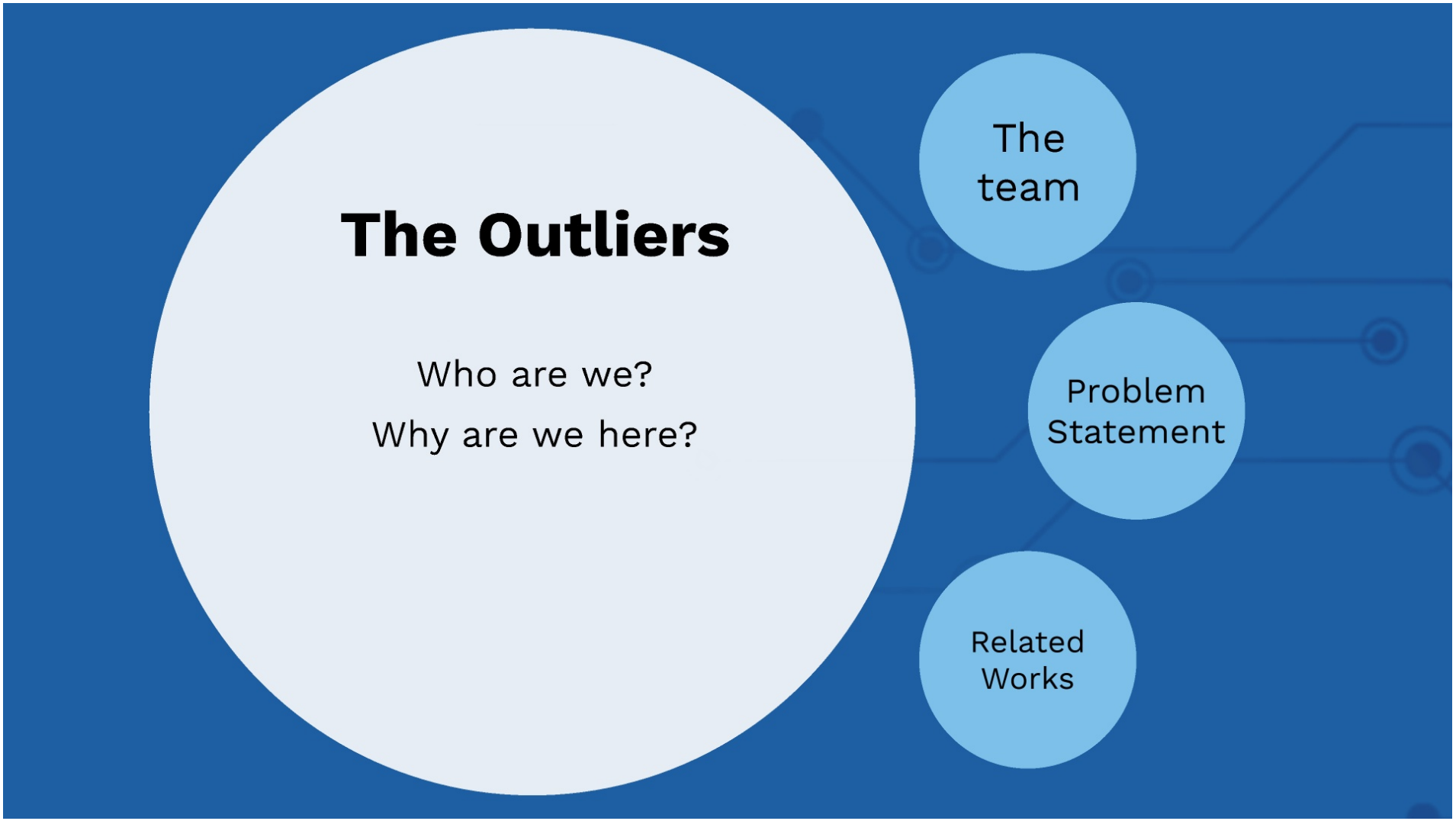
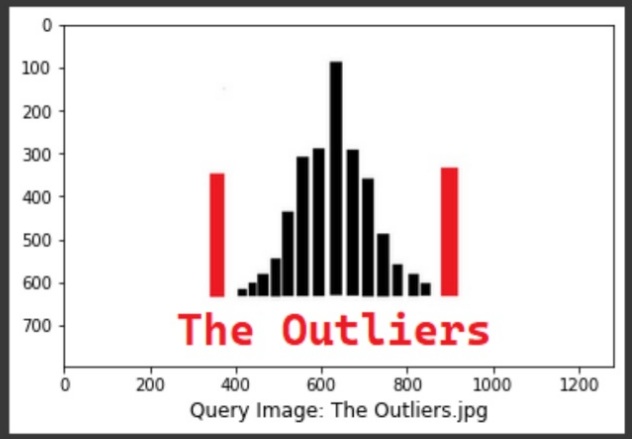
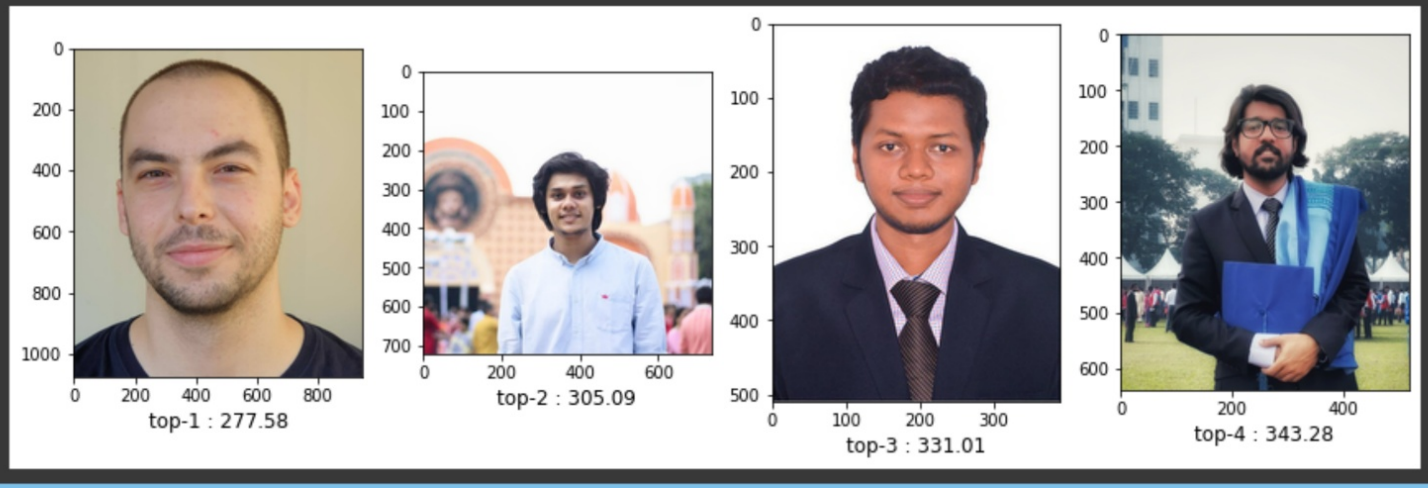

The Outliers
Who are we?
Why are we here?
The team
Problem Statement
Related Works

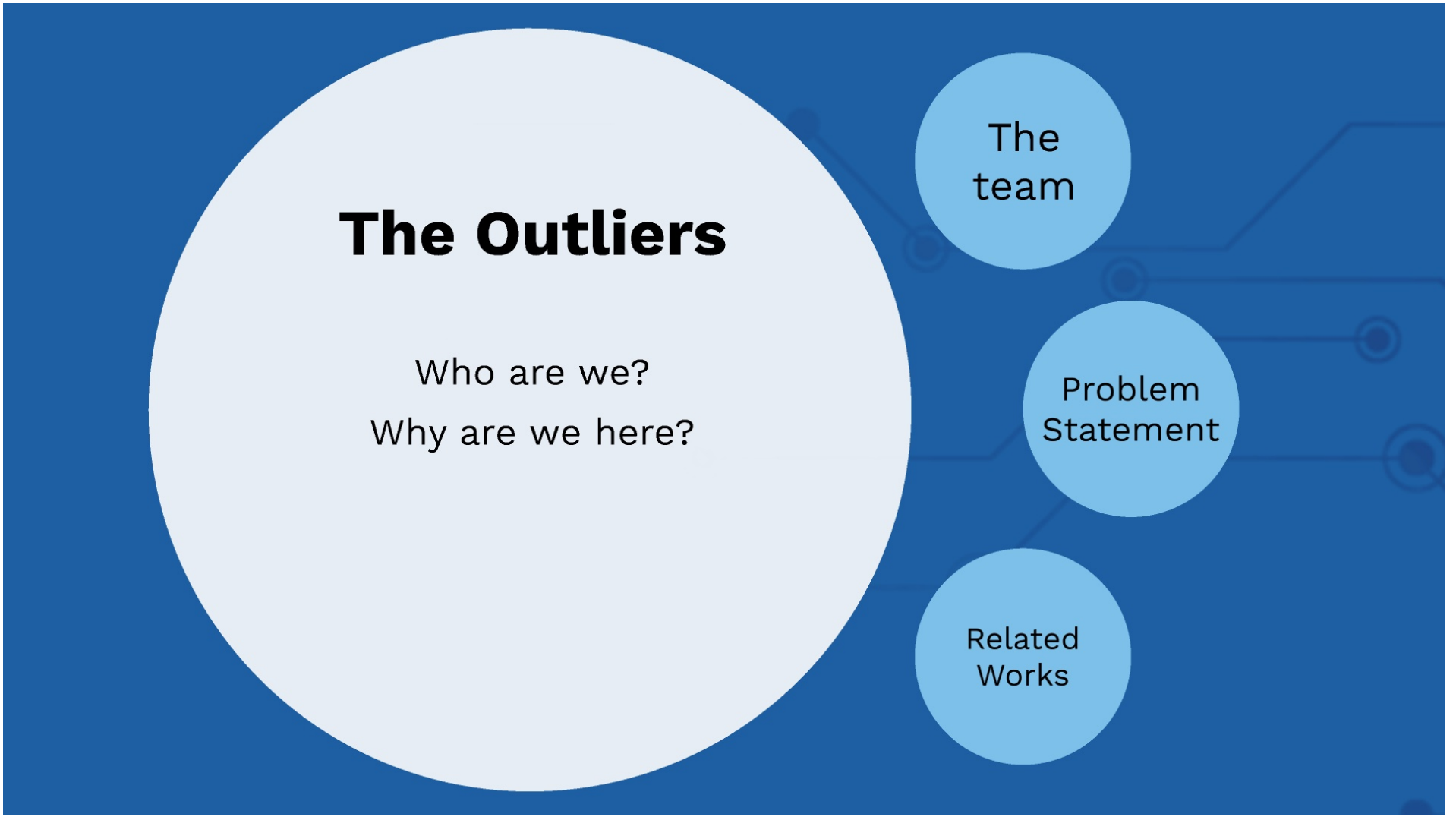
The Team

---------- Query Image --------



---------- Similar Images --------

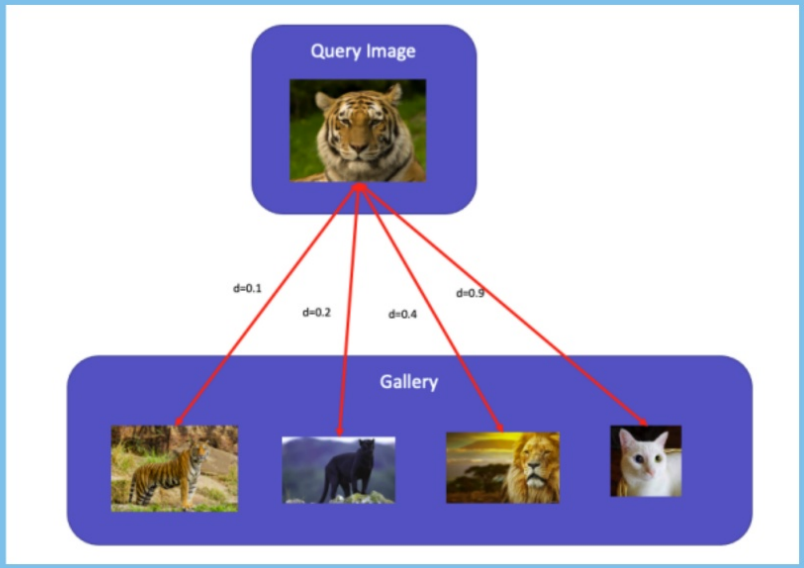


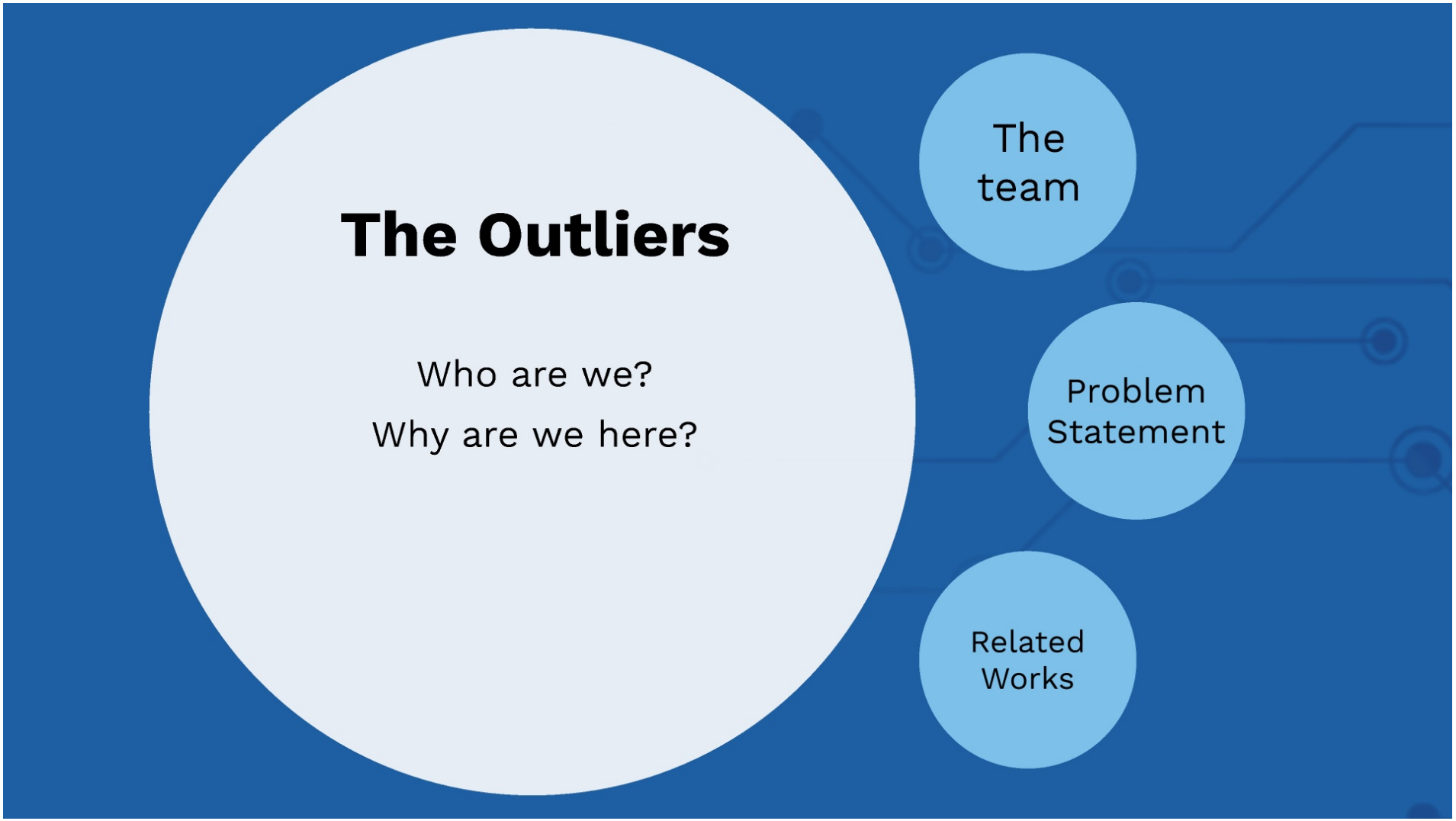

The Outliers
Who are we?
Why are we here?
The team
Problem Statement
Related Works

Problem Statement

*Create an image search engine where a **query** image is fed to a model that will return the most N similar images from a **gallery**.*

Given the input query image, the algorithm has to be capable of matching the input query image with another gallery image depicting the same animal.



