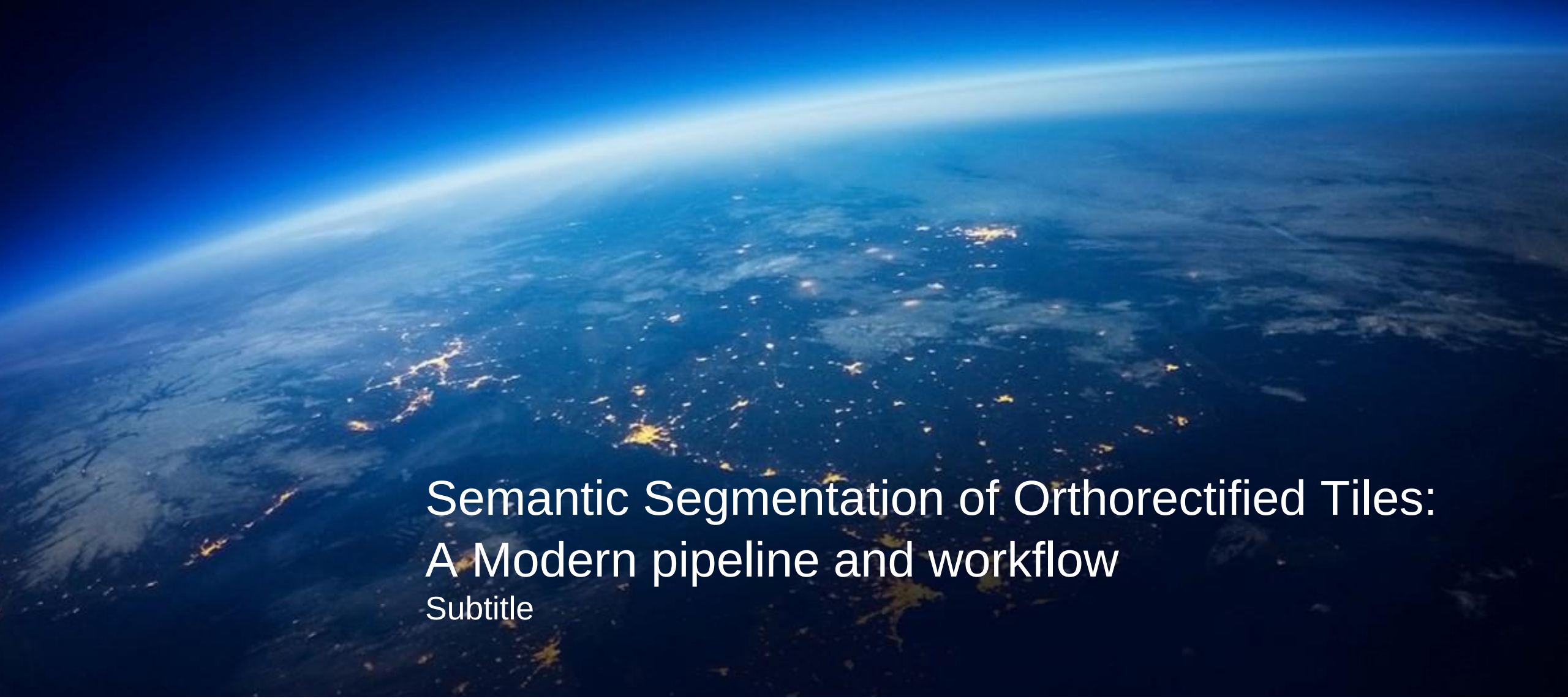


Semantic Segmentation of Orthorectified Tiles: A Modern pipeline and workflow

Subtitle

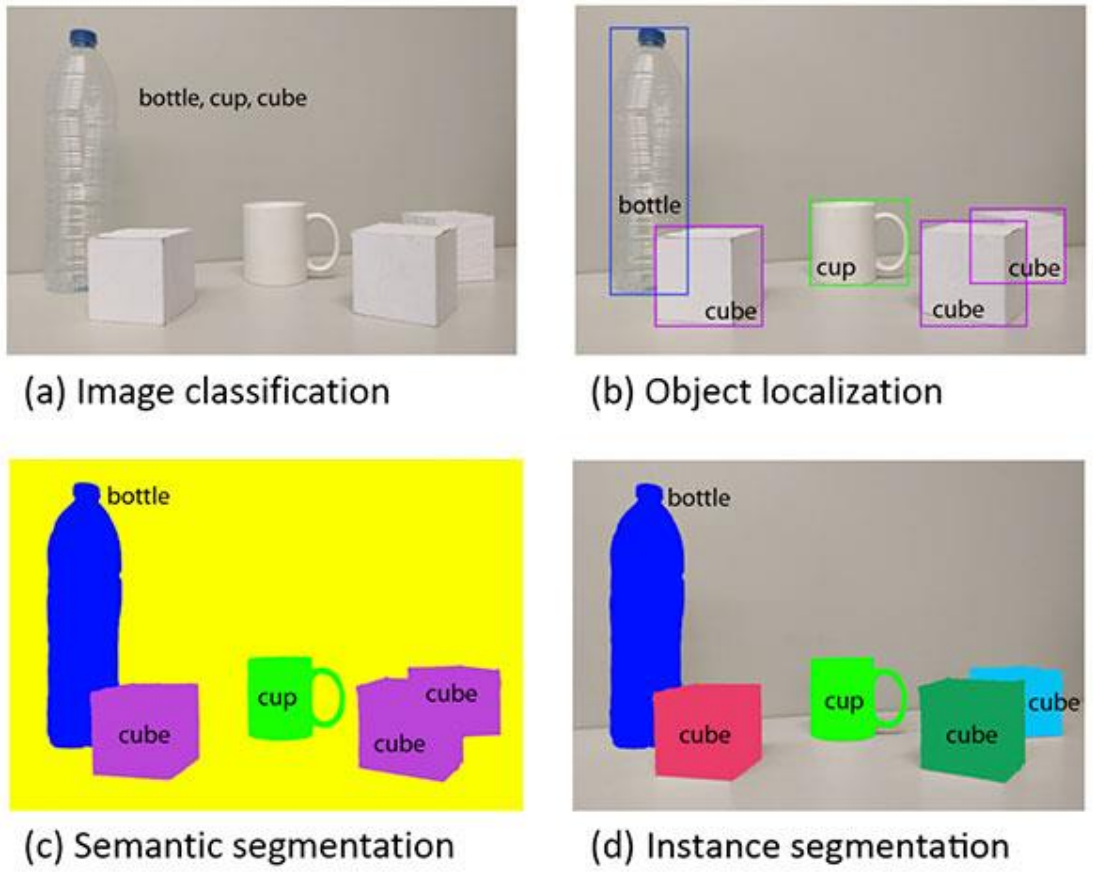
DEFENCE AND SPACE

Maher SEBAI – TIN3D

December 2019

AIRBUS

What is Semantic Segmentation in Deep Learning Context

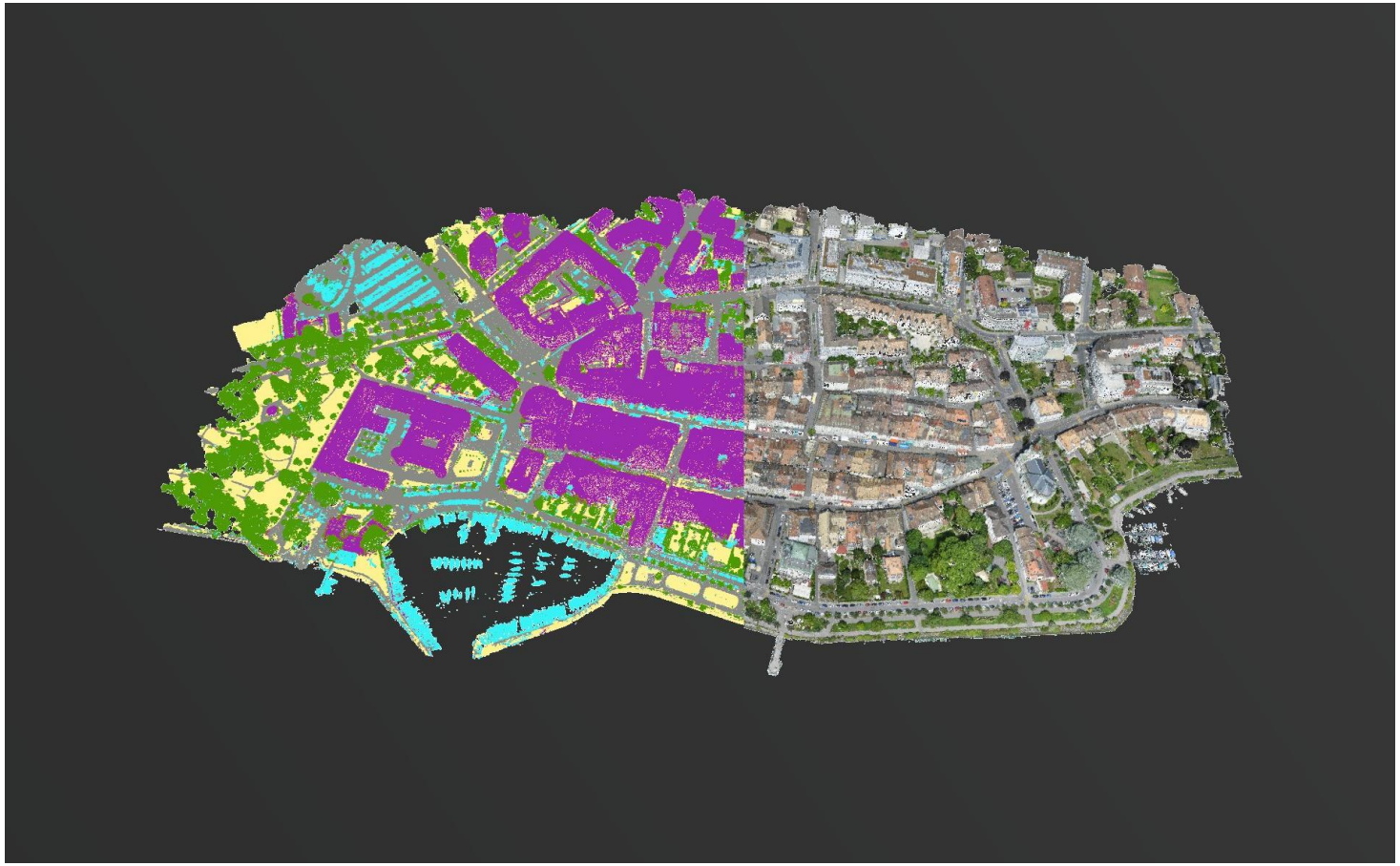


 Source: https://arxiv.org/abs/1704.06857

AIRBUS

Smart 3D City: Potential Benefits

- Product added Value
  - Point Cloud Semantic
  - Urban planning
  - Statistics …

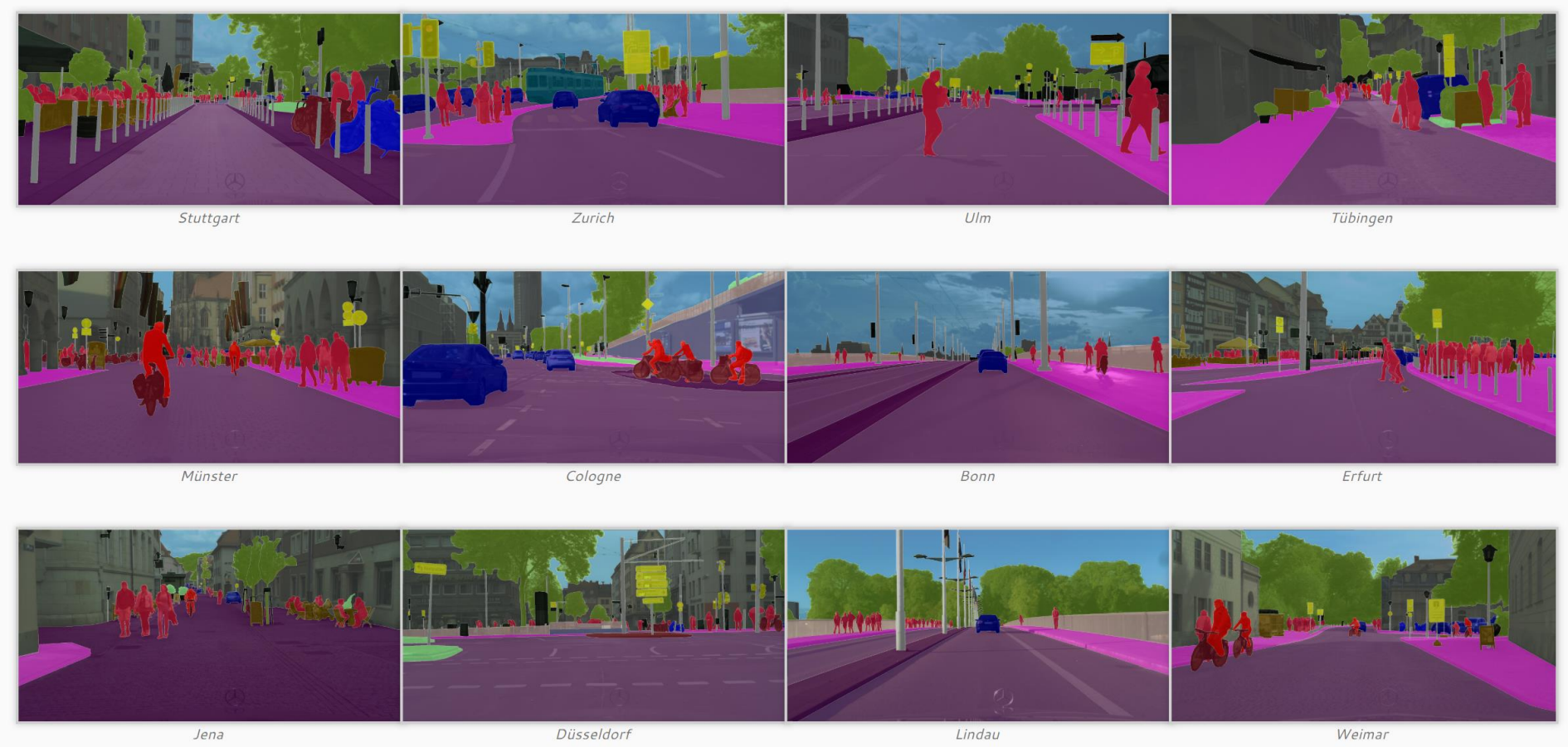


 Source: Pix4D

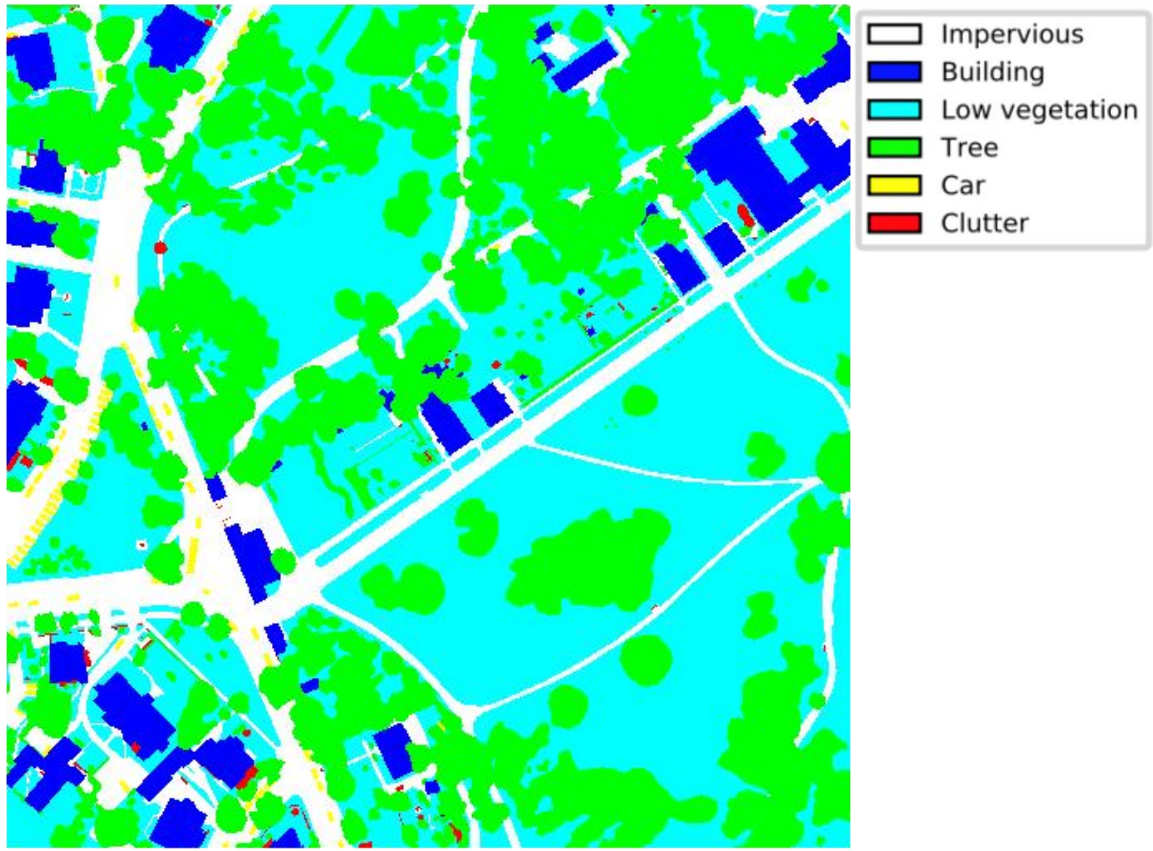
Supervised Semantic Segmentation Task Examples: MS COCO Dataset

 https://github.com/nightrome/cocostuff

Supervised Semantic Segmentation Task Examples: Cityscapes Dataset

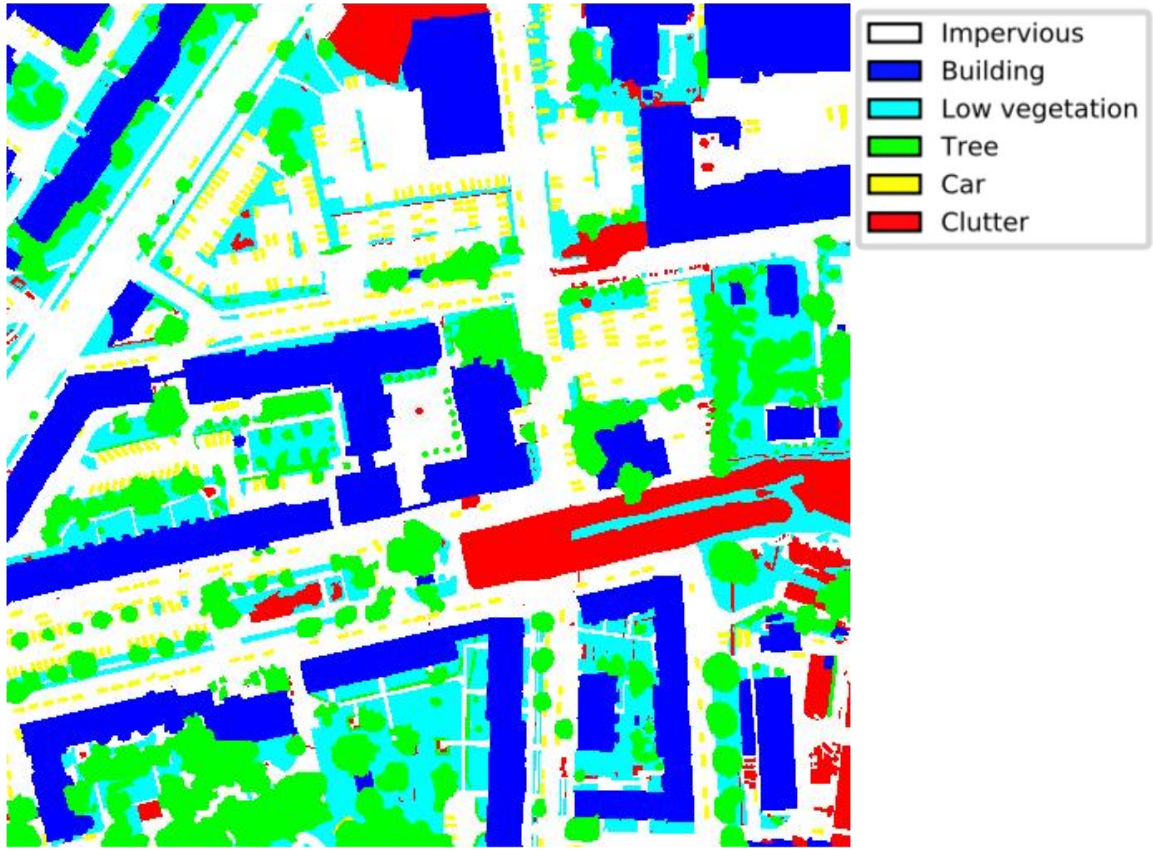


Supervised Semantic Segmentation Task Examples: ISPRS dataset (Potsdam)



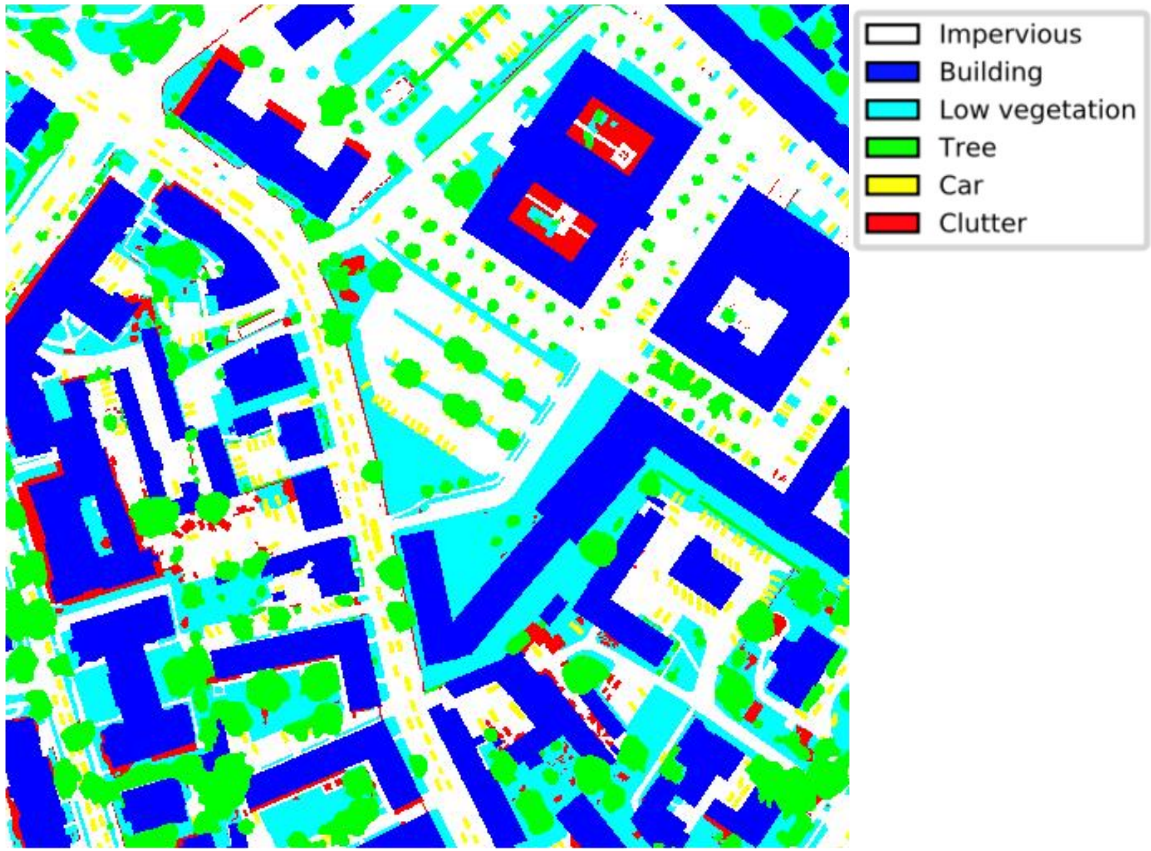
 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

Supervised Semantic Segmentation Task Examples: ISPRS dataset (Potsdam)



 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

Supervised Semantic Segmentation Task Examples: ISPRS dataset (Potsdam)



 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

ISPRS dataset details

Potsdam Benchmark dataset for ISPRS 2D Semantic labeling challenge in Potsdam:

- 38 tiles (6000x6000) at ≈ 5cm resolution
  - 24 Tiles dedicated for training
  - 14 Tiles for validation
- 4-bands IRRGB + DSM + NDSM
- Ground truth mask for 6 classes:
  - Building
  - Impervious Surfaces
  - Low Vegetation Tree
  - Tree
  - Car
  - Background

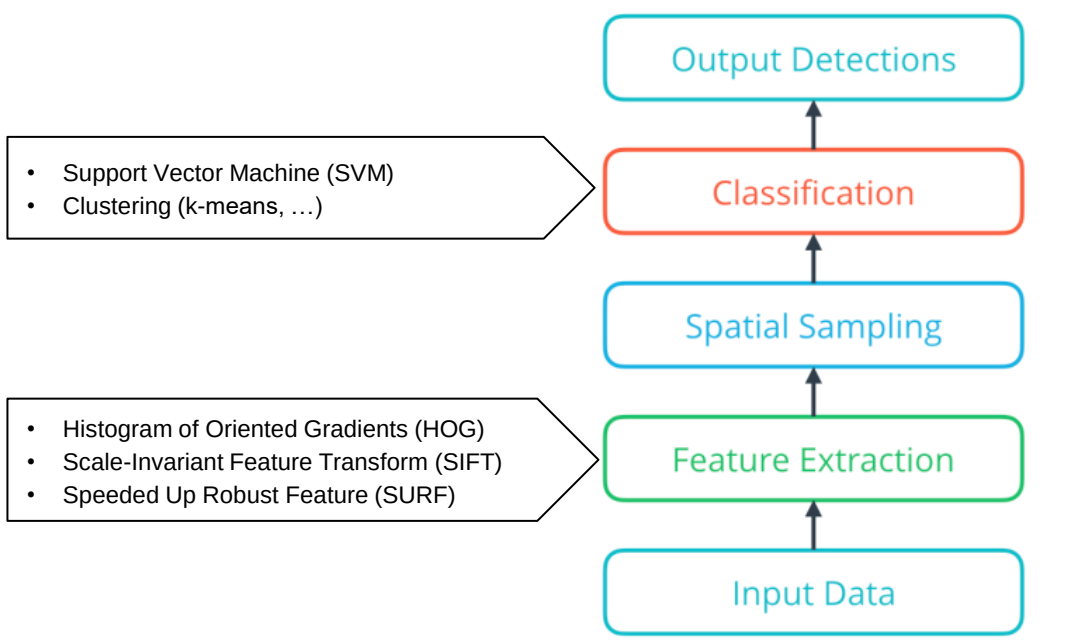
AIRBUS

Development environment

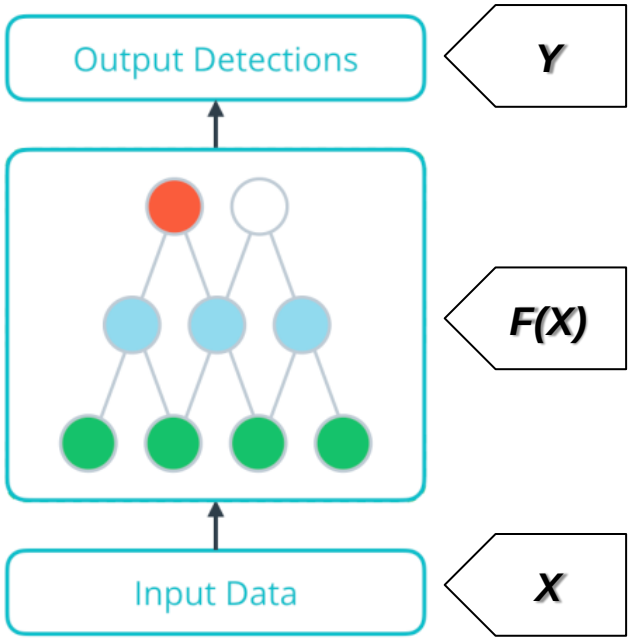
- Python 3.5
- Imageio 2.6
- Deep Learning Libraries:
  - Low-Level API: Tensorflow 2.0 (with eager_execution enabled)
  - High-Level API: Keras 2.2
  - Input pipeline API: Tensorflow.data
  - Monitoring API: TensorBoard

- Infrastructure:
  - 16-Core, 64GB RAM
  - Nvidia 16GB GPU (Tesla P100)
  - VM instance on GCP

Computer Vision Segmentation Approaches:



