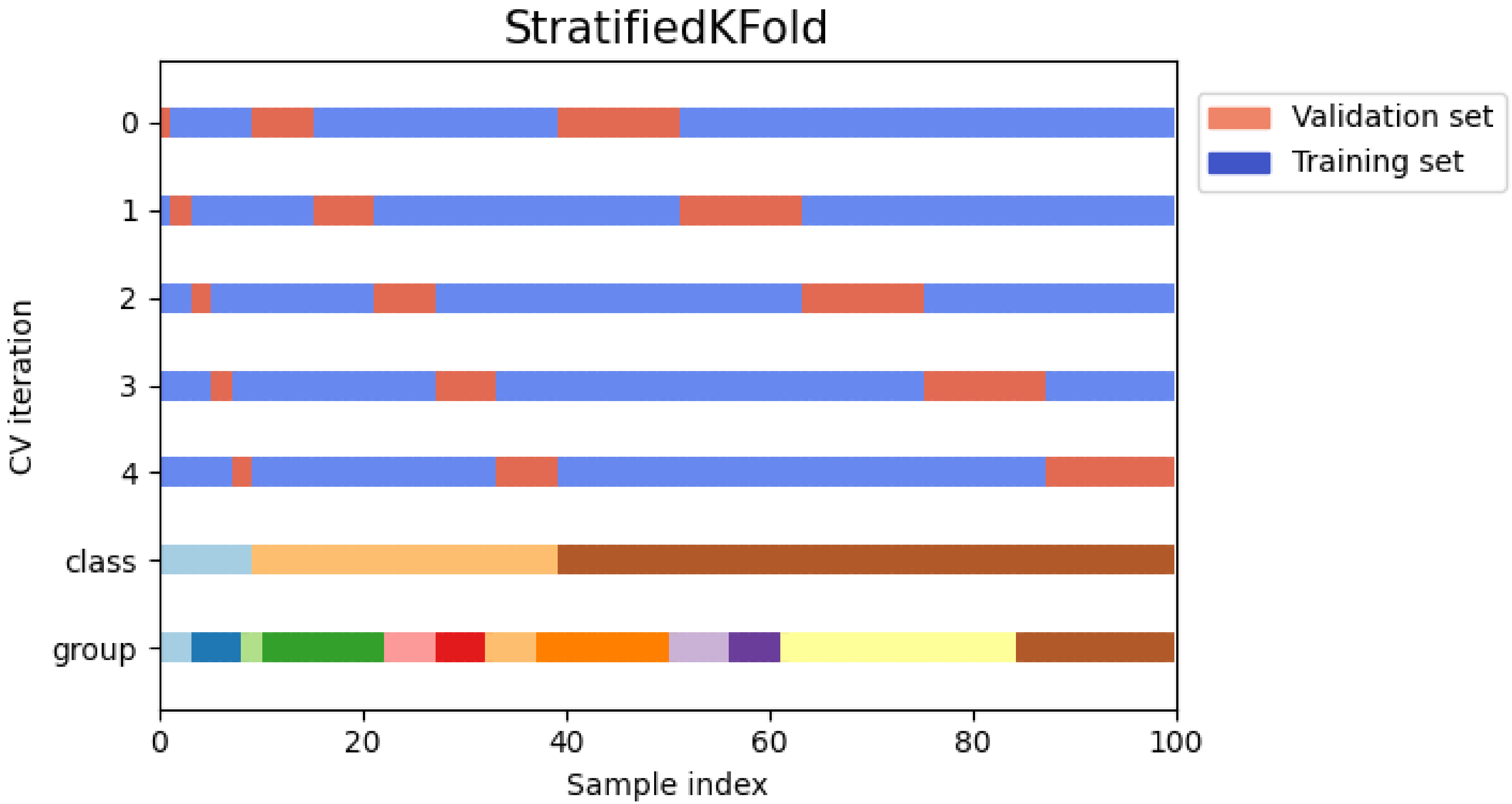

Healthy
Covid
Other Respiratory Disease
Data class
Performing experiment only with one split is not enough
Data group
0
20
40
60
80
100
Sample index