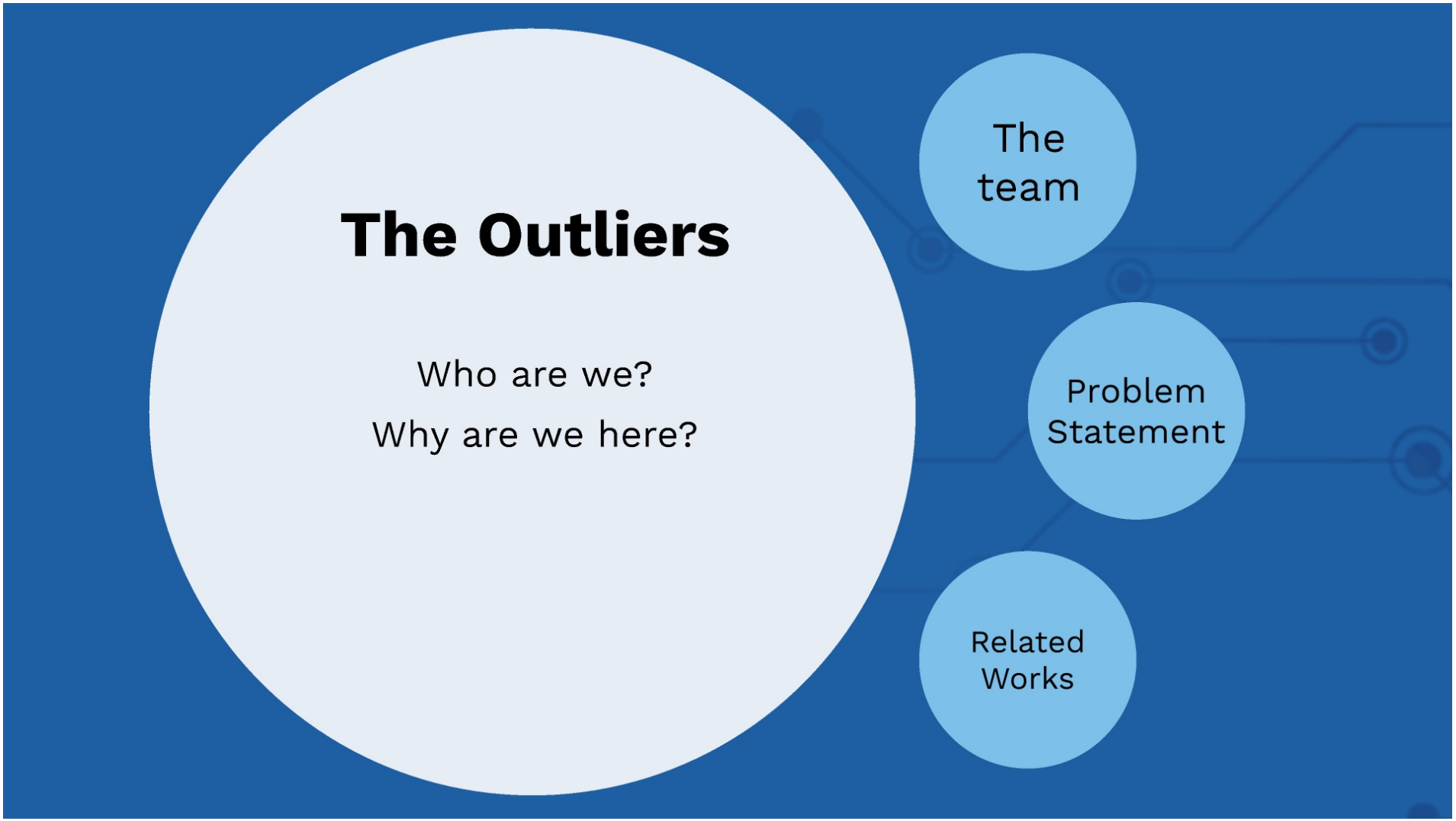

The Outliers
Who are we?
Why are we here?
The team
Problem Statement
Related Works

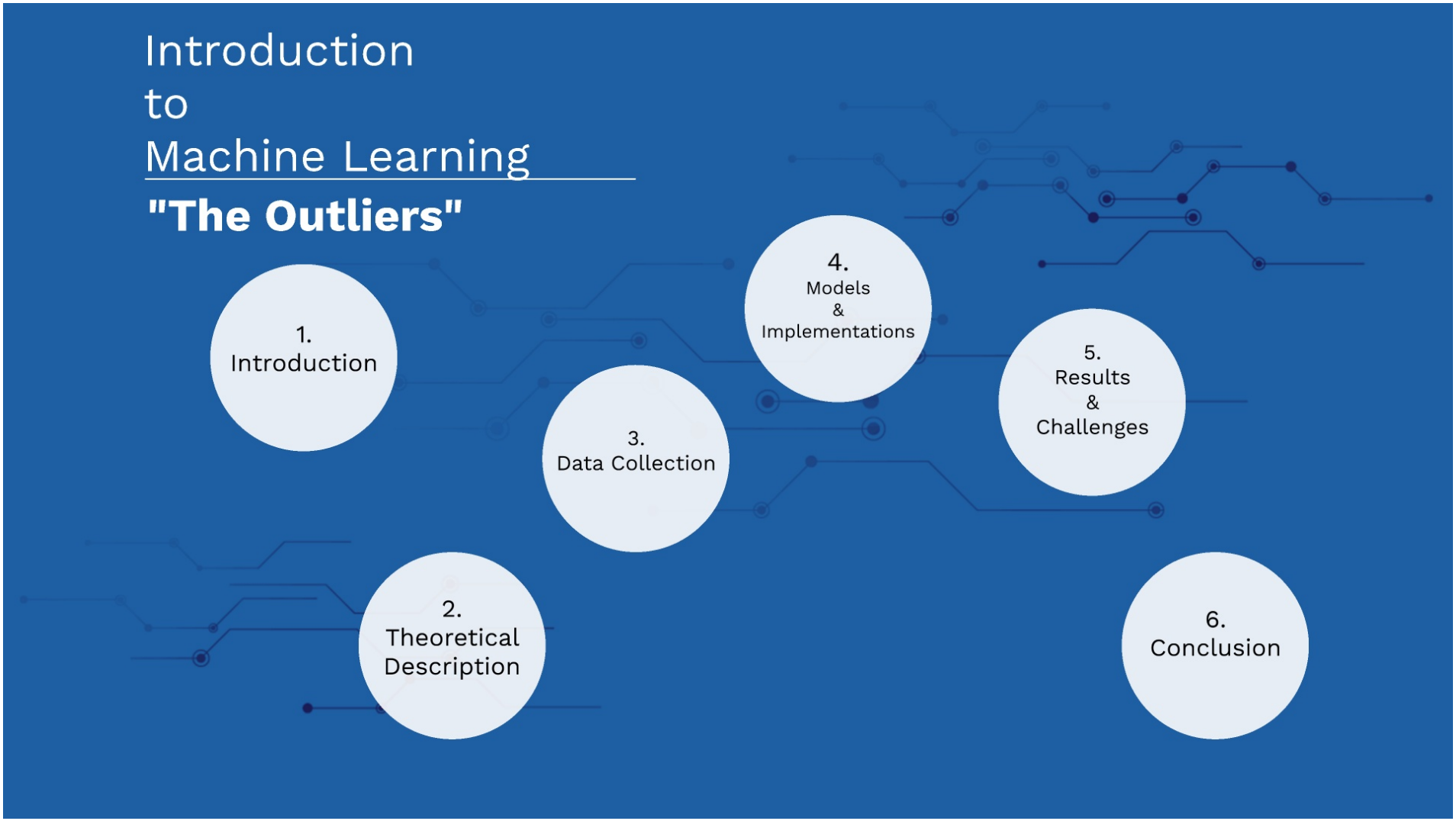
"Good artists copy, great artists steal"
Pablo Picasso

*Ramprasath, Muthukrishnan, M. Vijay Anand, and Shanmugasundaram Hariharan. "**Image classification using convolutional neural networks**." International Journal of Pure and Applied Mathematics 119.17 (2018): 1307-1319.*

*Chauhan, Rahul, Kamal Kumar Ghanshala, and R. C. Joshi. "**Convolutional neural network (CNN) for image detection and recognition**." 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC). IEEE, 2018.*

Hijazi, Samer, Rishi Kumar, and Chris Rowen. "**Using convolutional neural networks for image recognition**." Cadence Design Systems Inc.: San Jose, CA, USA 9 (2015).


The Outliers
Who are we?
Why are we here?
The team
Problem Statement
Related Works


Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion

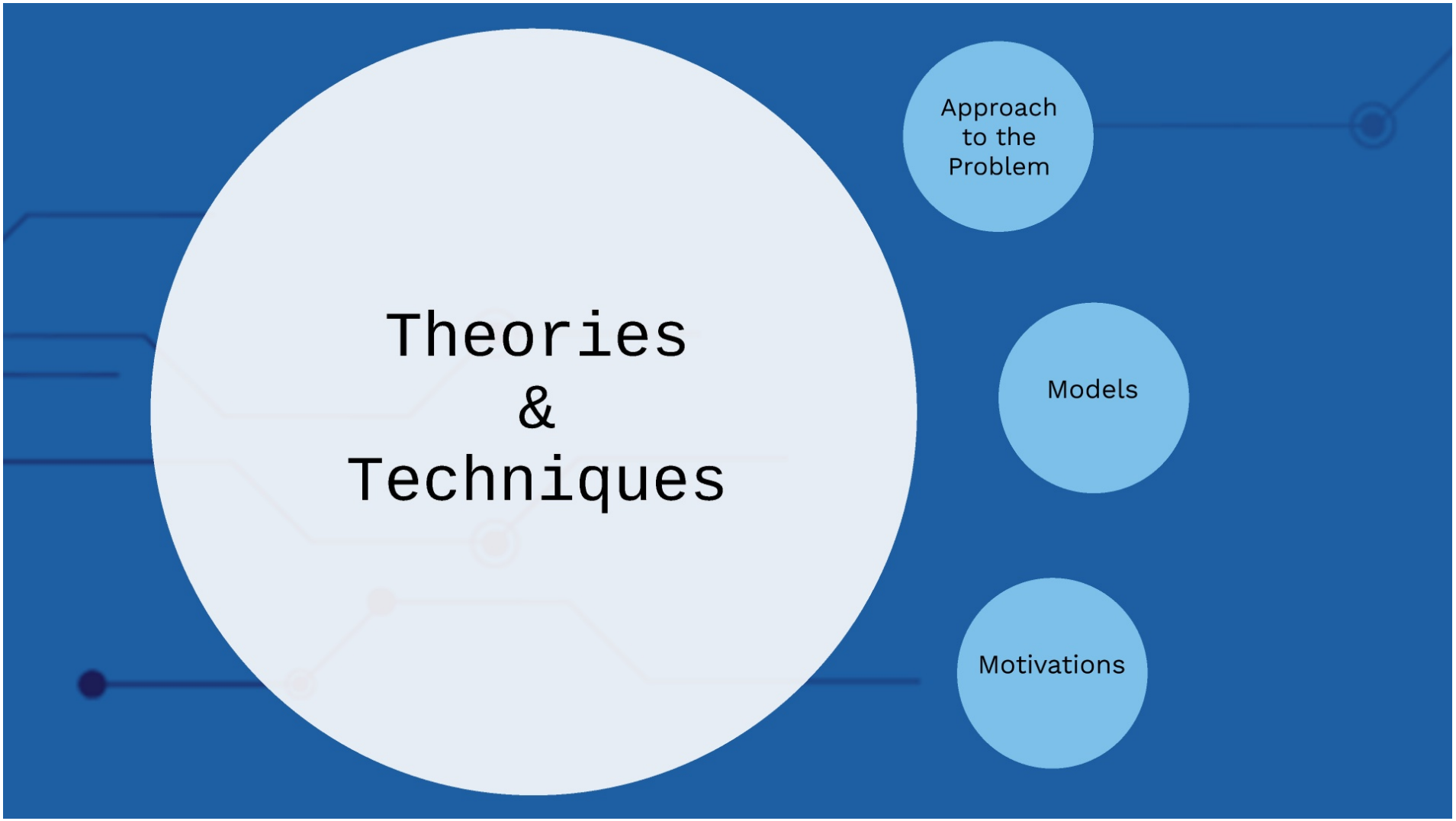
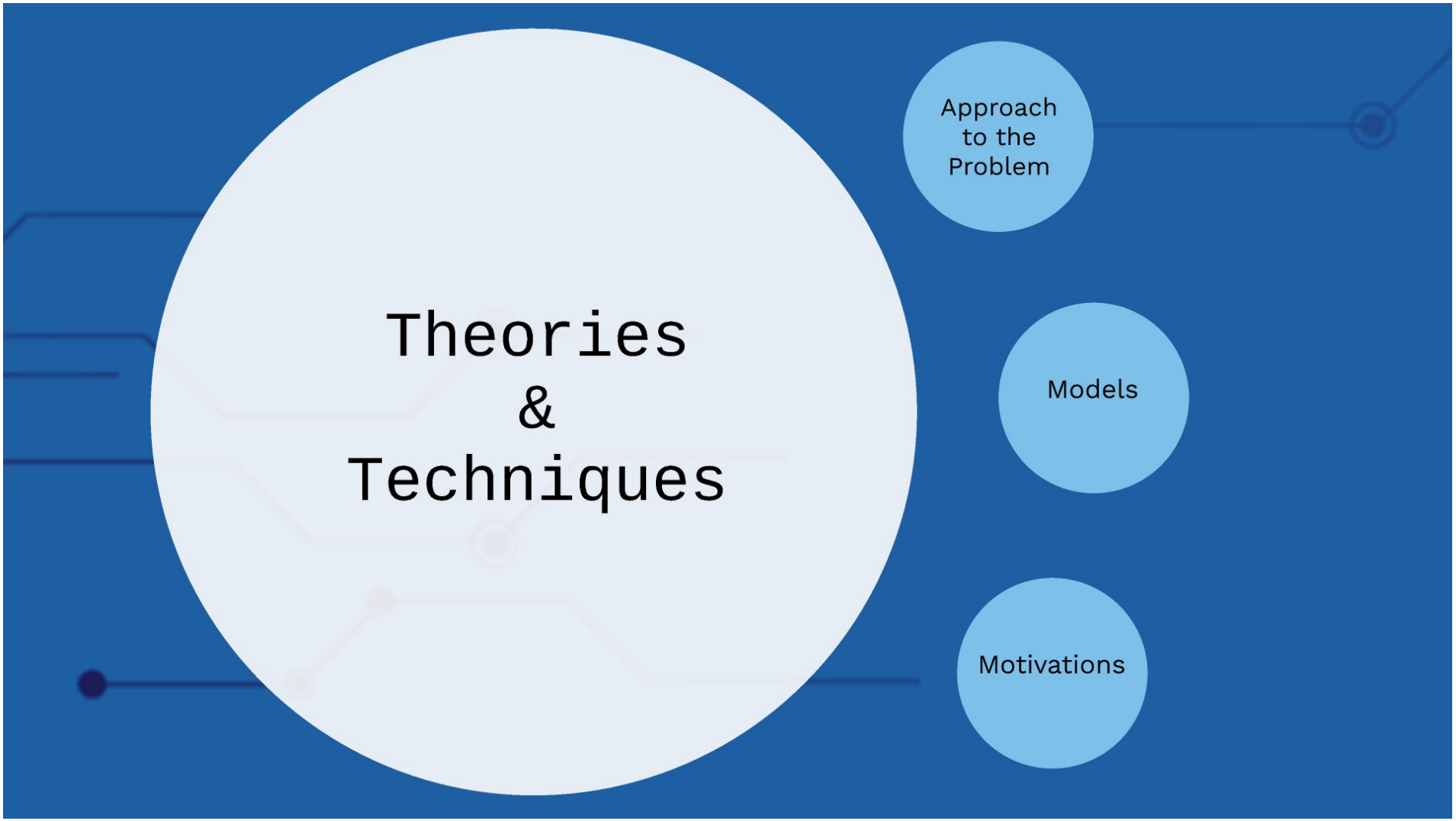

Theories
&
Techniques
Approach to the Problem
Models
Motivations

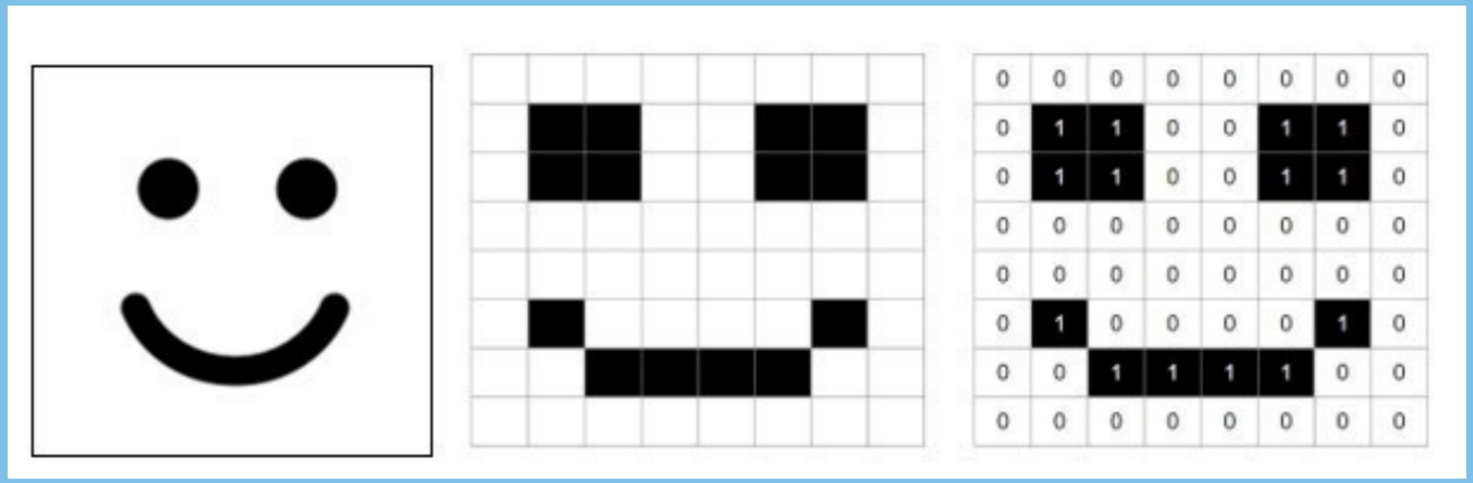
Image Recognition & Classification

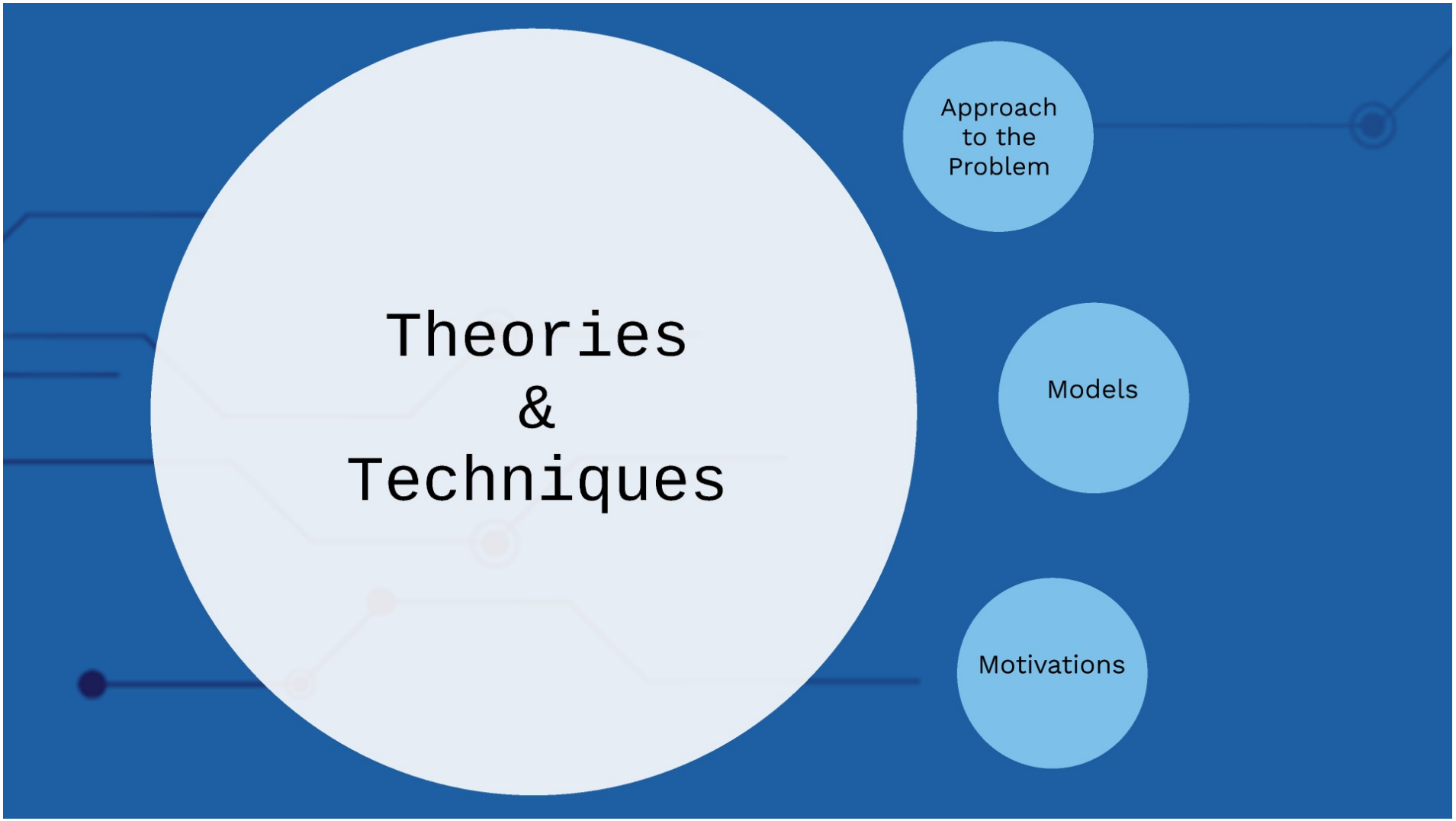
- Make some **research** and pick the most popular **solutions** & later **compare them**;
- Get and **pre-process data**;
- **Implement** and try all the possible solutions;
- Fit our work to make it feasible for the **competition**.