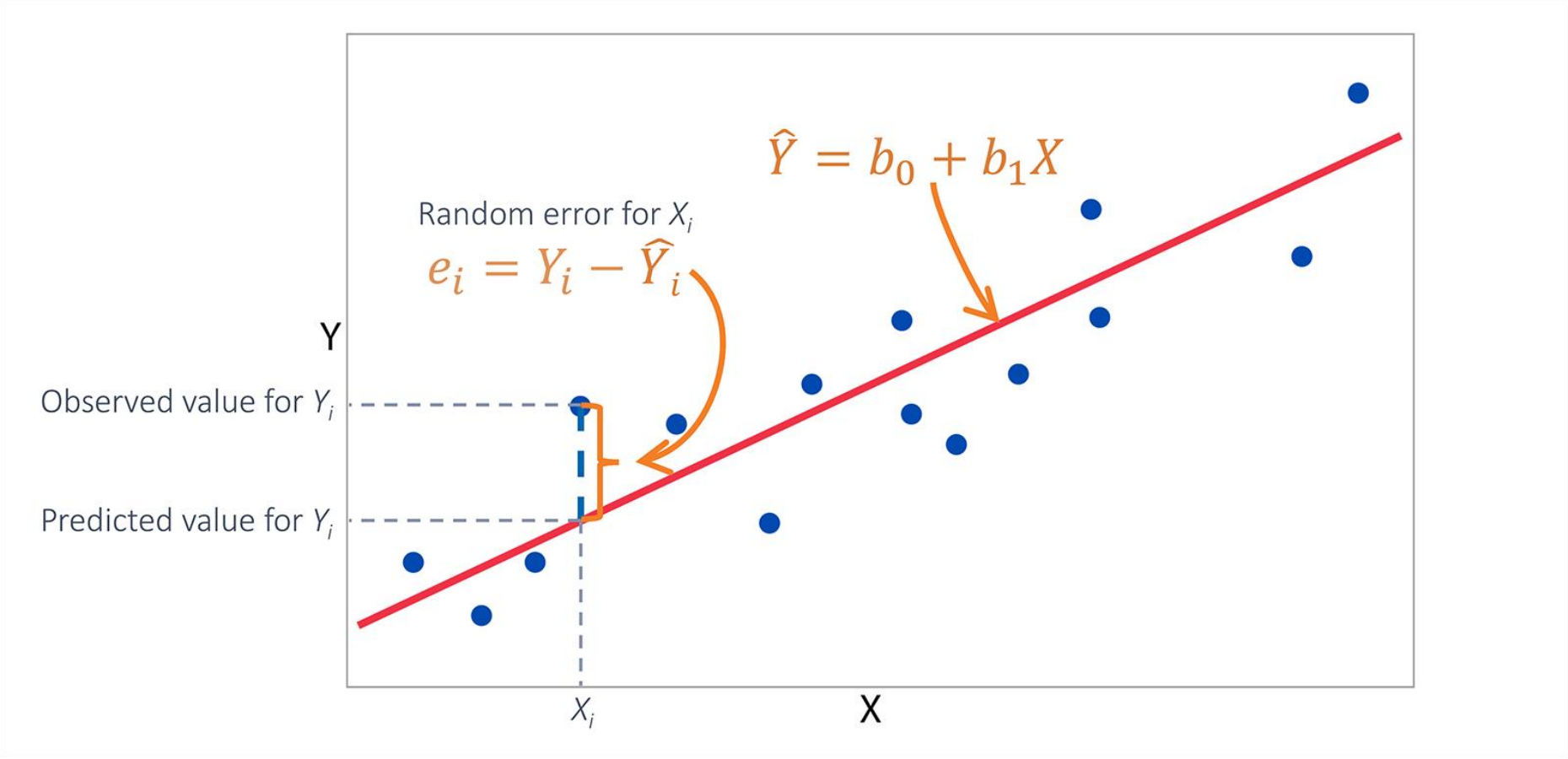
Traditional Computer Vision



Deep Neural Network

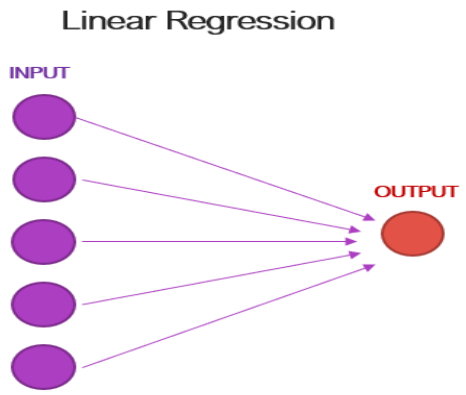
AIRBUS

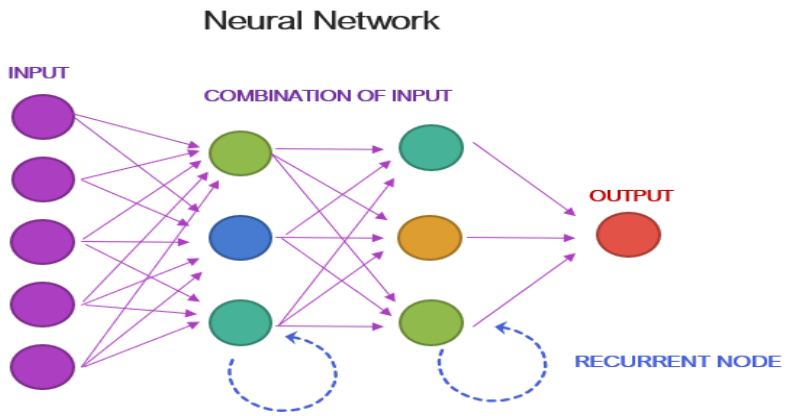
Parametric Modeling: Linear Regression



Linear Regression Vs Neural Networks:

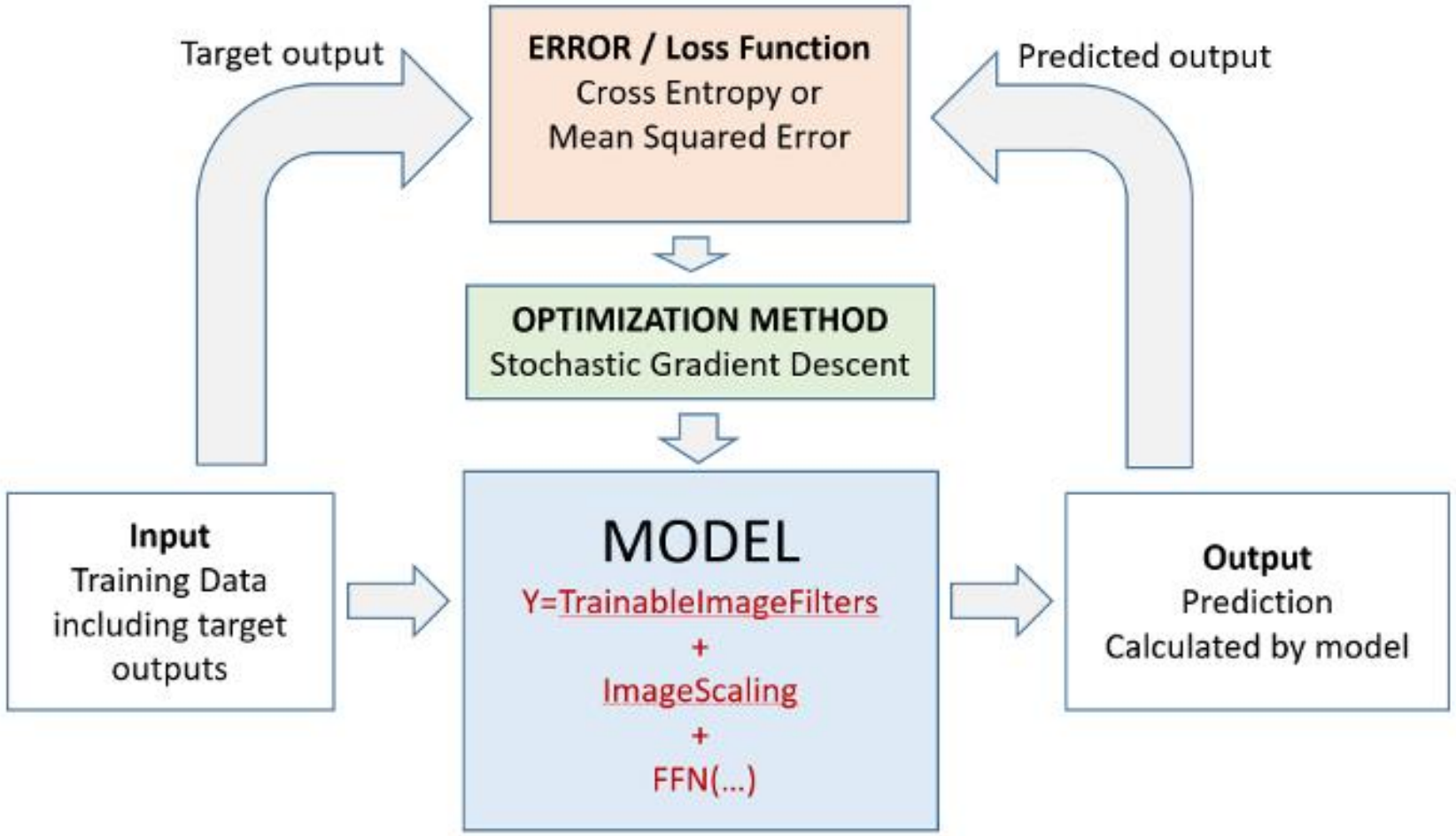
	Linear Regression	Neural Networks
Supervised Learning	Explanatory variables X Response Variables Y	Input Aerial image Pixels X Reference Mask Pixels Y
Parametric Model	$Y = b_0 + b_1 X$	Pixel Label = $F(X\|W_1W_2W_3W_4\ldots)$
Minimizing Loss Function (Error)	Mean Squared Error	Cross-entropy
Optimization Algorithm	Analytical Solution	Gradient Descent
Goodness of fit Metrics	R^2, F-test, p-value …	Accuracy, Jaccard Index, …





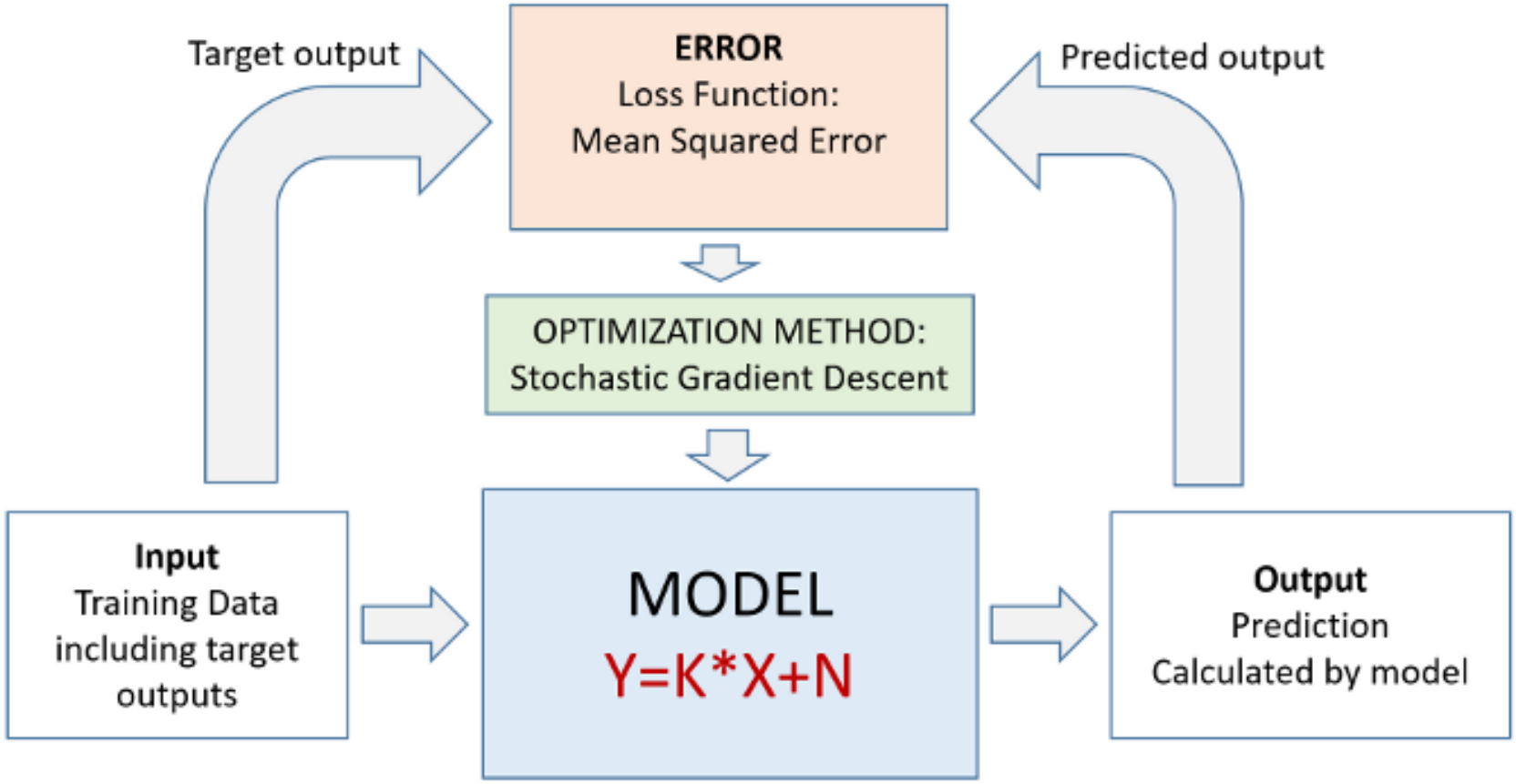
AIRBUS

Neural Network Training Flowchart



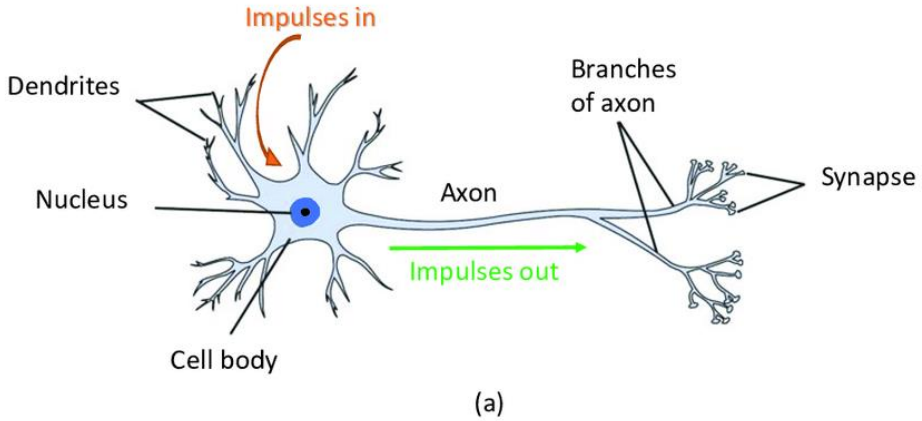
 http://www.deepnetts.com/blog/from-linear-regression-to-deep-learning-in-5-minutes.html

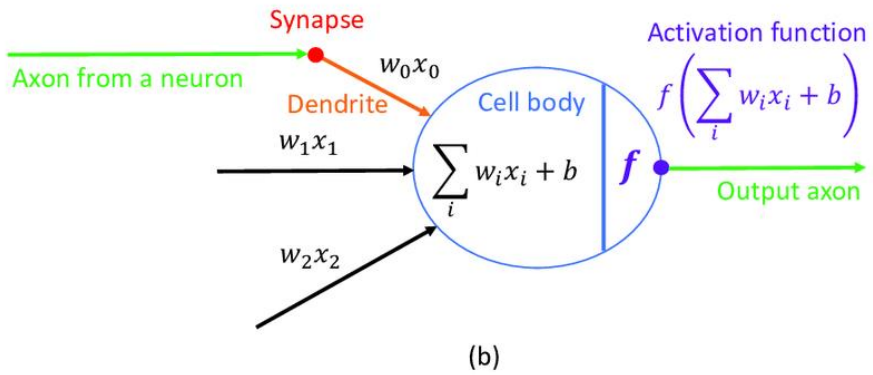
Linear Regression Training Flowchart:



 http://www.deepnetts.com/blog/from-linear-regression-to-deep-learning-in-5-minutes.html

Neural Networks Inspiration:

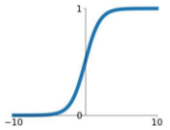




Activation Functions

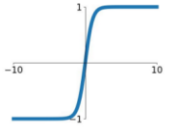
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



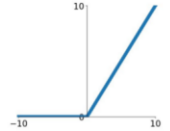
tanh

$\tanh(x)$



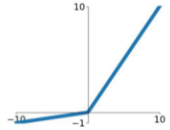
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

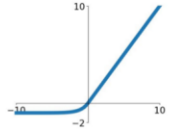


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

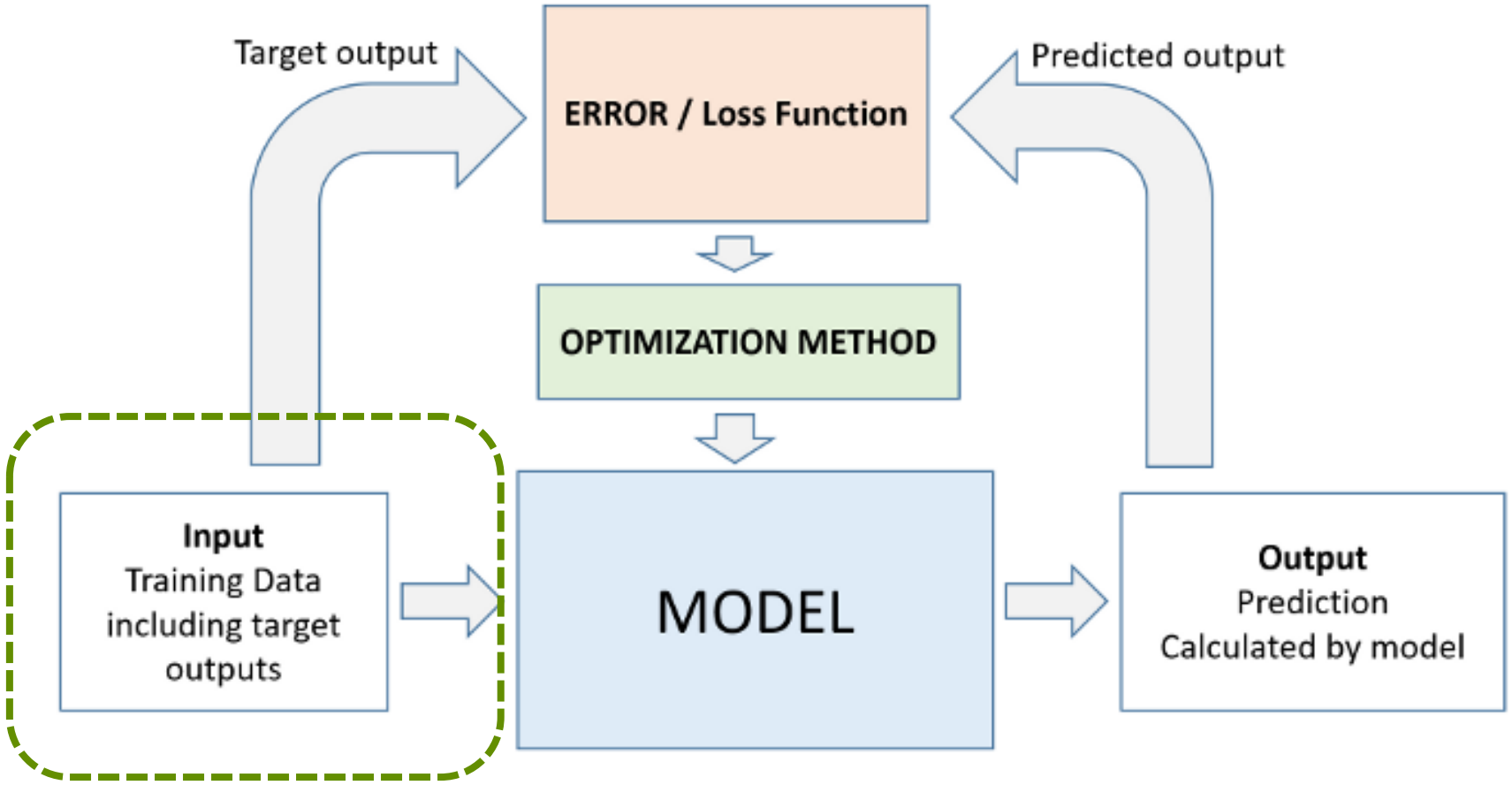
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



https://www.researchgate.net/figure/1-A-comparison-between-a-human-neuron-and-an-ANN-neuron-a-Shows-an-illustration-of-a_fig6_317679065

Input Pipeline: Tensorflow.data

Input Pipeline



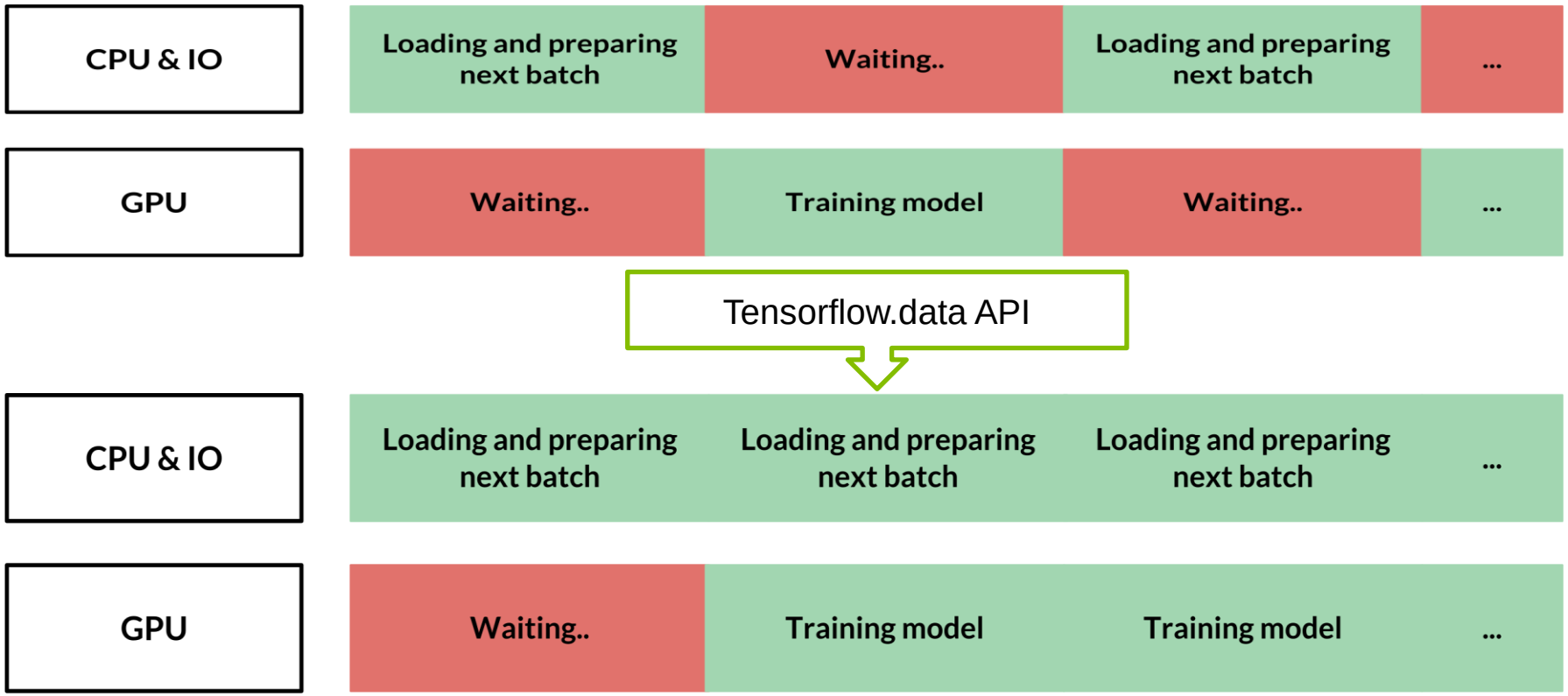
AIRBUS

Terminology:

- Training Dataset: The dataset that is used to compute the Loss function gradient to update the Models weights
- Validation Dataset: The dataset that is used to compute the Generalization Performance of the Model (Metrics)
- Epoch: One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
- Batch size: Total number of training examples present in a single batch.
- Example: a dataset made of 100 images with Batch size of 20 images => 1 epoch consists of 5 Batch iterations

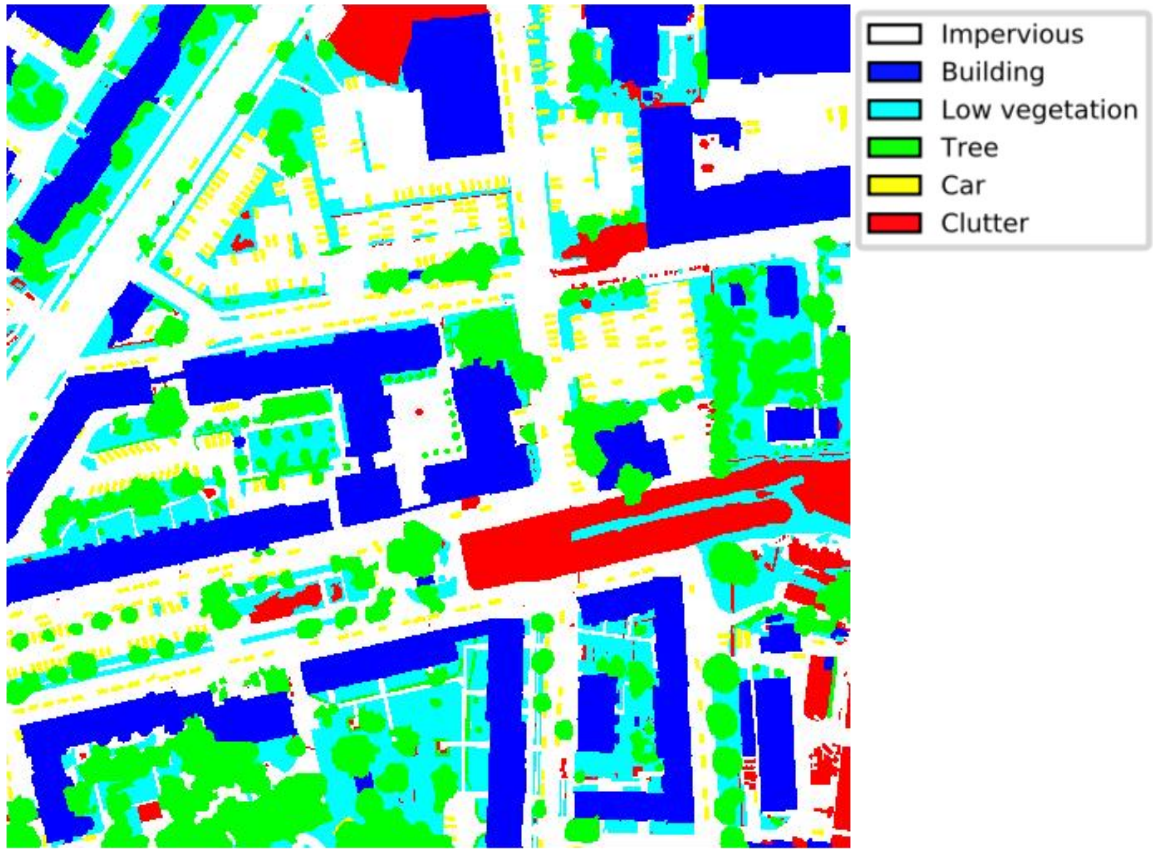
Input Pipeline challenges: avoid CPU-GPU bottleneck

- Goal: Decoupling data preprocessing from Accelerator Training



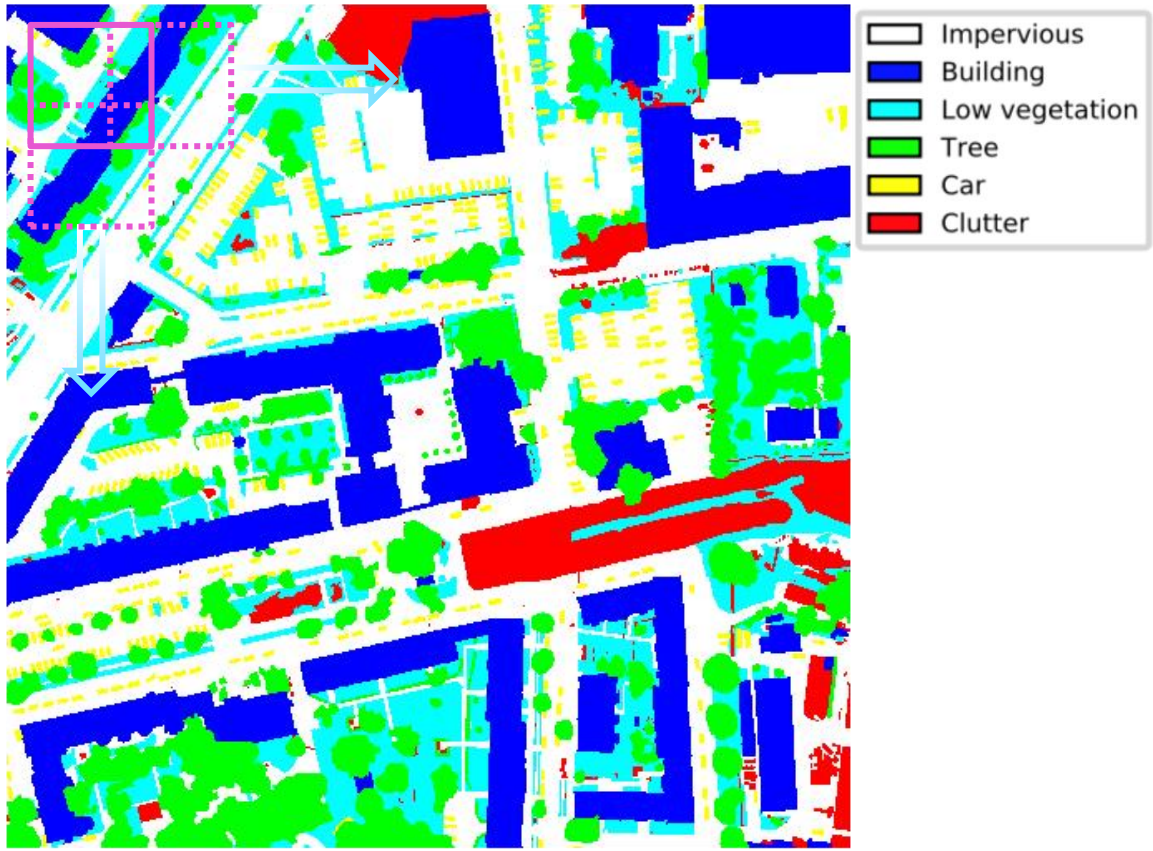
(Simplified, in reality there is always *some* idle time on both devices)

