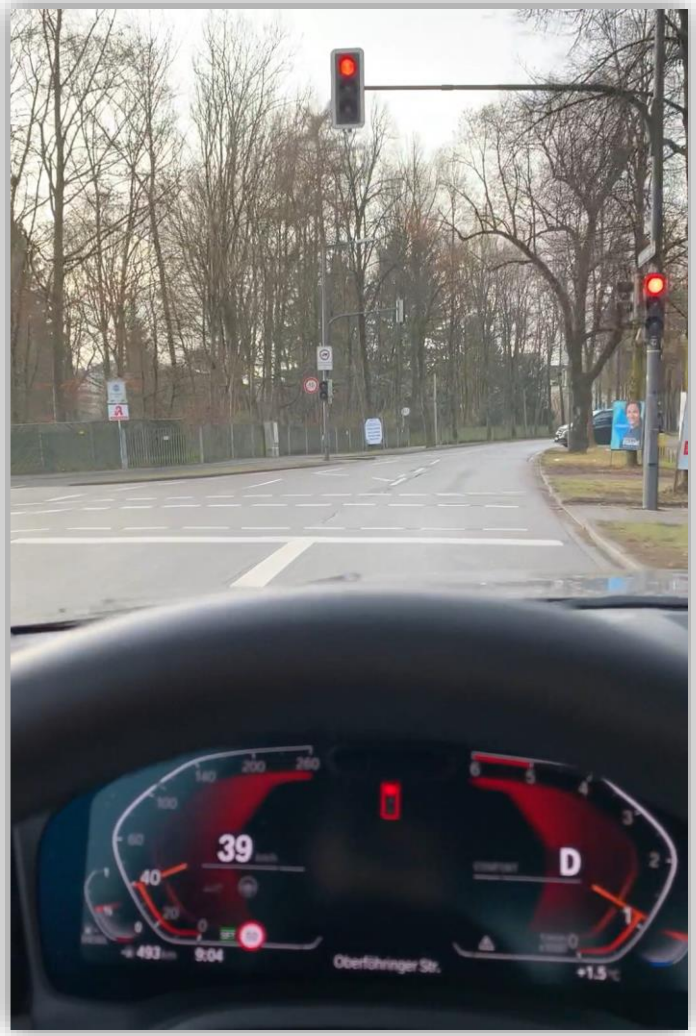
2021 embedded vision summit
VIRTUAL | MAY 25-27

Is My Model Performing Well? It Depends…

Vladimir Haltakov
BMW Group

How to Develop a Good Machine Learning Model?

- Experience with **traffic light recognition** in production
  - Red Light Warning
  - Urban Cruise Control
- Real world **challenges**
  - Optimizing the **customer function** instead of the model
  - **Conflicting requirements** in the project



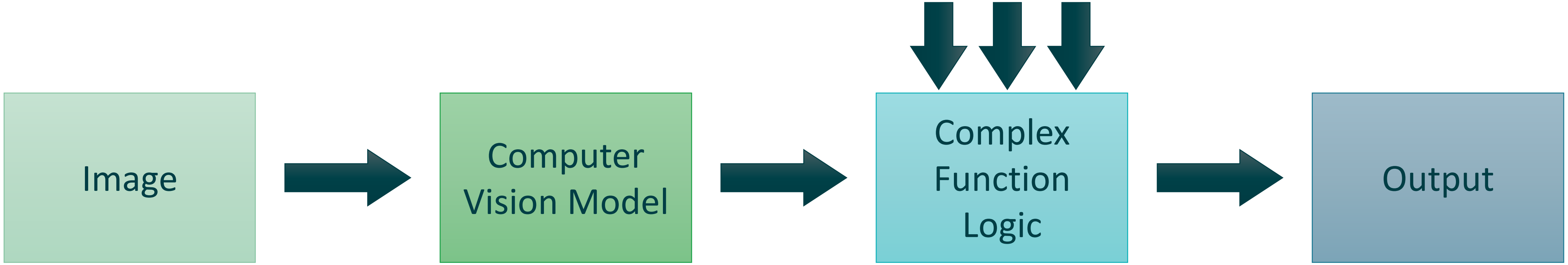
Classic Computer Vision Pipeline

Image → Computer Vision Model → Output

Well known **loss functions** and **evaluation measures**

Problem	**Loss Function**	**Evaluation Metrics**
Classification	Cross Entropy Loss	Recall, Precision, F1 Score
Regression	Mean Squared Error Loss	Mean Squared Error
Segmentation	Pixel-wise Cross Entropy Loss	Intersection-over-Union