Theories
&
Techniques
Approach to the Problem
Models
Motivations

CNN - Convolutional Neural Network

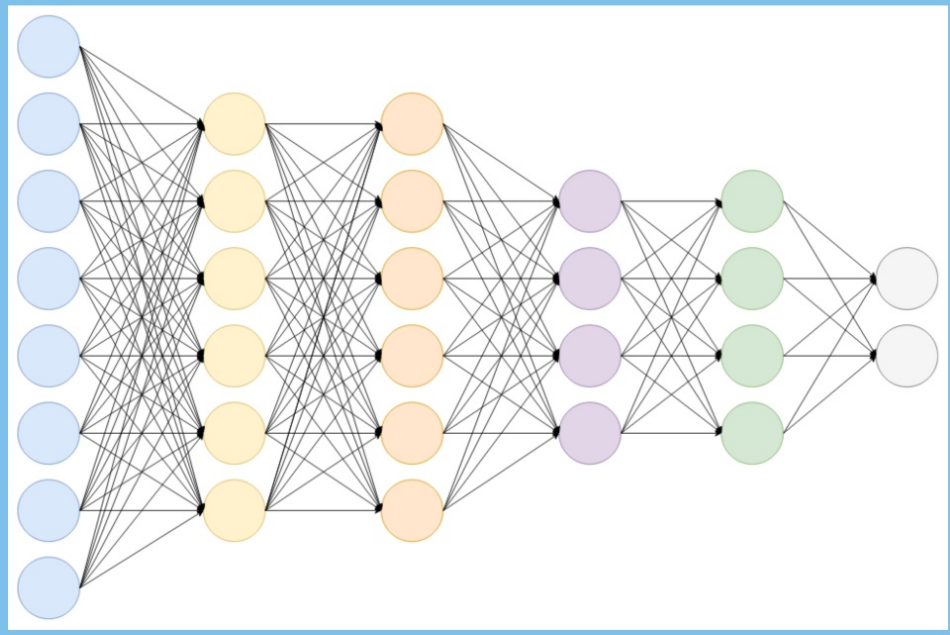
A Convolutional Neural Network, also known as CNN, is a class of **neural networks** that **specializes** in **processing data** that has a **grid-like topology**, such as an image.

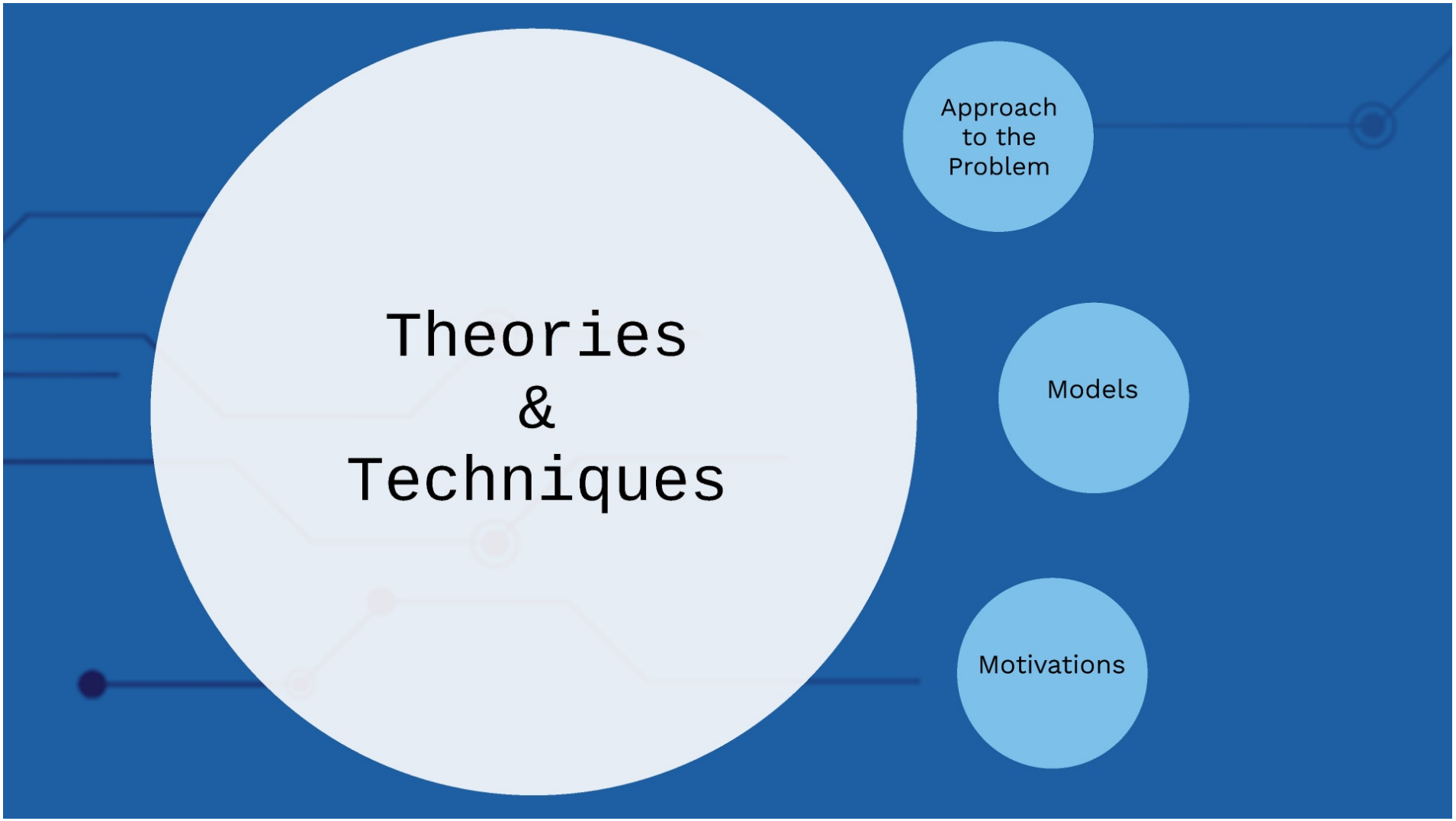


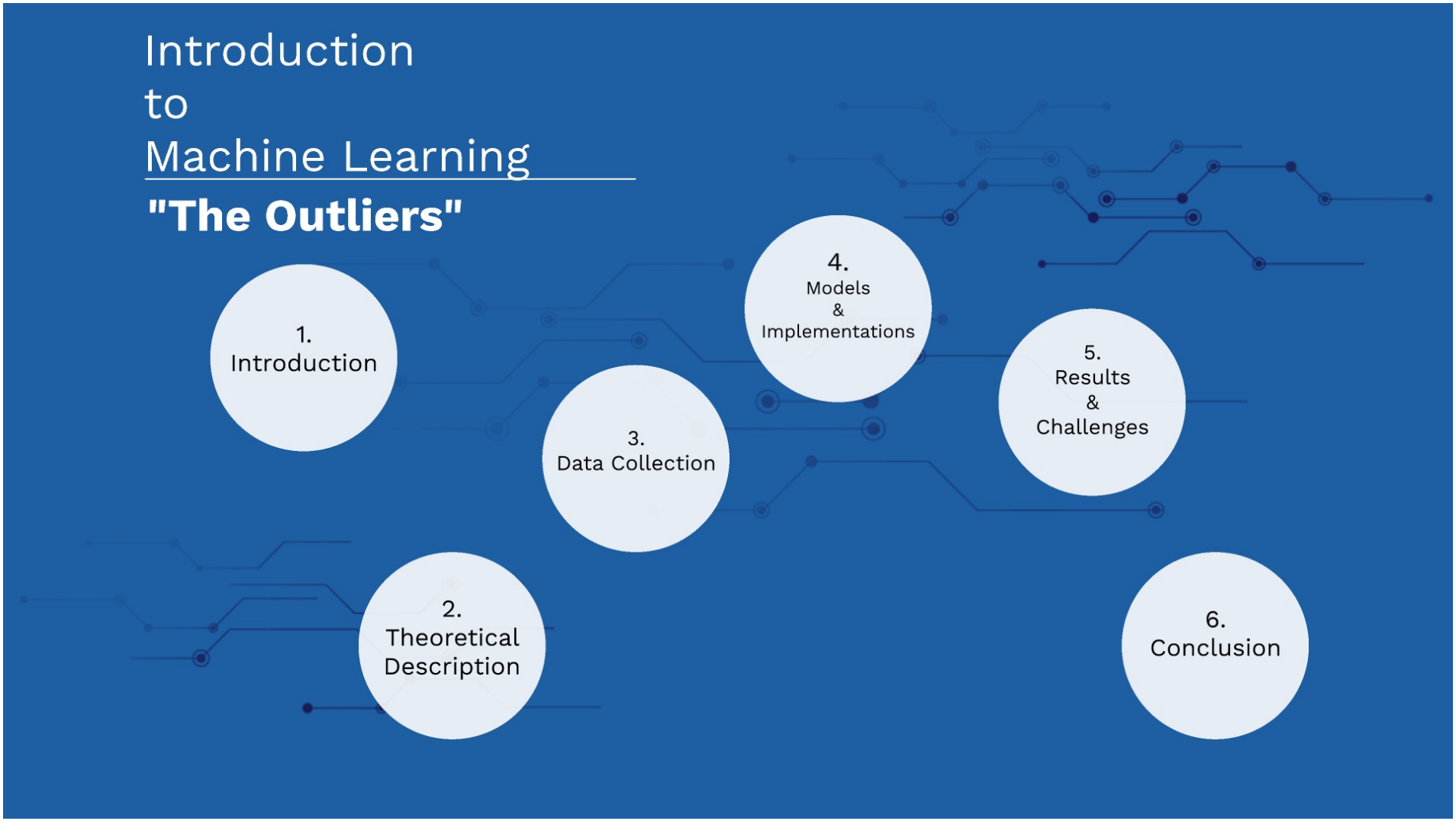

Theories
&
Techniques
Approach to the Problem
Models
Motivations

Why did we choose it among all?

1- CNNs is **specialised** in **pattern detection**;
2- There is a lot of **documentation** about CNNs;
3- There are many **pretrained models** which are applicable to this project problem.




Theories
&
Techniques
Approach to the Problem
Models
Motivations


Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion


Here comes the miners:
Data Collection
Image Scraping
Amount of Data
Characteristics of the Data

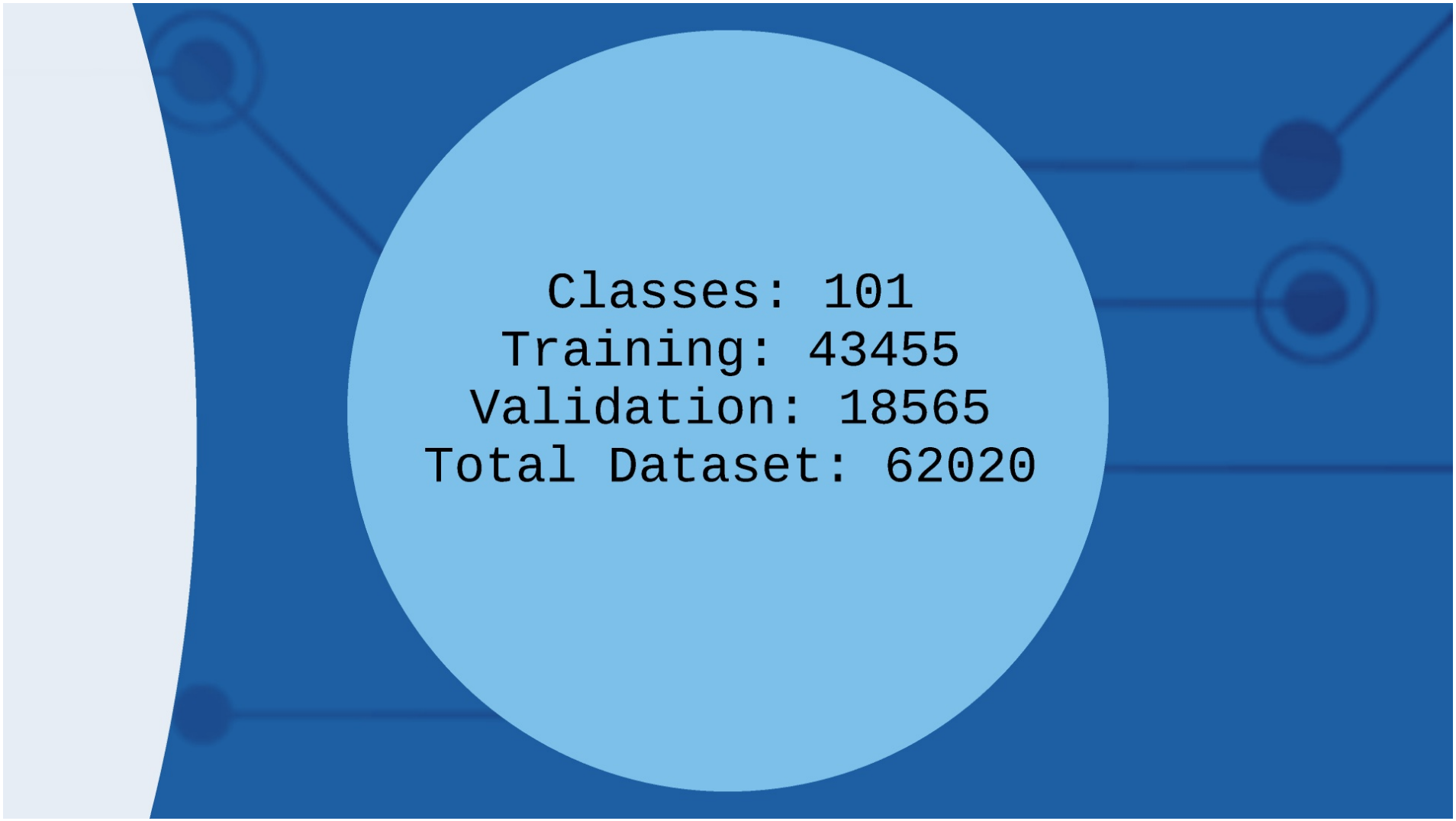
Approaches

- Image Download Tool
- Web Scraping
- Data Cleaning

Our Sources

- https://www.kaggle.com/datasets/iamsouravbanerjee/animal-image-dataset-90-different-animals/datasets
- https://www.image-net.org/challenges/LSVRC/2014/index#data
- https://www.mediasearch.dev/image


Here comes the miners:
Data Collection
Image Scraping
Amount of Data
Characteristics of the Data


Classes: 101
Training: 43455
Validation: 18565
Total Dataset: 62020

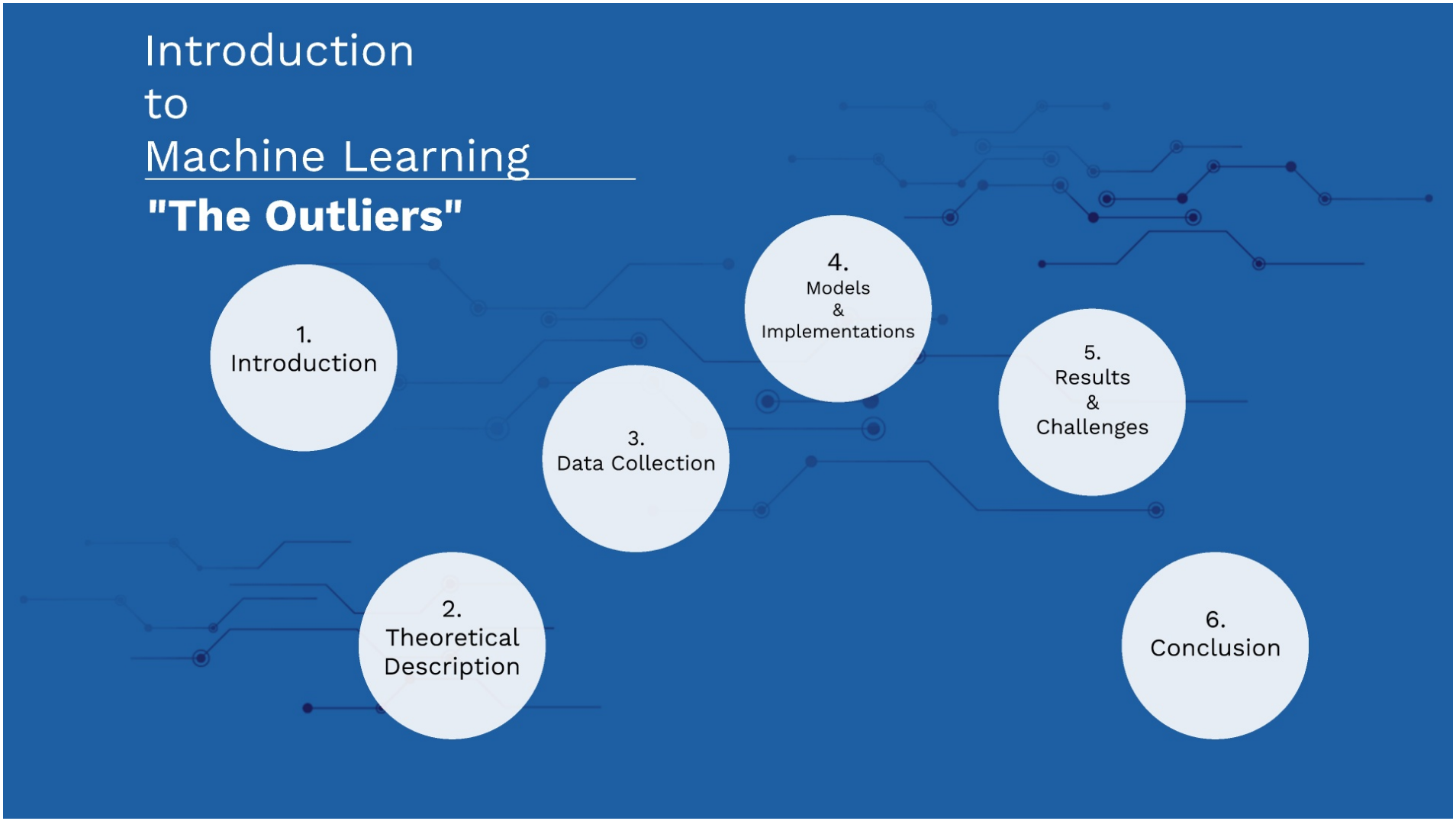

Here comes the miners:
Data Collection
Image Scraping
Amount of Data
Characteristics of the Data