Patch Extraction



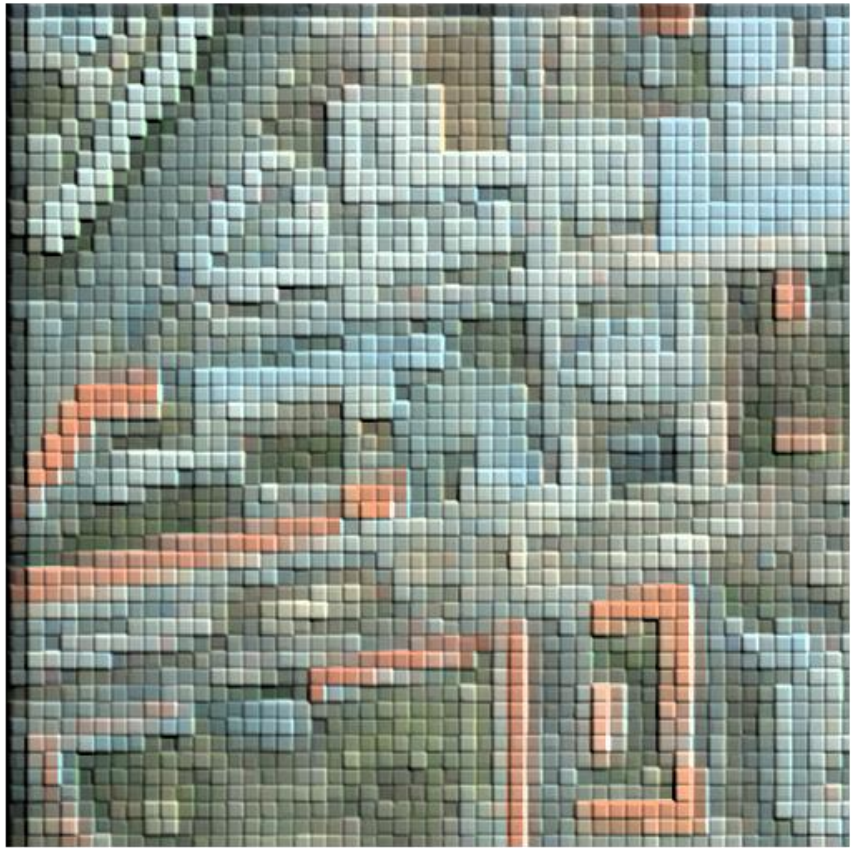
 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

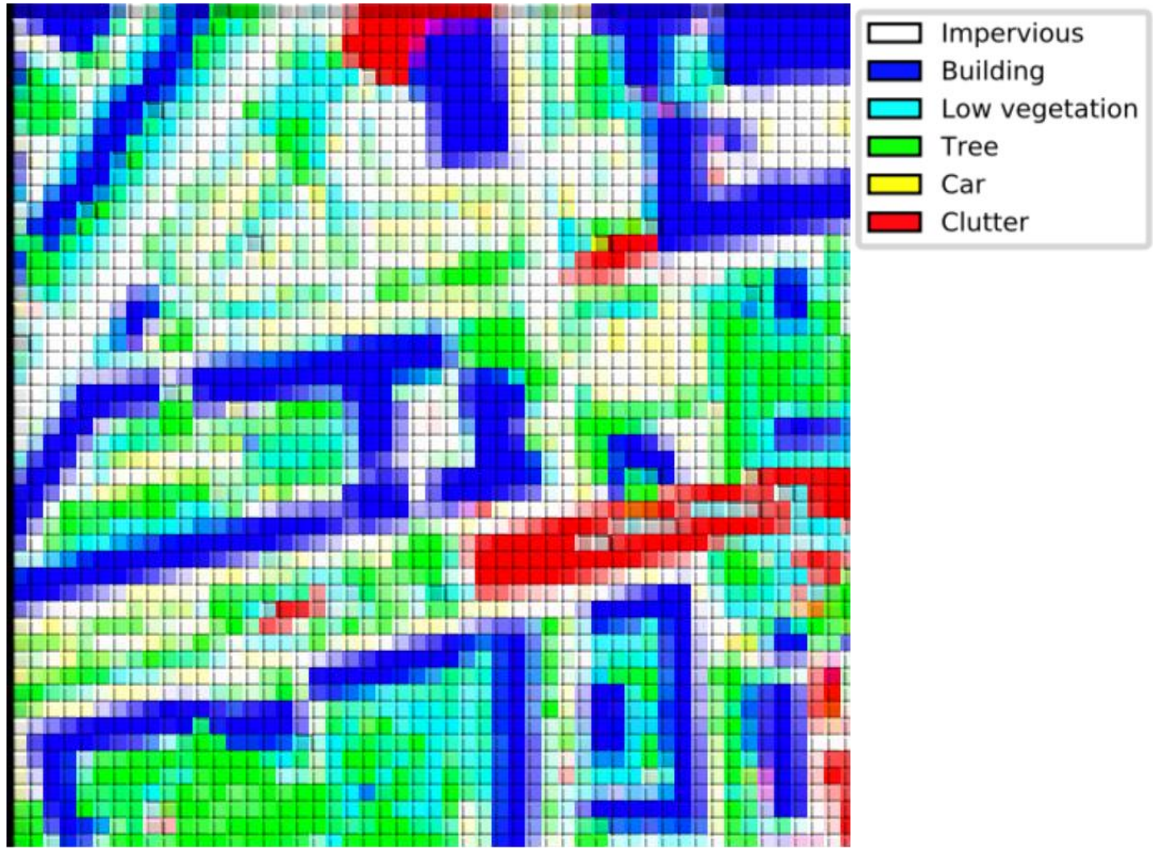
Patch Extraction



 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

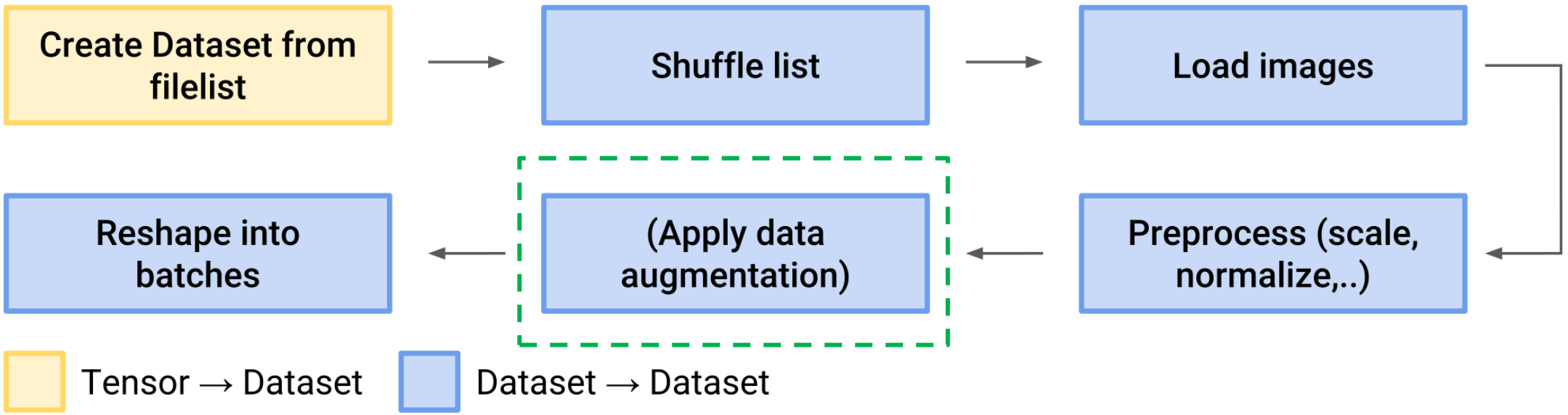
Supervised Semantic Segmentation Task Examples: ISPRS dataset (Potsdam)





 http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

- Practical advantage: Easy ETL (Extract-Load-Transform) chaining for complex data augmentation



Input Pipeline: Data Augmentation

- More data helps Model Performances
- Data Augmentation is a way to artificially produce more Annotated data at low cost to train the Model
- Apply same spatial transformation to both aerial and mask patch pair, color transformation only to aerial patch

Input Pipeline: implementation details

```
with tf.device('/CPU:0'):
    def get_dataset(img_dir, mask_dir, batch_size, shuffle=True, repeat=False, augment=True, scaled=True, cachefile=None):
        images_path  = Path(img_dir)
        masks_path   = Path(mask_dir)
        images_files = sorted([str(x) for x in images_path.iterdir() if x.is_file() and x.suffix == '.tif'])
        masks_files  = sorted([str(x) for x in masks_path.iterdir()  if x.is_file() and x.suffix == '.tif'])

        # checking the equality of the two list lenghts
        assert len(images_files) == len(masks_files), " (Error) The Aerial image count does not match Mask counts!"
        len_images = len(images_files)

        if cachefile:
            cache_file_name = '/tmp/' + cachefile
          else:
            cache_file_name = ''

        # images file names dataset
        images_ds = tf.data.Dataset.from_tensor_slices(images_files)
        masks_ds = tf.data.Dataset.from_tensor_slices(masks_files)
        .
        .
        .
```

Input Pipeline: implementation details

```
with tf.device('/CPU:0'):
    def get_dataset(img_dir, mask_dir, batch_size, shuffle=True, repeat=False, augment=True, scaled=True, cachefile=None):
        .
        .
        .

        # zipping the aerial and mask dataset
        patch_pair_ds = tf.data.Dataset.zip((images_ds, masks_ds))
        if shuffle:
            patch_pair_ds = patch_pair_ds.shuffle(len_images)
        patch_pair_ds = patch_pair_ds.map(tf_get_image, num_parallel_calls= AUTOTUNE).prefetch(AUTOTUNE)

        # start making patches
        if scaled:
            patch_pair_ds = patch_pair_ds.map(make_patches_scaled, num_parallel_calls= AUTOTUNE)
        else:
            patch_pair_ds = patch_pair_ds.map(make_patches, num_parallel_calls= AUTOTUNE)
        patch_pair_ds = patch_pair_ds.apply(tf.data.Dataset.unbatch).cache(filename=cache_file_name)
        if augment:
            patch_pair_ds = patch_pair_ds.map(make_augmentation, num_parallel_calls= AUTOTUNE).prefetch(AUTOTUNE)
        else:
            patch_pair_ds = patch_pair_ds.map(no_augmentation, num_parallel_calls= AUTOTUNE).prefetch(AUTOTUNE)
        # repeat is not necessary as keras fit method will use epochs parameter for repetition
        if repeat:
            patch_pair_ds = patch_pair_ds.repeat()
        if shuffle:
            patch_pair_ds = patch_pair_ds.shuffle(600)
        patch_pair_ds = patch_pair_ds.batch(batch_size)
        patch_pair_ds = patch_pair_ds.prefetch(AUTOTUNE)
        return patch_pair_ds
```

Input Pipeline Performance

Without Caching:

```
[24]: timepipeline(train_ds, batch_size=BATCH_SIZE)

100% 1000/1000 [03:40<00:00, 4.53it/s]

batche size: 50
1000 batches: 220.8209400177002 s
4.52856 Batches/s
226.42780 Images/s
```

With Disk Caching:

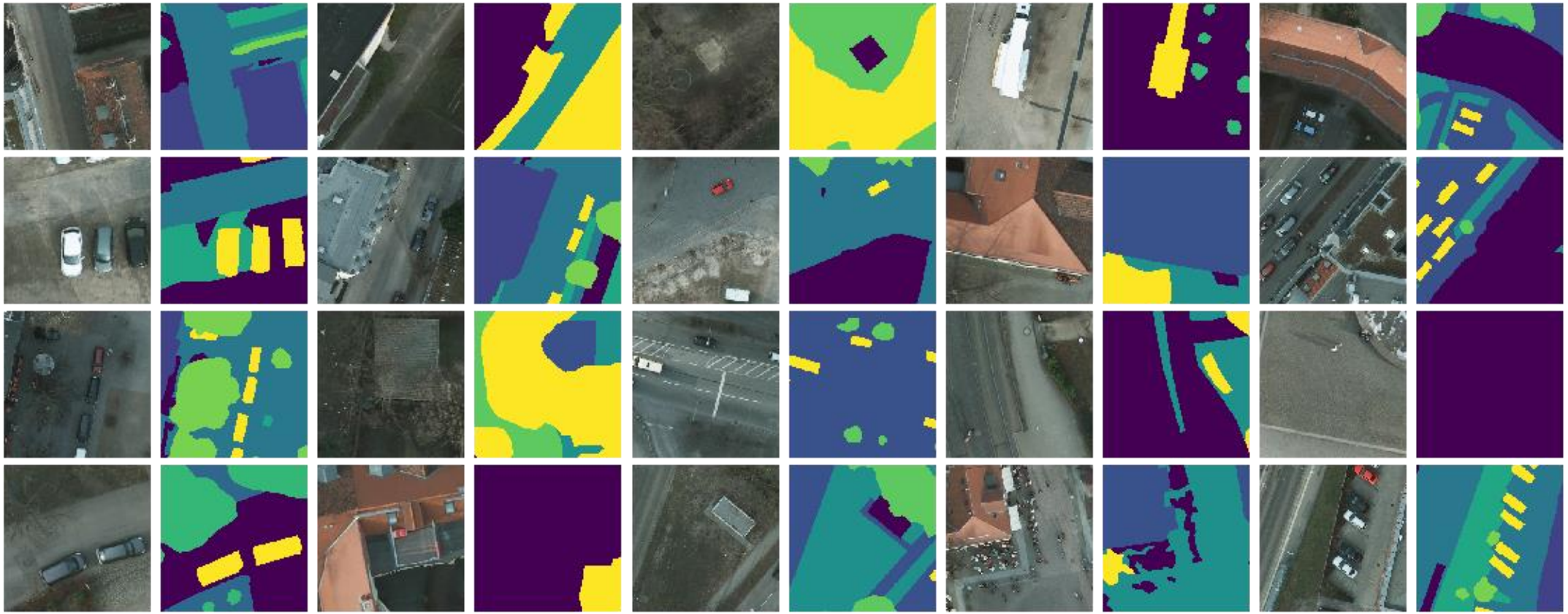
```
[35]: timepipeline(train_ds, batch_size=BATCH_SIZE)

100% 1000/1000 [01:14<00:00, 13.48it/s]

batche size: 50
1000 batches: 74.26196789741516 s
13.46584 Batches/s
673.29215 Images/s
```

With Memory Caching:

```
[38]: timepipeline(train_ds, batch_size=BATCH_SIZE)

100% 1000/1000 [01:13<00:00, 13.56it/s]

batche size: 50
1000 batches: 73.79464364051819 s
13.55112 Batches/s
677.55595 Images/s
```

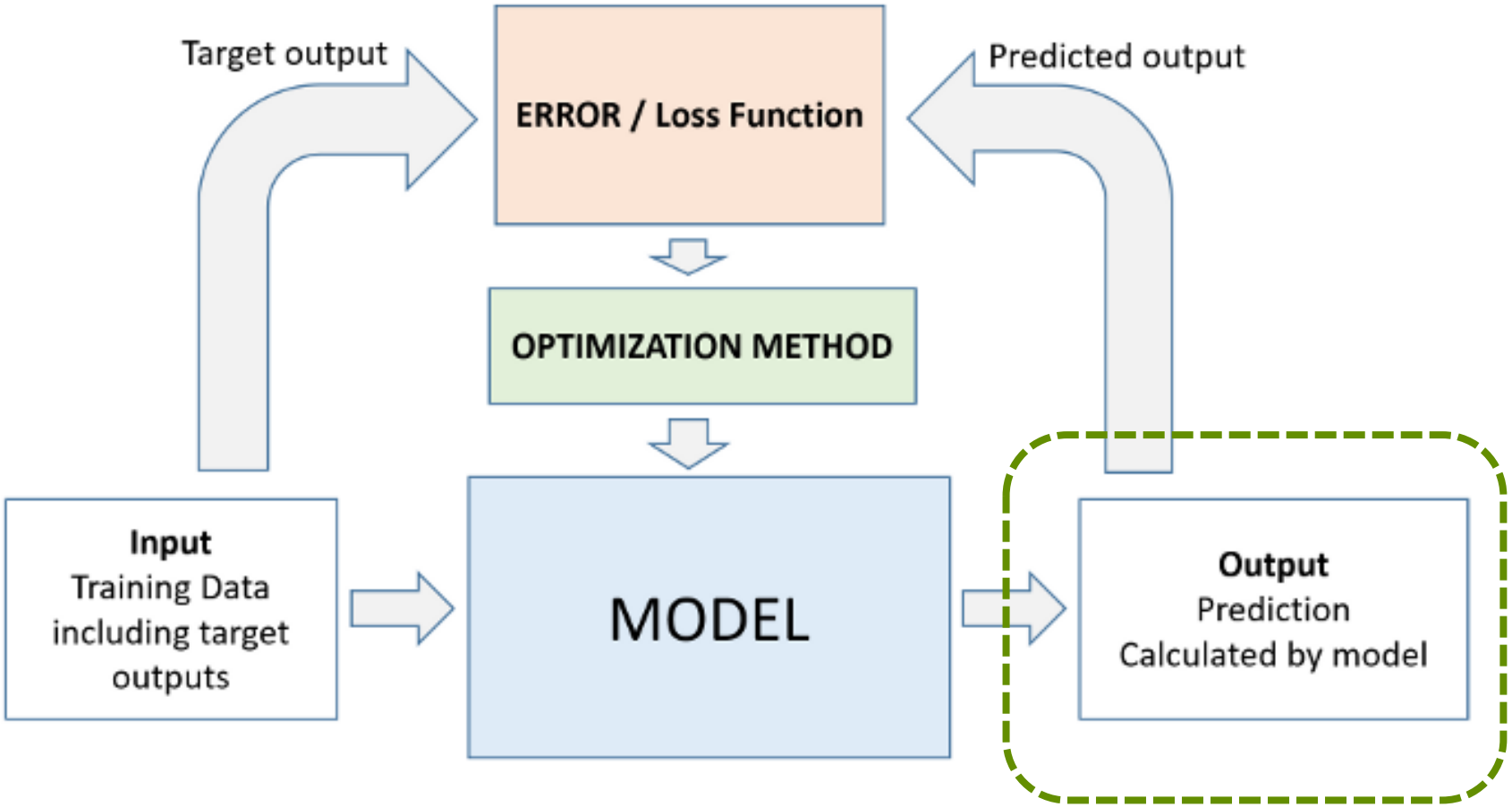
Input Pipeline Patch Pairs Example

```
[47]: train_ds = get_dataset(img_dir=train_img_dir, mask_dir=train_mask_dir, batch_size=BATCH_SIZE, shuffle=True, repeat=True, augment=True, scaled=False, cachefile=None)

[ ]: val_ds = get_dataset(img_dir=val_img_dir, mask_dir=val_mask_dir, batch_size=BATCH_SIZE, shuffle=False, repeat=True, augment=False, scaled=False, cachefile=None)
```

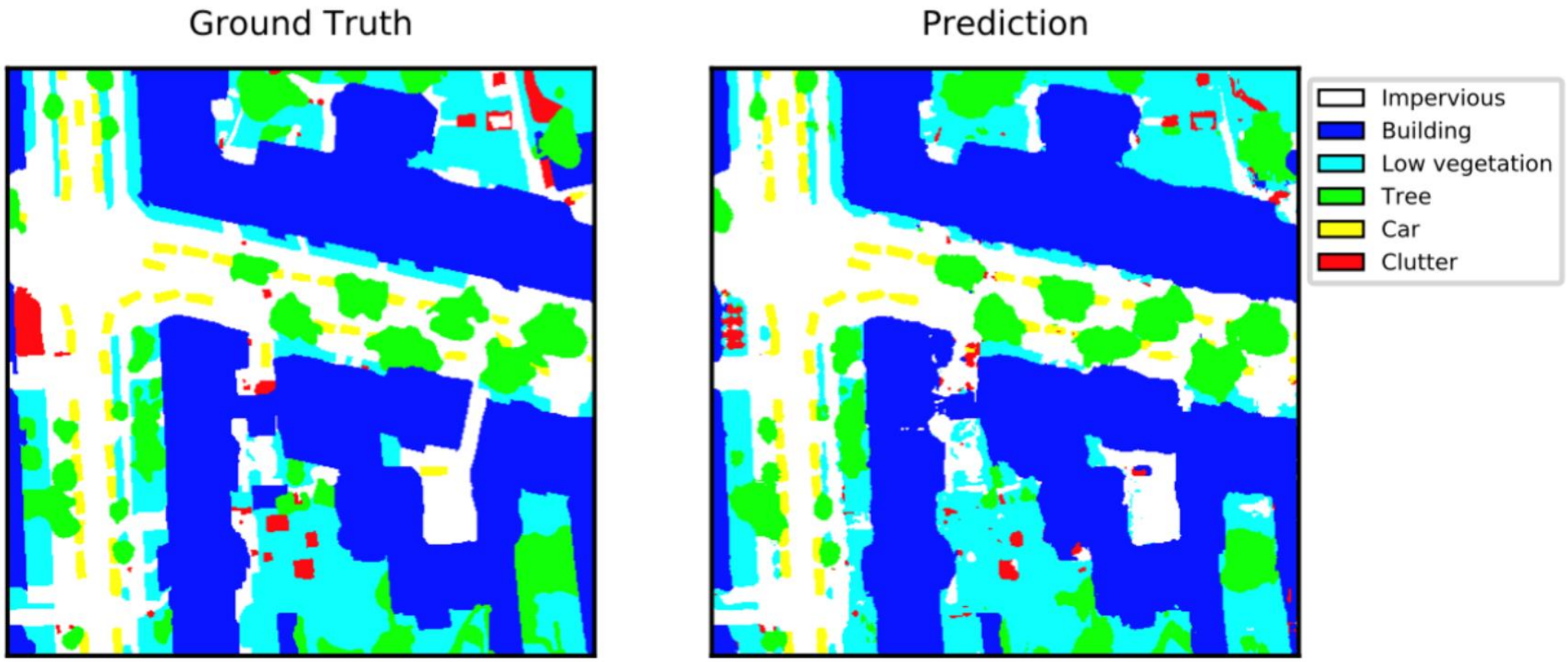


Model Performance Metrics

Performance Metrics

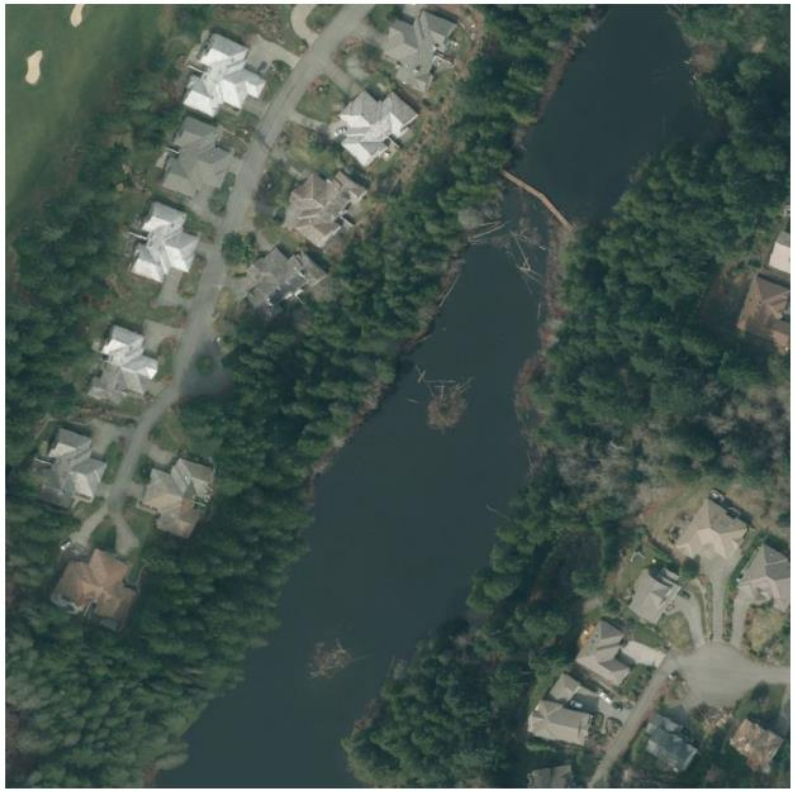


Natural Metric for Image Comparison: Accuracy



AIRBUS

Performance Metrics: Accuracy is not enough For Segmentation



Kitsap County, WA

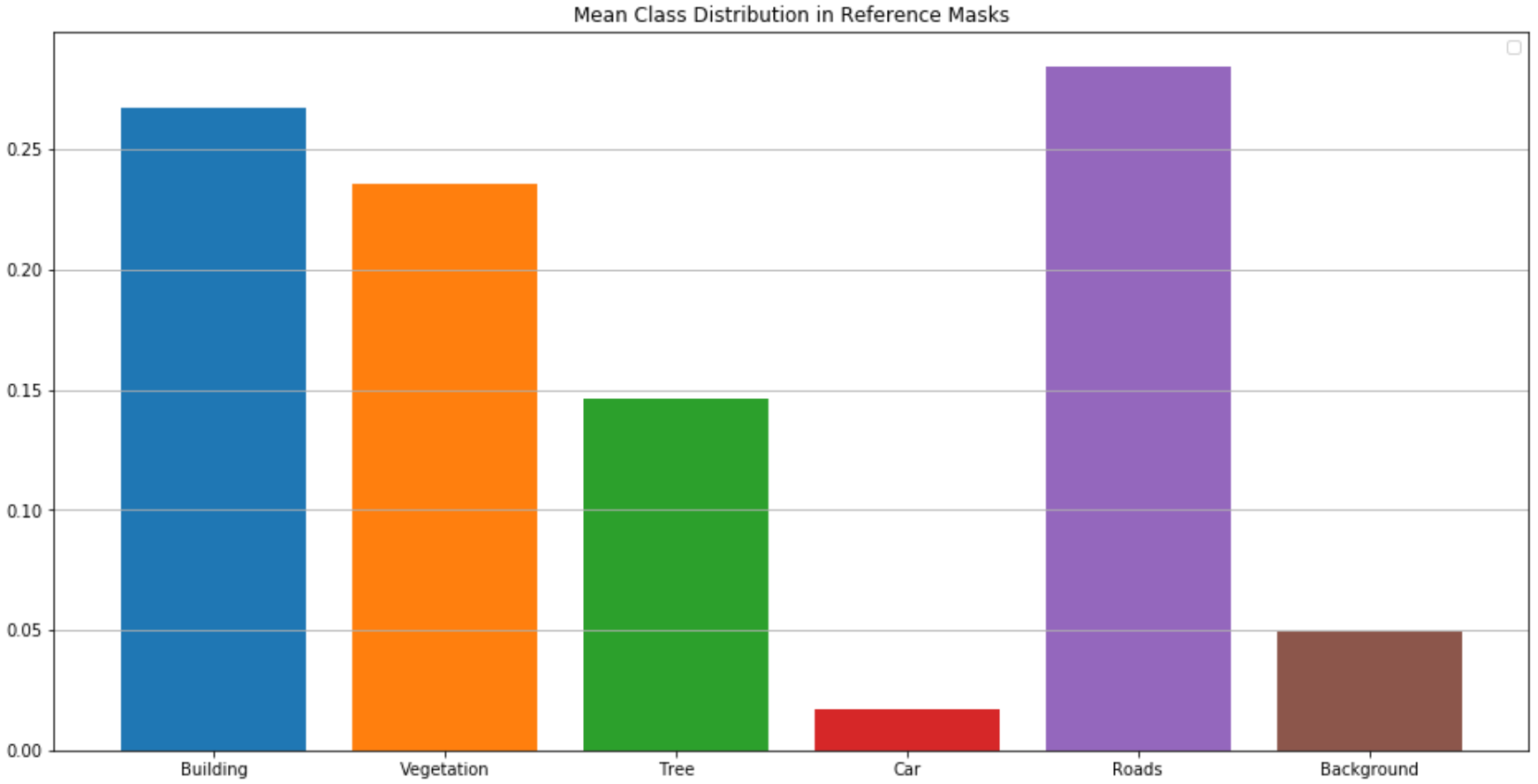
Kitsap County, WA – Reference

≈ 10% Building

Model = all Background

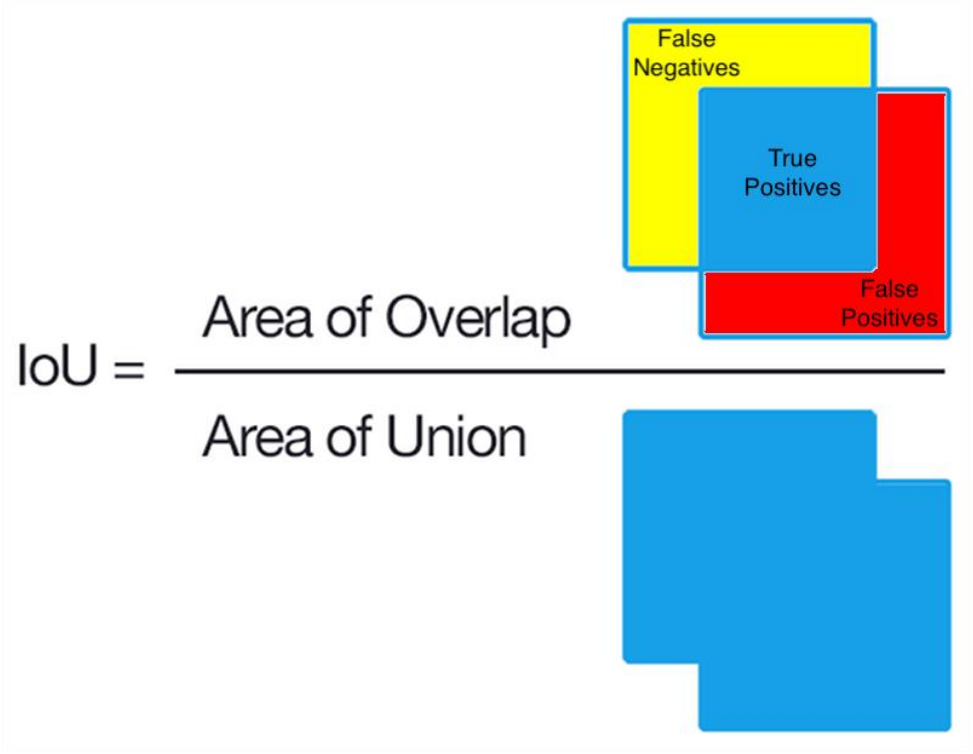
≈ 90% Accuracy

ISPRS Class Distribution



AIRBUS

A Better Segmentation Metric: Jaccard Index (Intersection Over Union)