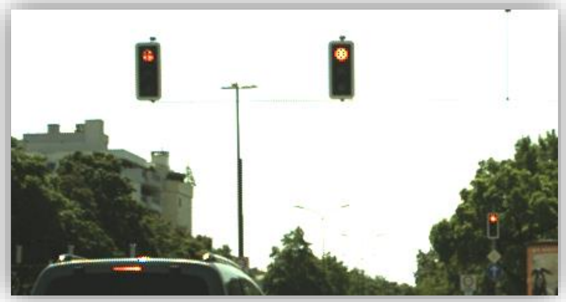
Complex Computer Vision Pipeline



- How does **the model accuracy** correlate with the **function accuracy**?
- How do **false positives** and **false negatives** influence the **output**?
- Which **loss function** to choose for **training**?
- When is my computer vision model **good enough**?
- Why did I **improve** the **accuracy** of the model, but the **application** got **worse**?

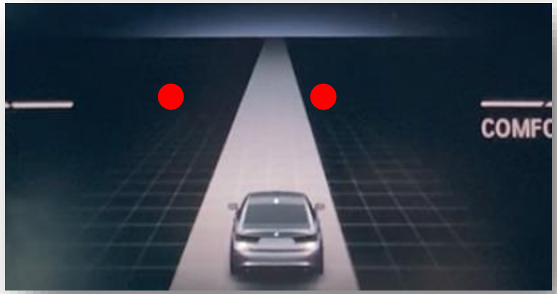
Example Pipeline – Traffic Light Warning

Image → Computer Vision Model → Complex Function Logic → Output



camera image

detected traffic lights



3D estimation + fusion with map

red light warning

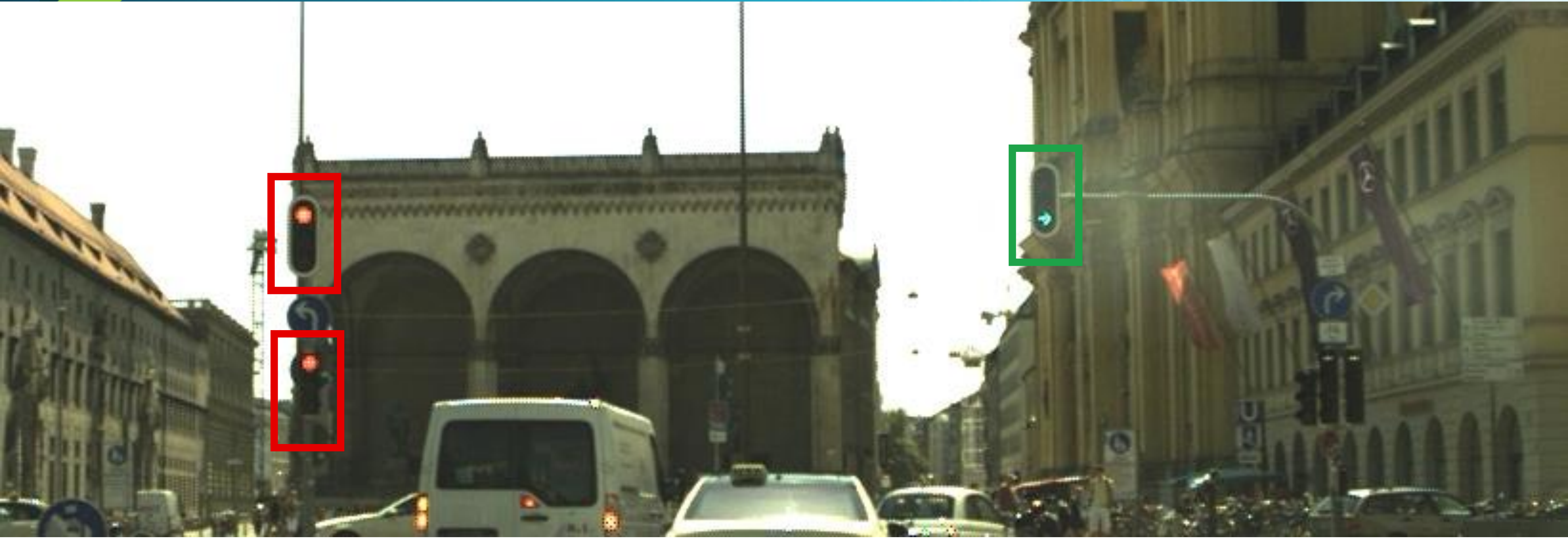
Example Scenario – Redundant Traffic Lights