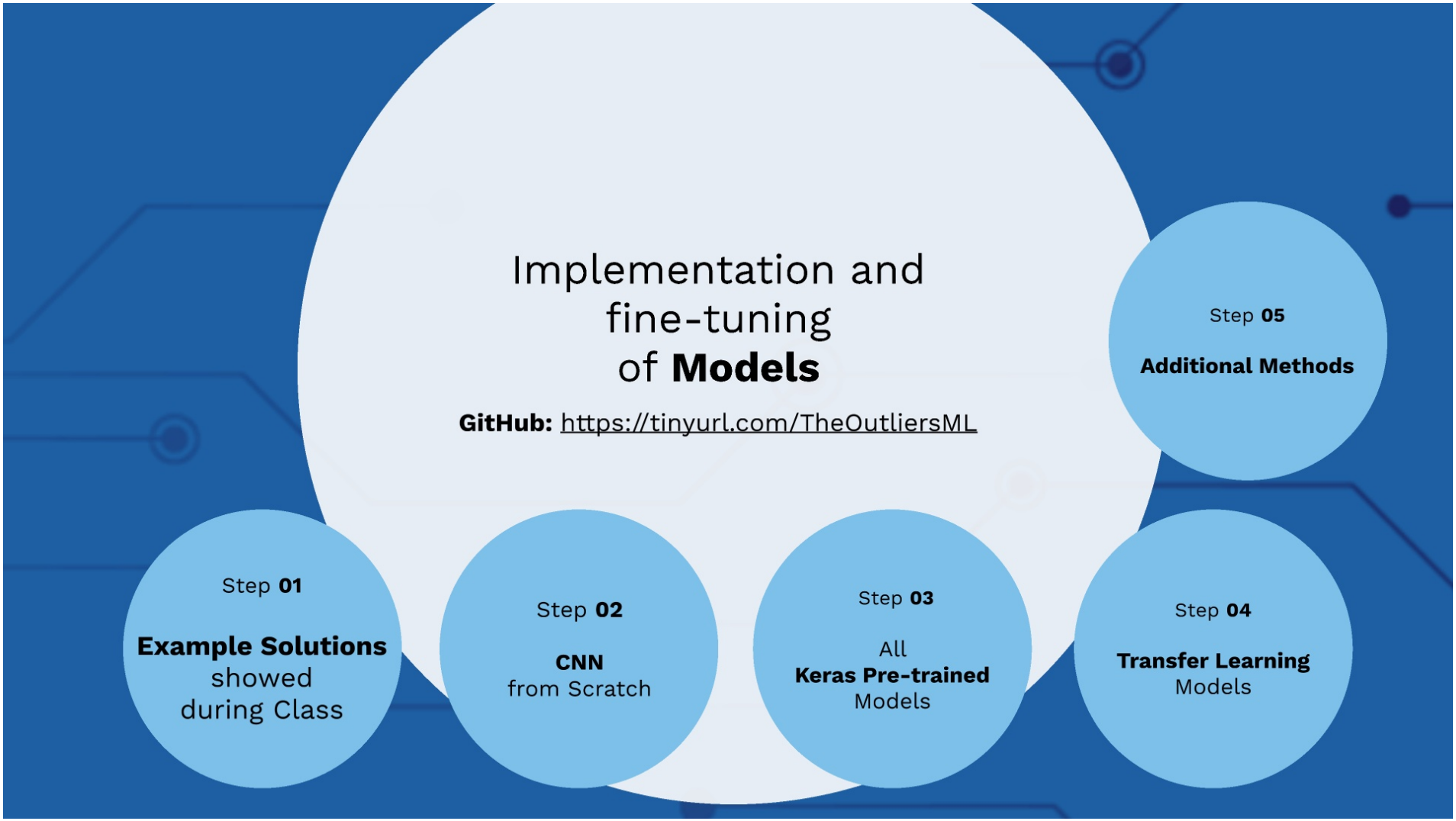
Data Augmentation

- Rescale
- Shear Range
- Zoom Range
- Rotation range
- Horizontal Flip

70/30 split for Train & Test

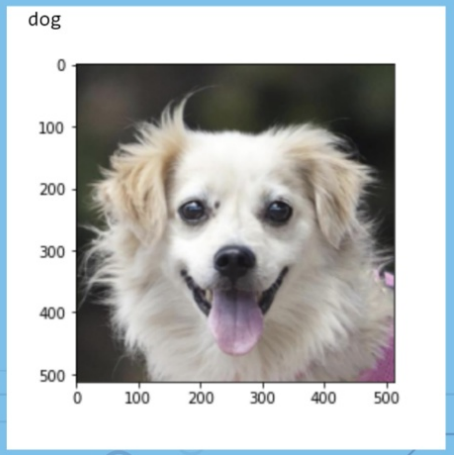

Here comes the miners:
Data Collection
Image Scraping
Amount of Data
Characteristics of the Data

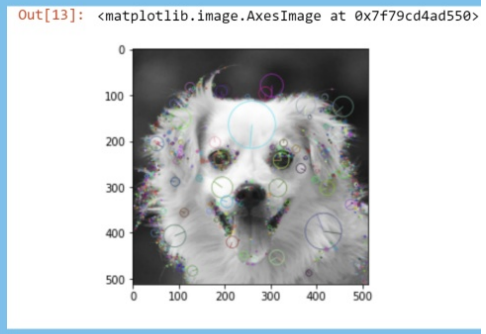

Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion


Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods

SIFT BASED SOLUTION

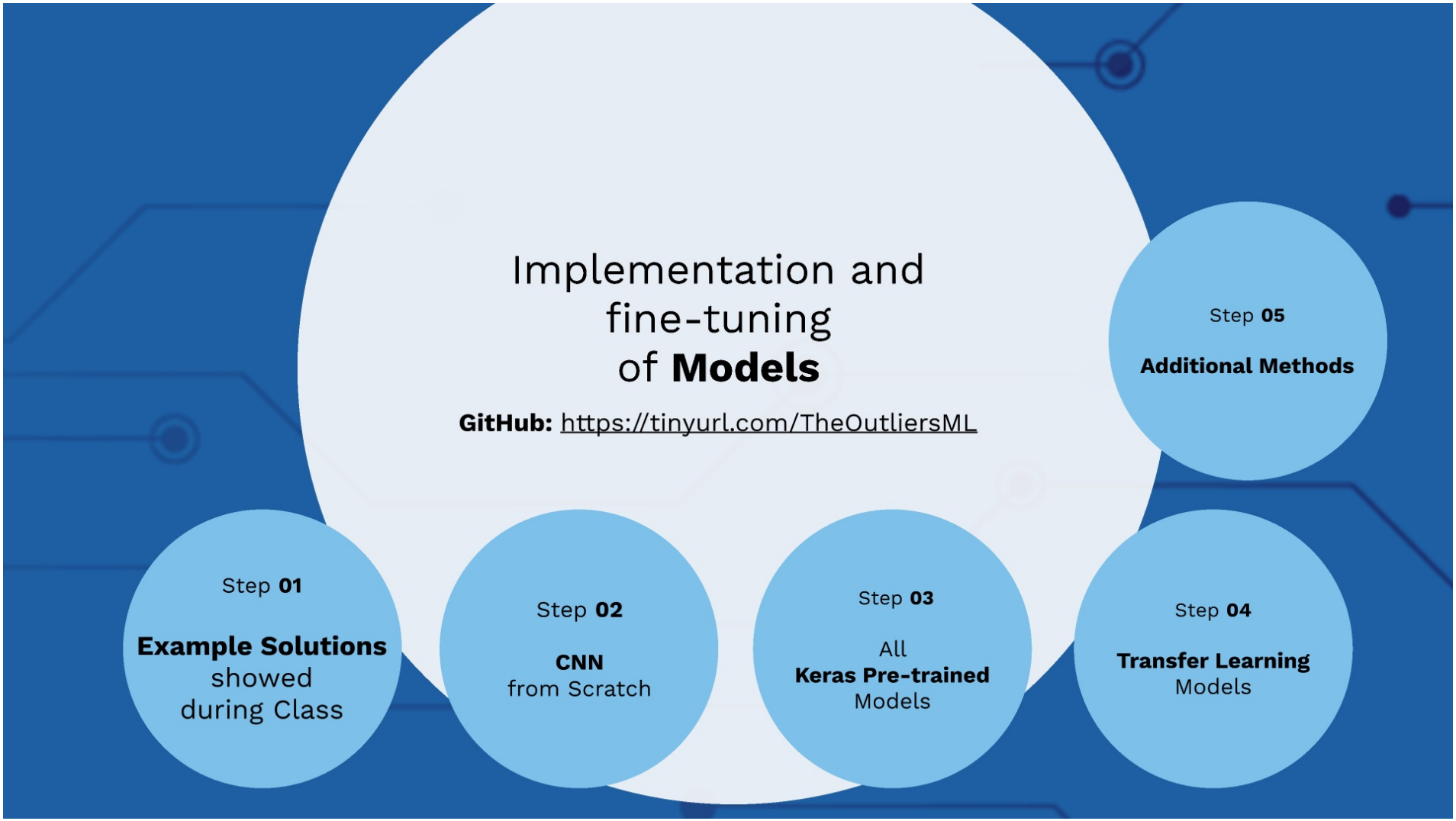
- Image alignment and object detection.
- Keypoints/Interest point
- SIFT detector
- SIFT descriptor





Result on our Dataset

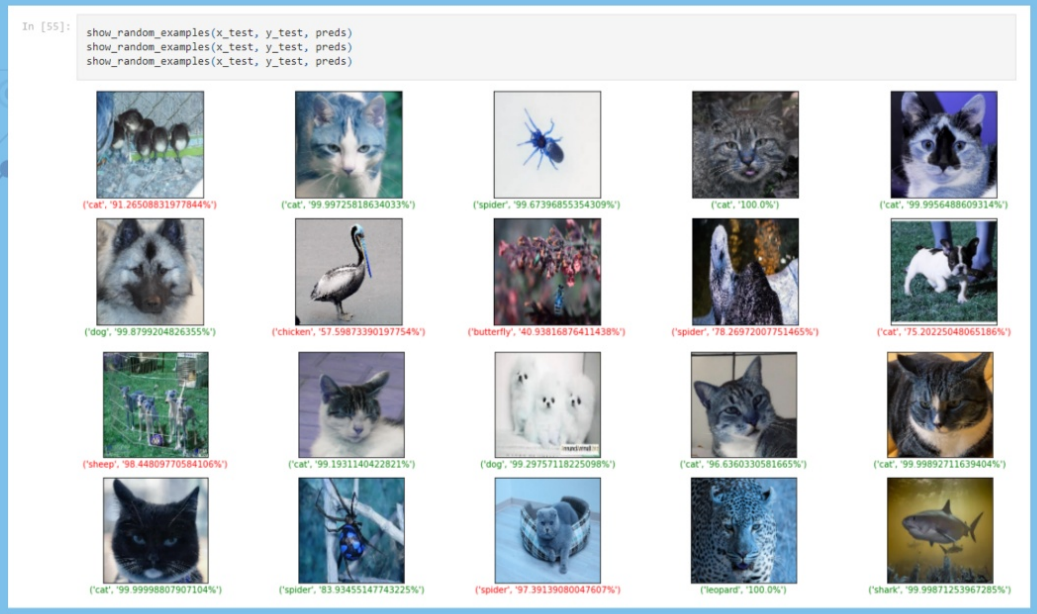
```
########## RESULTS ##########
--> Top-1 Accuracy: 0.192
--> Top-3 Accuracy: 0.500
--> Top-10 Accuracy: 0.615
```


Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods

CNN from Scratch - 01

https://tinyurl.com/TheOutliersCNNscratch

- **Preprocessed** the data
- Converted to **Numpy Array**
- **LabelBinarizer** to scale Label data
- Train/test **70:30** split
- **Training** the model:
  - epochs=10
  - batch_size=128
  - img_size = 360

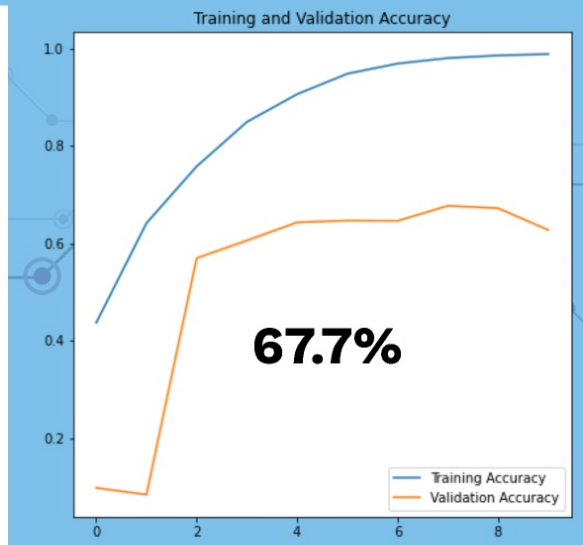


CNN from Scratch - 02

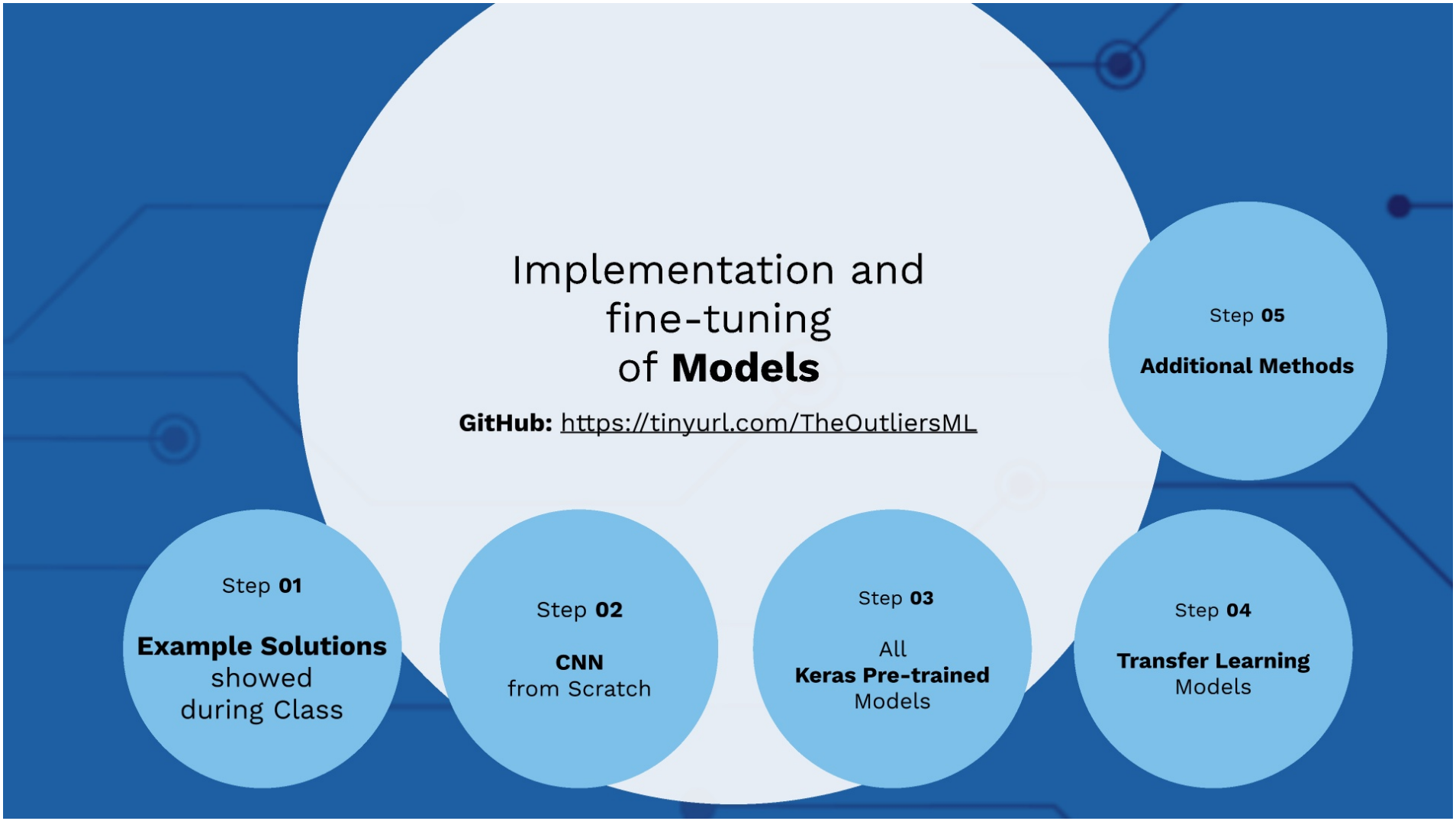
Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      896

 batch_normalization (BatchN  (None, 128, 128, 32)     128
 ormalization)

 conv2d_1 (Conv2D)           (None, 126, 126, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 63, 63, 64)        18496

 batch_normalization_1 (Batc  (None, 63, 63, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 61, 61, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 30, 30, 128)       73856

 batch_normalization_2 (Batc  (None, 30, 30, 128)      512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 28, 28, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 98)                2458722