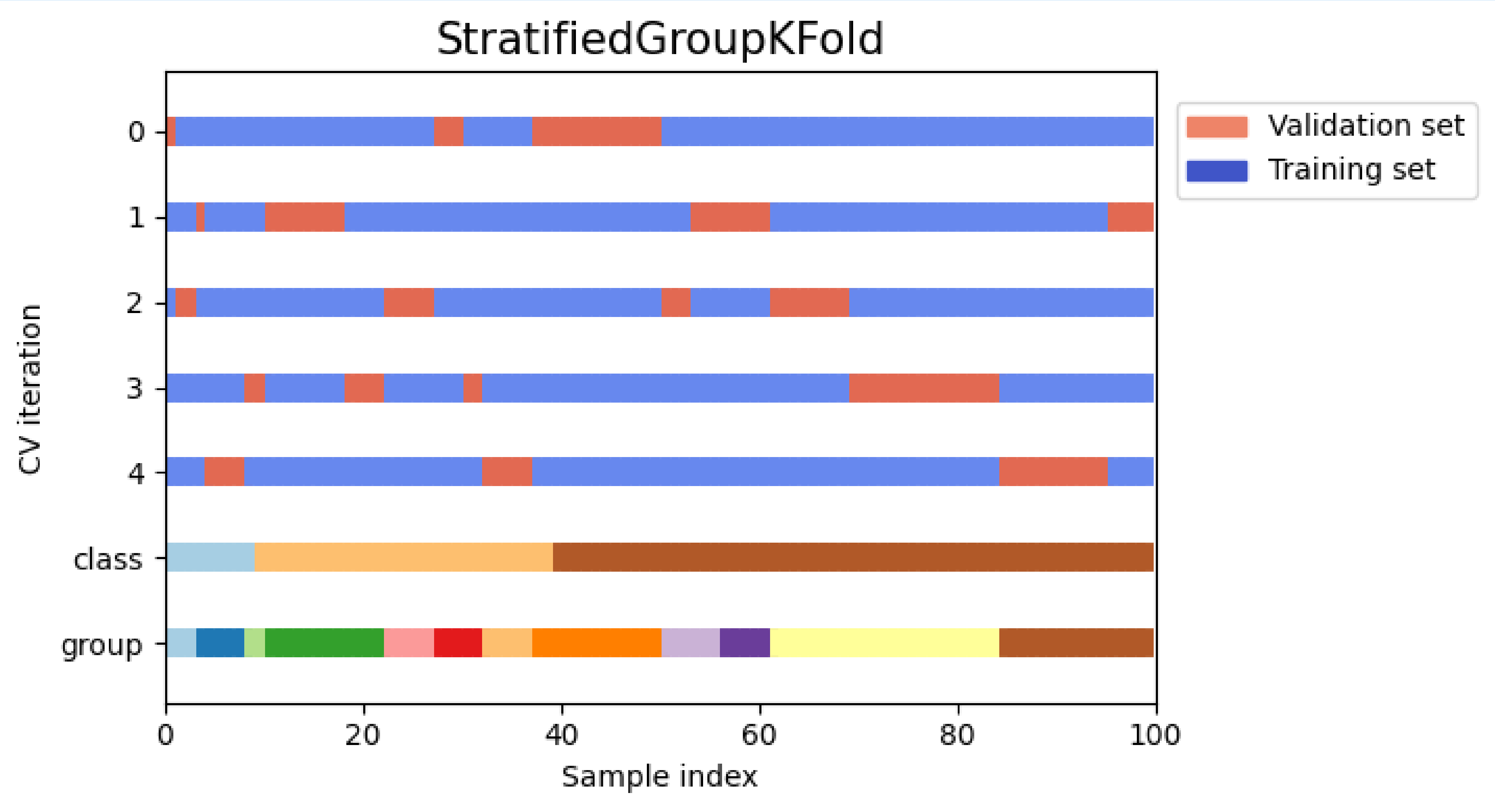


KFold
Validation set
Training set
CV iteration
0
1
2
3
4
class
group
0
20
40
60
80
100
Sample index



*A different distribution of groups compared to the datasets is used for easier visualization