False negatives and false positives **don't hurt** the **performance** a lot

Example Scenario – Single Traffic Light

False negatives and green false positives **reduce** the function **availability**

Example Scenario – Different Signal Groups



Green false negatives lead to **false warnings**

Example Scenario – No Traffic Light

Red false positives lead to **false warnings**

BMW GROUP

The Problem

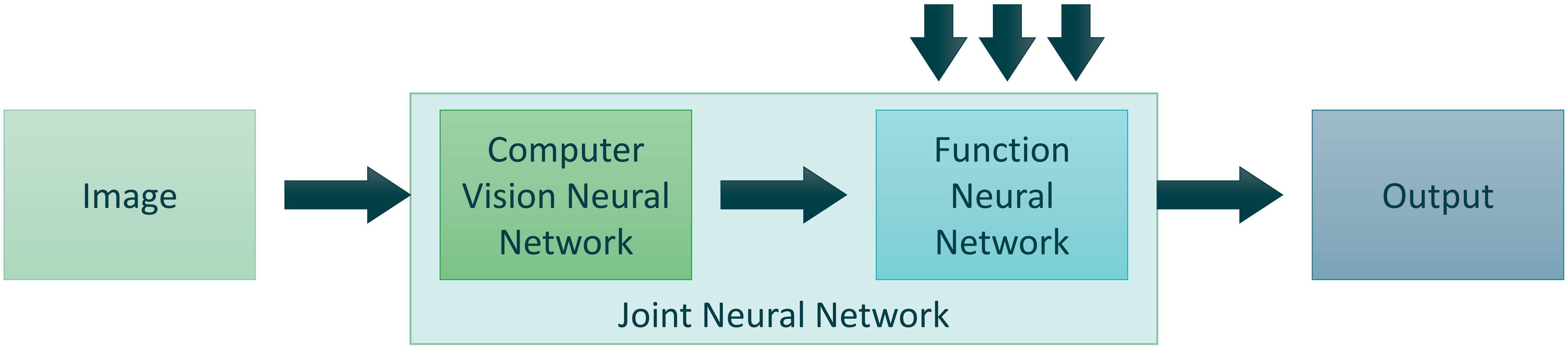
Measurable **model problems** are **not directly correlated** to problems on the **output**.

Sometimes **improvement** of the model's **accuracy** lead to **regression in the application**

Strategies to deal with the problem

1. Formulate a **joint optimization**
2. Adapt **the loss function** to the problem
3. Optimize **application specific error** measures

1. Formulate a Joint Optimization



- Model the **function logic** as a **neural network**
- Optimize the **computer vision** and the **function logic** networks **jointly**

✓ Get the **most performance** out of the data

✗ May be **infeasible** for complex functions and too **data intensive**

2. Adapt the Loss Function to the Problem

Add **application specific** terms to the **loss function** of the model

Examples for the traffic light warning function

- Penalize **FN more** for big traffic lights (closer to the car)
- Penalize **FN less** if there are other objects of the same type
- Penalize **FP less** if there are already objects of the same type present

✓ Constraints can be **automatically optimized**

✗ **Not always** possible

✗ Hyper-parameter **tuning is difficult** (see Resources for help)