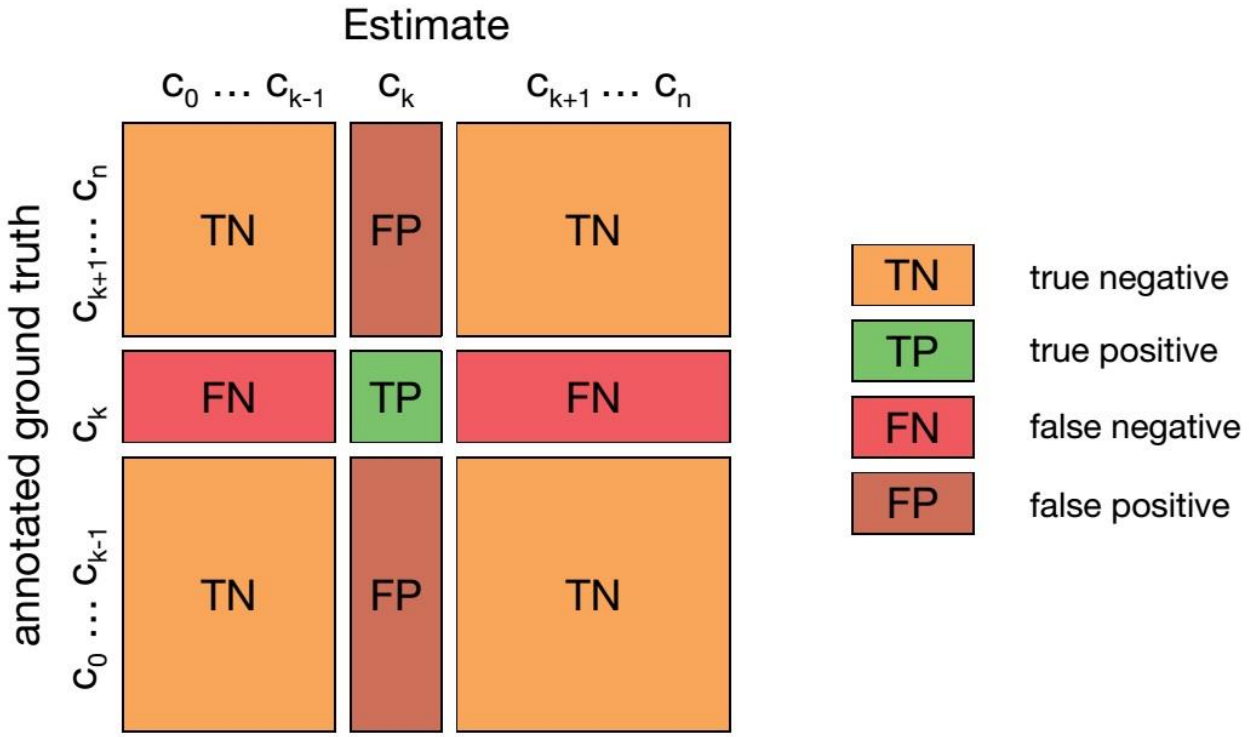


AIRBUS

Metrics Formulas:

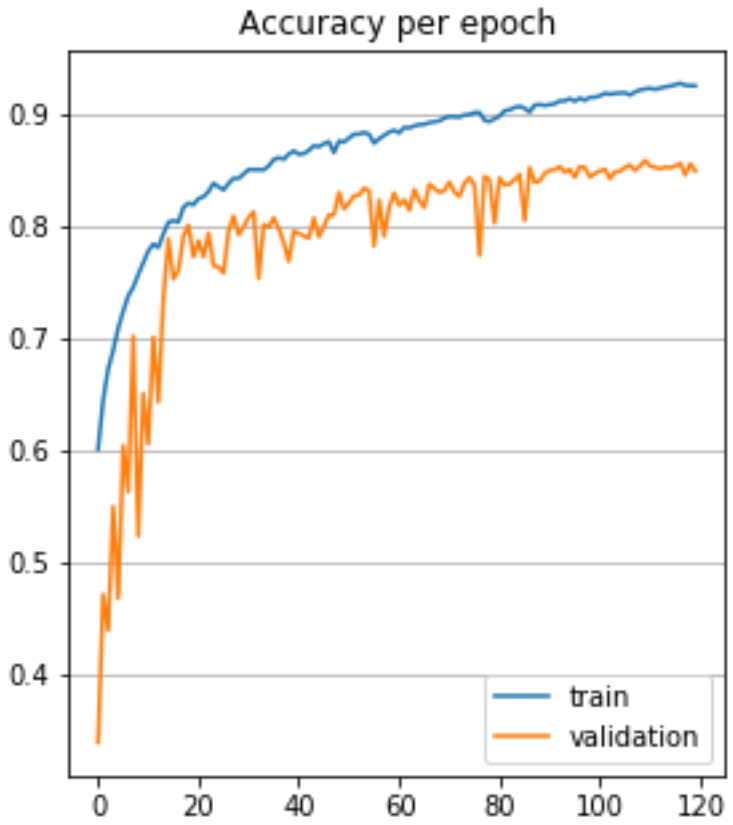
$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

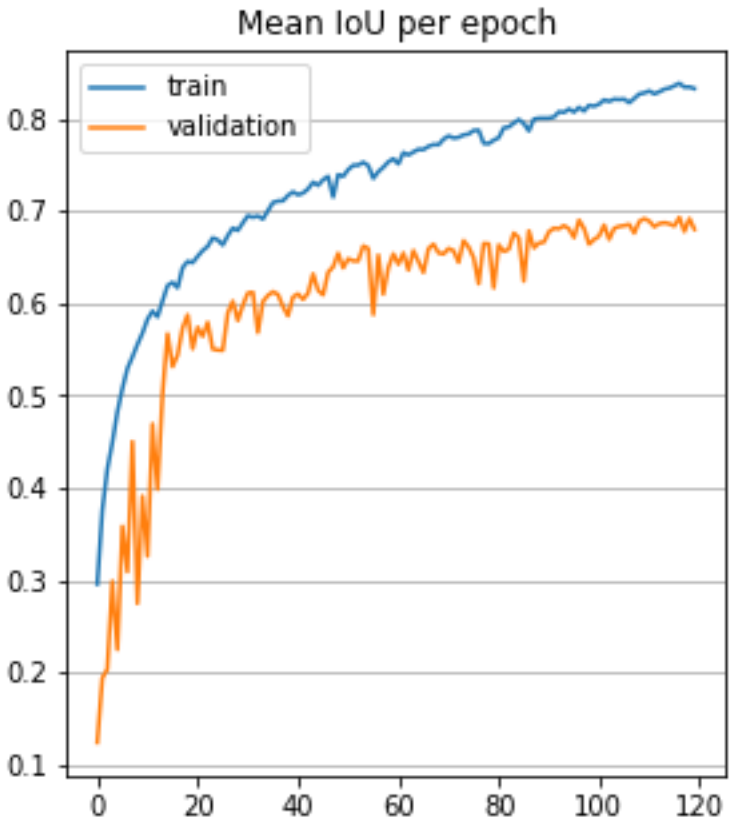
$$Jaccard\ Index\ (IoU) = \frac{TP}{TP + FP + FN}$$

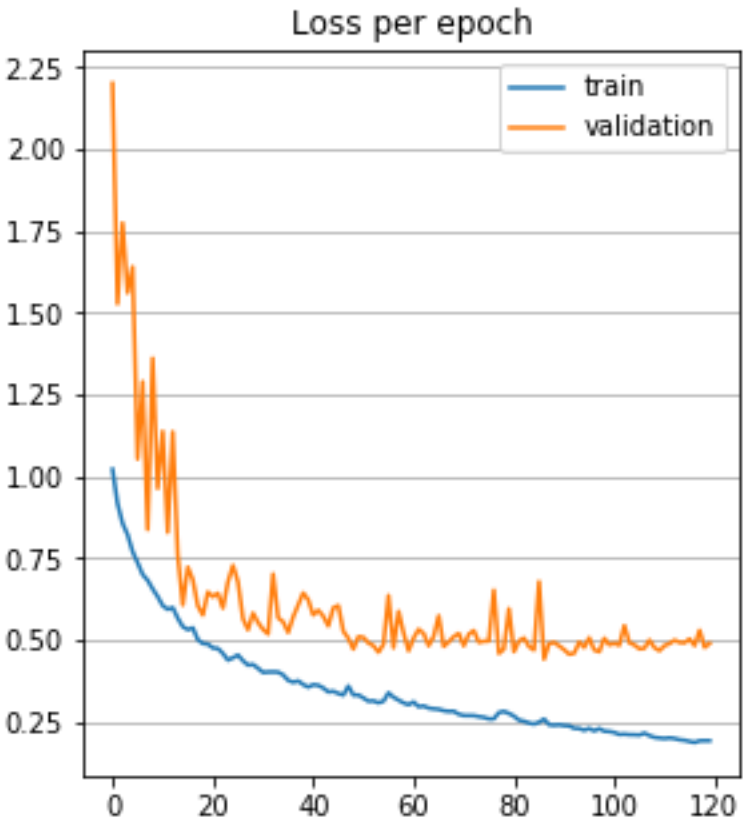


Confusion Matrix

Monitoring Metrics During Training







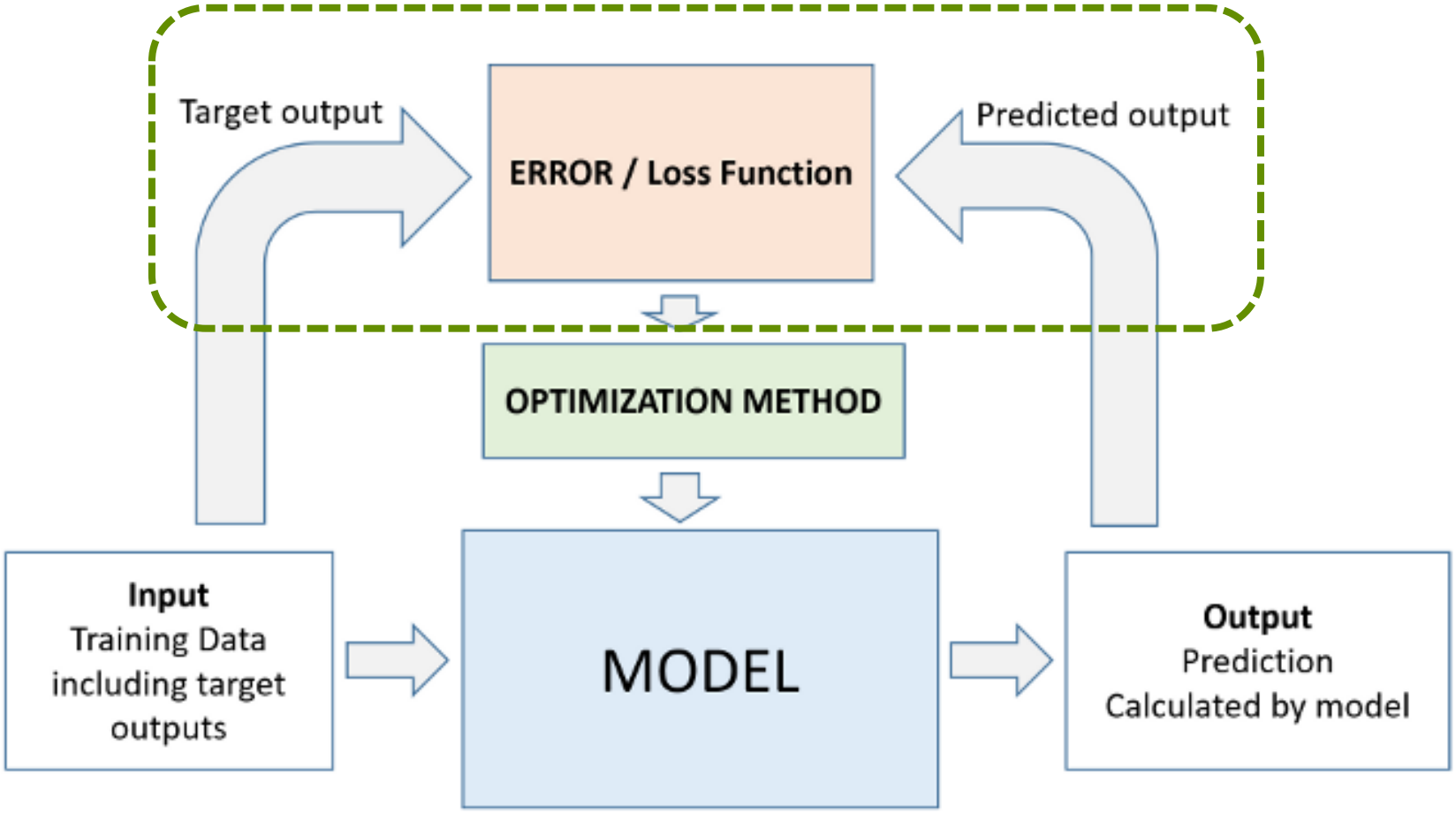
AIRBUS

Training Callbacks

- Keras Callbacks are Predefined Functions automatically executed at the end of every epoch:
  - Checkpoint Callback: monitor a metric on validation dataset (IOU for instance) and stores the model weights only if the Performance improved
  - Reduce on Plateau Callback: reduce the learning Rate by a factor if the Performance Matric did not improve after a number of epochs
  - Early stopping callback: automatically stops training if Performance Metric did not improve after a number of epochs
  - Tensorboard callback: stores training and Validation Metrics for live Monitoring

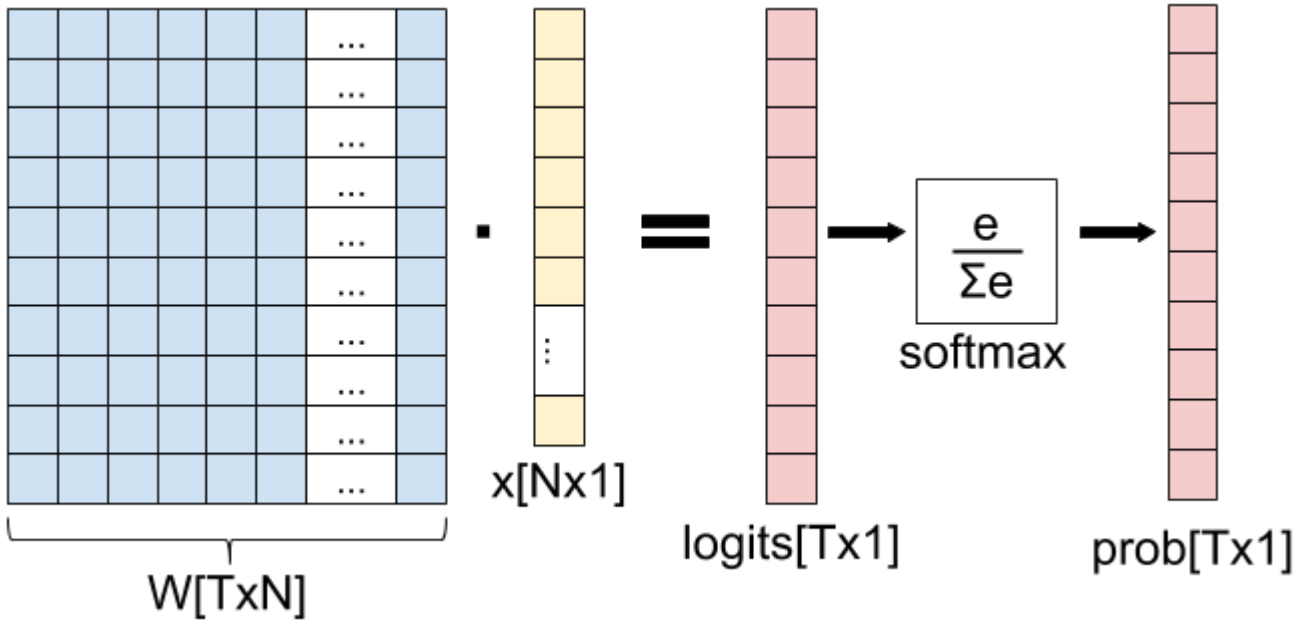
Model Loss Function

Loss Function

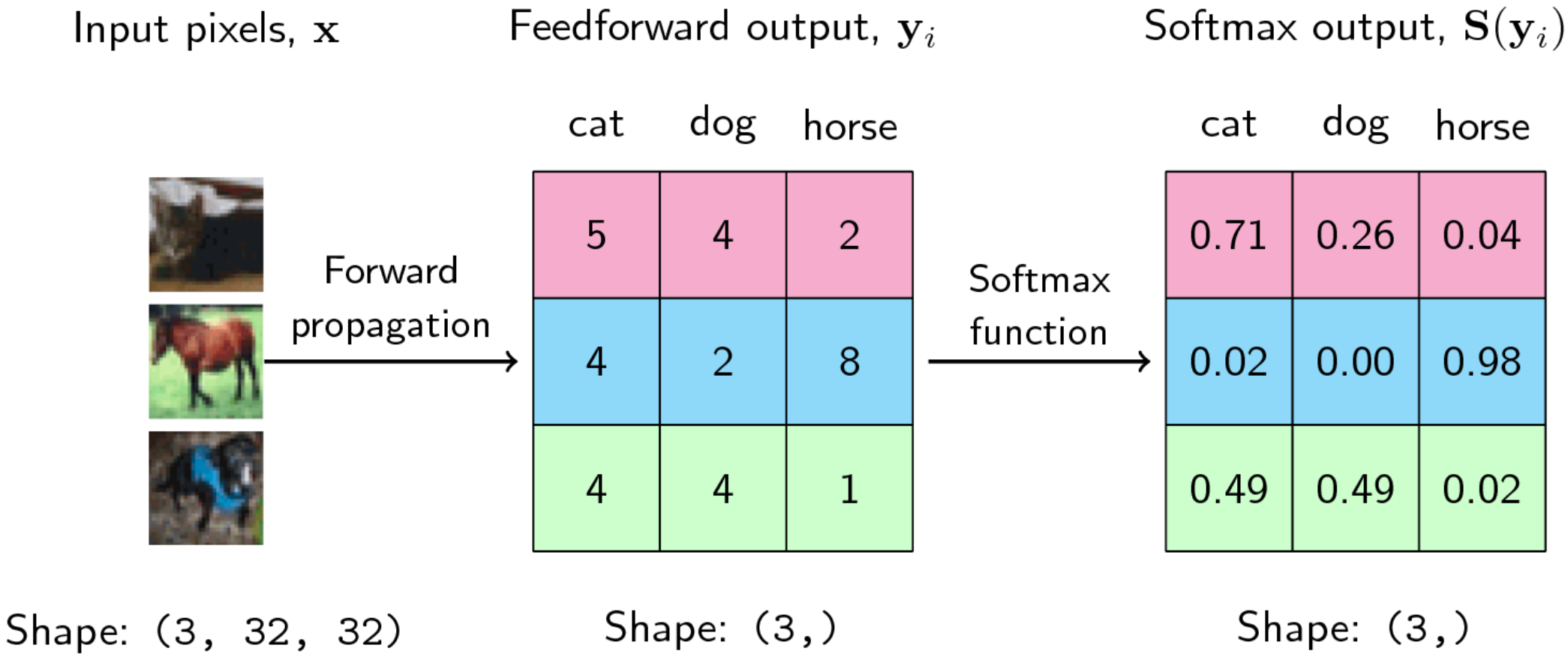


AIRBUS

The Softmax Layer



The Softmax Layer



Used Loss Functions:

- Multiclass Cross Entropy:

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

- Focal Loss:

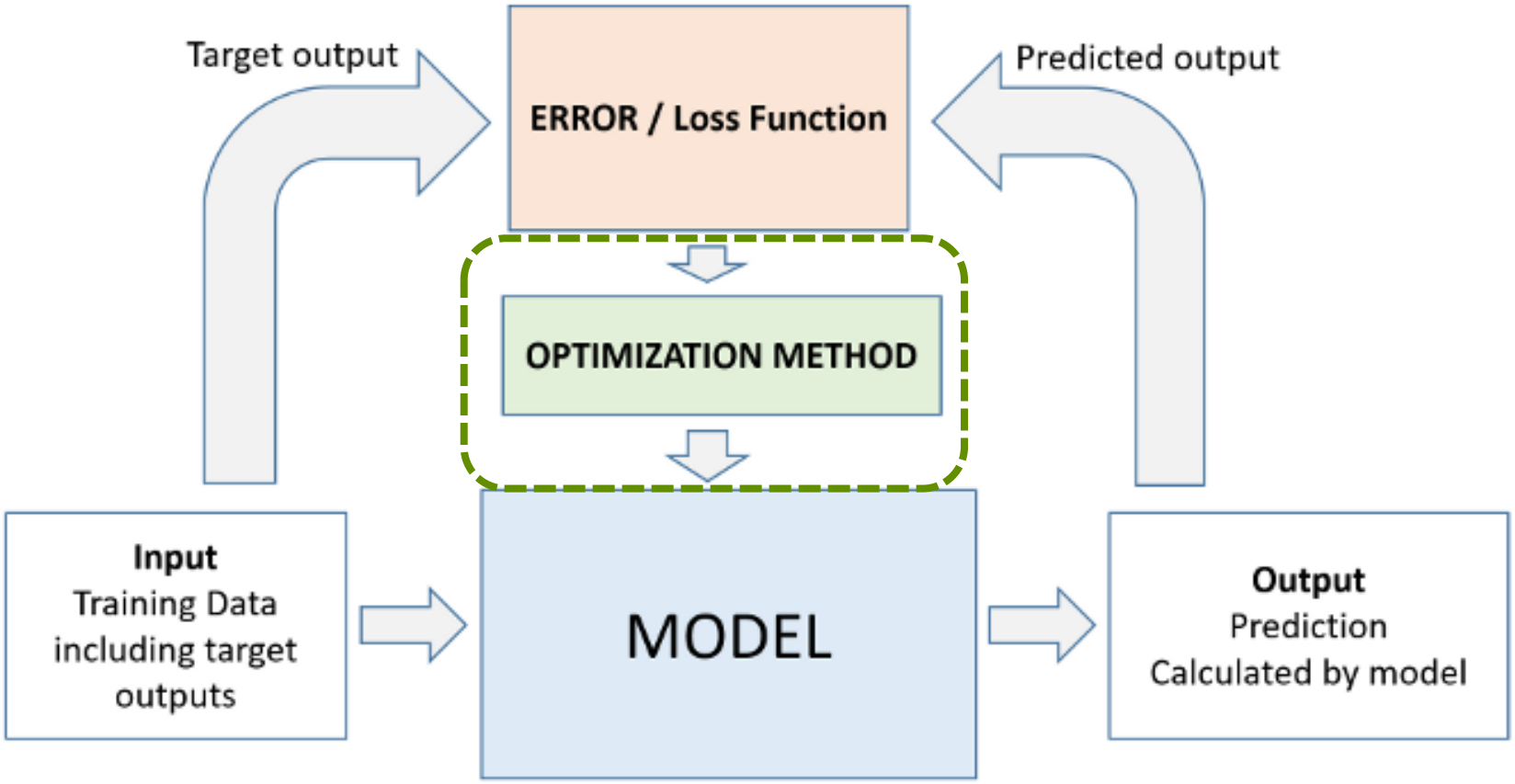
$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1 - p_\mathrm{t})^\gamma \log(p_\mathrm{t}).$$

- Jaccard Index Loss:

$$L_{IoU} = 1 - IoU = 1 - \frac{I(X)}{U(X)}$$

Model Optimization

Model Optimization

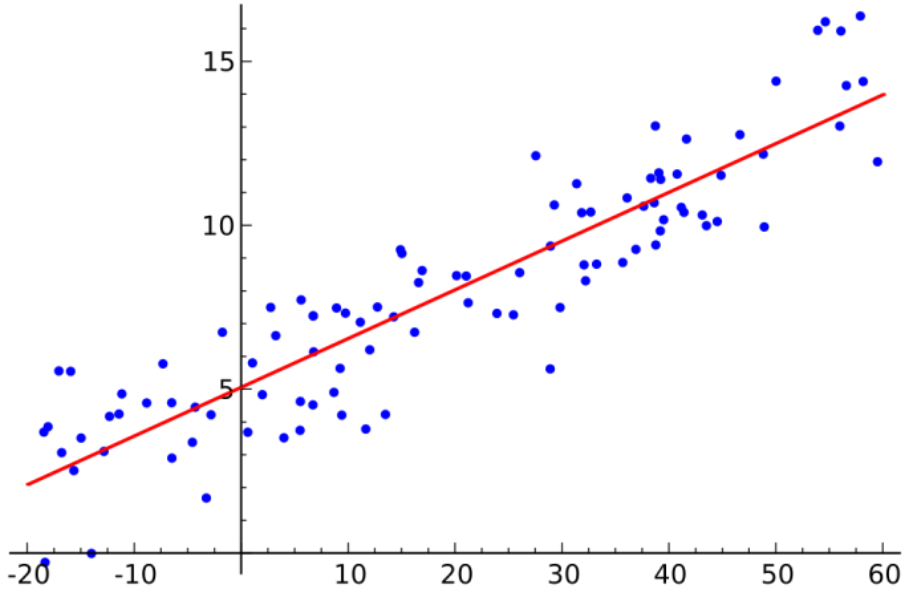


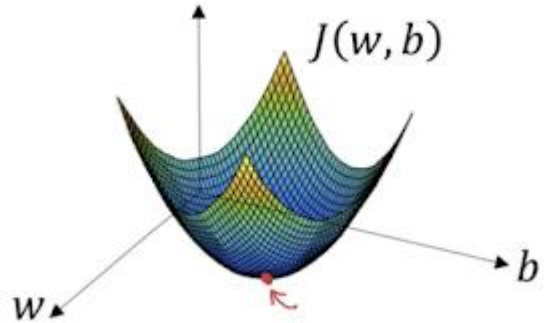
AIRBUS

Optimization Algorithms

- The goal is to find the set of weights that minimizes the Loss function

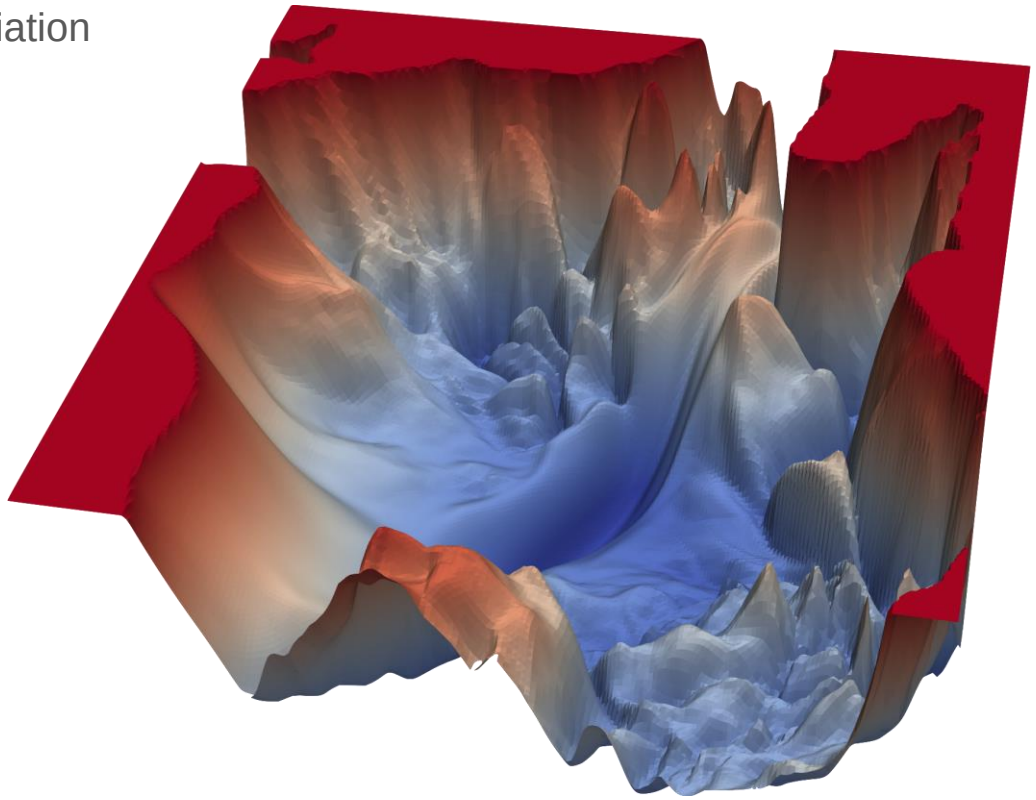
- For Linear Regression:
  - Model: $Y = w X + b$
  - Loss Function (Squared Mean Error): $J(w,b) = ||Y – w X – b||^2$





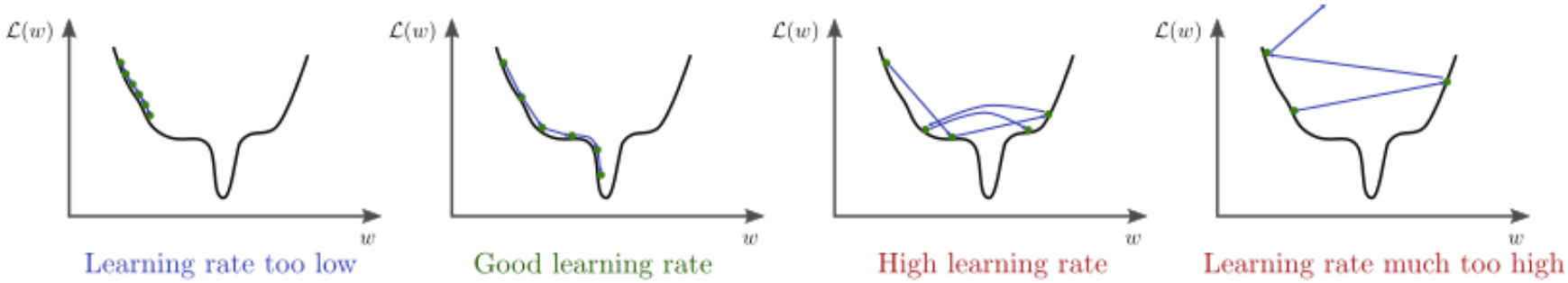
Loss Function Landscape for Neural Network