StratifiedKFold
Validation set
Training set
CV iteration
0
1
2
3
4
class
group
0
20
40
60
80
100
Sample index


StratifiedGroupKFold
Validation set
Training set
CV iteration
0
1
2
3
4
class
group
0
20
40
60
80
100
Sample index

Up Next: Regularization

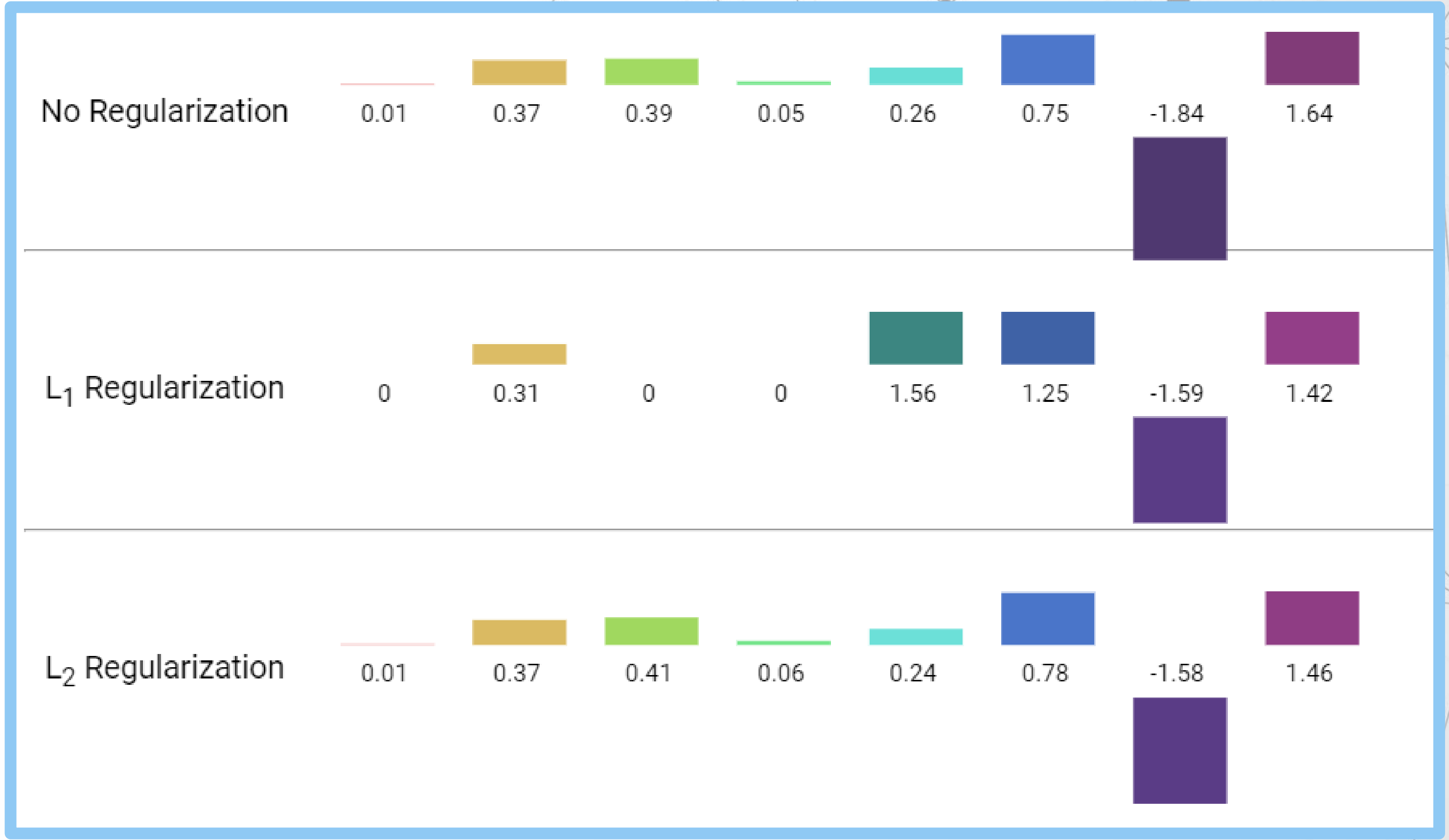
Regularization

$$\mathcal{L}_1 = \sum_{i=1}^{N} |w_i|$$

$$Total\ Cost\ Function = Loss + \mathcal{L}_1$$

$$\mathcal{L}_1 = \sum_{i=1}^{N} |w_i|^2$$

$$Total\ Cost\ Function = Loss + \mathcal{L}_1$$



Optimizers

```
tf.keras.optimizers.SGD(learning_rate = 0.01)
```

```
tf.keras.optimizers.SGD(learning_rate = 0.01,
                        momentum=0.9)
```

```
tf.keras.optimizers.RMSprop(learning_rate = 0.01)
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

Optimizers

```