3. Optimize Application Specific Error Metrics

Work on solving **only problems** of the model that **hurt the application**

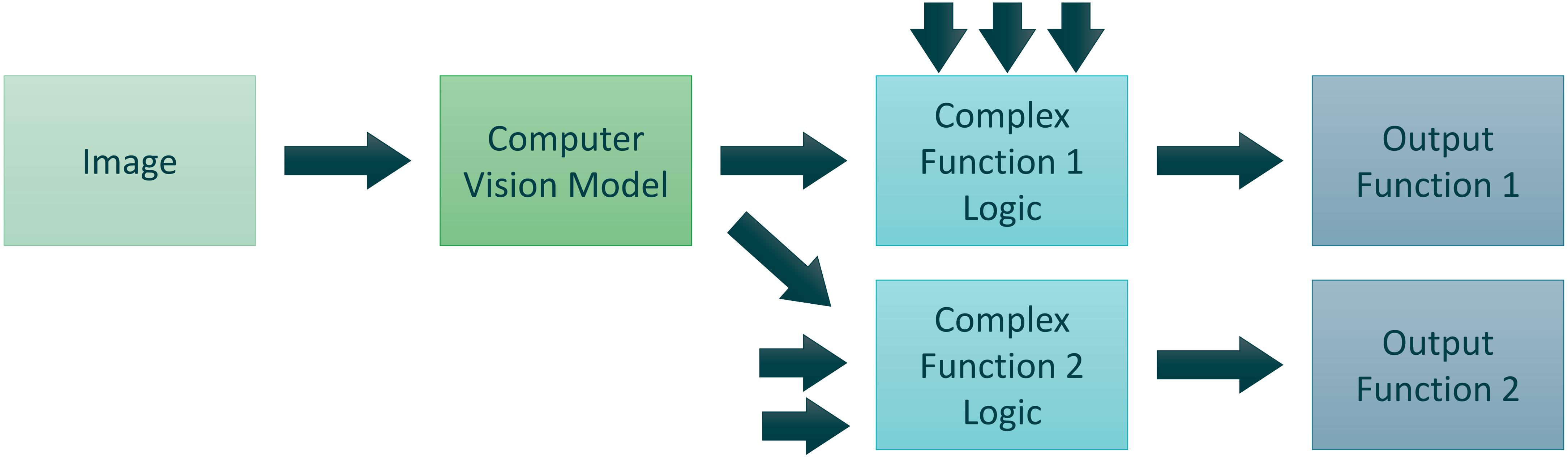
1. **Evaluate** the **final output** according to the **application specific** metrics
2. **Review** the **error cases** and **categorize** the model's issues
3. Work on **solving** these **categories** without reducing overall performance

Example

10x False Warning → 6x FP red light, 2x FN green light, 5x Wrong distance estimation

20x Missed Warning → 5x Wrong distance estimation, 13x FN red light, 4x FP green light

Conflicting Requirements

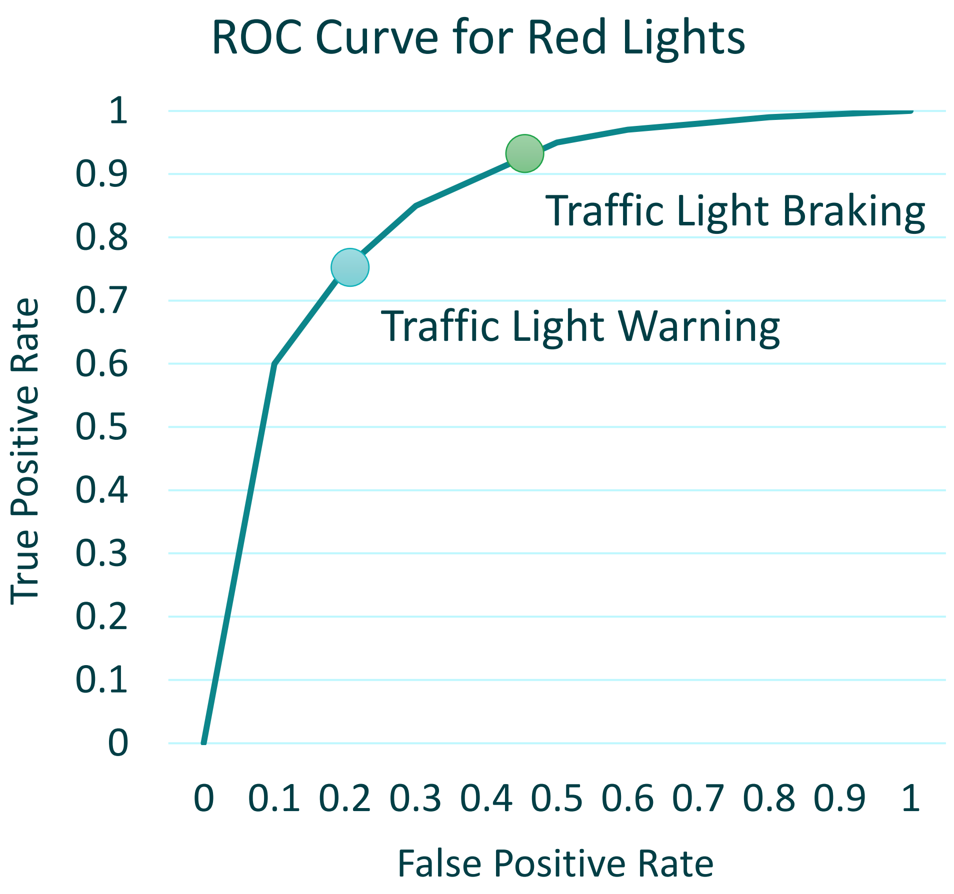


- Conflicting requirements **depending** on the **scenario**
- Conflicting requirements **between functions**