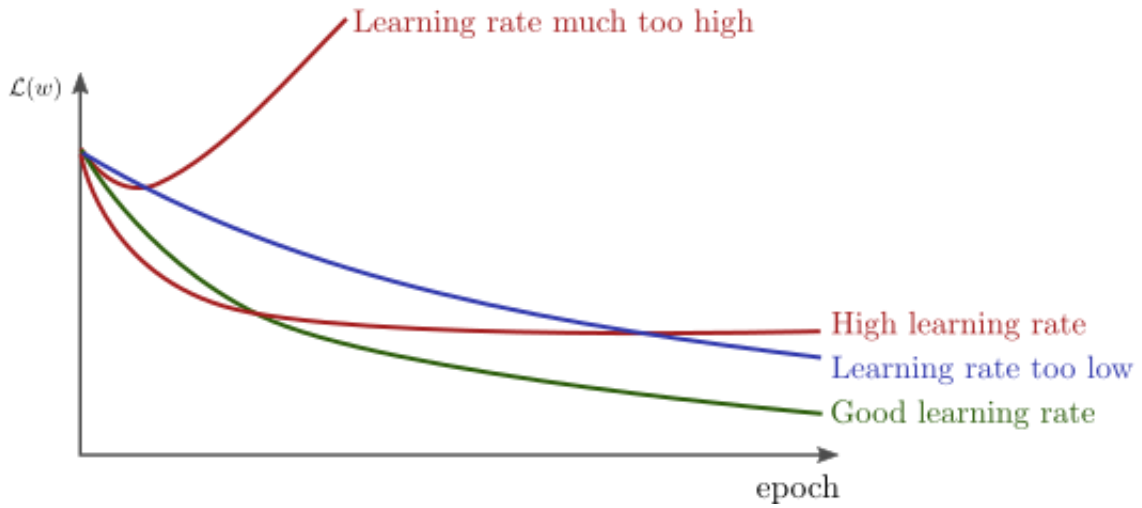
- All Optimization Algorithm are variation of Gradient Descent and are provided by Tensorflow library
- Gradient of loss function is computed using Automatic-differentiation
- Some of the best known are: SGD, RMSPROP, ADAM
  - In our work SGD with momentum is used
- The common algorithm Parameter: Learning Rate



 https://www.cs.umd.edu/~tomg/projects/landscapes/

Finding a Good Learning Rate is critical

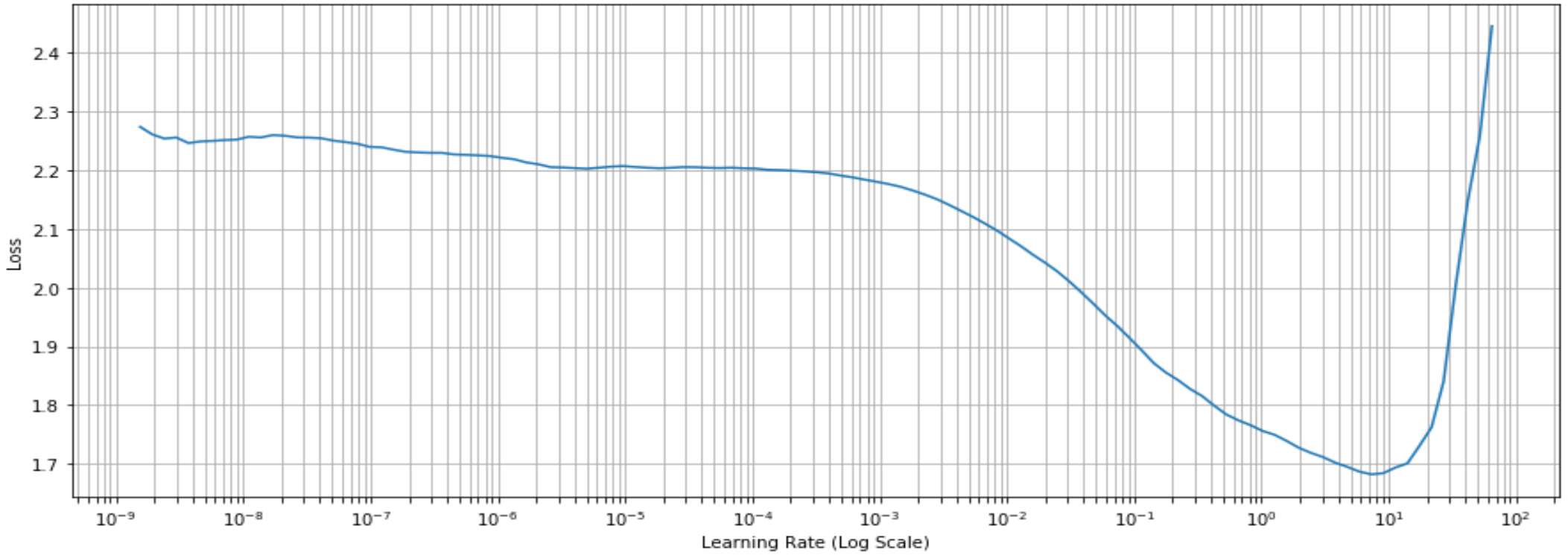




 http://www.bdhammel.com/learning-rates/

Learning Rate Finder

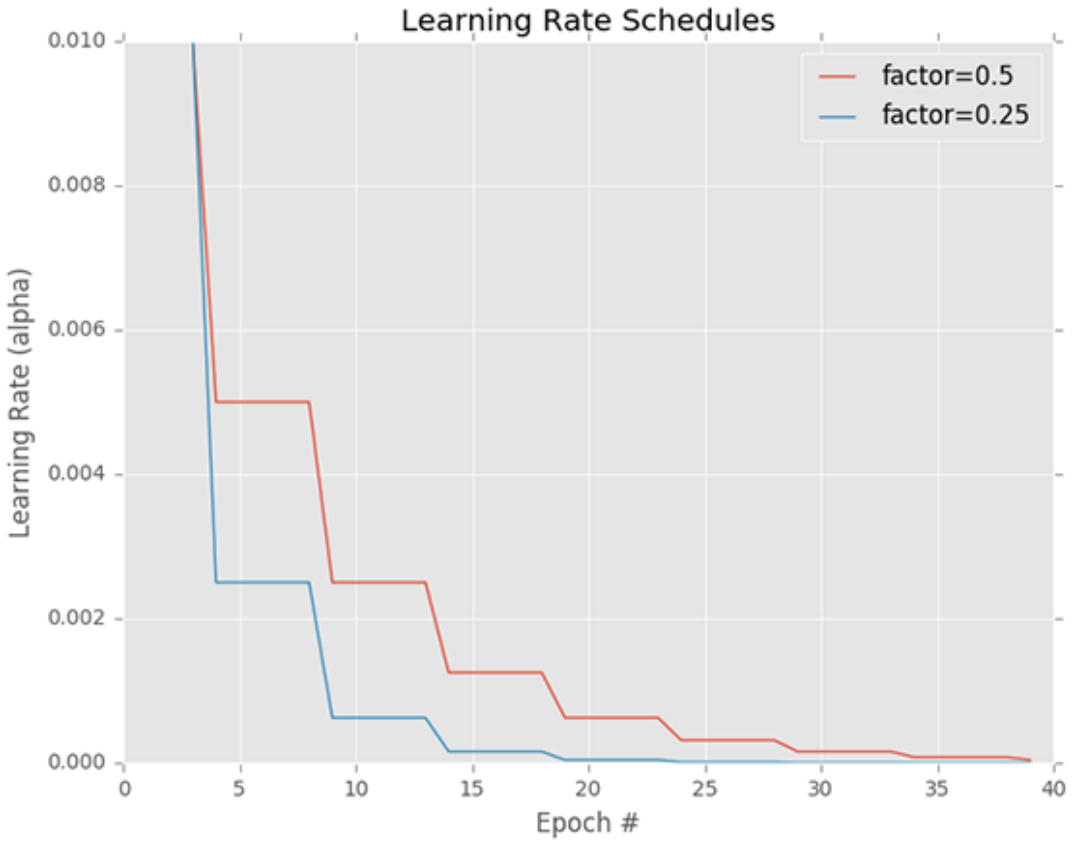
```
[ ]: lrf = LearningRateFinder(model)
     lrf.find(train_ds, 1e-9, 1e+2, epochs=1, stepsPerEpoch=TRAIN_STEPS_PER_EPOCH)
     lrf.plot_loss()
```



 Paper: https://arxiv.org/abs/1506.01186

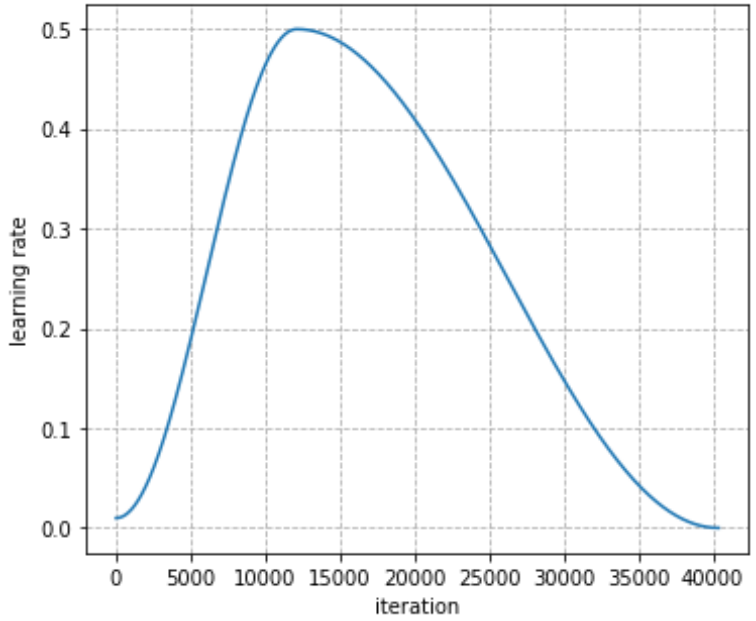
AIRBUS

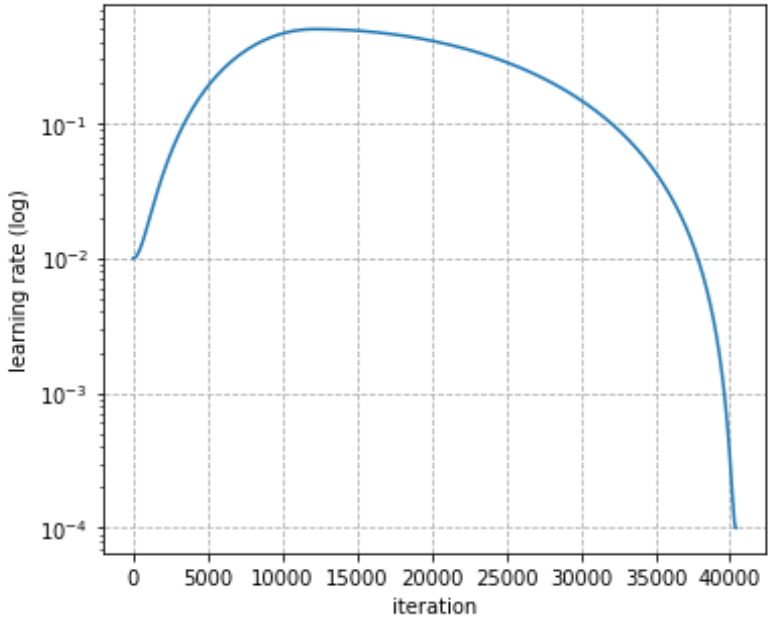
Training Strategy: Reduce on Plateau (Keras Callback)

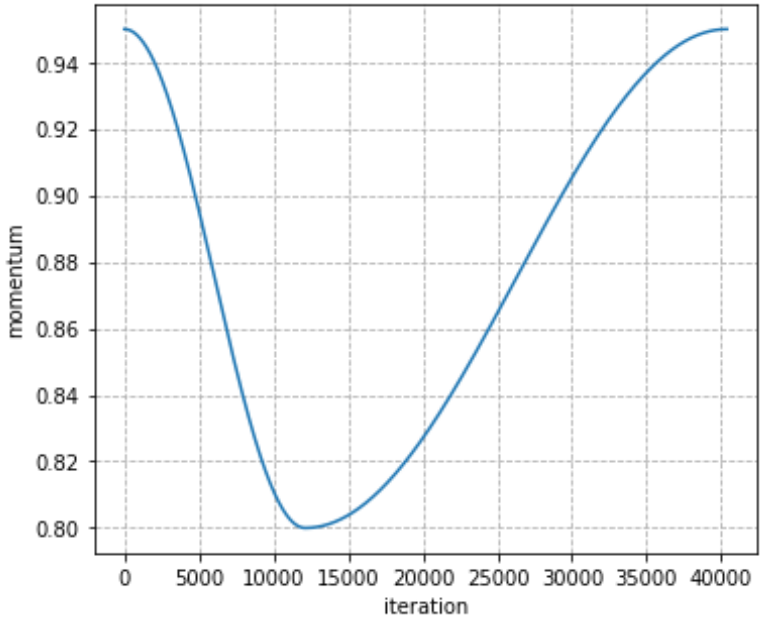


AIRBUS

Training strategy: One Cycle Policy (Keras Callback)