tf.keras.optimizers.SGD(learning_rate = 0.01)
```

Calculate Prediction

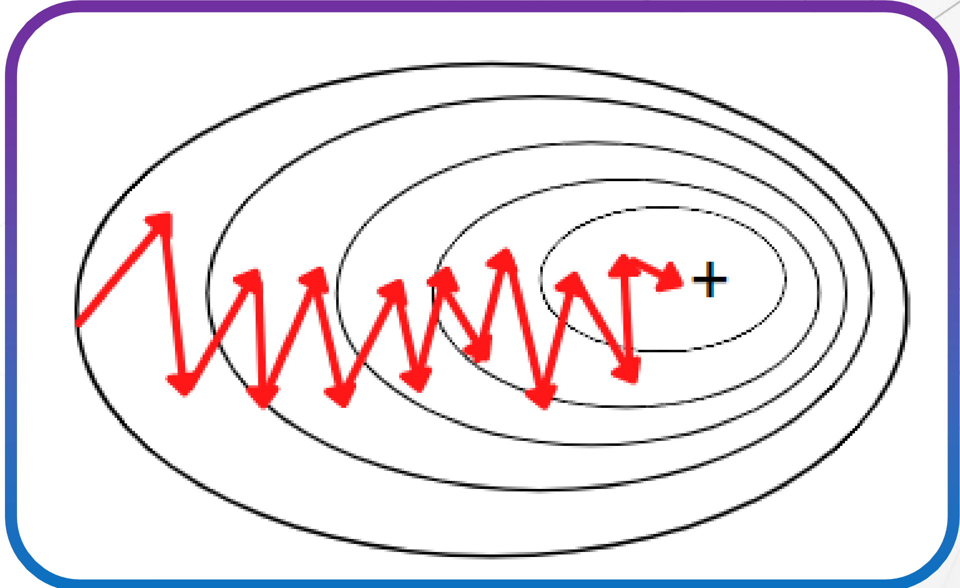
$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$

Estimate Error

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$$w = w - \alpha \frac{\partial J}{\partial w} \qquad b = b - \alpha \frac{\partial J}{\partial b}$$



Optimizers

```
tf.keras.optimizers.SGD(learning_rate = 0.01,
                        momentum=0.9)
```

Calculate Prediction

$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$

Estimate Error

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$w = w - \alpha V_{dw} \quad b = b - \alpha V_{db}$

$V_{dw} = \beta V_{dw} + (1- \beta)\, V_{dw}$

$V_{db} = \beta V_{db} + (1- \beta)\, V_{db}$

Which amounts to $\frac{1}{1-\beta}$ *values*

Optimizers