=================================================================
Total params: 2,746,626
Trainable params: 2,746,178
Non-trainable params: 448
_________________________________________________________________
```

Training and Validation



https://tinyurl.com/TheOutliersCNNscratch

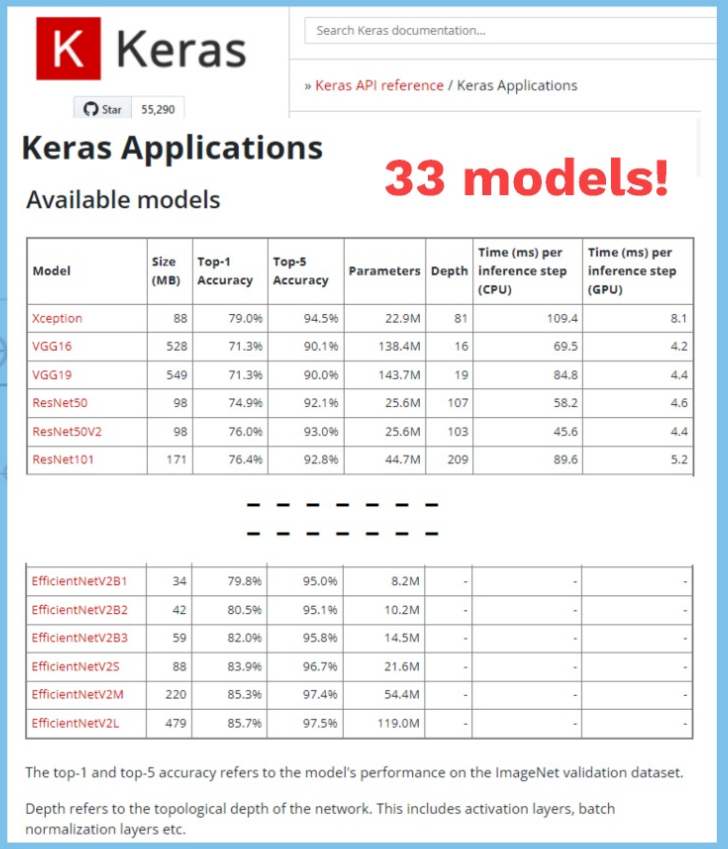

Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods

Keras Pre-trained Models - 01

Yes! We tried them **ALL**!!!

We are that **CRAZY**!!!

https://keras.io/api/applications/



Keras

Search Keras documentation...

» Keras API reference / Keras Applications

Star 55,290

Keras Applications

33 models!

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
- - - - - - - - - - - - - -							
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

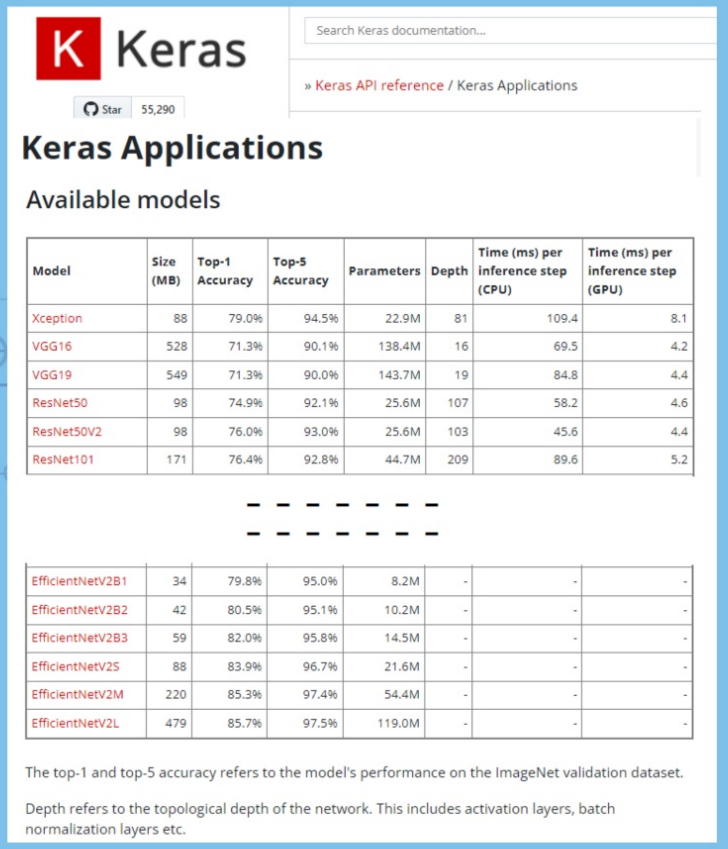
The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Keras Pre-trained Models - 02

- Wrote **re-usable code templates** to run all the models
- Run with **default parameters**
- Compared **results**
  - Finalized **EfficientNetB7**
- **Fine-tuned** model:
  - batch_size = 64
  - img_size = 224
  - PCA n_components = 500
  - distance: Cosine
  - Nearest neighbor algo: KD Tree

https://keras.io/api/applications/



Keras

Search Keras documentation...

» Keras API reference / Keras Applications

Star 55,290

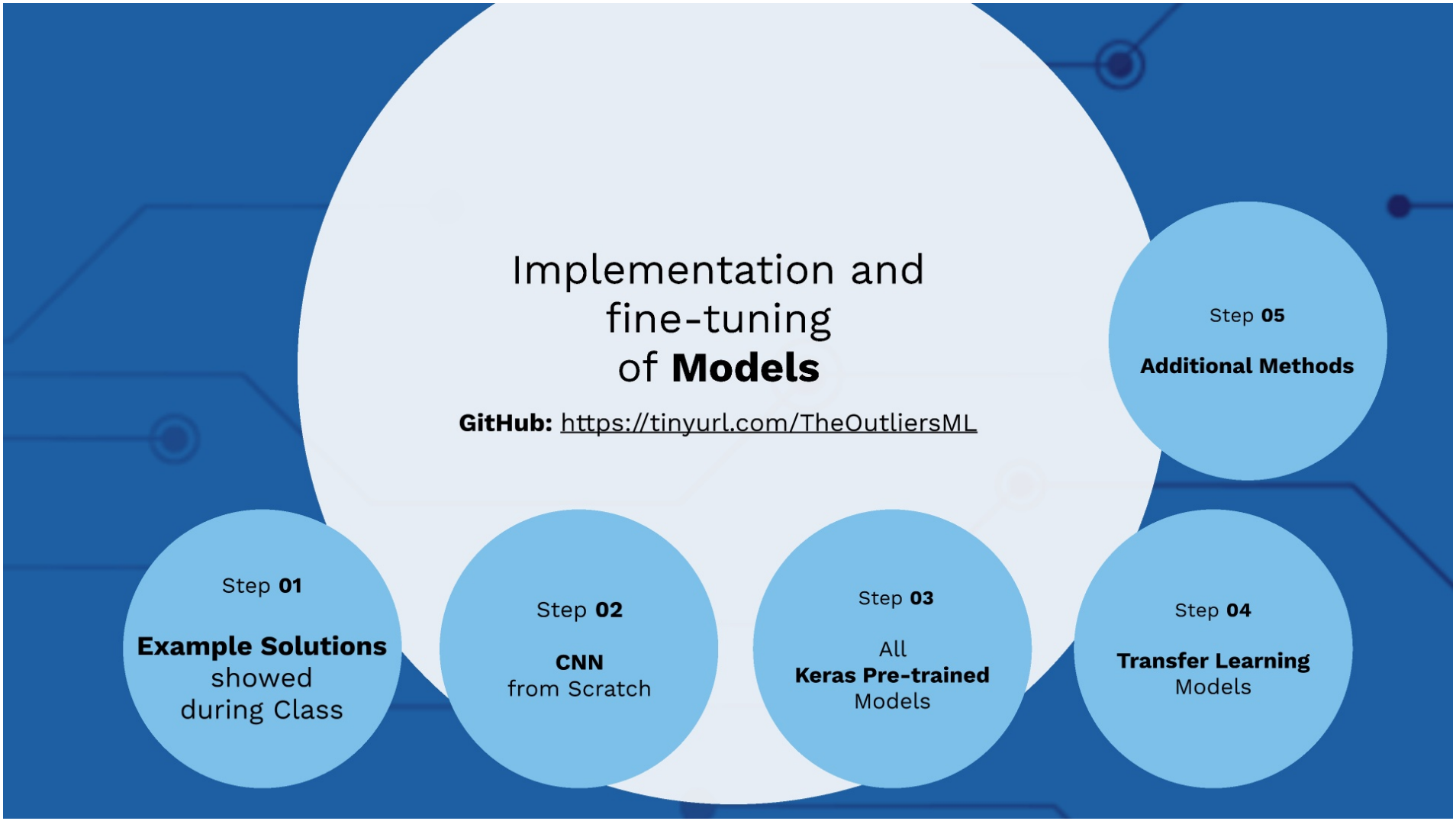
Keras Applications

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
- - - - - - -							
- - - - - - -							
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

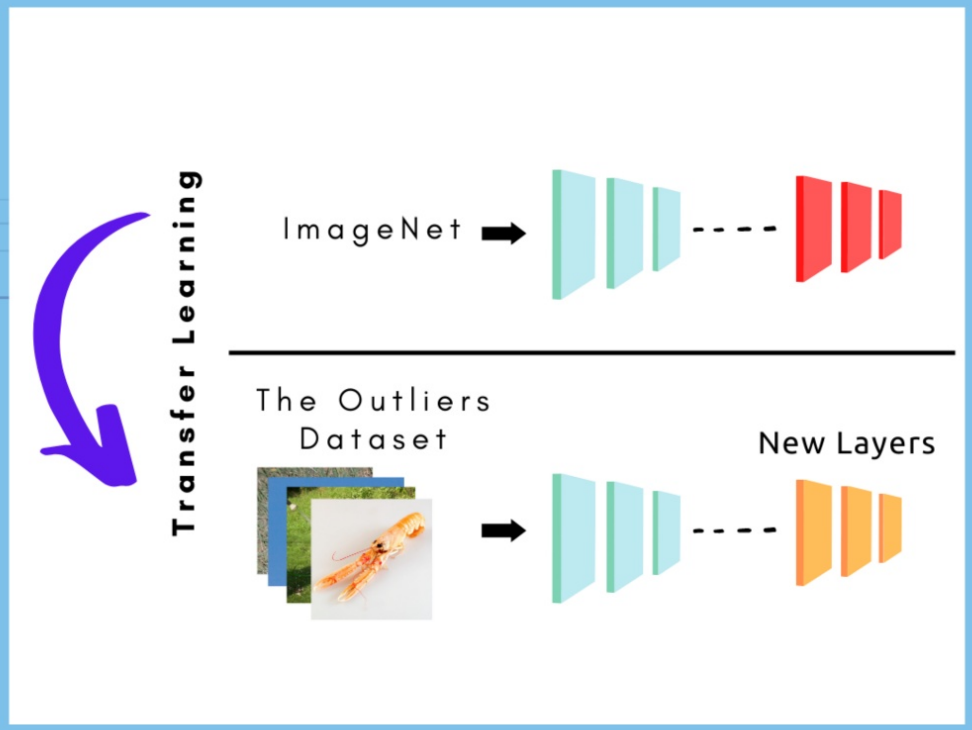
The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.


Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods

Transfer Learning Models - 01

- VGG16
- Resnet50
- Resnet50v2
- MobileNet
- EfficientNetV2B3



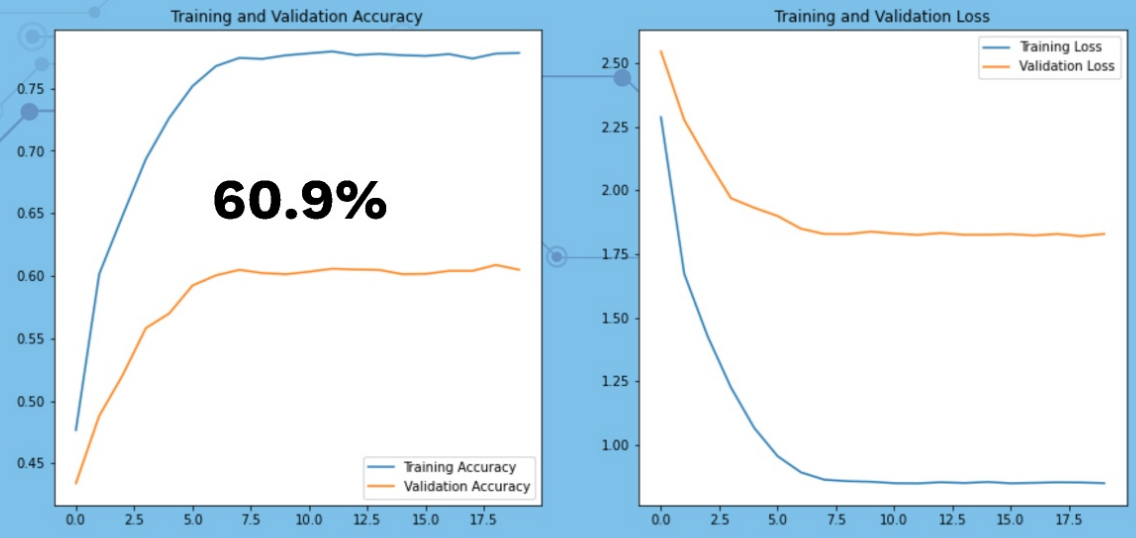
Transfer Learning Models - 02

VGG16 TL Model: https://tinyurl.com/TheOutliersVGG16TL

```
# our layers - you can add more if you want
x = Flatten()(vgg16.output)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vgg16.input, outputs=prediction)

# Make last 4 layers trainable
model.get_layer('block5_conv3').trainable = True
model.get_layer('block5_conv2').trainable = True
model.get_layer('block5_conv1').trainable = True
model.get_layer('block4_conv3').trainable = True

# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

- - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - -

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 101)               2533989

=================================================================
Total params: 17,248,677
Trainable params: 2,533,989
Non-trainable params: 14,714,688
_________________________________________________________________
```

Training and Validation



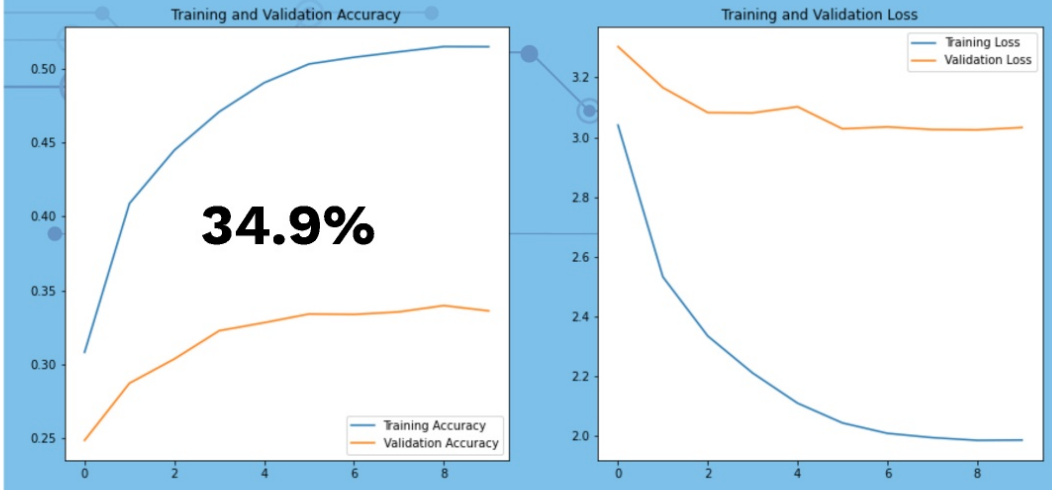
Transfer Learning Models - 03

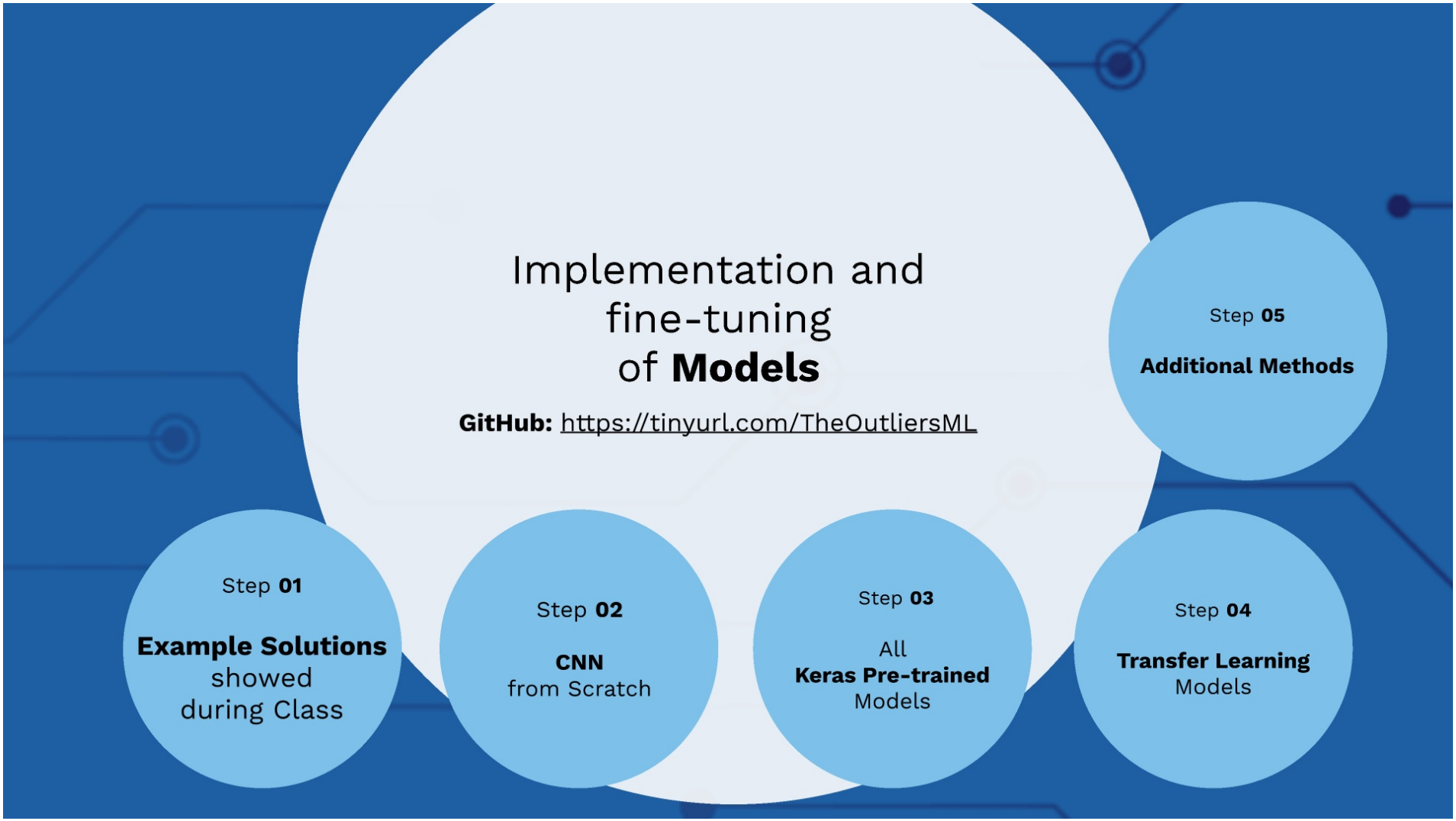
ResNet50 Model: https://tinyurl.com/TheOutliersResNet50TL

```
# our layers - you can add more if you want
x = Flatten()(resnet50.output)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=resnet50.input, outputs=prediction)

# Make last 4 layers trainable
model.get_layer('conv5_block3_3_conv').trainable = True
model.get_layer('conv5_block3_2_conv').trainable = True
model.get_layer('conv5_block3_1_conv').trainable = True
model.get_layer('conv5_block2_3_conv').trainable = True

# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 224, 224, 3       0         []
                             )]

conv1_pad (ZeroPadding2D)    (None, 230, 230, 3)       0         ['input_1[0][0]']

conv1_conv (Conv2D)          (None, 112, 112, 64       9472      ['conv1_pad[0][0]']
                             )

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

conv5_block3_3_conv (Conv2D) (None, 7, 7, 2048)        1050624   ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal (None, 7, 7, 2048)      8192      ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)       (None, 7, 7, 2048)        0         ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation) (None, 7, 7, 2048)       0         ['conv5_block3_add[0][0]']

flatten (Flatten)            (None, 100352)            0         ['conv5_block3_out[0][0]']

dense (Dense)                (None, 101)               10135653  ['flatten[0][0]']

=================================================================================================
Total params: 33,723,365
Trainable params: 10,135,653
Non-trainable params: 23,587,712
_________________________________________________________________________________________________
```

Training and Validation




Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods

Additional Methods for improving performance

- Implementation of **LearningRateScheduler**

In [15]:
```
from keras.callbacks import LearningRateScheduler
def lr_scheduler(epoch, lr):
    decay_rate = 0.85
    decay_step = 1
    if epoch % decay_step == 0 and epoch:
        return lr * pow(decay_rate, np.floor(epoch / decay_step))
    return lr
```

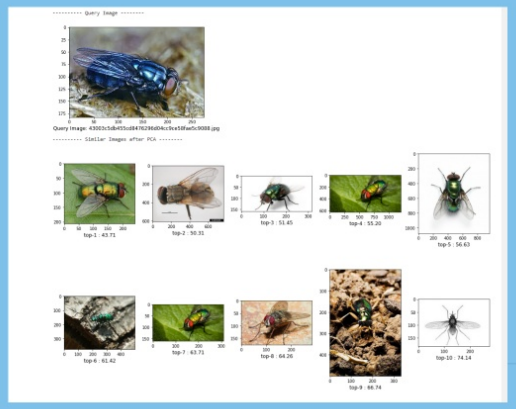
- Implementation of **PCA**

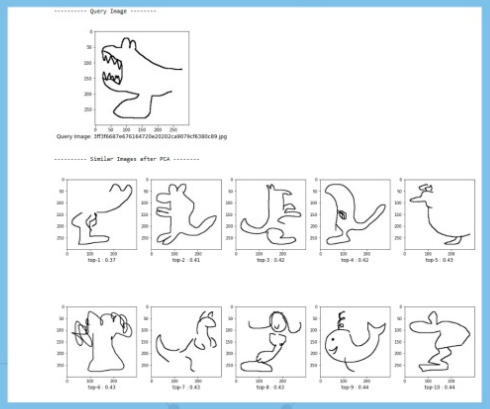
To overcome this we will apply PCA to our features and reduce the dimensions and try to find the similar features again.

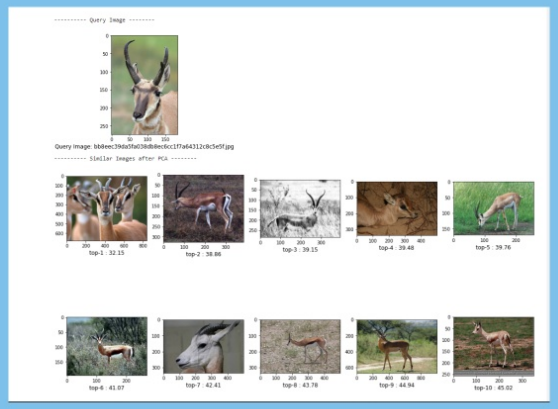
In [27]:
```
pca = PCA(n_components=100)
pca.fit(feature_list)
compressed_features = pca.transform(feature_list)
```

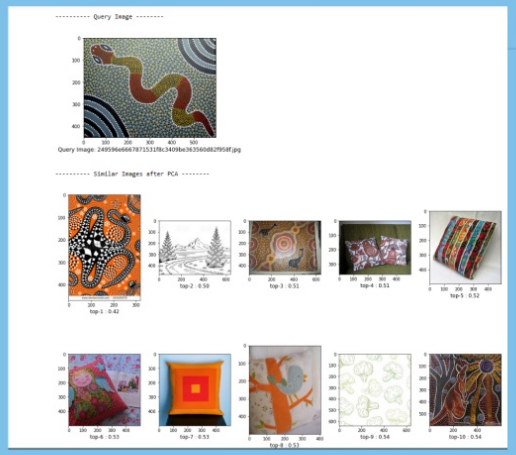
In [28]:
```
neighbors_pca_features = NearestNeighbors(n_neighbors=5,
                                          algorithm='ball_tree',
                                          metric='euclidean').fit(compressed_features)
```

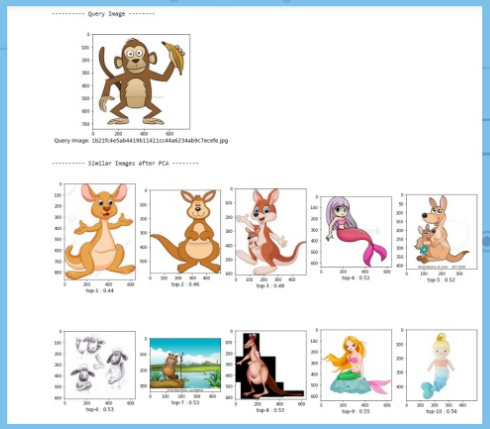
Some results from the **Competetion**