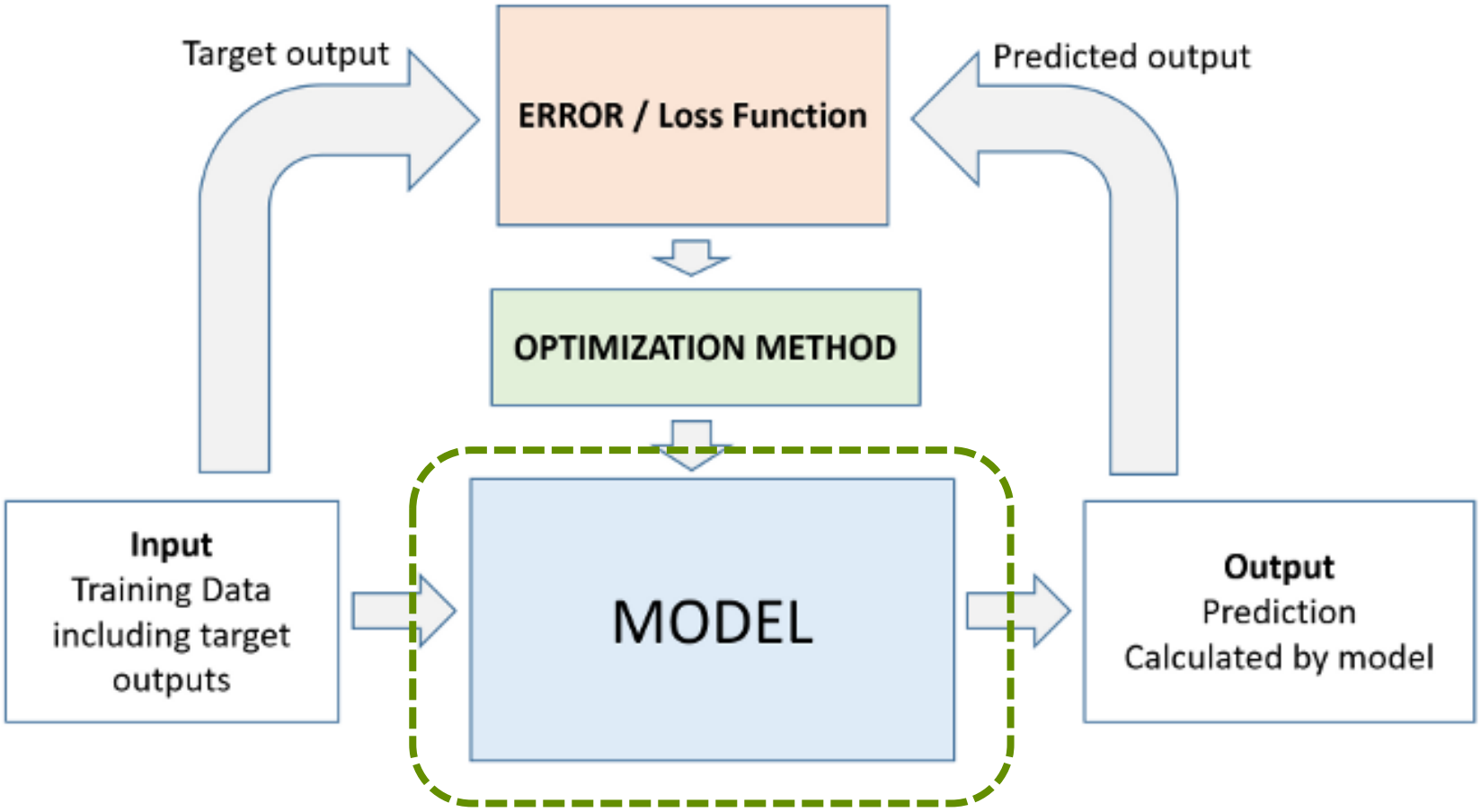




AIRBUS

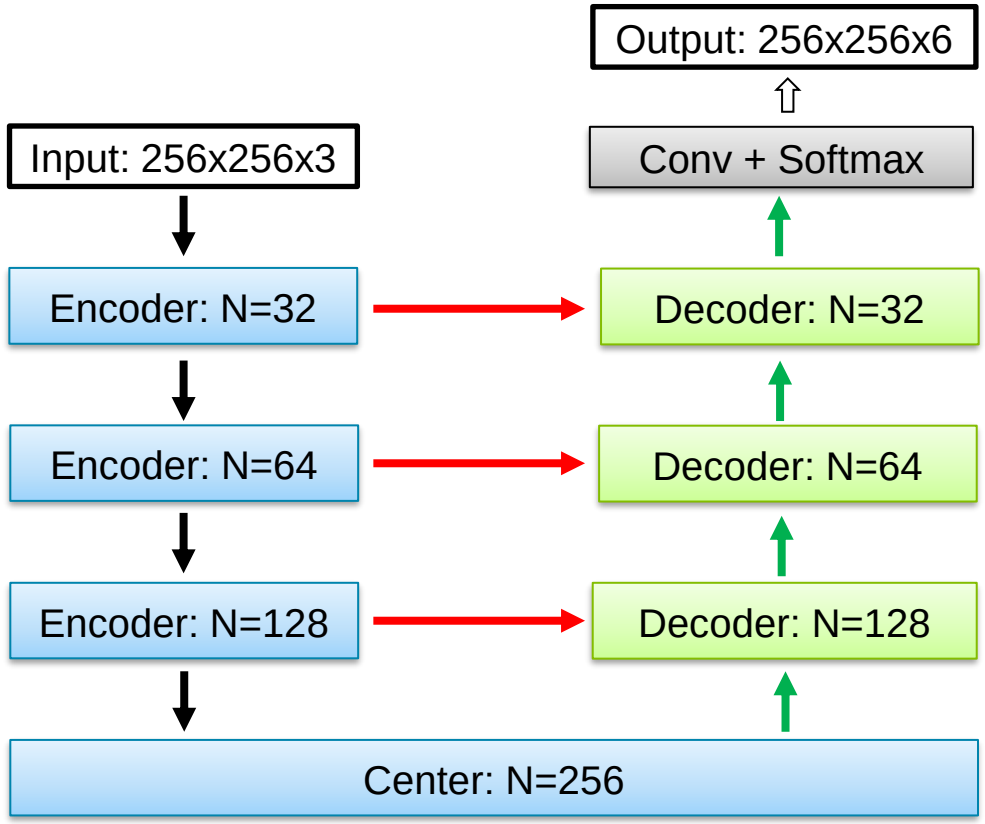
Model Architecture

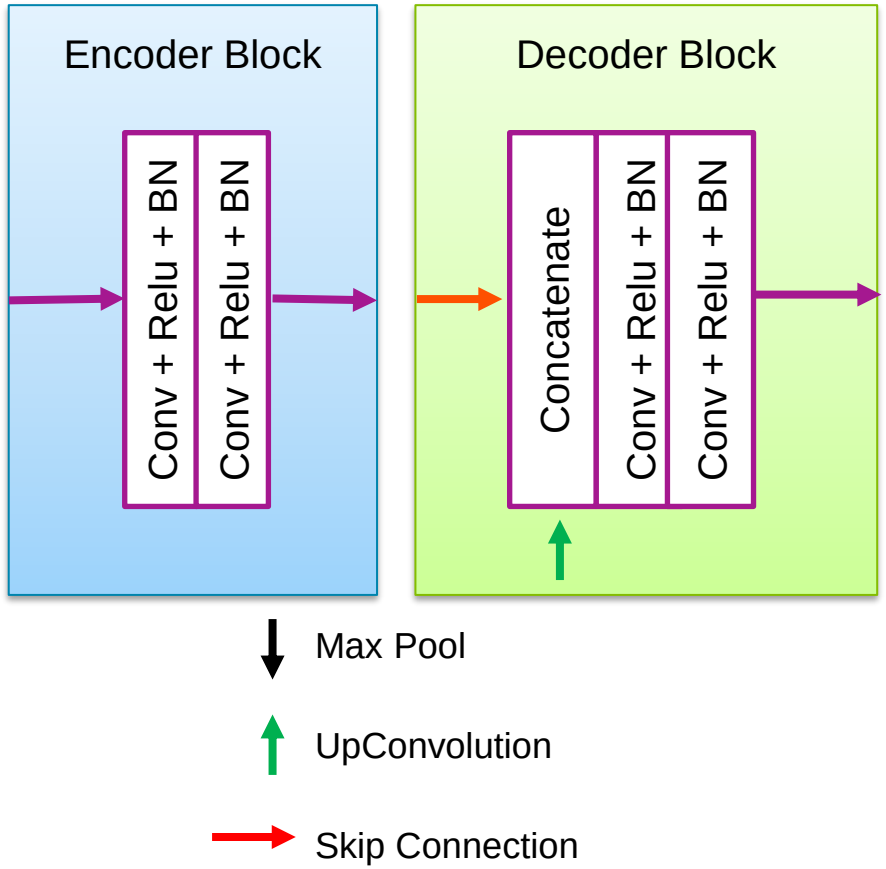
Model Architecture



AIRBUS

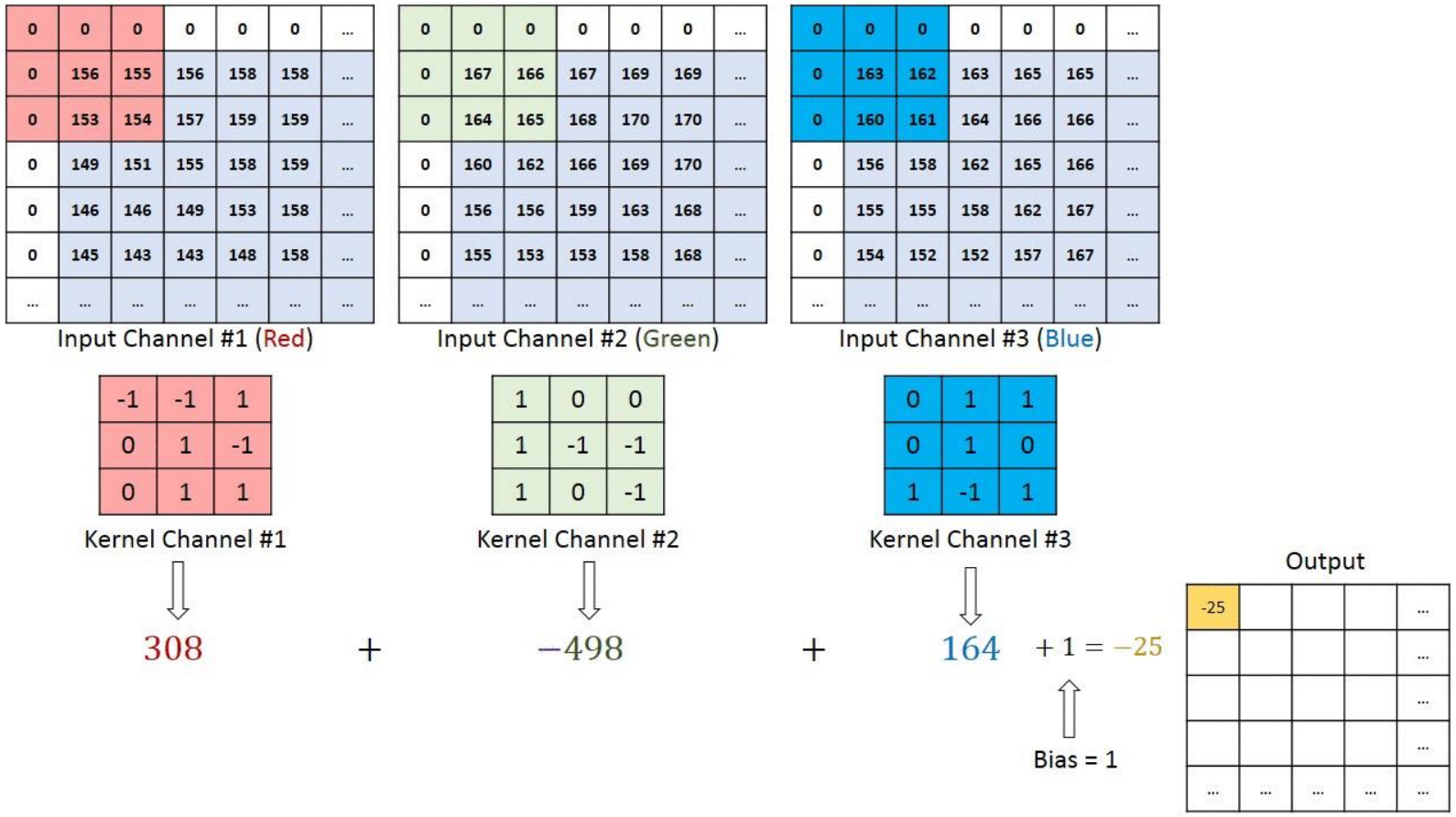
Unet: 3 Skips



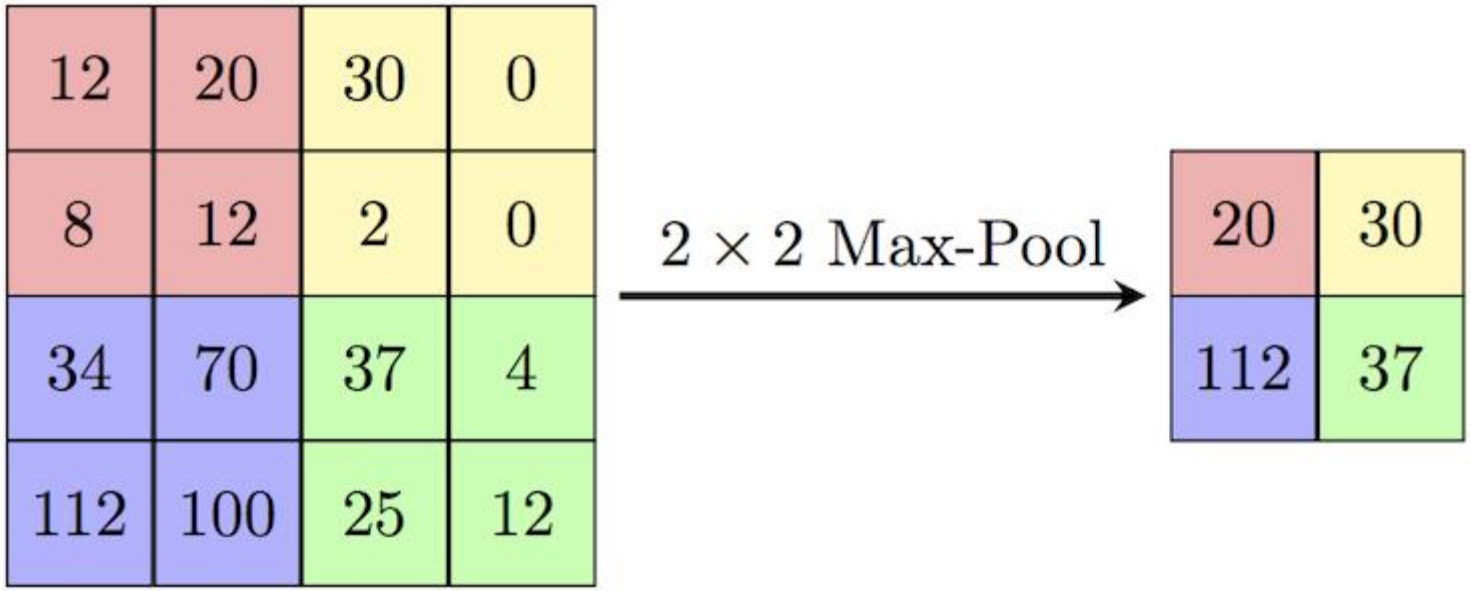


AIRBUS

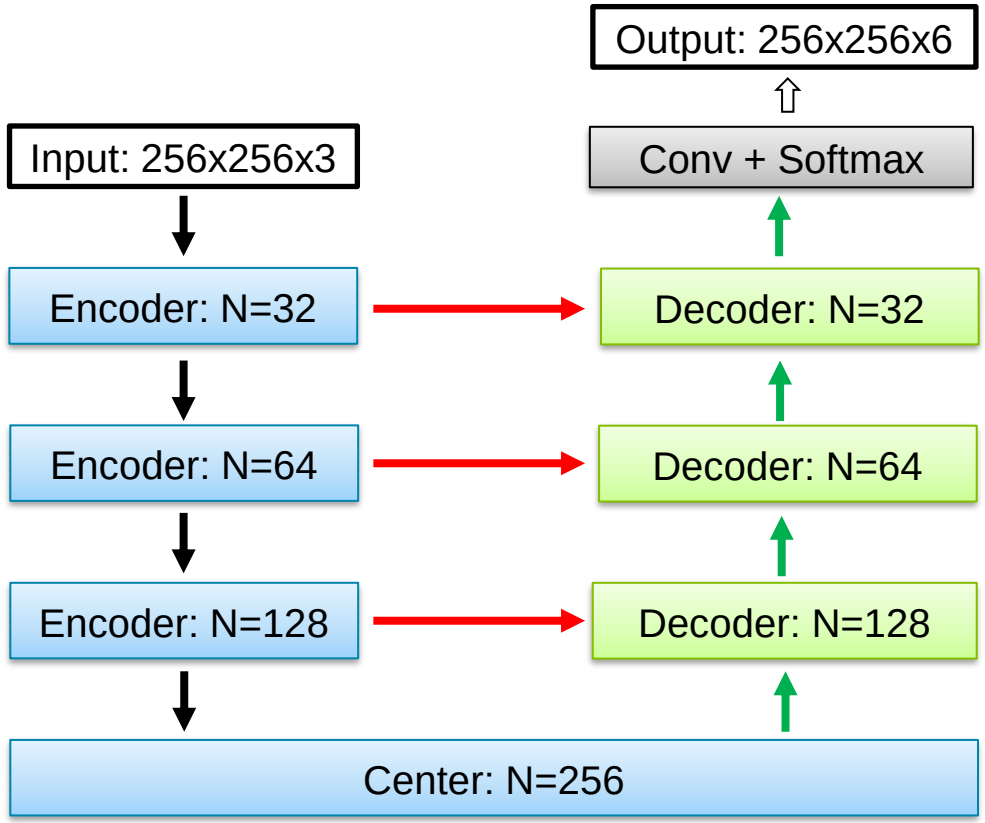
Convolution Layer in Neural Networks

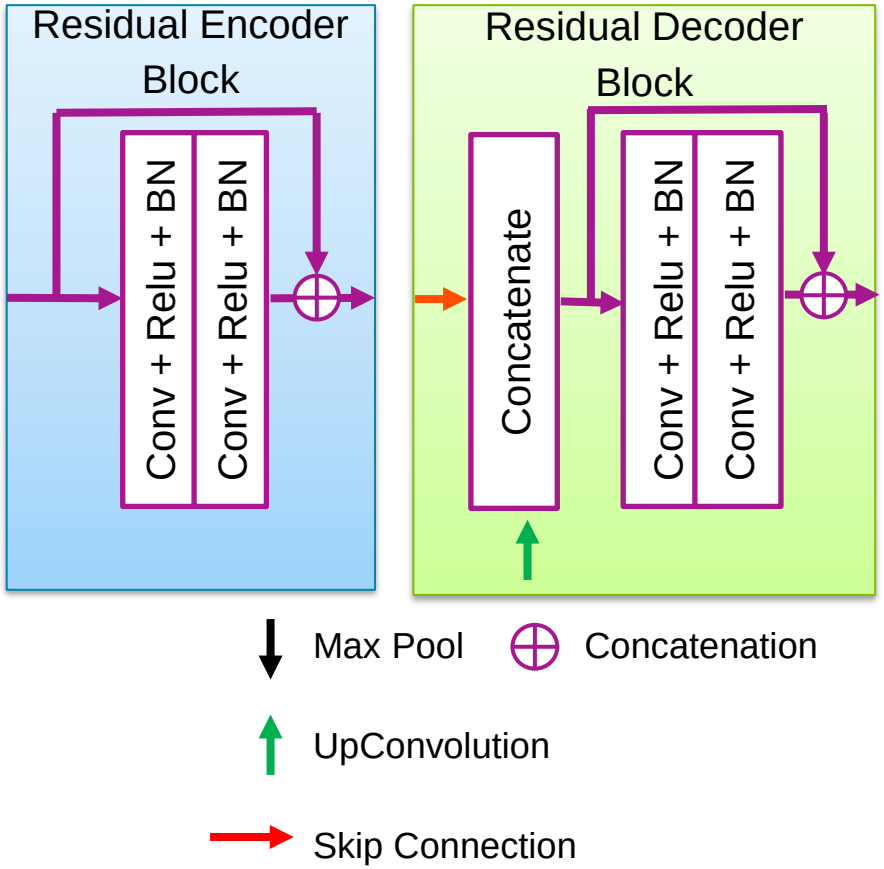


Max Pooling Layer in Neural Networks



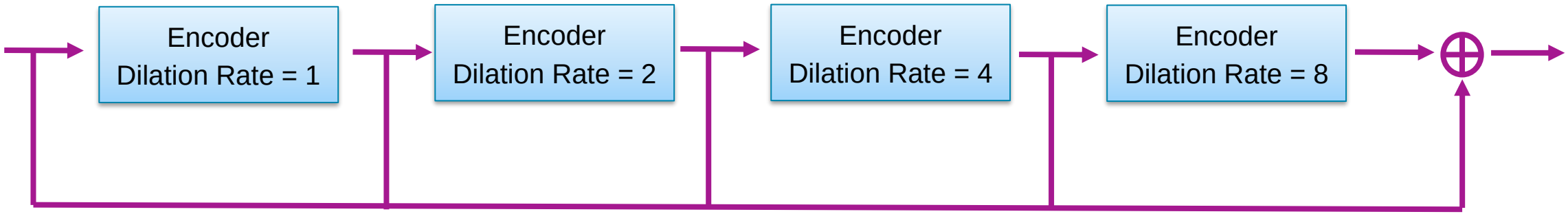
Unet: 3 Skips + Residual Blocks (like ResNet)



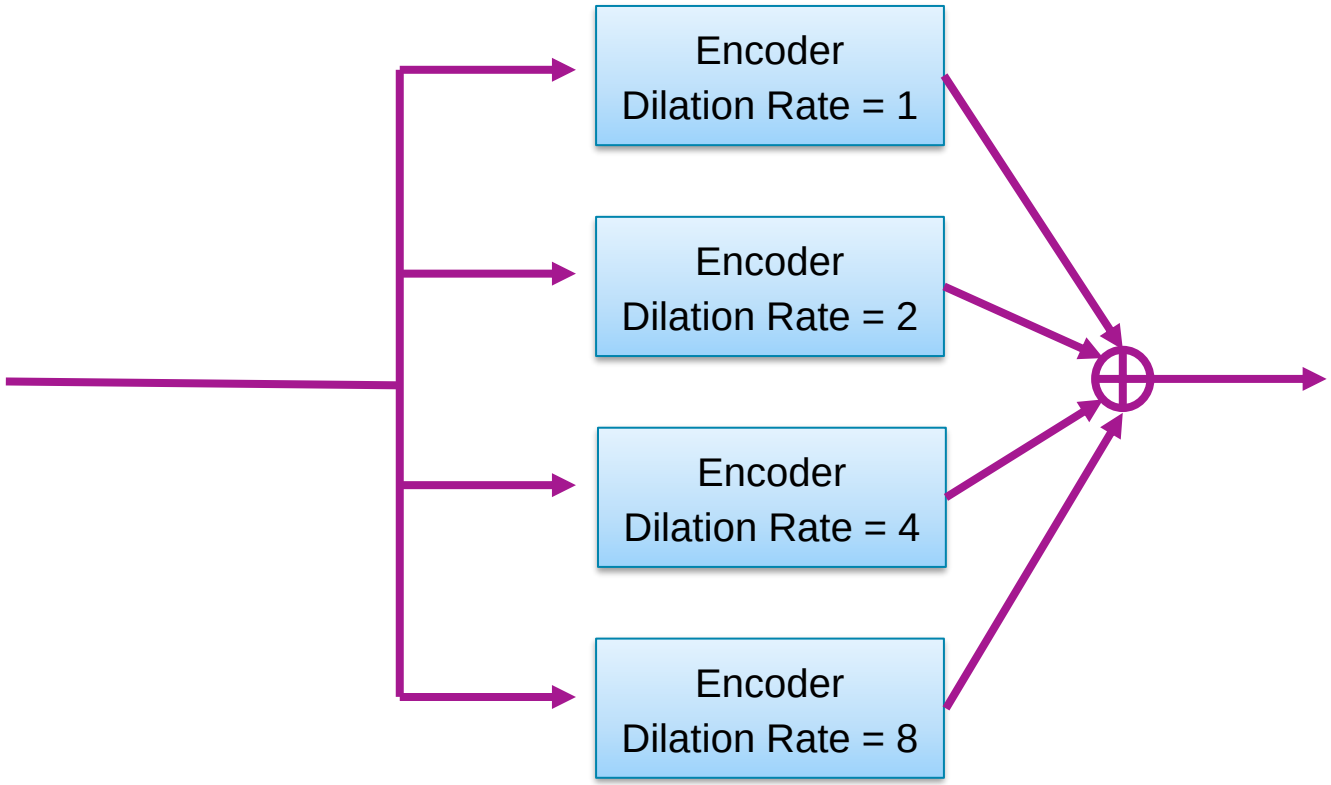


AIRBUS

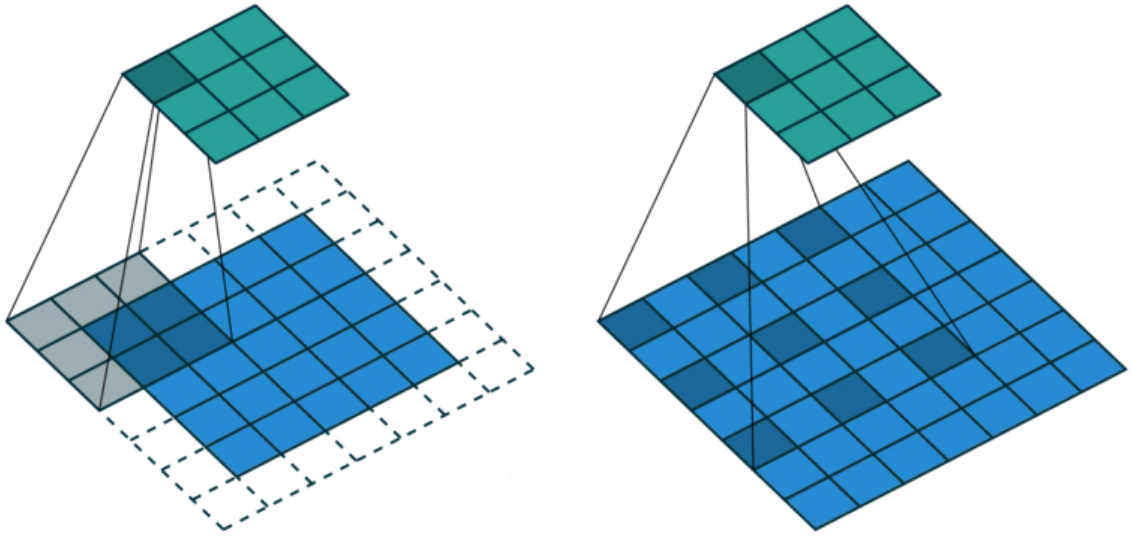
Unet: Serial Dilated Center (like DeepLab)

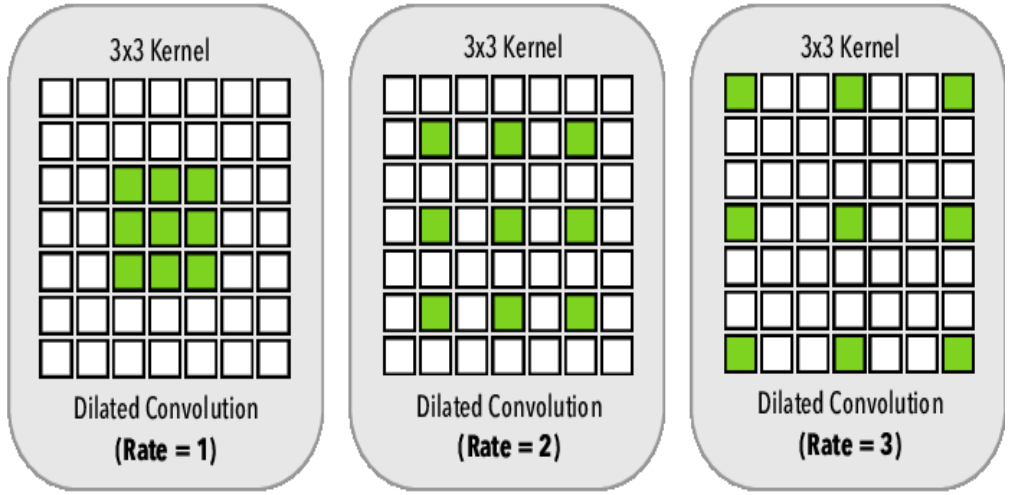


Unet: Parallel Dilated Center (like DeepLab)



Dilated Convolution Layer



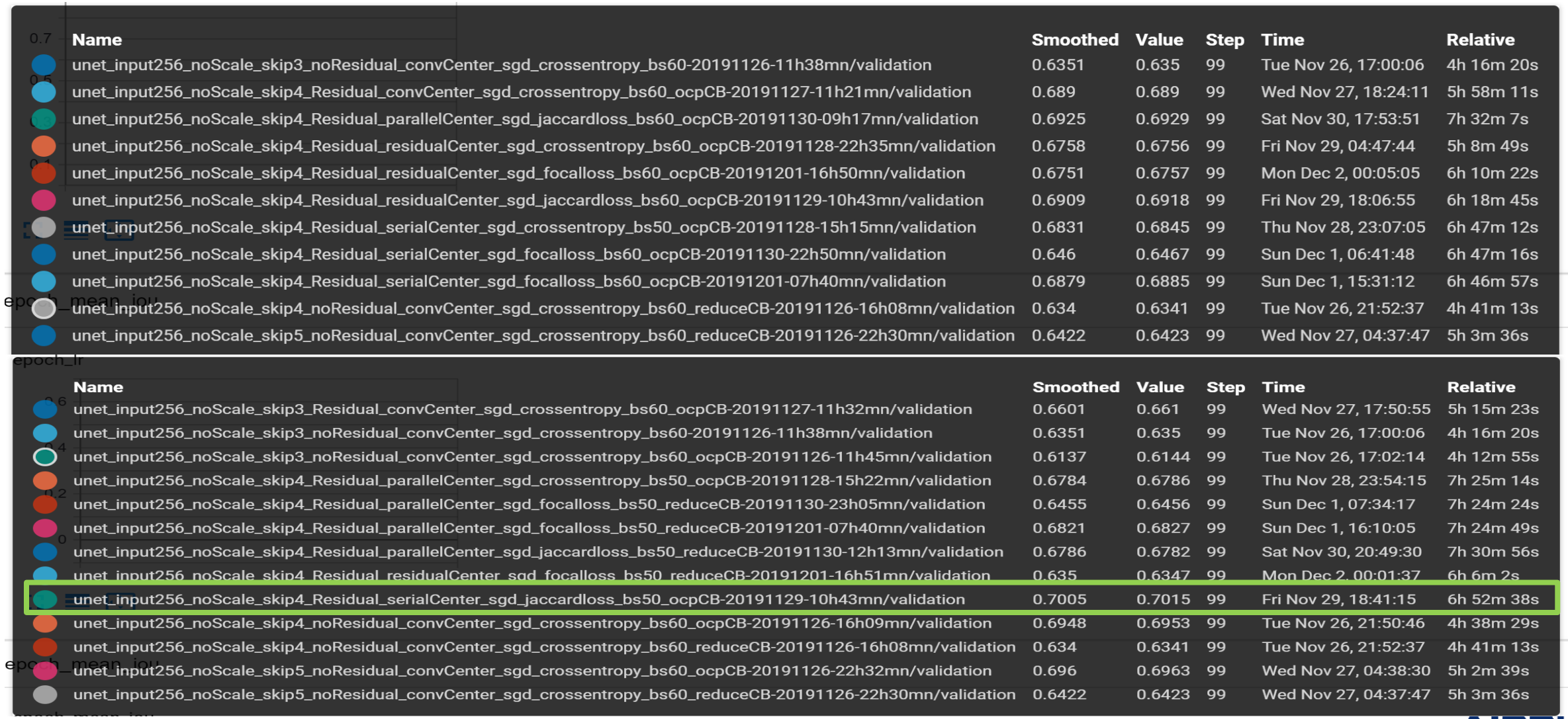


AIRBUS

Tensorboard Experiment Results

AIRBUS

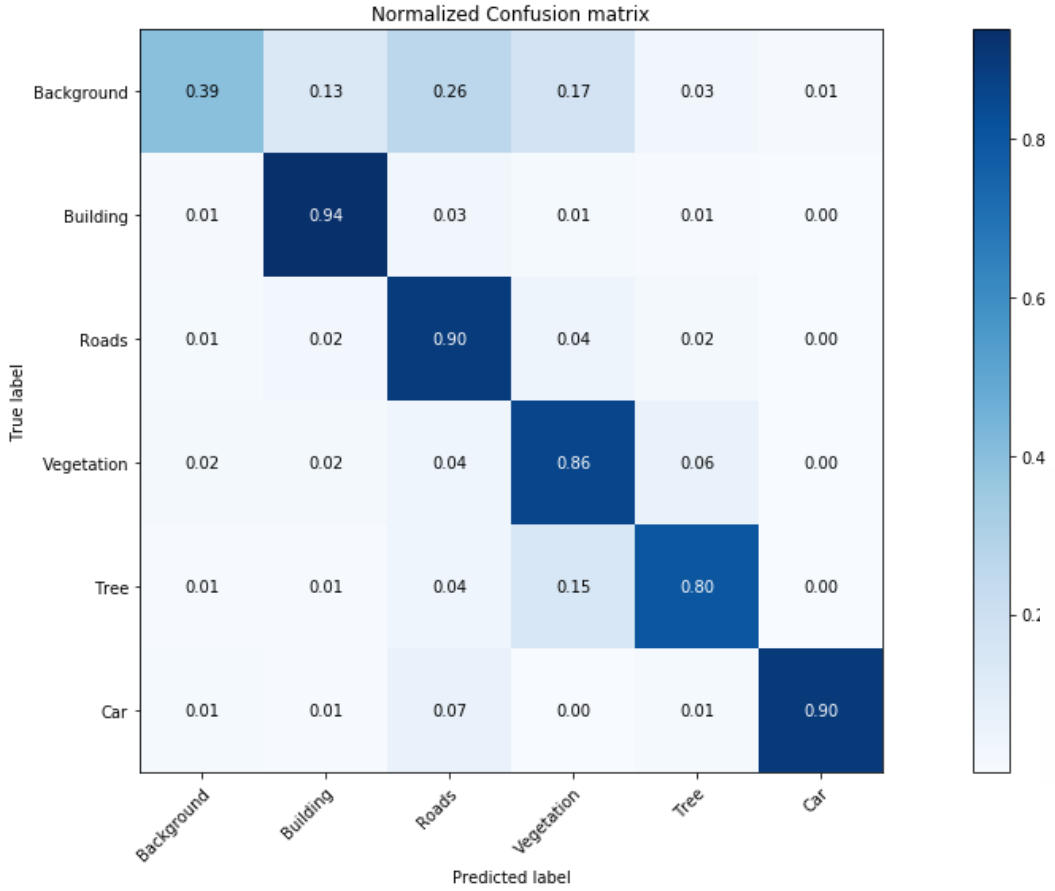
Experiment Results



Name	Smoothed	Value	Step	Time	Relative
unet_input256_noScale_skip3_noResidual_convCenter_sgd_crossentropy_bs60-20191126-11h38mn/validation	0.6351	0.635	99	Tue Nov 26, 17:00:06	4h 16m 20s
unet_input256_noScale_skip4_Residual_convCenter_sgd_crossentropy_bs60_ocpCB-20191127-11h21mn/validation	0.689	0.689	99	Wed Nov 27, 18:24:11	5h 58m 11s
unet_input256_noScale_skip4_Residual_parallelCenter_sgd_jaccardloss_bs60_ocpCB-20191130-09h17mn/validation	0.6925	0.6929	99	Sat Nov 30, 17:53:51	7h 32m 7s
unet_input256_noScale_skip4_Residual_residualCenter_sgd_crossentropy_bs60_ocpCB-20191128-22h35mn/validation	0.6758	0.6756	99	Fri Nov 29, 04:47:44	5h 8m 49s
unet_input256_noScale_skip4_Residual_residualCenter_sgd_focalloss_bs60_ocpCB-20191201-16h50mn/validation	0.6751	0.6757	99	Mon Dec 2, 00:05:05	6h 10m 22s
unet_input256_noScale_skip4_Residual_residualCenter_sgd_jaccardloss_bs60_ocpCB-20191129-10h43mn/validation	0.6909	0.6918	99	Fri Nov 29, 18:06:55	6h 18m 45s
unet_input256_noScale_skip4_Residual_serialCenter_sgd_crossentropy_bs50_ocpCB-20191128-15h15mn/validation	0.6831	0.6845	99	Thu Nov 28, 23:07:05	6h 47m 12s
unet_input256_noScale_skip4_Residual_serialCenter_sgd_focalloss_bs60_ocpCB-20191130-22h50mn/validation	0.646	0.6467	99	Sun Dec 1, 06:41:48	6h 47m 16s
unet_input256_noScale_skip4_Residual_serialCenter_sgd_focalloss_bs60_ocpCB-20191201-07h40mn/validation	0.6879	0.6885	99	Sun Dec 1, 15:31:12	6h 46m 57s
unet_input256_noScale_skip4_noResidual_convCenter_sgd_crossentropy_bs60_reduceCB-20191126-16h08mn/validation	0.634	0.6341	99	Tue Nov 26, 21:52:37	4h 41m 13s
unet_input256_noScale_skip5_noResidual_convCenter_sgd_crossentropy_bs60_reduceCB-20191126-22h30mn/validation	0.6422	0.6423	99	Wed Nov 27, 04:37:47	5h 3m 36s

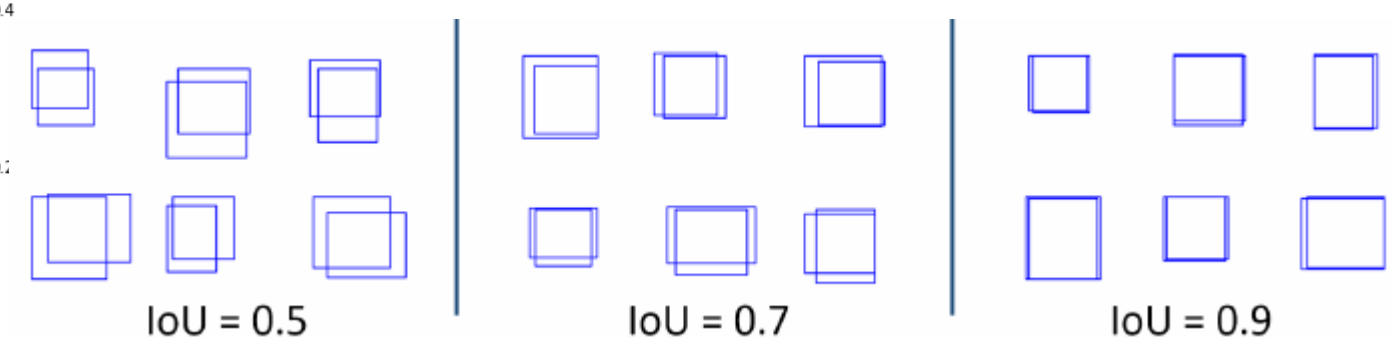
Name	Smoothed	Value	Step	Time	Relative
unet_input256_noScale_skip3_Residual_convCenter_sgd_crossentropy_bs60_ocpCB-20191127-11h32mn/validation	0.6601	0.661	99	Wed Nov 27, 17:50:55	5h 15m 23s
unet_input256_noScale_skip3_noResidual_convCenter_sgd_crossentropy_bs60-20191126-11h38mn/validation	0.6351	0.635	99	Tue Nov 26, 17:00:06	4h 16m 20s
unet_input256_noScale_skip3_noResidual_convCenter_sgd_crossentropy_bs60_ocpCB-20191126-11h45mn/validation	0.6137	0.6144	99	Tue Nov 26, 17:02:14	4h 12m 55s
unet_input256_noScale_skip4_Residual_parallelCenter_sgd_crossentropy_bs50_ocpCB-20191128-15h22mn/validation	0.6784	0.6786	99	Thu Nov 28, 23:54:15	7h 25m 14s
unet_input256_noScale_skip4_Residual_parallelCenter_sgd_focalloss_bs50_reduceCB-20191130-23h05mn/validation	0.6455	0.6456	99	Sun Dec 1, 07:34:17	7h 24m 24s
unet_input256_noScale_skip4_Residual_parallelCenter_sgd_focalloss_bs50_reduceCB-20191201-07h40mn/validation	0.6821	0.6827	99	Sun Dec 1, 16:10:05	7h 24m 49s
unet_input256_noScale_skip4_Residual_parallelCenter_sgd_jaccardloss_bs50_reduceCB-20191130-12h13mn/validation	0.6786	0.6782	99	Sat Nov 30, 20:49:30	7h 30m 56s
unet_input256_noScale_skip4_Residual_residualCenter_sgd_focalloss_bs50_reduceCB-20191201-16h51mn/validation	0.635	0.6347	99	Mon Dec 2, 00:01:37	6h 6m 2s
unet_input256_noScale_skip4_Residual_serialCenter_sgd_jaccardloss_bs50_ocpCB-20191129-10h43mn/validation	0.7005	0.7015	99	Fri Nov 29, 18:41:15	6h 52m 38s
unet_input256_noScale_skip4_noResidual_convCenter_sgd_crossentropy_bs60_ocpCB-20191126-16h09mn/validation	0.6948	0.6953	99	Tue Nov 26, 21:50:46	4h 38m 29s
unet_input256_noScale_skip4_noResidual_convCenter_sgd_crossentropy_bs60_reduceCB-20191126-16h08mn/validation	0.634	0.6341	99	Tue Nov 26, 21:52:37	4h 41m 13s
unet_input256_noScale_skip5_noResidual_convCenter_sgd_crossentropy_bs60_ocpCB-20191126-22h32mn/validation	0.696	0.6963	99	Wed Nov 27, 04:38:30	5h 2m 39s
unet_input256_noScale_skip5_noResidual_convCenter_sgd_crossentropy_bs60_reduceCB-20191126-22h30mn/validation	0.6422	0.6423	99	Wed Nov 27, 04:37:47	5h 3m 36s

Best Model Detailed Results Computed on Validation Set



Background	Building	Roads	Vegetation	Tree	Car
0.3132	0.8701	0.8030	0.6928	0.7061	0.8243

Validation Dataset Jaccard Index (IoU) per Class

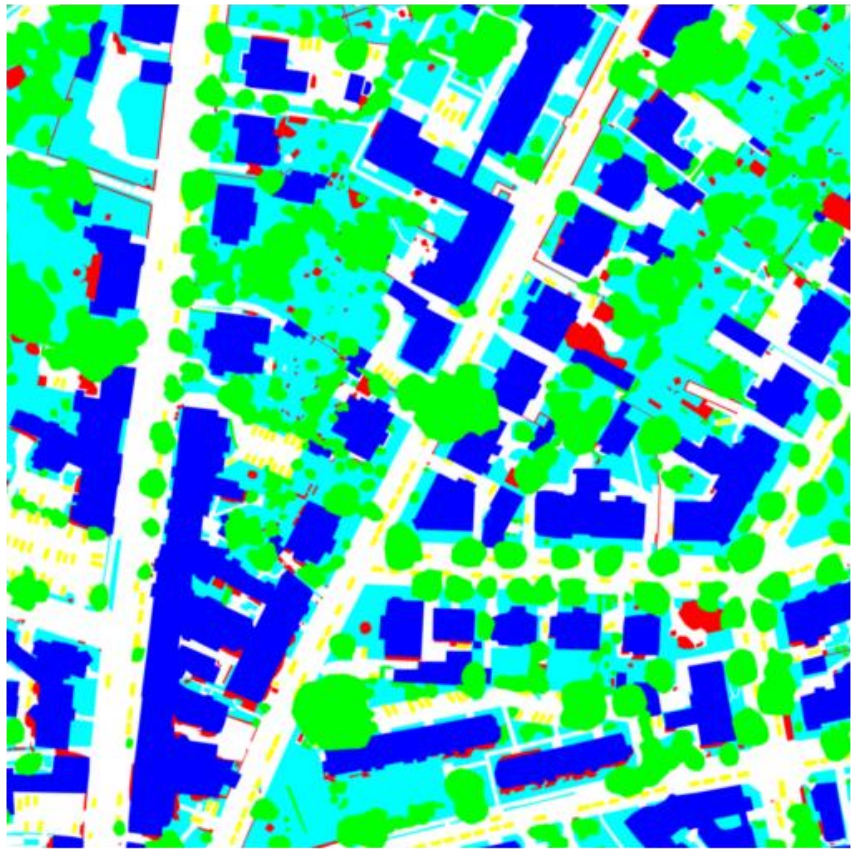


AIRBUS

Predicting on a new Tile (from Validation set)