```
tf.keras.optimizers.SGD(learning_rate = 0.01,
                        momentum=0.9)
```

Calculate Prediction

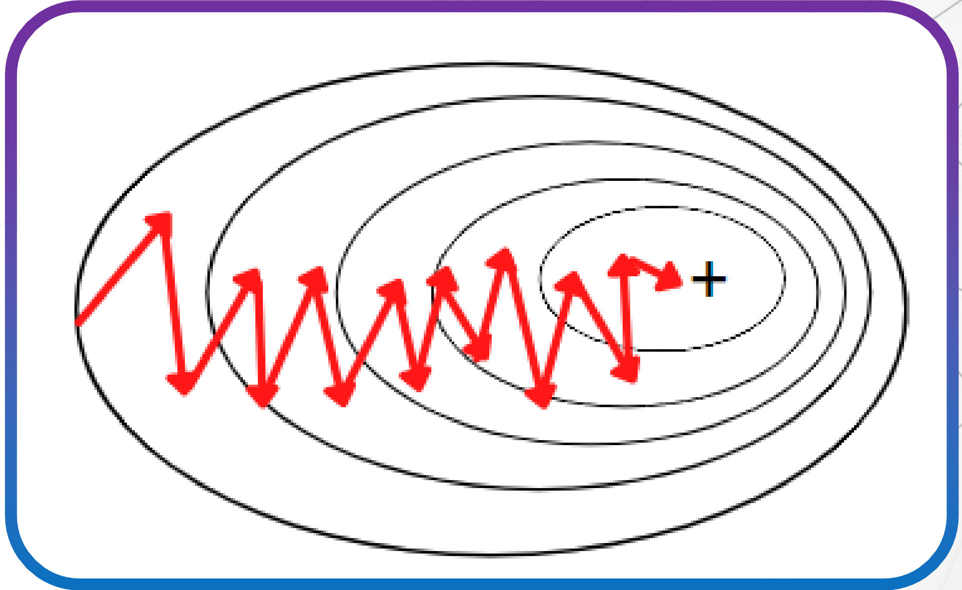
$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$

Estimate Error

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$w = w - \alpha V_{dw} \quad b = b - \alpha V_{db}$



Optimizers

```
tf.keras.optimizers.RMSprop(learning_rate = 0.01)
```

Calculate Prediction

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

Estimate Error

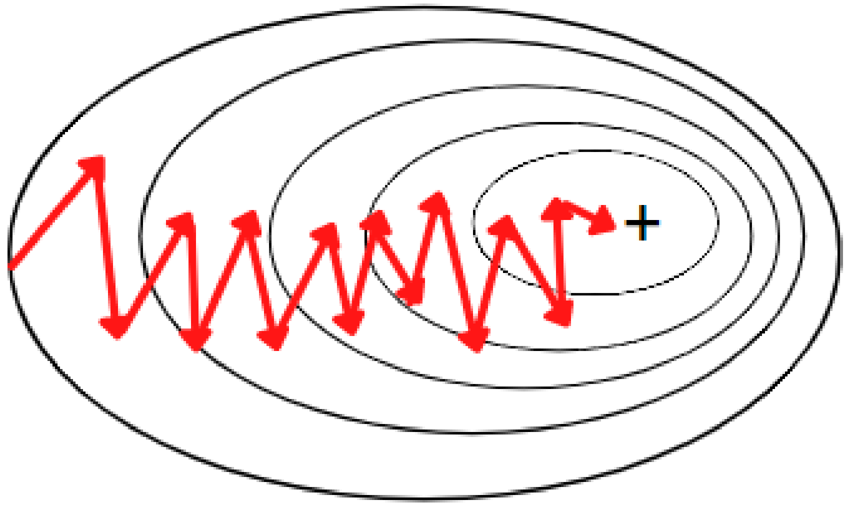
$$\boldsymbol{J}(\boldsymbol{w}, \boldsymbol{b}) = \frac{\boldsymbol{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{m}} (\hat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}_{\boldsymbol{actual}}})^{\boldsymbol{2}}$$

Update Parameters

$$\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{\alpha} \frac{\boldsymbol{dw}}{\sqrt{S_{dw}}} \quad \boldsymbol{b} = \boldsymbol{b} - \boldsymbol{\alpha} \frac{\boldsymbol{db}}{\sqrt{S_{db}}}$$

$$S_{dw} = \beta S_{dw} + (1- \beta)\, dw^2$$

$$S_{db} = \beta S_{db} + (1- \beta)\, db^2$$



Optimizers

```
tf.keras.optimizers.SGD(learning_rate = 0.01)
```

```
tf.keras.optimizers.SGD(learning_rate = 0.01,
                        momentum=0.9)
```

```
tf.keras.optimizers.RMSprop(learning_rate = 0.01)
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

Any Question?