Example: Traffic Light **Warning** vs. Traffic Light **Braking**

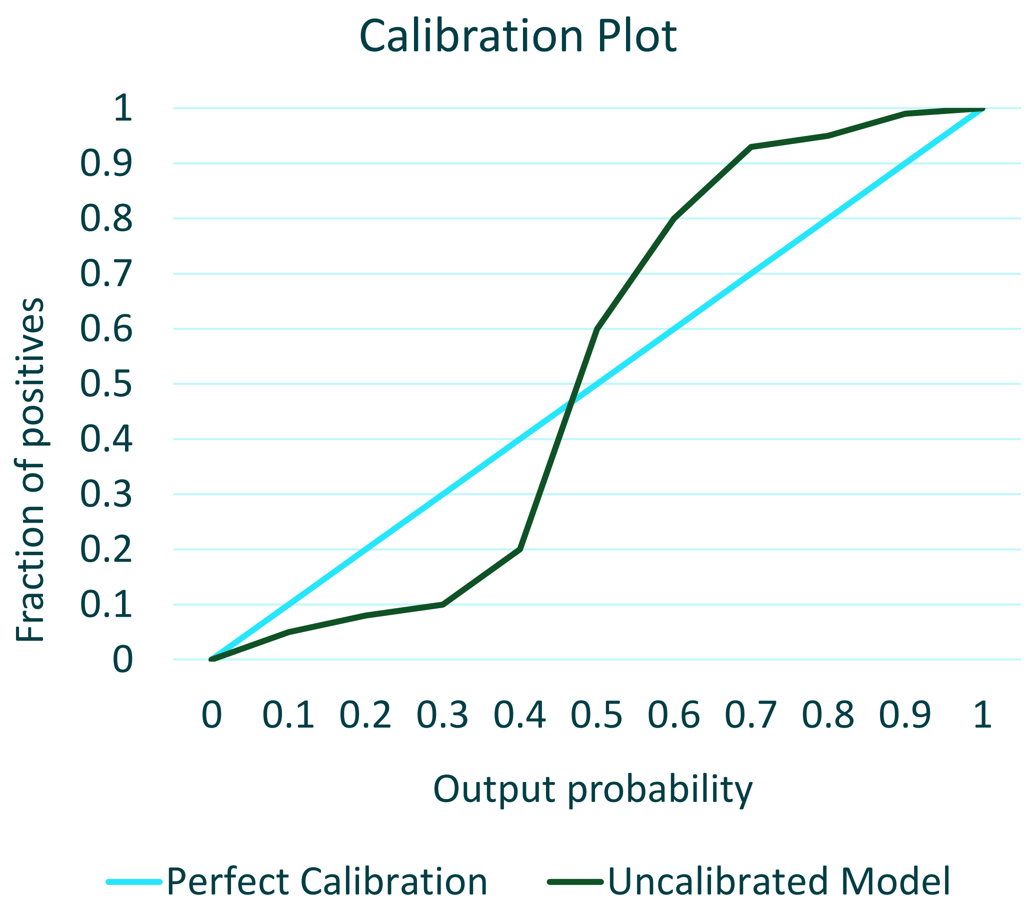
Dealing with Conflicting Requirements

- Provide **more information** from the model
  - Existence **probabilities**
  - Classification **scores**
  - Position **covariances**
  - Tracker **uncertainties**
- The functions need to **choose which data** to use
- The functions are able to **control** the **trade-offs**



Model Calibration

- Interpret the **output** of the model as a **probability**
- The **probability** usually **doesn't match** the real distribution
- Most models are **not calibrated** (e.g. **neural networks**, SVM, decision trees)
- Various **methods for calibration:** Platt Scaling, Isotonic Regression (see Resources)



Summary and Important Lessons

- In **complex systems**, the **accuracy** of the computer vision model **does not directly correlate** to the **performance** of the final application
- Define **problem specific evaluation metrics** and **loss** functions
- Provide **more information** from the computer vision model to handle **conflicting requirements**
- **Calibrate** the **model** output

Resources

Model tuning and calibration

How we can make machine learning algorithms tunable?

https://engraved.ghost.io/how-we-can-make-machine-learning-algorithms-tunable/

Model Calibration

https://scikit-learn.org/stable/modules/calibration.html

2021 Embedded Vision Summit

Watch my other talk

“Data Collection in the Wild.”