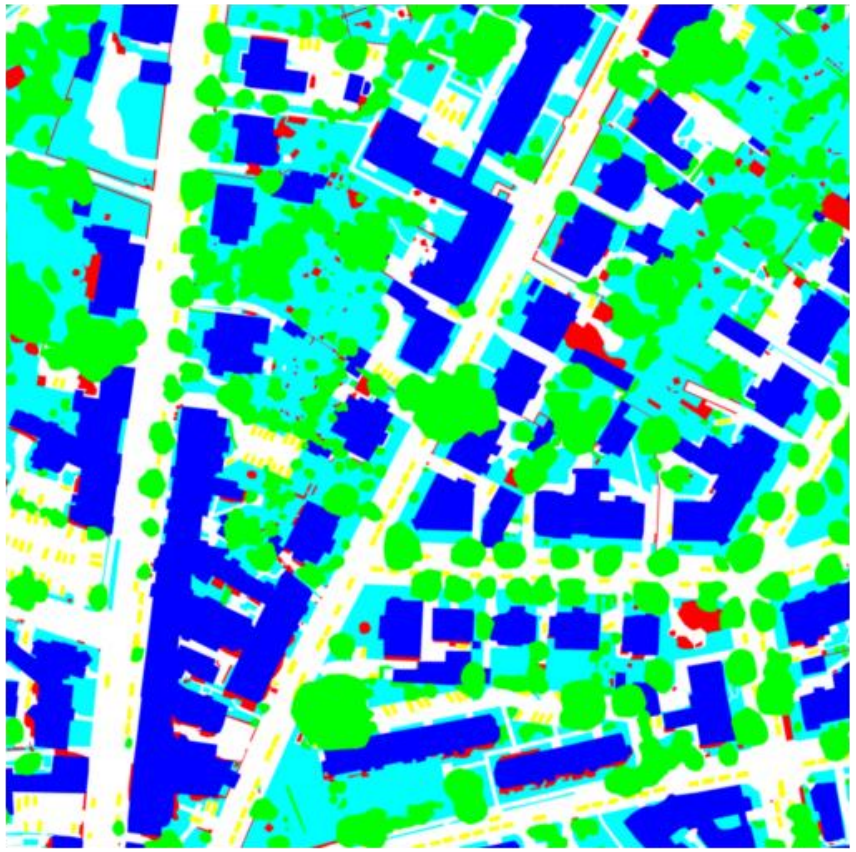
Aerial Tile

Reference Mask

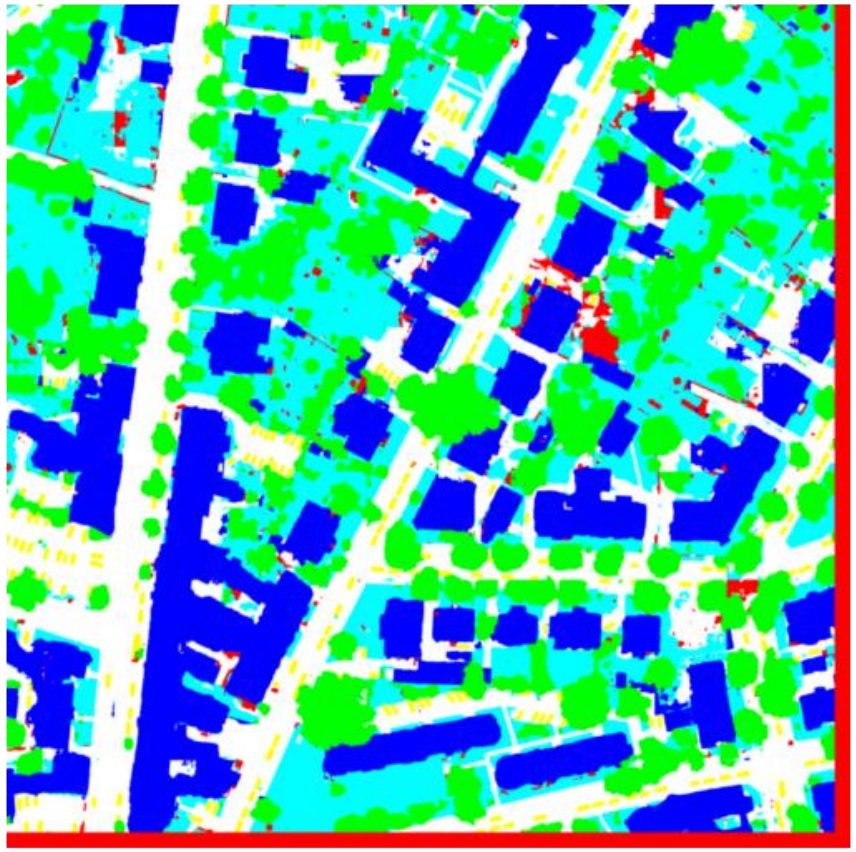


Predicting on a new Tile (from Validation set)

Reference Mask



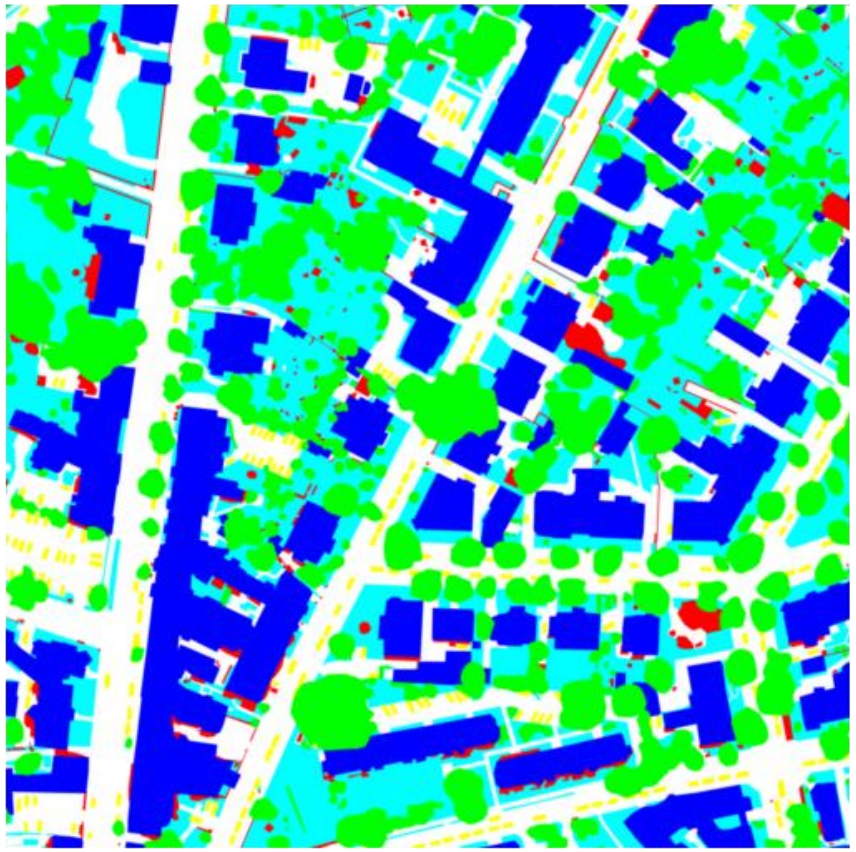
Prediction



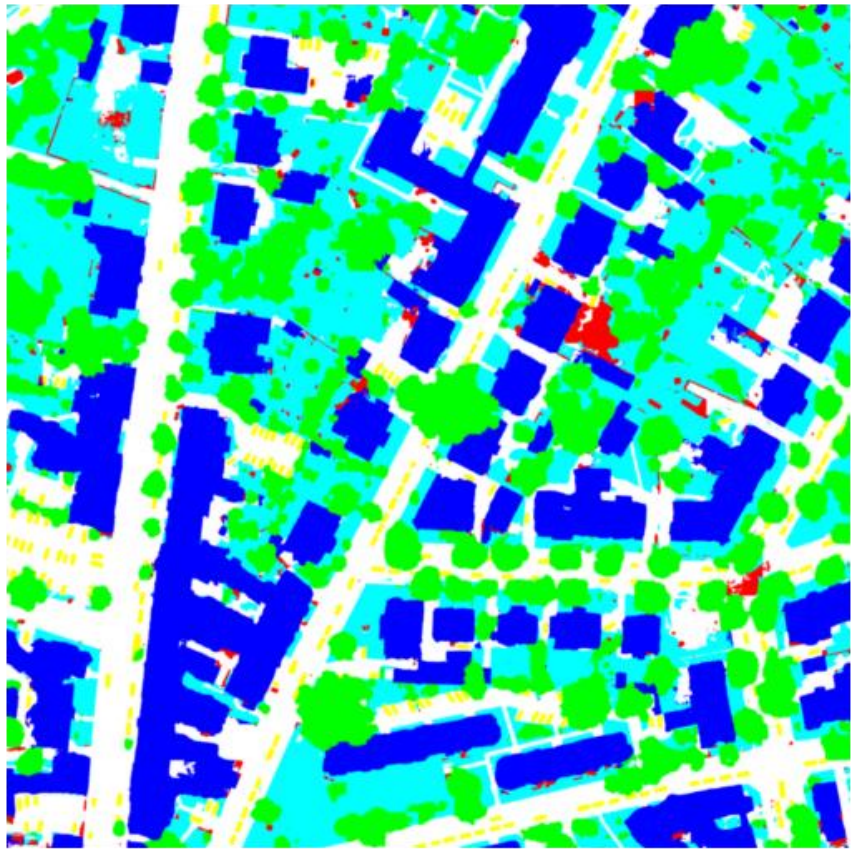
Background	Building	Roads	Vegetation	Tree	Car
0.1141	0.8755	0.7486	0.6756	0.7393	0.8093

Predicting on a new Tile with Test Time Augmentation

Reference Mask



Prediction

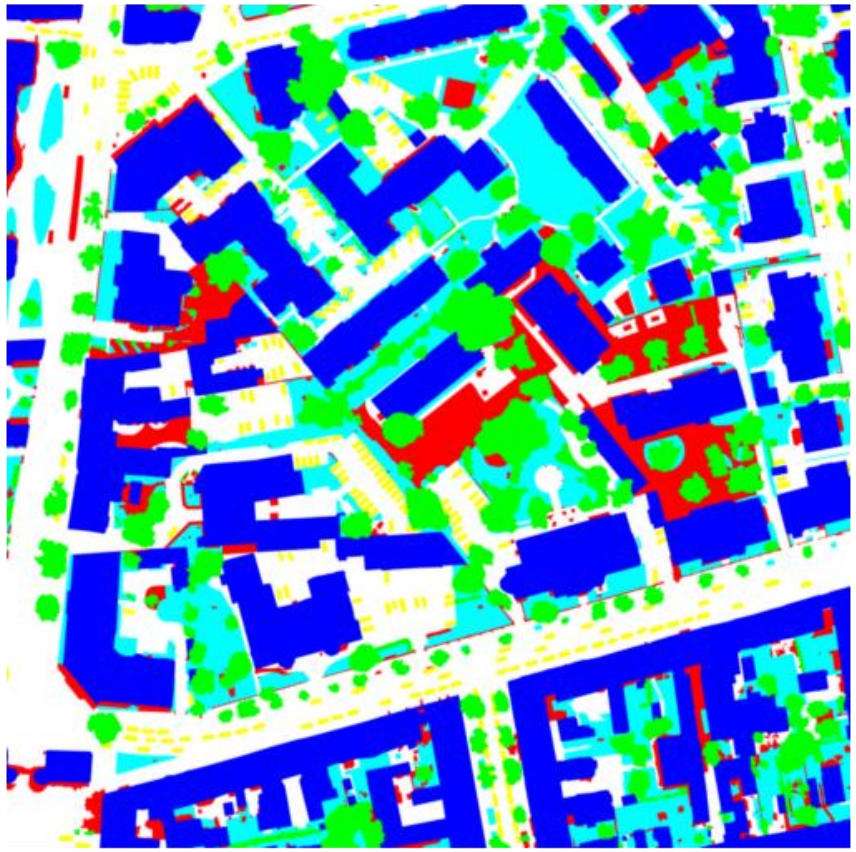


Background	Building	Roads	Vegetation	Tree	Car
0.2250	0.9143	0.7920	0.7171	0.7769	0.8416

Predicting on a new Tile (Example 2)

Aerial Tile

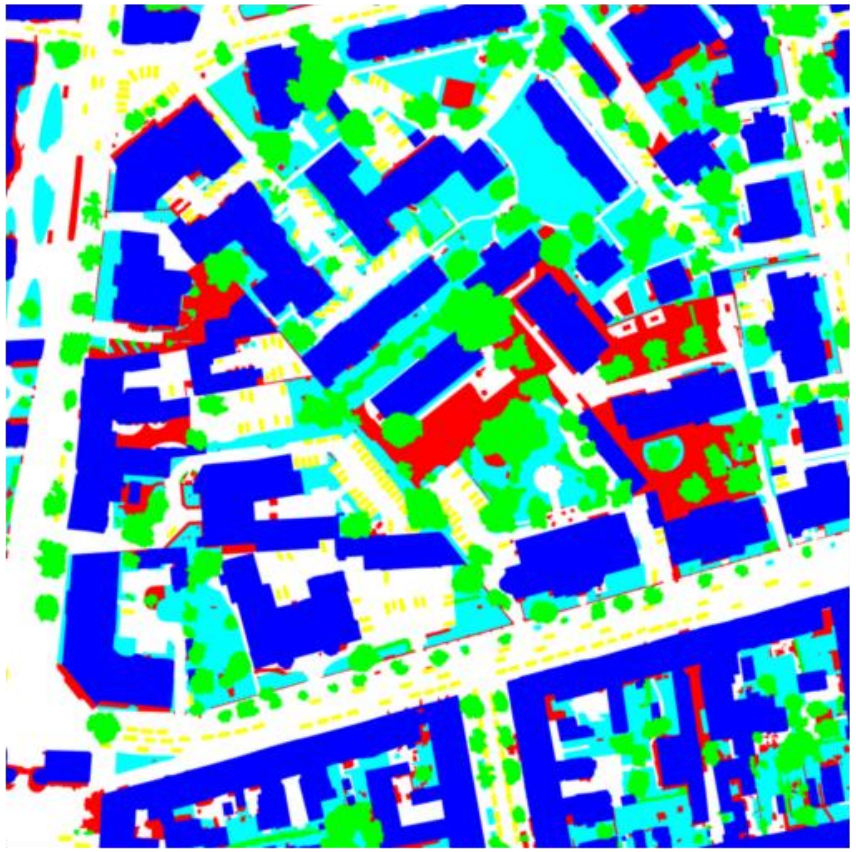
Reference Mask



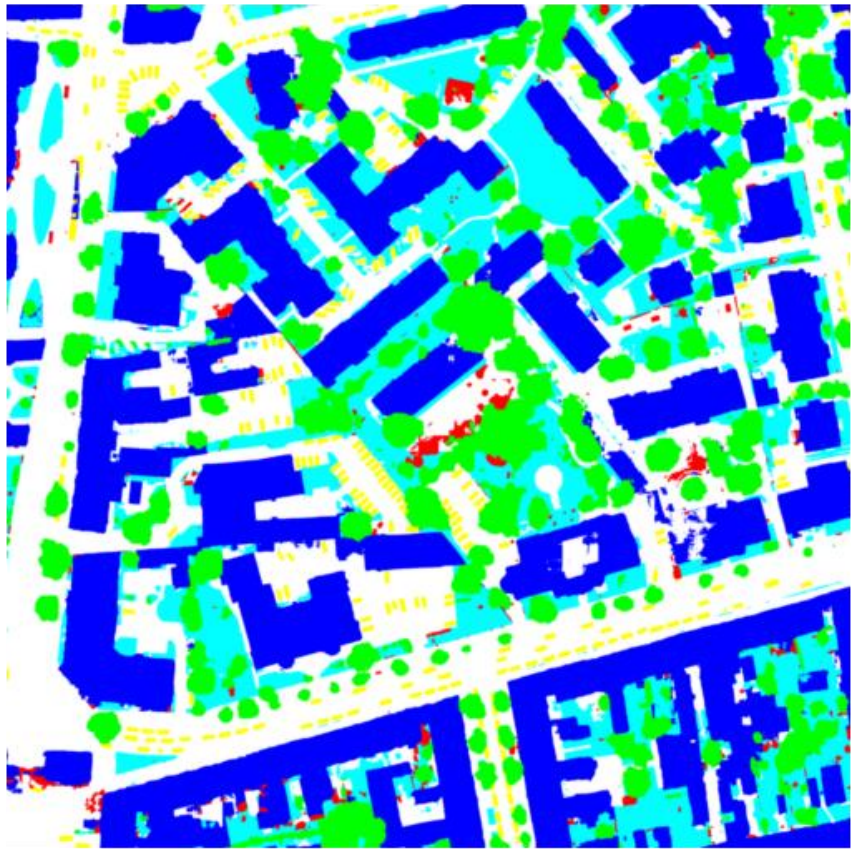
AIRBUS

Predicting on a new Tile (Example 2)

Reference Mask



Prediction

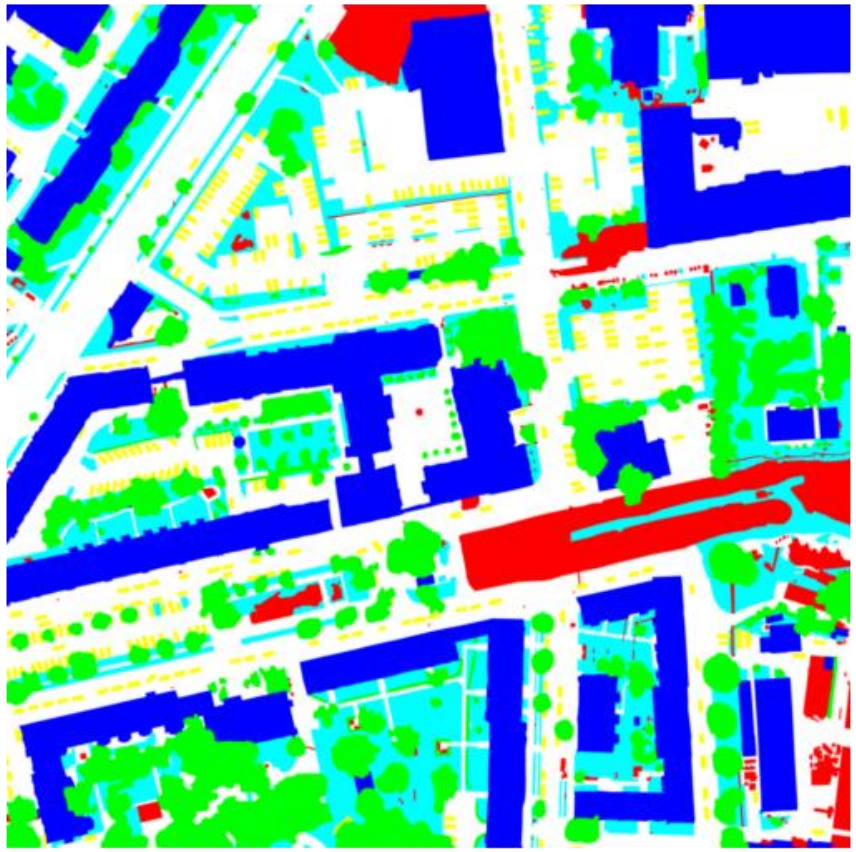


Background	Building	Roads	Vegetation	Tree	Car
0.1259	0.9212	0.7887	0.6878	0.7341	0.8615

Predicting on a new Tile (Example 3)

Aerial Tile

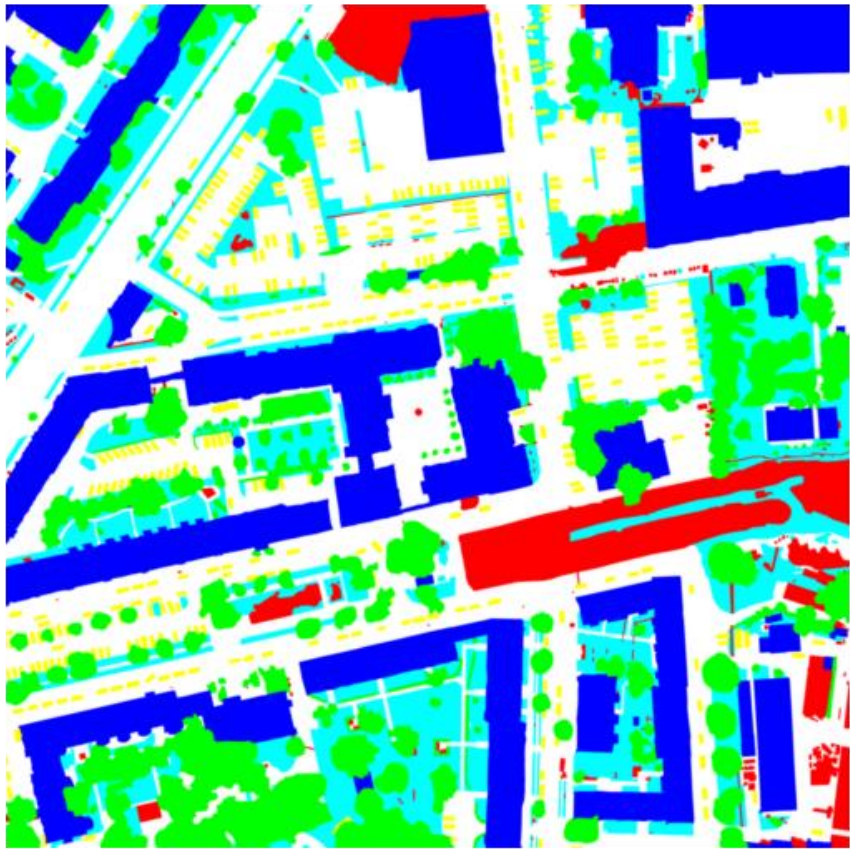
Reference Mask



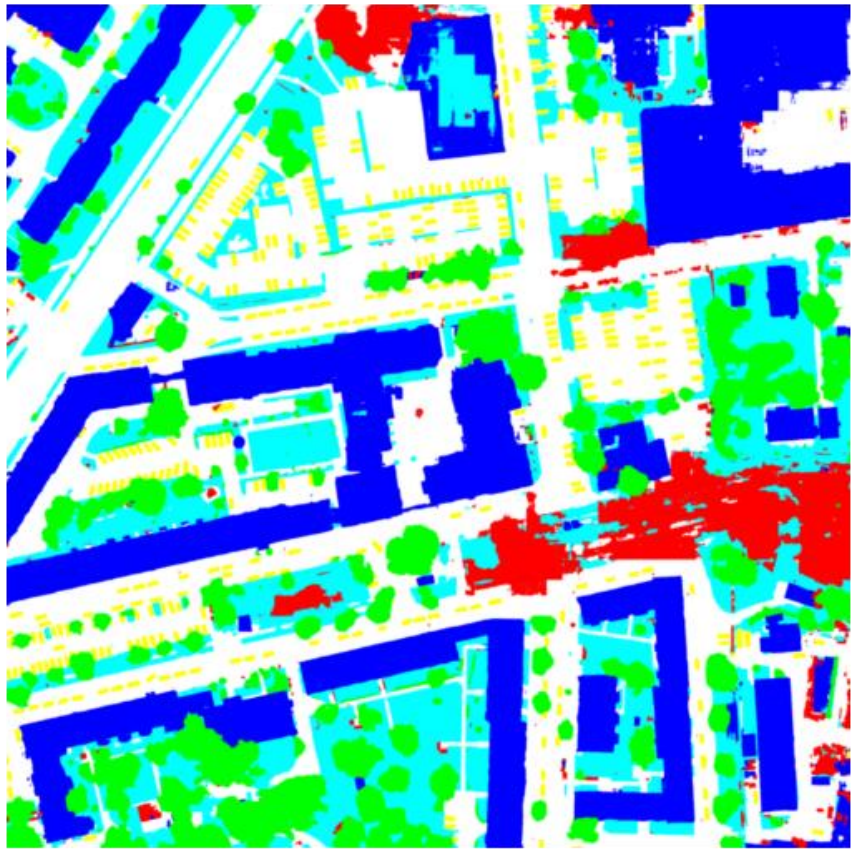
AIRBUS

Predicting on a new Tile (Example 3)

Reference Mask



Prediction



Background	Building	Roads	Vegetation	Tree	Car
0.5344	0.8806	0.8454	0.6298	0.7001	0.8502

Model into the wild

Divider subtitle

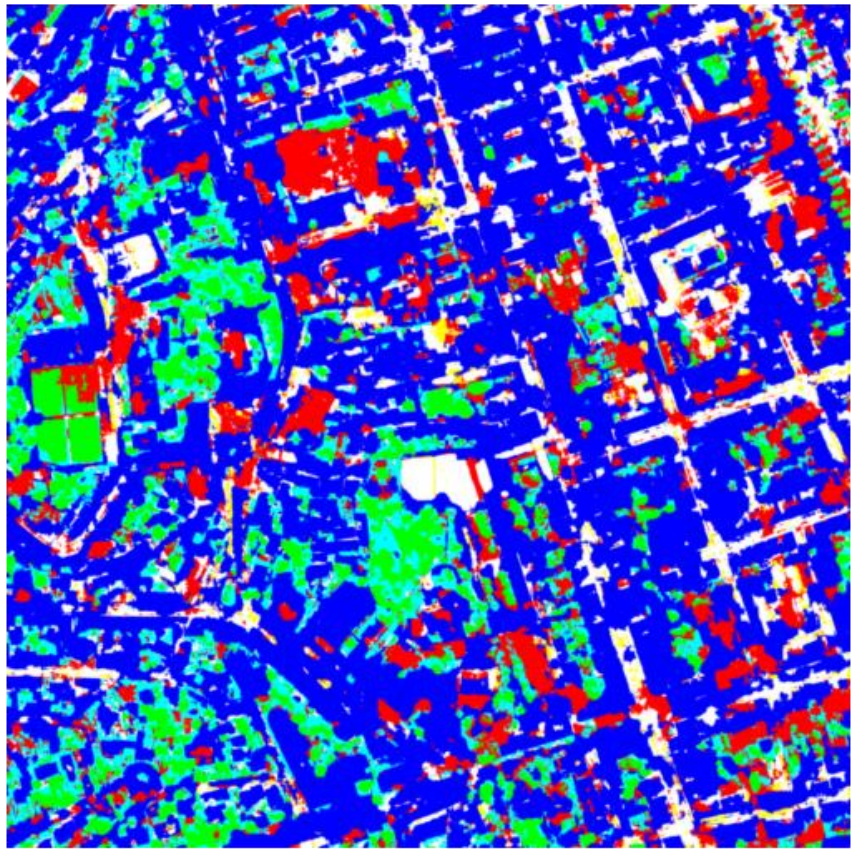
Predicting on Marseille

AIRBUS

Predicting on Marseille

Aerial Tile

Prediction

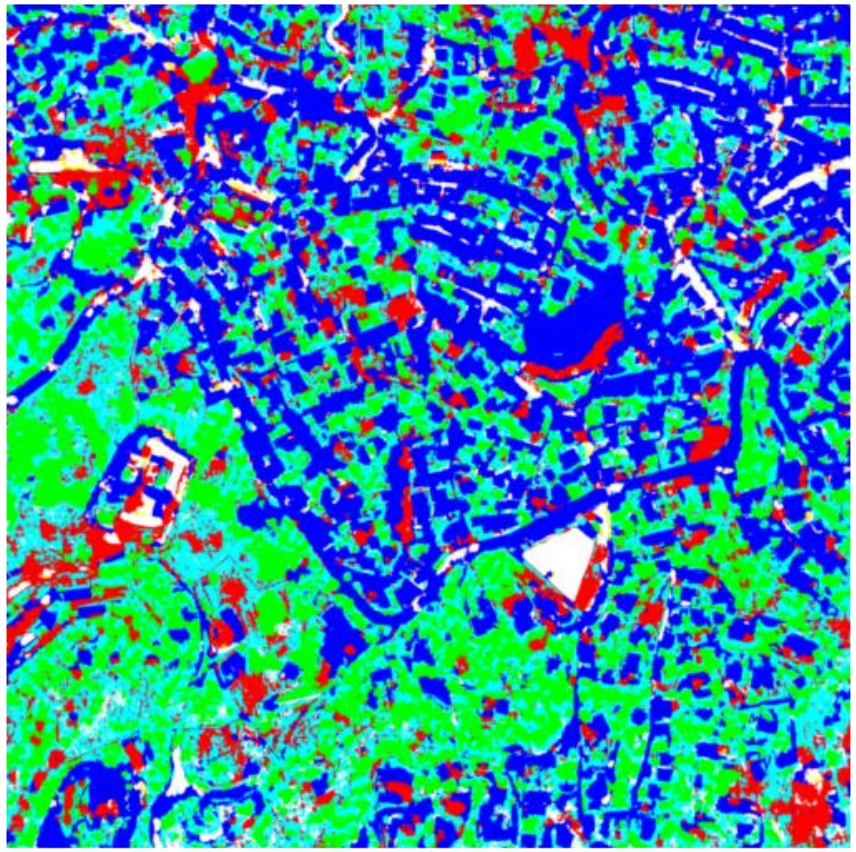


AIRBUS

Predicting on Marseille

Aerial Tile

Prediction

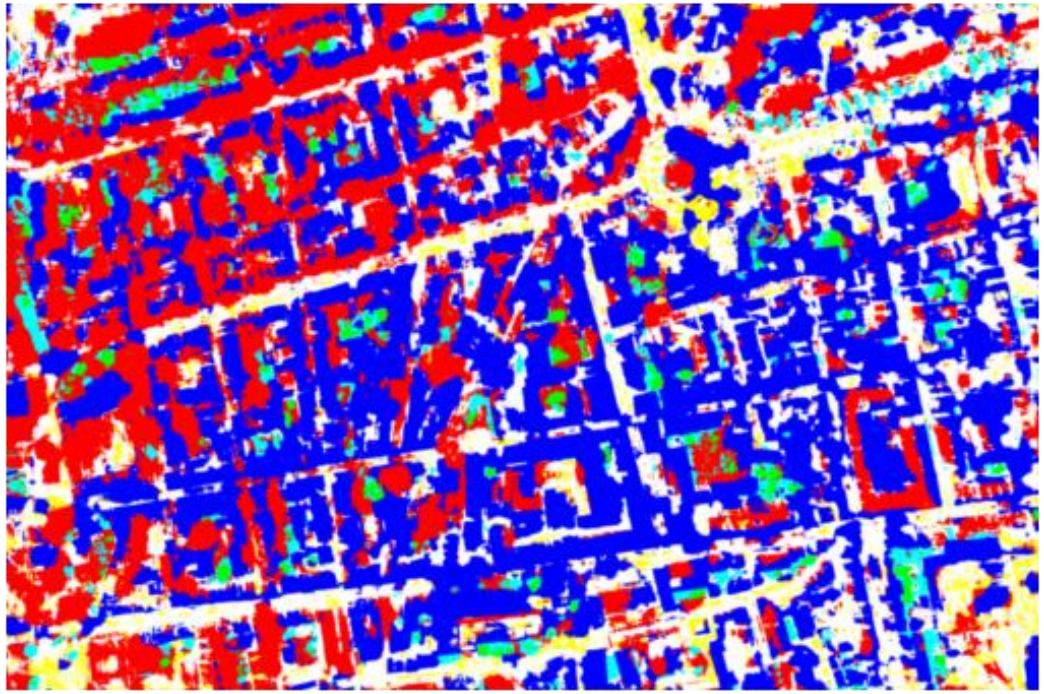


AIRBUS

Predicting on Toulouse (NADIR)

Aerial Tile

Prediction



AIRBUS

What Happened ?

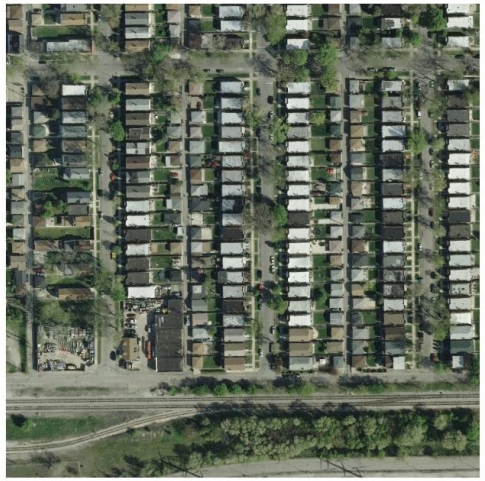
- Our Model did not generalize well.