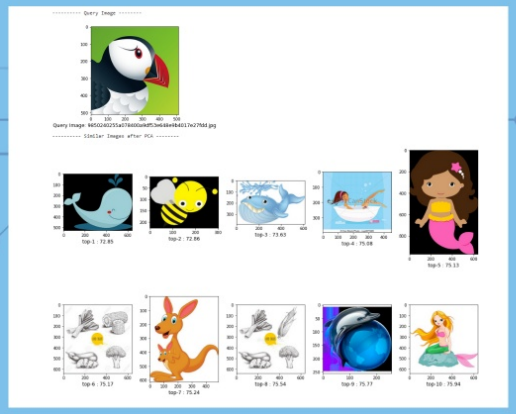


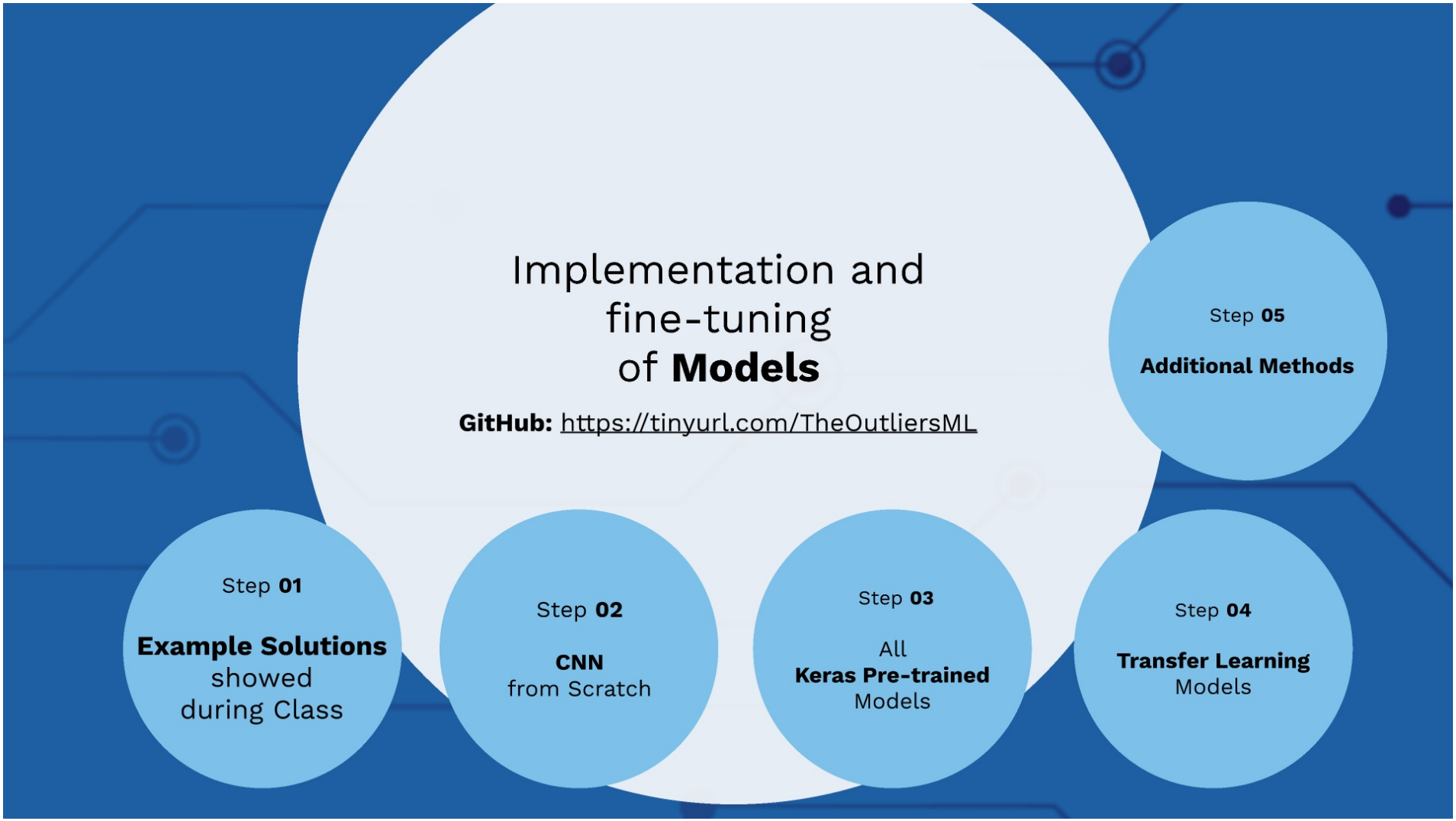


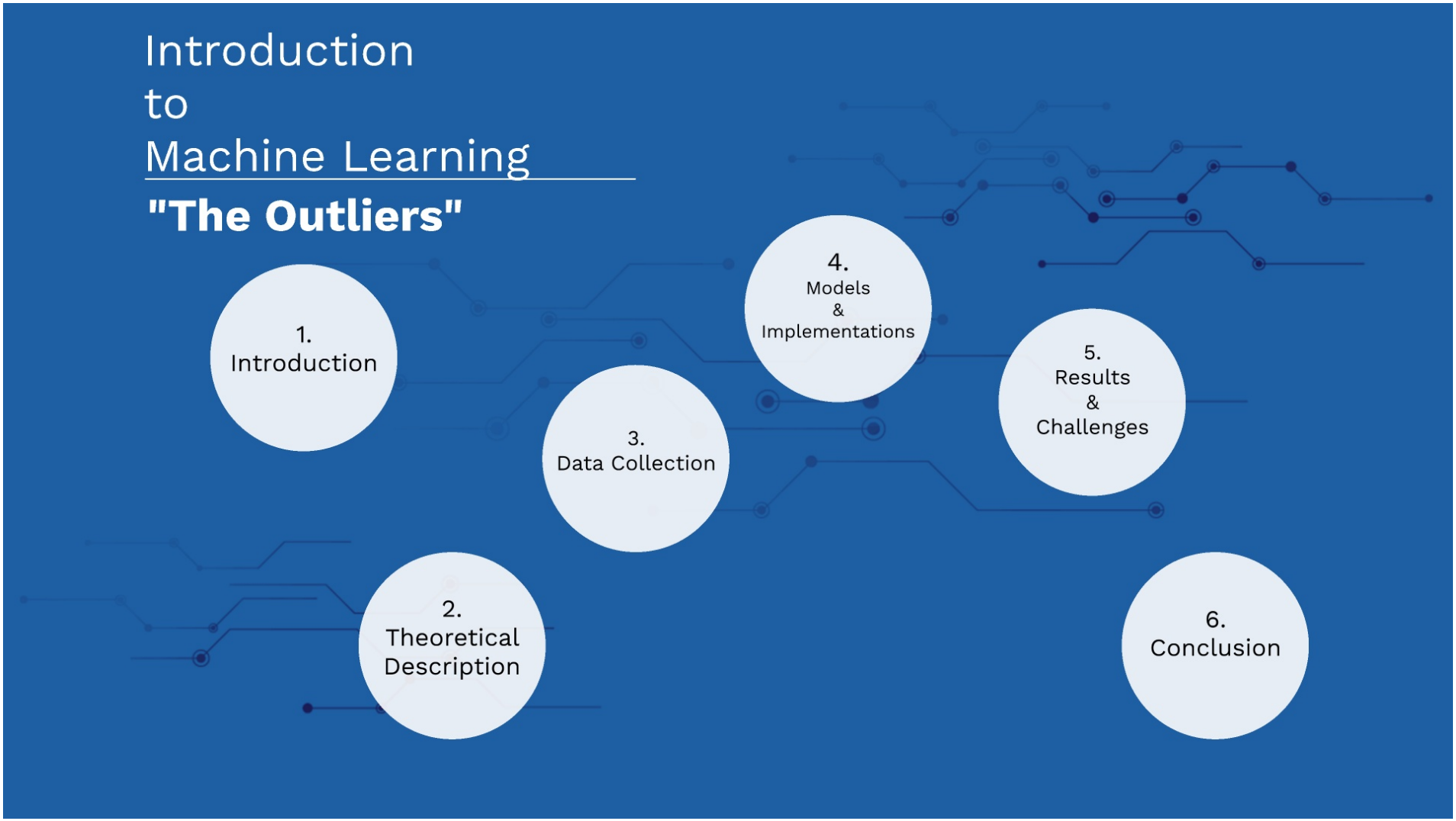


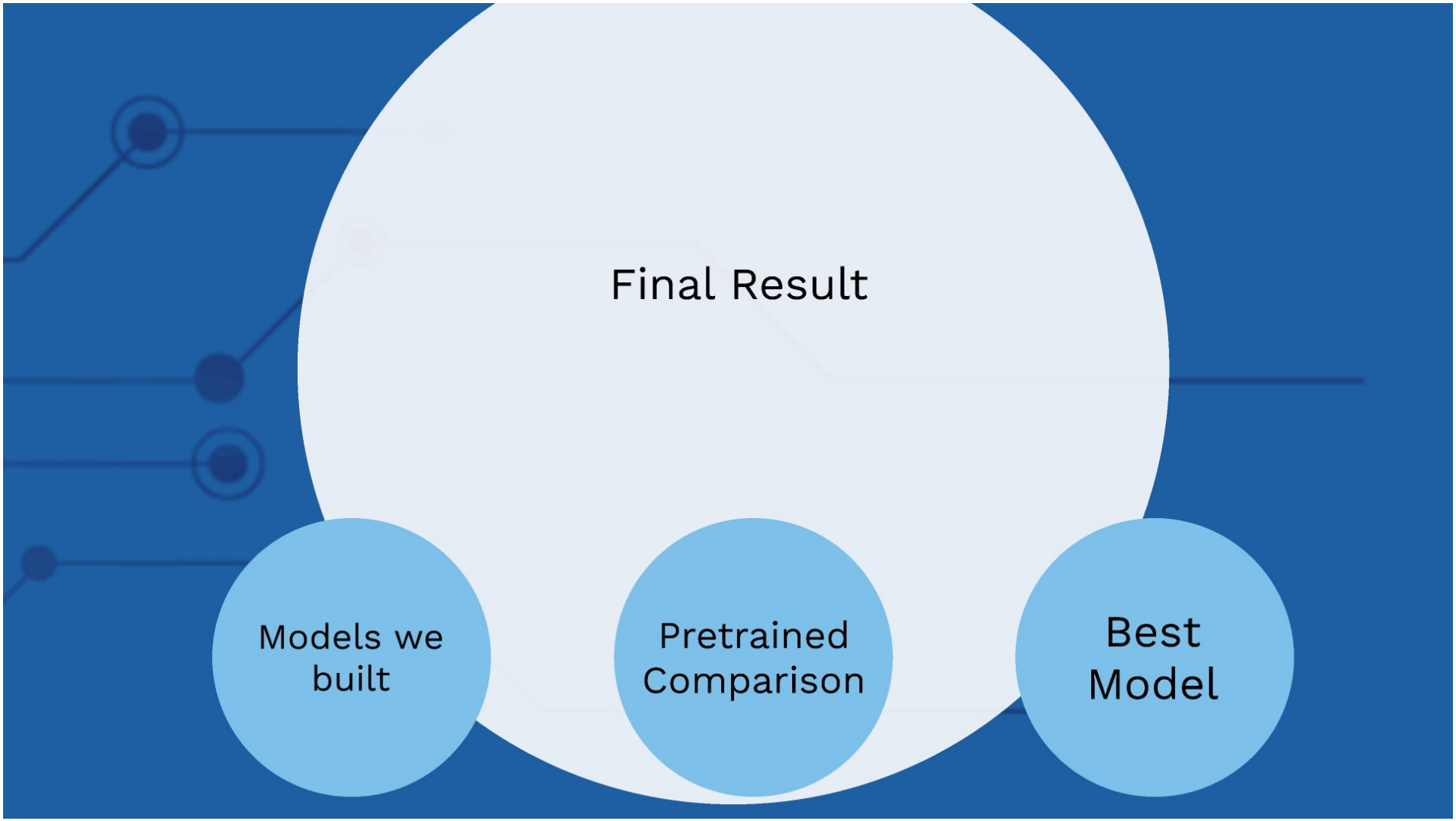






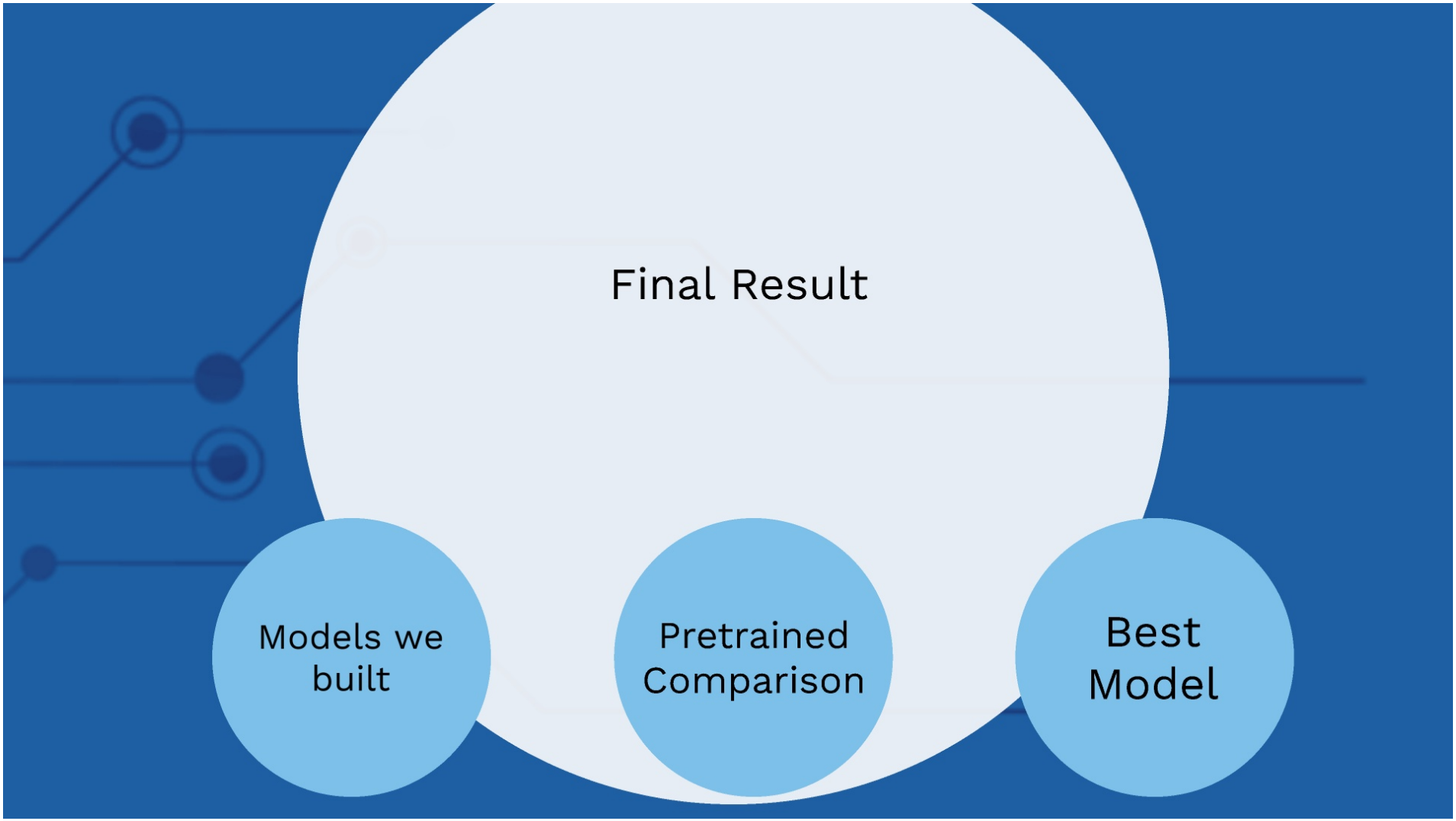

Implementation and fine-tuning of **Models**
GitHub: https://tinyurl.com/TheOutliersML
Step **01**
Example Solutions showed during Class
Step **02**
CNN from Scratch
Step **03**
All **Keras Pre-trained** Models
Step **04**
Transfer Learning Models
Step **05**
Additional Methods


Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion


Final Result
Models we built
Pretrained Comparison
Best Model

Comparison of Models we built:

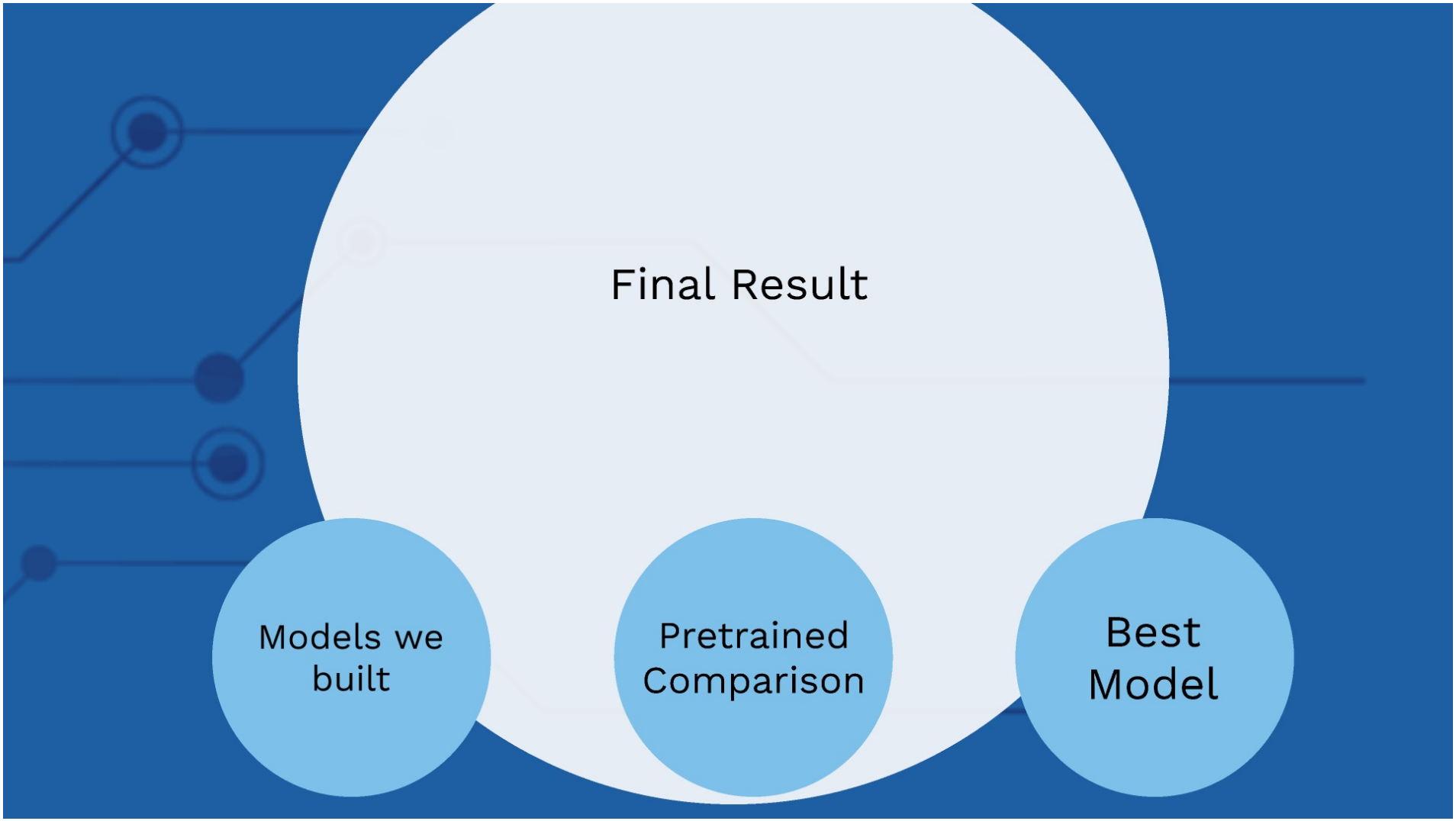
Models	Top 1	Top 5	Top 10
SIFT Based Model	0.192	0.5	0.615
Image matching example	0.65	0.62	0.62
CNN from scratch	0.33	0.31	0.4
Transfer Learning on VGG16	0.4	0.59	0.597
Transfer Learning on ResNet50	0.42	0.45	0.6
Transfer Learning on MobileNet	0.5	0.62	0.69
Transfer Learning on ResNet50V2	0.49	0.5	0.58
Transfer Learning on EfficientNetV2B3	0.47	0.57	0.59


Final Result
Models we built
Pretrained Comparison
Best Model

Comparison of **pretrained** models:

Models	Top 1	Top 5	Top 10
Transfer Learning on MobileNet	0.5	0.62	0.69
Transfer Learning on ResNet50V2	0.49	0.5	0.58
Xception	0.51	0.7	0.72
VGG16	0.53	0.56	0.62
VGG19	0.55	0.6	0.72
ResNet50V2	0.58	0.66	0.7
ResNet50	0.54	0.55	0.59
InceptionV3	0.6	0.72	0.73
MobileNet	0.62	0.63	0.7
EfficientNetB7	0.7	0.8	0.815

EfficientNetB7	Top 1	Top 5	Top 10
"Batch = 8 Img_size = 360 n_neighbors = 10 n_components = 500"	0.7	0.8	0.815
Batch = 8 Img_size = 360 n_neighbors = 15 n_components = 600	0.72	0.76	0.8
Batch = 5 Img_size = 255 n_neighbors = 15 n_components = 500	0.707	0.8	0.815
Batch = 5 Img_size = 300 n_neighbors = 10 n_components = 500	**0.73**	**0.8**	**0.83**


Final Result
Models we built
Pretrained Comparison
Best Model

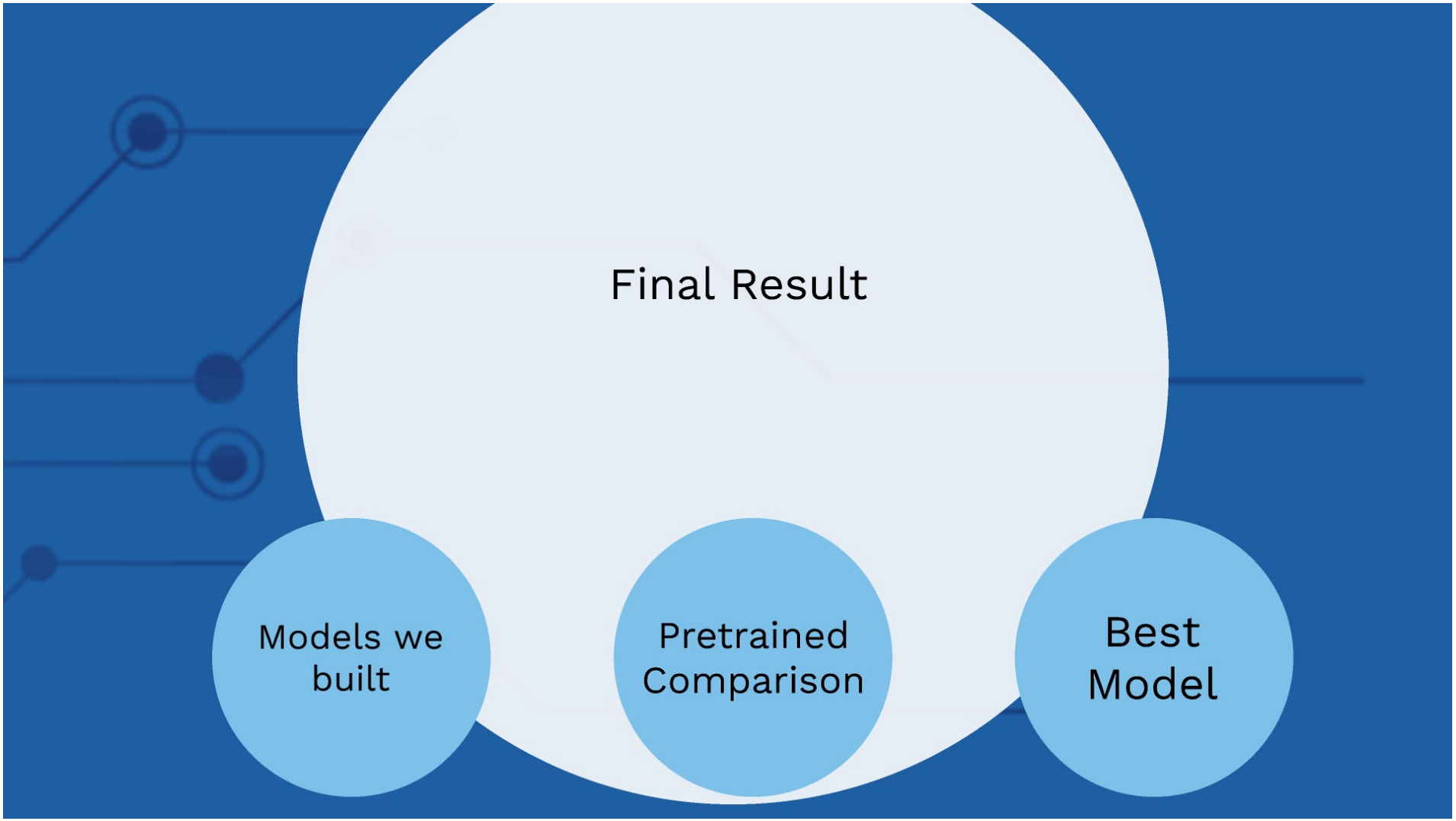
The Best Model so far:

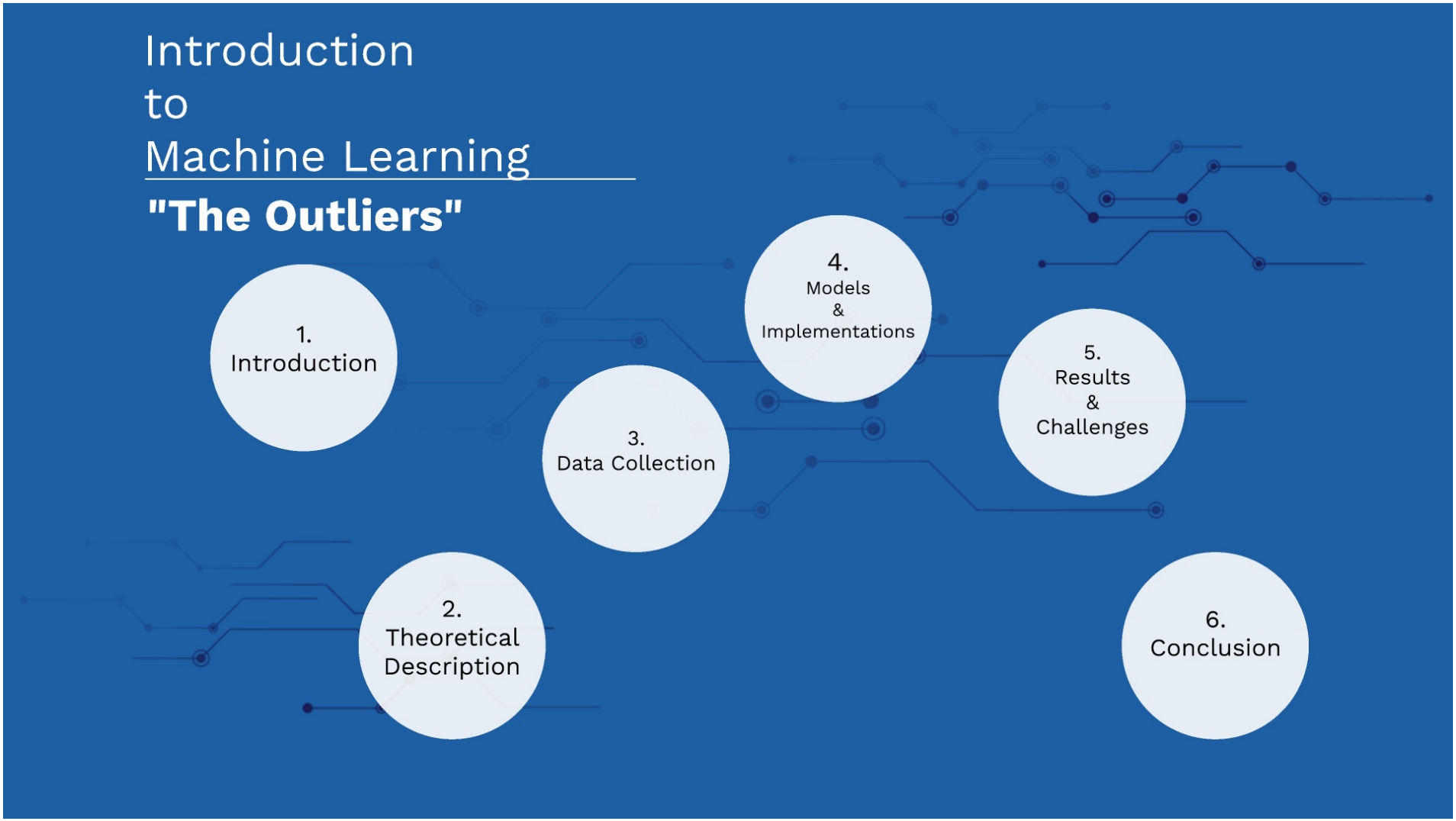
EfficientNetB7 Pretrained Model

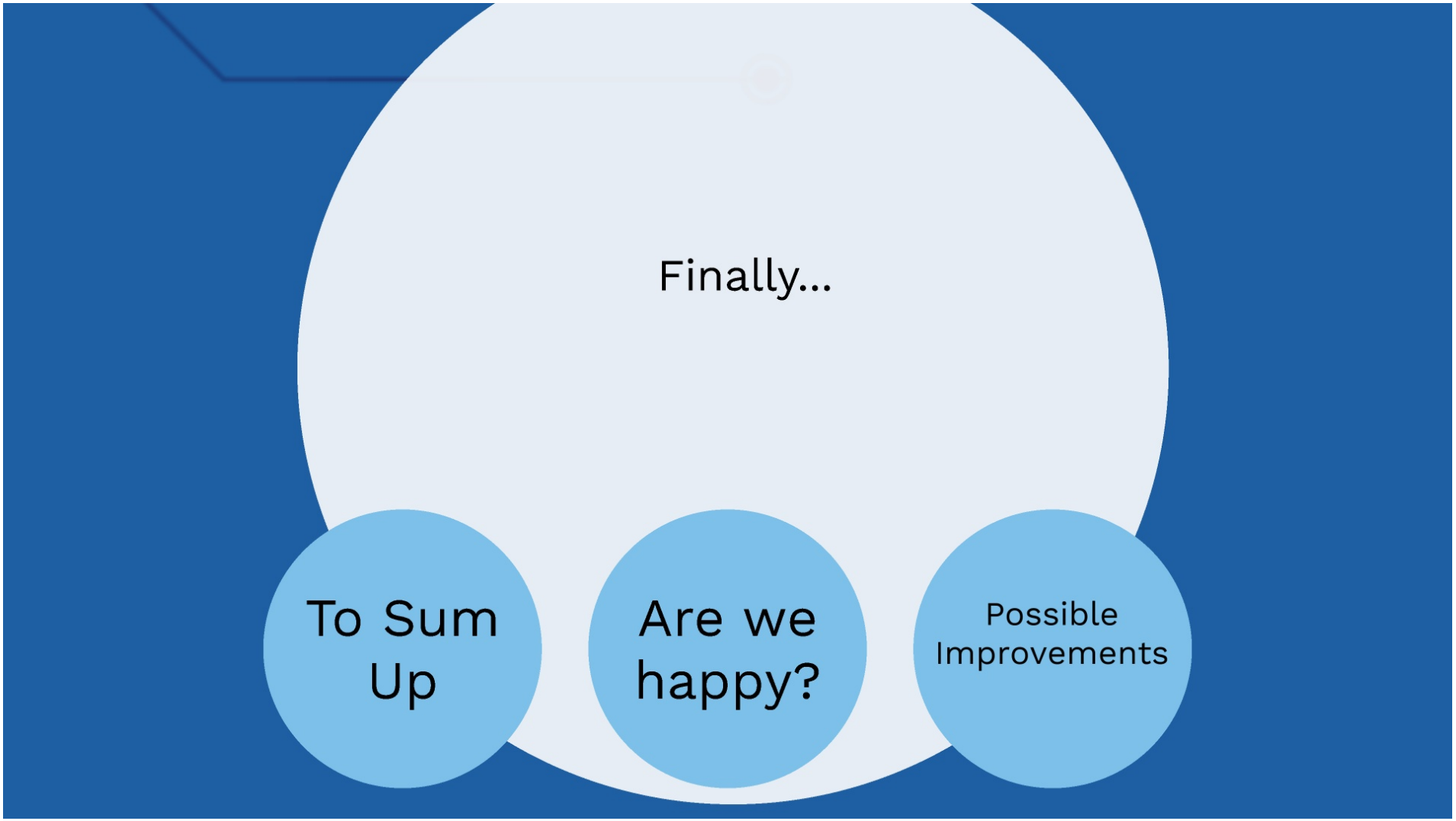
- **Top 1** Accuracy: **0.73**
- **Top 5** Accuracy: **0.8**
- **Top 10** Accuracy: **0.83**

Optimal Parameters:

- Batch size: 5
- image size: 300
- Distance: Cosine
- n_component: 500
- Nearest neighbor algo: KD Tree


Final Result
Models we built
Pretrained Comparison
Best Model


Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion

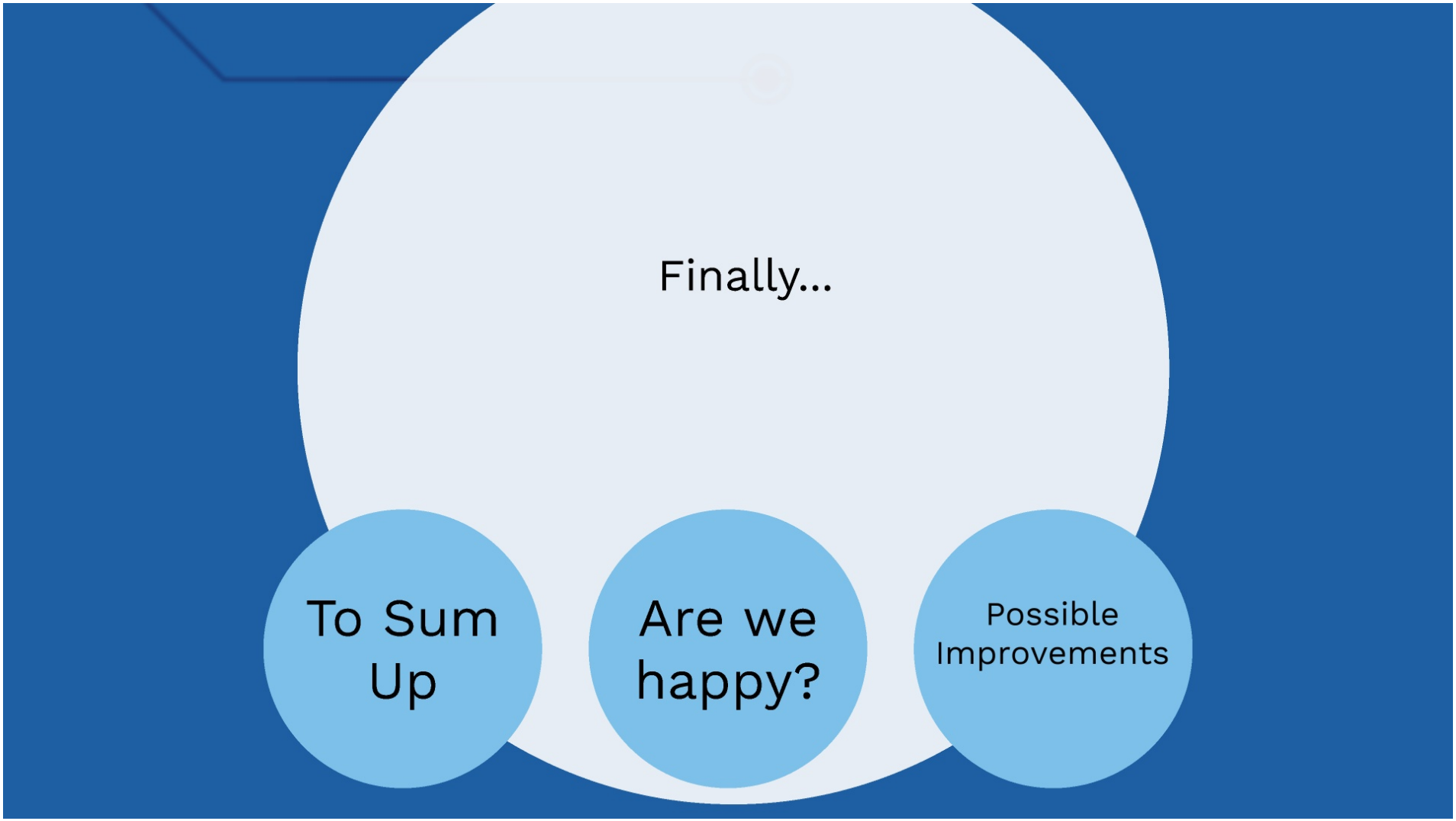

Finally...
To Sum Up
Are we happy?
Possible Improvements

It was a great experience

What did we learn?

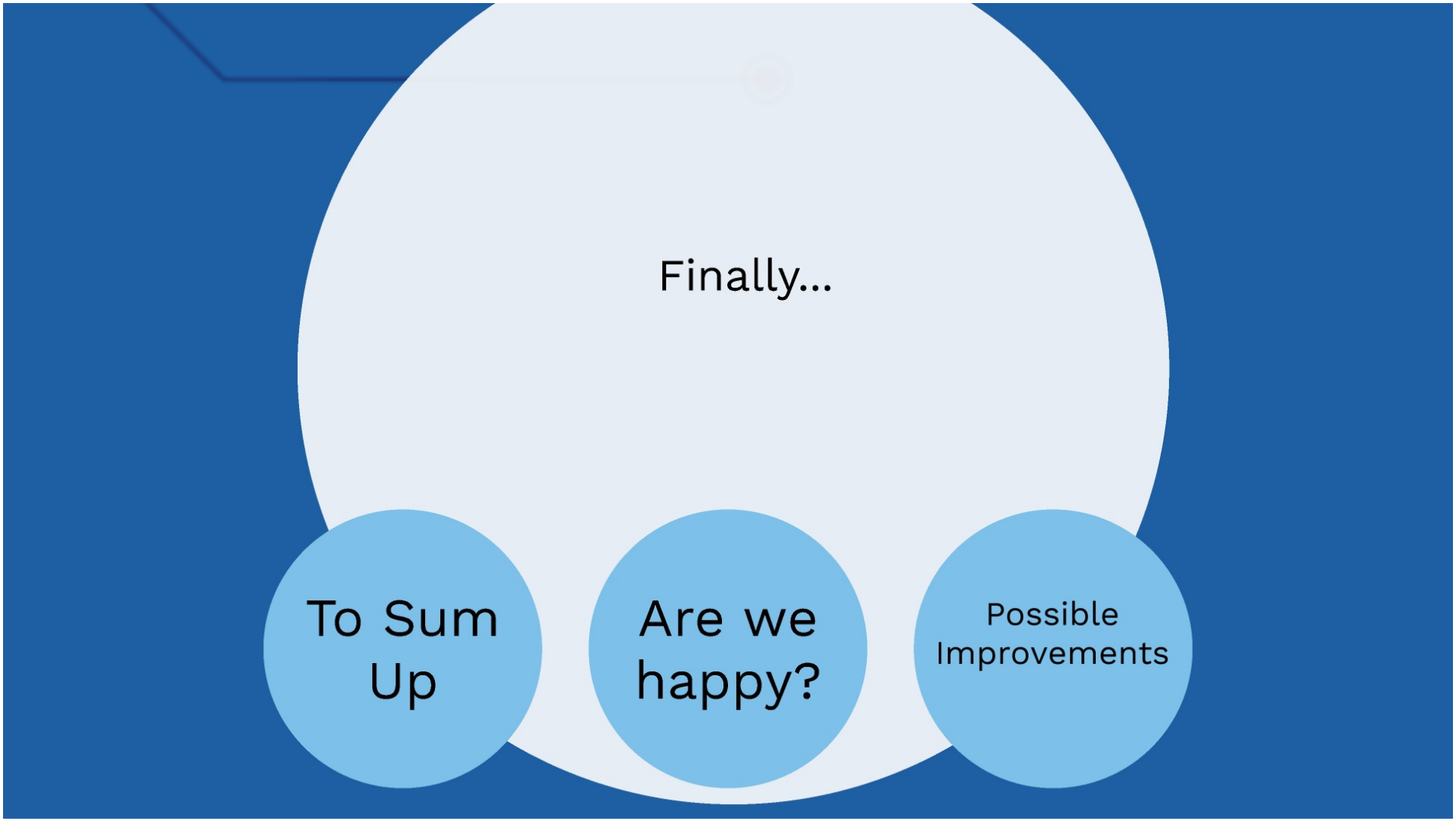
A first hand experience in

-Data Collection
-Various ML Algorithms
-Deep Learning techniques
-Neural Networks
-Hyper-parameter tuning
-Similarity Metrics


Finally...
To Sum Up
Are we happy?
Possible Improvements

"We can get no satisfaction"

We would have loved to have even more time to get our hands in depth, but in the end it is a project which allowed us to see how far could we get, which are our strengths and which one are our weaknesses


Finally...
To Sum Up
Are we happy?
Possible Improvements

Possible improvements of our work

-Larger and more diverse Dataset
-Improved training
-Ensemble methods can be explored
-More advanced similarity methods for scalability

	Team Member	Contribution
The Outliers	Raul Fabio Riva	Content Creator/Research & Reference
		Statistical Modelling
		Fine Tuning
	Swaviman Kumar	Collaborative filtering approach
		VGG16 Transfer Model
		MobileNet Transfer Model
		Ensemble learning
		ANNOY Indexing for image similarity
	Md Ashraful Alam Hazari	Prepare Dataset
		Prepare Dataset Manually using Image downloader and Clean data
		Image Scraping
		Fine Tuning all the pre-trained models
		Test the models with our own gallery and query
	Shaker Mahmud Khandaker	Analysis and fine-tuning of the example SIFT Solution
		Implementation of CNN from Scratch
		Writing Re-usable codes to implement any models
		Implementation and fine-tuning of all Keras Pre-trained Models
		Implementation and fine-tuning of Transfer Learning Models:- VGG16- Resnet50- Resnet50v2- EfficientNetV2B3
		Implementation of LearningRateScheduler
		Implementation of PCA
		Implementation of Submission API

The Outliers

Thanks for the attention :)

Shaker Mahmud Khandaker

Machine Learning Engineer

Raul Fabio Riva

Research & Development,
Content Creator

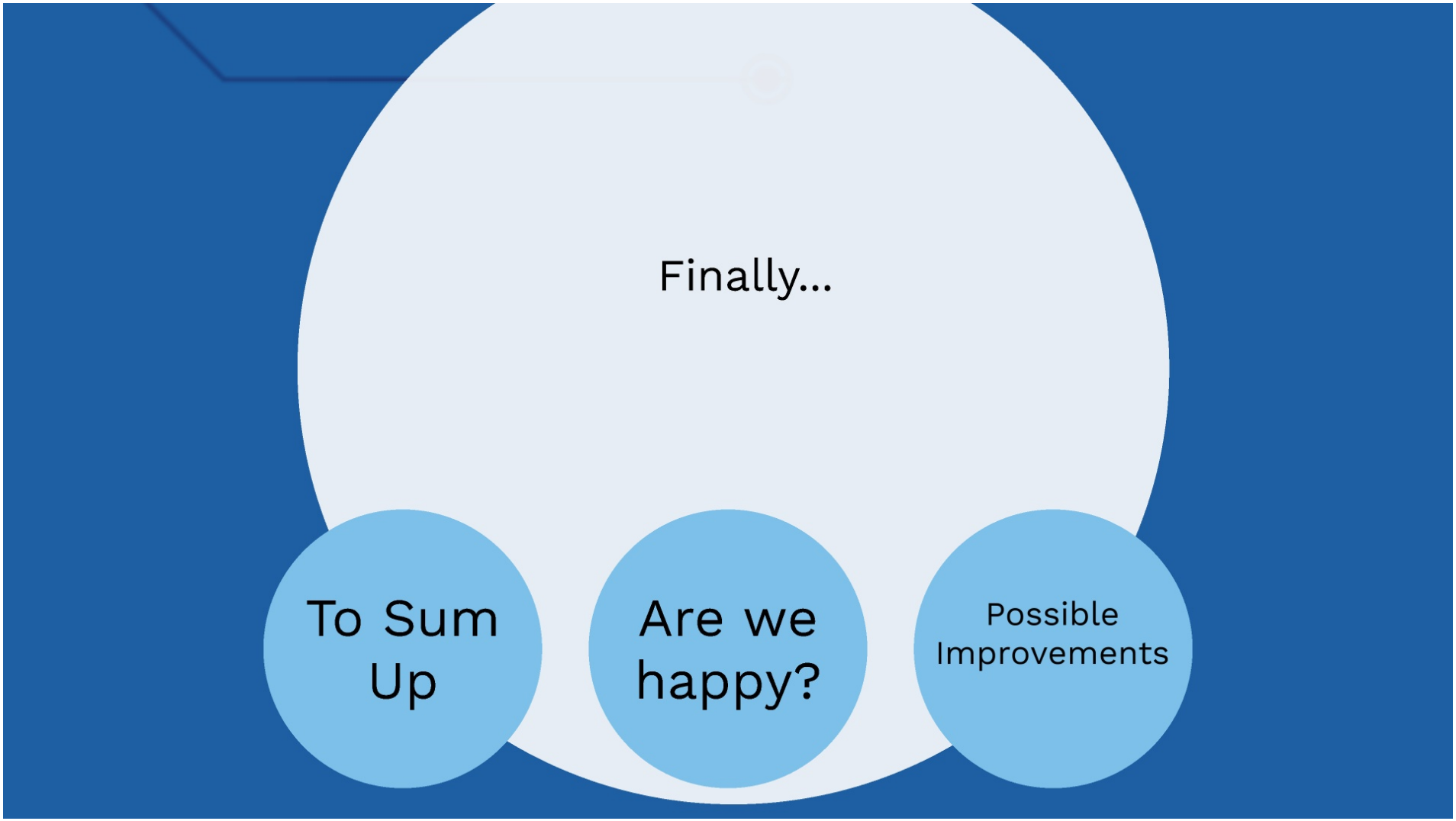
Ashraful Alam Hazari

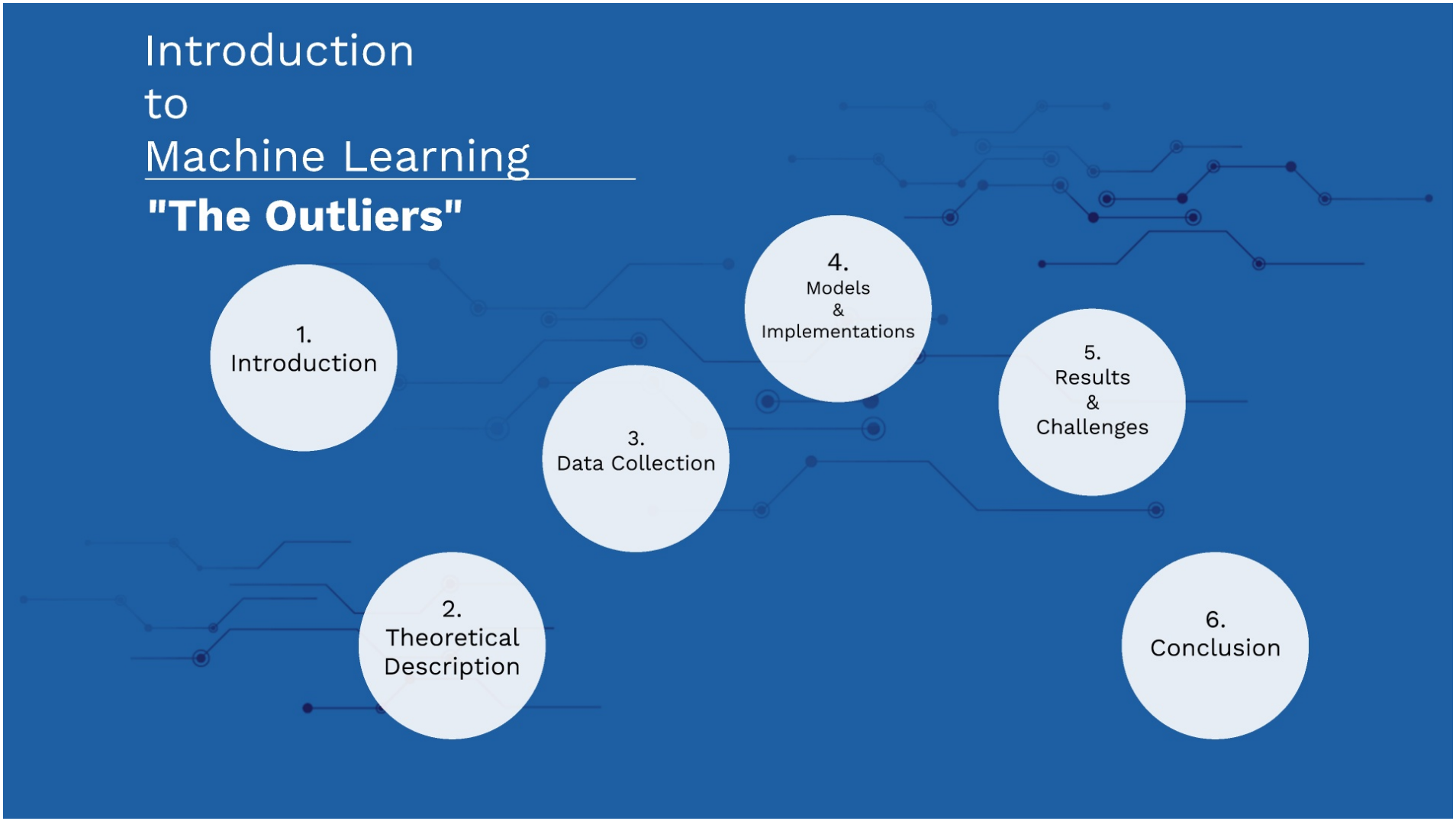
Data Miner

Swaviman Kumar

Software Engineer

Github: https://tinyurl.com/TheOutliersML


Finally...
To Sum Up
Are we happy?
Possible Improvements


Introduction
to
Machine Learning
"The Outliers"
1.
Introduction
2.
Theoretical
Description
3.
Data Collection
4.
Models
&
Implementations
5.
Results
&
Challenges
6.
Conclusion