- Possible root causes:
  - Training set is too small: limited to one city, tile resolution
  - Implemented Data Augmentation is limited: need more
  - Inference tile have challenging lightning aspects
  - Off-Nadir images does not provide good results

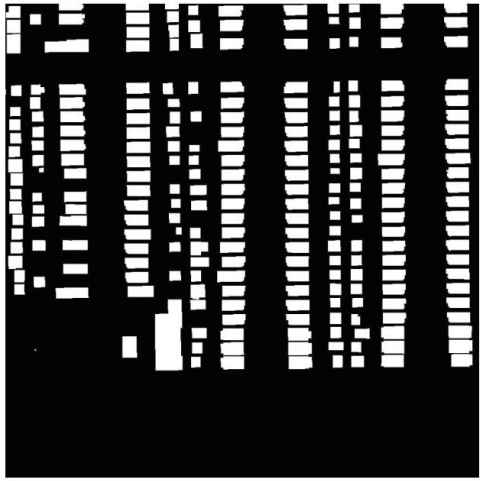
INRIA Aerial Building Dataset

INRIA Building Dataset

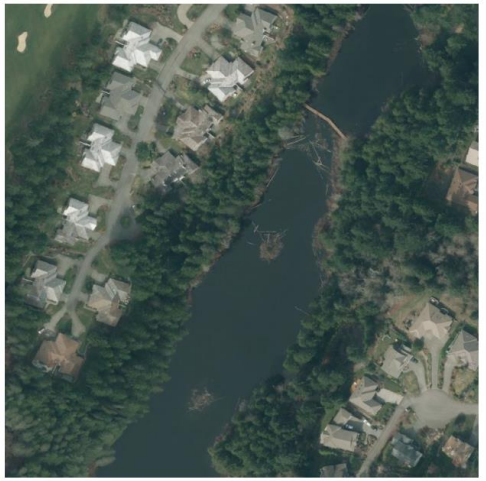
- The training set contains 180 color image tiles of size 5000×5000, covering a surface of 1500 m × 1500 m each (at a 30 cm resolution).
- There are 36 tiles for each of the following regions:
  - Austin
  - Chicago
  - Kitsap County
  - Western Tyrol
  - Vienna



Chicago



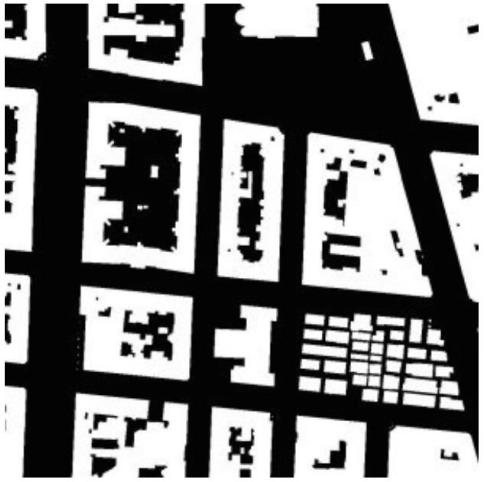
Chicago – reference



Kitsap County, WA

Kitsap County, WA – Reference

Vienna



Vienna – Reference

INRIA: Defining the Train and the Validation dataset objects

[20]:
```
train_ds = get_dataset(img_dir=inria_train_img_dir, mask_dir=inria_train_mask_dir, batch_size=BATCH_SIZE, shuffle=True, repeat=True, augment=True, cachefile='inria_train_cache')
val_ds   = get_dataset(img_dir=inria_val_img_dir, mask_dir=inria_val_mask_dir,batch_size=BATCH_SIZE, shuffle=False, repeat=True, augment=False, cachefile=None)
print("train patch number = {}, valid patch number = {}".format(TRAIN_PATCH_NUMBER, VALID_PATCH_NUMBER))
print("train steps = {}, valid steps = {}".format(TRAIN_STEPS_PER_EPOCH, VAL_STEPS_PER_EPOCH))
```

```
Removing old cache file: /tmp/inria_train_cache.index
Removing old cache file: /tmp/inria_train_cache.data-00000-of-00001
train patch number = 361, valid patch number = 361
train steps = 1120, valid steps = 181
```

[22]:
```
timepipeline(train_ds, batch_size=BATCH_SIZE, steps=4000)
```

Error displaying widget: model not found

```
batche size: 50
4000 batches: 414.9433560371399 s
9.63987 Batches/s
481.99350 Images/s
```

[28]:
```
timepipeline(val_ds, batch_size=BATCH_SIZE, steps=1000)
```

Error displaying widget: model not found

```
batche size: 50
1000 batches: 41.72989892959595 s
23.96363 Batches/s
1198.18167 Images/s
```

INRIA: Defining the Model

[47]:
```
inria_model = get_unet_model(MODEL_SHAPE, filters=32, n_block=4, n_class=2, mode=None, residual=True)
inria_model.summary()
```

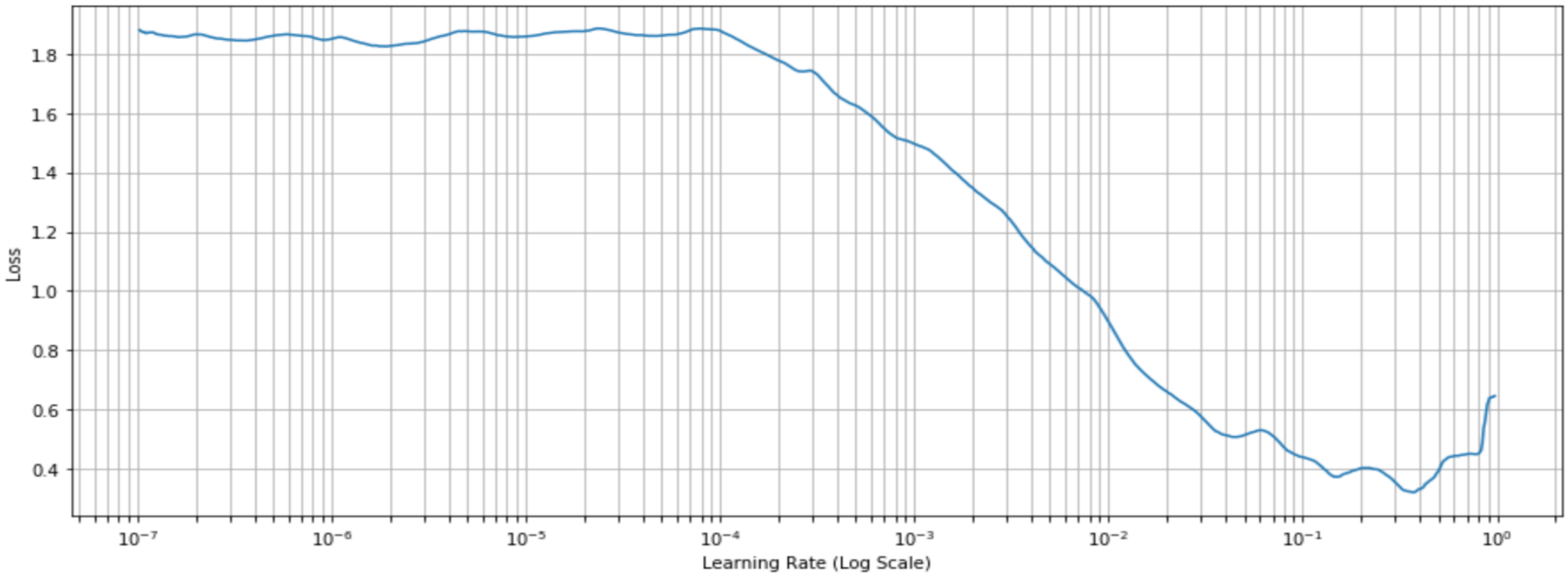
```
Total params: 10,730,984
Trainable params: 10,725,096
Non-trainable params: 5,888
```

[ ]:
```
experiment_name = "inria_unet_input256_noScale_skip4_Residual_residualCenter_sgd_cross_entropy_bs50_ocpCB"
sgd_optim = tf.keras.optimizers.SGD(lr=0.1, nesterov=True)
acc_metric = tf.keras.metrics.SparseCategoricalAccuracy(name='Accuracy')
iou_metric = tf.keras.metrics.MeanIoU(num_classes=NUM_CLASSES)
fixed_iou  = CategoricalMeanIoU(num_classes=NUM_CLASSES)
inria_model.compile(optimizer=sgd_optim, loss='sparse_categorical_crossentropy', metrics=[acc_metric,fixed_iou])
```

INRIA: Finding the Learning Rate

[41]:
```
lrf = LearningRateFinder(inria_model)
lrf.find(train_ds, 1e-7, 1, epochs=1, stepsPerEpoch=TRAIN_STEPS_PER_EPOCH)
lrf.plot_loss()
```

```
Train for 1120 steps
1120/1120 [==============================] - 794s 709ms/step - loss: 1.3063 - Accuracy: 0.6399 - mean_iou: 0.3948
```



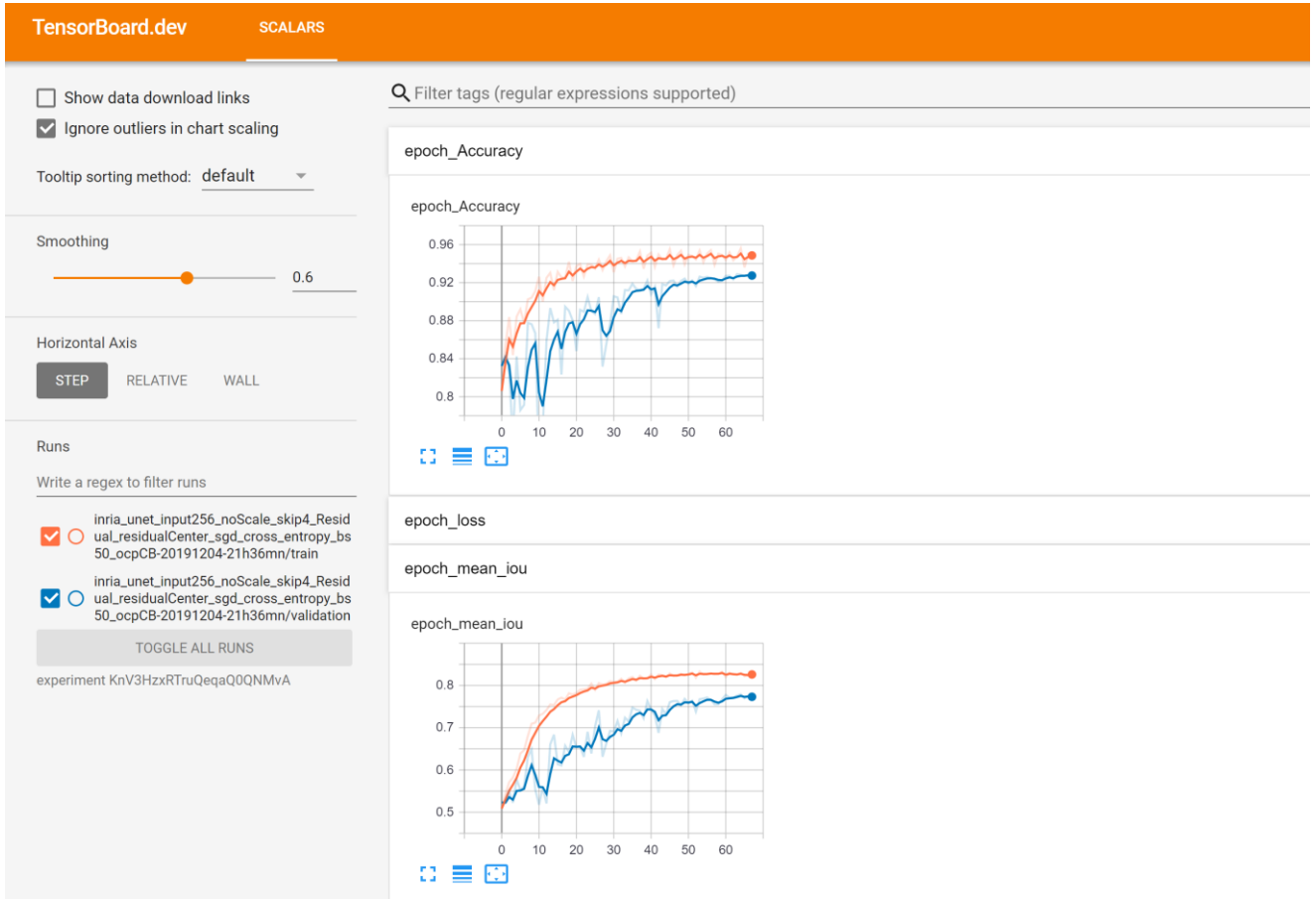
AIRBUS

INRIA: Training Phase

[ ]:
```
# One cycle policy training
checkpoint_path = "../inria_checkpoint_dir/" + experiment_name + datetime.now().strftime("-%Y%m%d-%Hh%Mmn") + "/best-cp-" + experiment_name
tb_log_dir      = "../inria_log_dir/" + experiment_name + datetime.now().strftime("-%Y%m%d-%Hh%Mmn")
# callbacks
ocp_cb = OneCycleScheduler(max_lr=1e-1)
checkpoint_cb  = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path, monitor='val_mean_iou', mode='max', save_weights_only=True, save_best_only=True, verbose=1)
tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir=tb_log_dir)

keras_history = inria_model.fit(train_ds,
                    steps_per_epoch=TRAIN_STEPS_PER_EPOCH,
                    epochs=70,
                    validation_data=val_ds,
                    validation_steps=VAL_STEPS_PER_EPOCH,
                    callbacks=[ocp_cb, checkpoint_cb, tensorboard_cb],
                    verbose=1)
```

```
Train for 1120 steps, validate for 181 steps
Epoch 1/70
1119/1120 [============================>.] - ETA: 0s - loss: 0.5033 - Accuracy: 0.8061 - mean_iou: 0.5086 - OneCycleScheduler, lr: 0.0045361, momentum: 0.9491623

Epoch 00001: val_mean_iou improved from -inf to 0.52302, saving model to ../inria_checkpoint_dir/inria_unet_input256_noScale_skip4_Residual_residualCenter_sgd_cross_entropy_bs50_ocpCB-2019
esidual_residualCenter_sgd_cross_entropy_bs50_ocpCB
1120/1120 [==============================] - 833s 744ms/step - loss: 0.5031 - Accuracy: 0.8062 - mean_iou: 0.5086 - val_loss: 0.4252 - val_Accuracy: 0.8324 - val_mean_iou: 0.5230
Epoch 2/70
1119/1120 [============================>.] - ETA: 0s - loss: 0.3631 - Accuracy: 0.8538 - mean_iou: 0.5426 - OneCycleScheduler, lr: 0.0061325, momentum: 0.9466680

Epoch 00002: val_mean_iou did not improve from 0.52302
1120/1120 [==============================] - 830s 741ms/step - loss: 0.3631 - Accuracy: 0.8538 - mean_iou: 0.5426 - val_loss: 0.3866 - val_Accuracy: 0.8473 - val_mean_iou: 0.5223
Epoch 3/70
 296/1120 [======>.......................] - ETA: 9:31 - loss: 0.3196 - Accuracy: 0.8729 - mean_iou: 0.5586
```

INRIA: Monitoring the Training Phase



AIRBUS

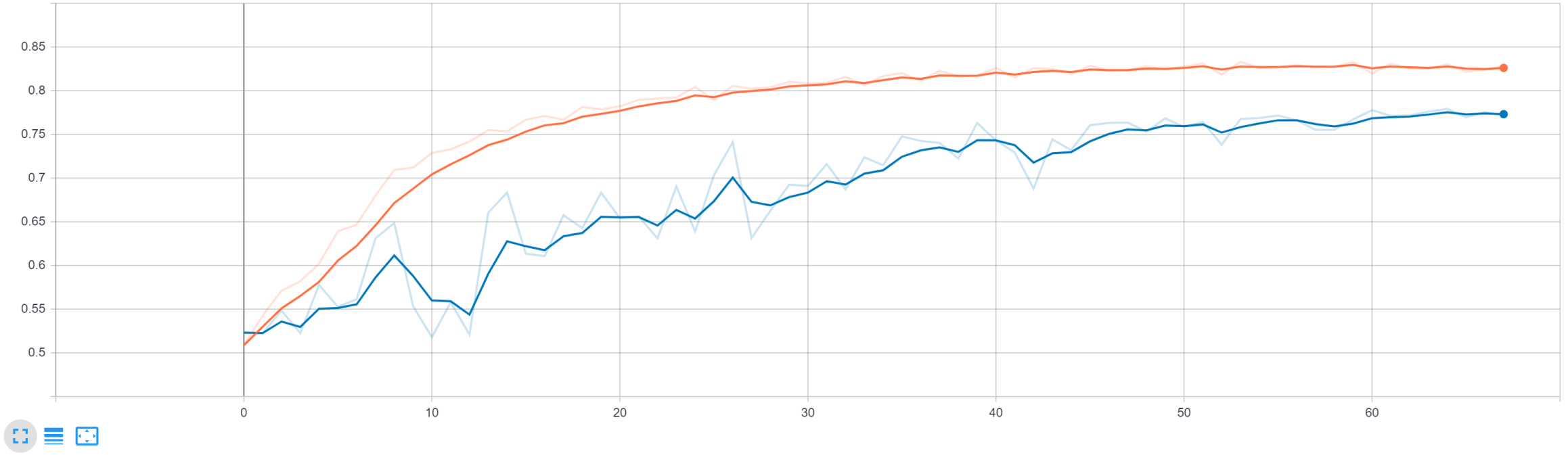
INRIA: Monitoring the Training Phase

epoch_loss

epoch_mean_iou

Name	Smoothed	Value	Step	Time	Relative
inria_unet_input256_noScale_skip4_Residual_residualCenter_sgd_cross_entropy_bs50_ocpCB-20191204-21h36mn/train	0.8261	0.828	67	Thu Dec 5, 14:18:31	15h 28m 13s
inria_unet_input256_noScale_skip4_Residual_residualCenter_sgd_cross_entropy_bs50_ocpCB-20191204-21h36mn/validation	0.7731	0.7718	67	Thu Dec 5, 14:18:31	15h 28m 13s

epoch_mean_iou



AIRBUS

INRIA: Evaluation

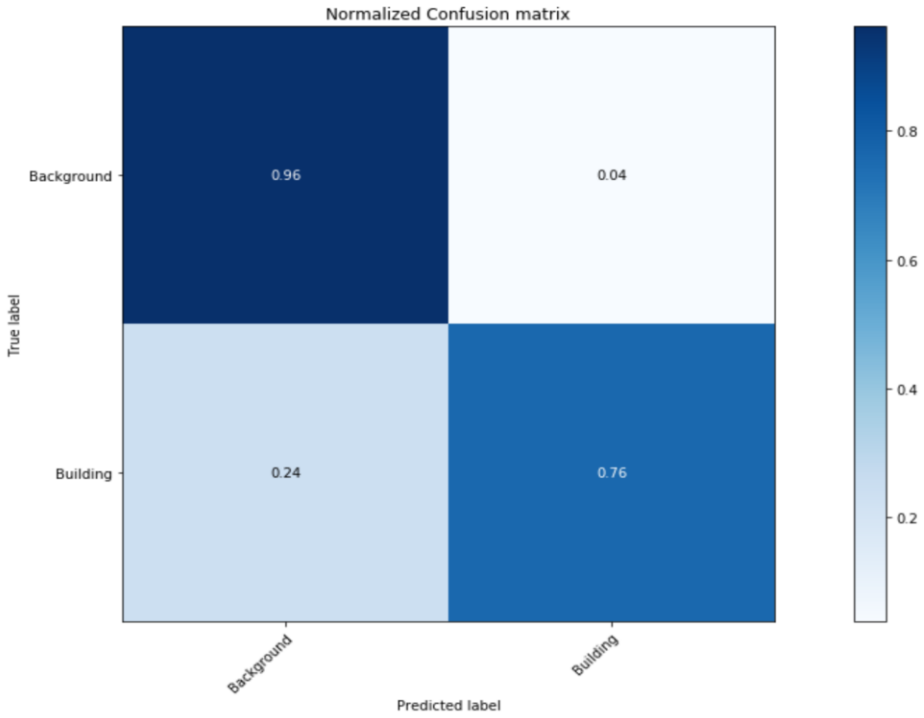
```
[ ]: inria_model.load_weights('../inria_checkpoint_dir/.../best-cp-inria_unet_input256_noScale_skip4_Residual_residualCenter_sgd_cross_entropy_bs50_ocpCB')
```

```
[46]: inria_model.evaluate(val_ds, steps=VAL_STEPS_PER_EPOCH)
```

```
181/181 [==============================] - 53s 293ms/step - loss: 0.1810 - Accuracy: 0.9283 - mean_iou: 0.7791
[46]: [0.1810000821824561, 0.92829496, 0.77909243]
```

```
[50]: compute_dataset_iou(inria_model, val_ds, VAL_STEPS_PER_EPOCH)
```

Error displaying widget: model not found



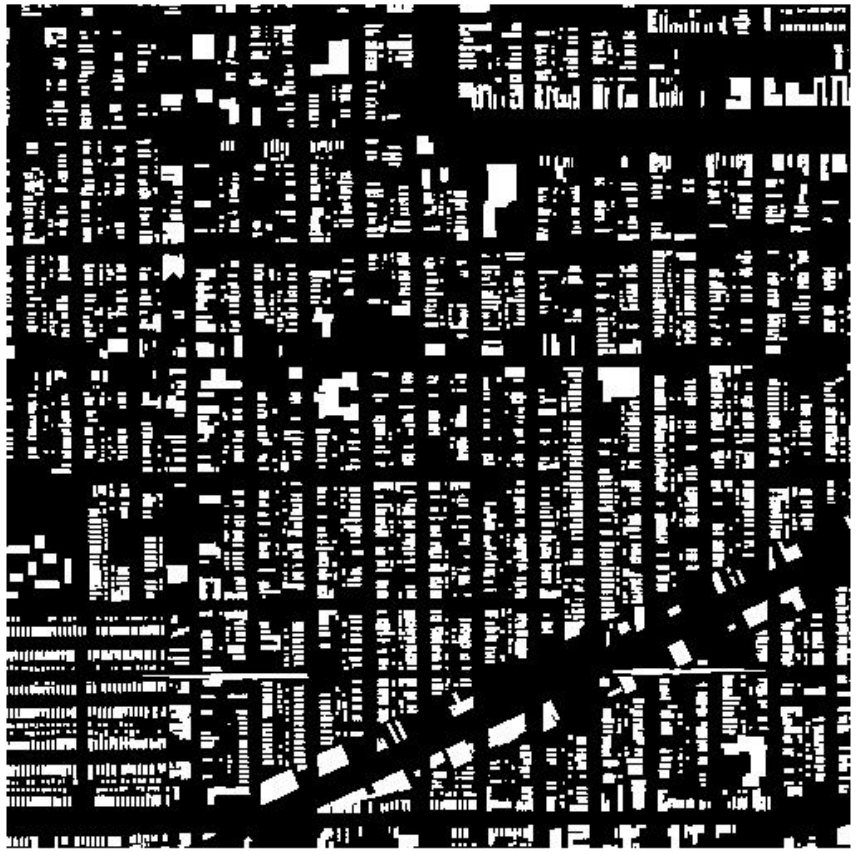
[50]:

	Background	Building
0	0.9178	0.6404

INRIA: predicting on a Validation tile (chicago)

Aerial Tile

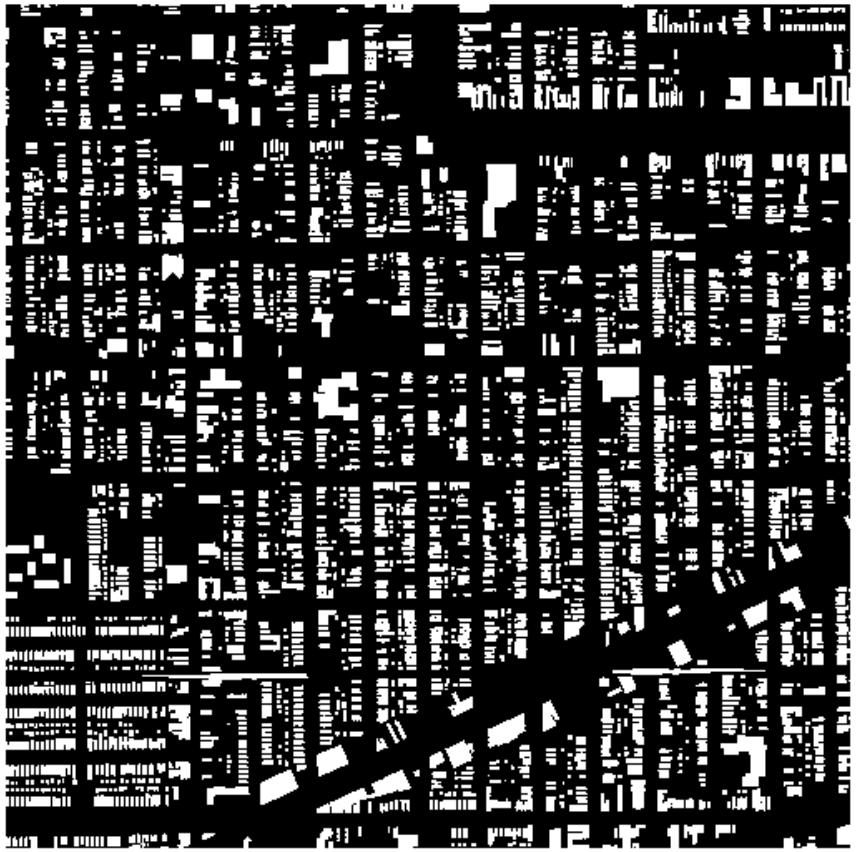
Reference Mask



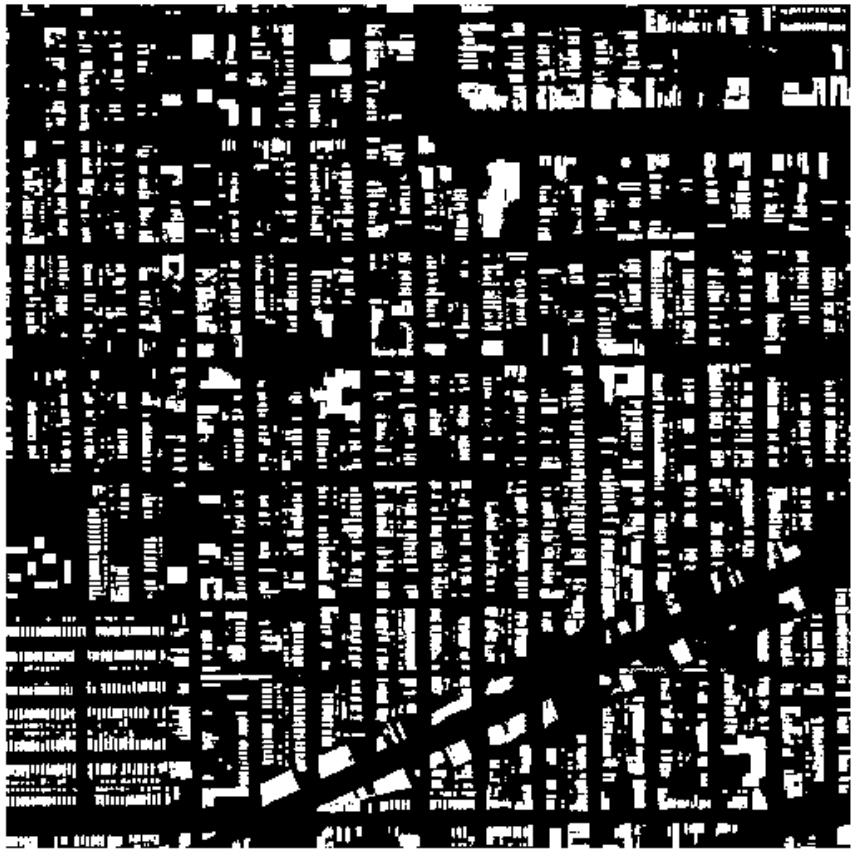
AIRBUS

INRIA: predicting on a Validation tile (chicago)

Reference Mask



Prediction



```
[159]: get_tile_iou(msk, y_pred2)
```

[159]:

	Background	Building
0	0.8908	0.6419

AIRBUS

INRIA: Predicting on Marseille

AIRBUS

INRIA: Predicting on Marseille

Aerial Tile

Prediction

AIRBUS

INRIA: Predicting on Marseille (oblique image)

Aerial Tile

Prediction

AIRBUS

Conclusion

- Produced a fast, flexible and robust input pipeline that make optimal usage of CPU & GPU resources
- Trained Deep learning Neural Network for segmentation with transparent performance metrics and reproductible flow
- Tried cutting-edge research technique to train neural networks (learning rate finder, One Cycle Policy)
- Tried New Loss function (Focal Loss, Jaccard Index Loss)
- Reached state of the Art results on the ISPRS dataset
- Learned (the hard way) that Model Performance is tightly coupled with its training data: Variety and Quantity are paramount

Where to go from here

- Synthetic datasets: Generative model for data augmentation
- Combine Geometric Computer vision with Deep Learning
- Prediction post processing

Bibliography

- **UNet: Convolutional Networks for Biomedical Image Segmentation:** https://arxiv.org/abs/1505.04597
- **Cyclical Learning Rates for Training Neural Networks:** https://arxiv.org/abs/1506.01186
- **A disciplined approach to neural network hyper-parameters: Part 1 learning rate, batch size, momentum, and weight decay:** https://arxiv.org/abs/1803.09820
- **Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates:** https://arxiv.org/abs/1708.07120
- **Focal Loss for Dense Object Detection:** https://arxiv.org/abs/1708.02002
- **Deep Residual Learning for Image Recognition:** https://arxiv.org/abs/1512.03385
- **Densely Connected Convolutional Networks:** https://arxiv.org/abs/1608.06993
- **Semantic Labeling in Very High Resolution Images via a Self-Cascaded Convolutional Neural Network:** https://www.researchgate.net/publication/321970661_Semantic_Labeling_in_Very_High_Resolution_Images_via_a_Self-Cascaded_Convolutional_Neural_Network_ISPRS_JPRS_IF6942

Thank you