Carvana Image Masking Challenge: Implementing a U-net

Jiawei Yu

Feb. 22, 2019

Kaggle task link:

https://www.kaggle.com/c/carvana-image-masking-challenge

Outline

- Task overview
- U-net
- Data preprocessing & generation
- Implementing & tweaking a U-net
- Data augmentation
- Results analyses & visualization

1.

Task Overview

Training/testing data visualization, train/val/test split plan.

“

Develop an algorithm that ***automatically*** removes the photo studio background

Training data

Image

Mask

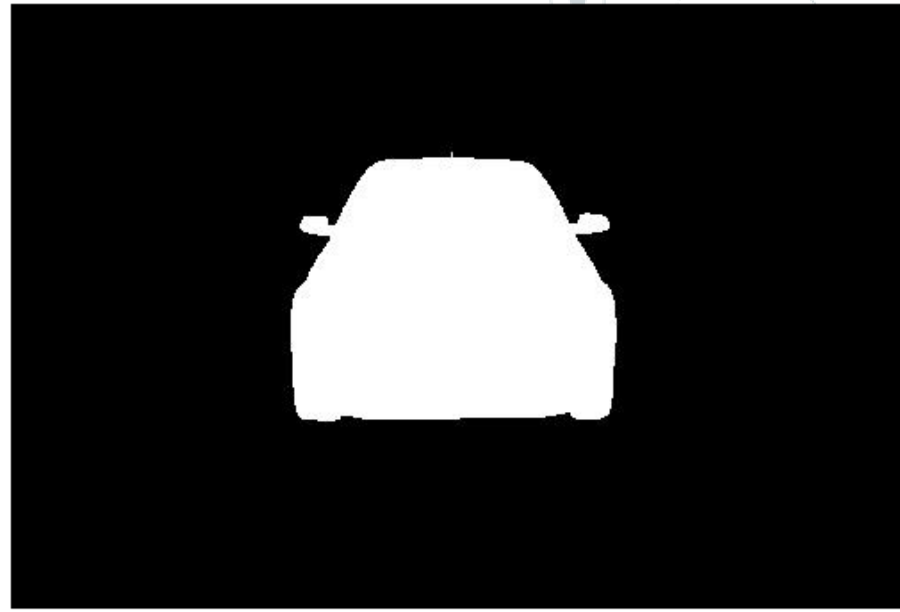


Image information

JPG format

RGB mode (3 channels)

'uint8' data ranging from 0 to 255

Size of 1918 × 1280 pixels

Mask information

GIF format

INDEX mode (1 channel)

'uint8' data with 0 and 1.

Size of 1918 × 1280 pixels

Train data w label (mask)

Total **5088** images/masks.

318 unique cars, each has **16** images taken from 16 different angles.

Test data w/o label

Total **100064** images/masks.

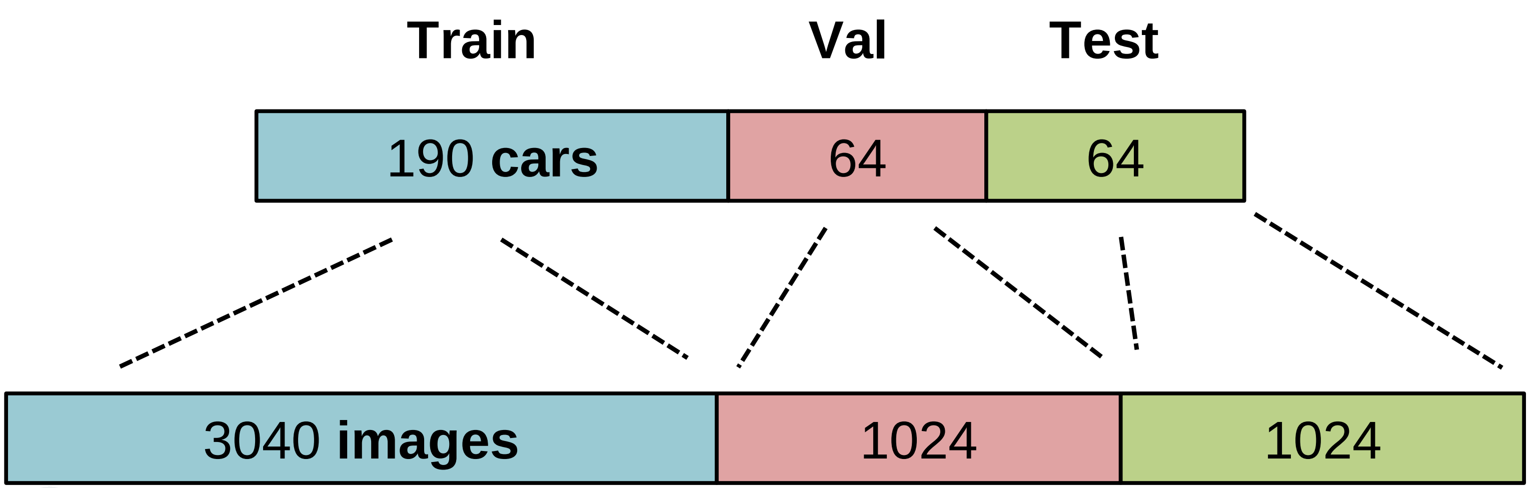
6254 unique cars, each has **16** images taken from 16 different angles.

Train data w label (mask)

Total **5088** images/masks.

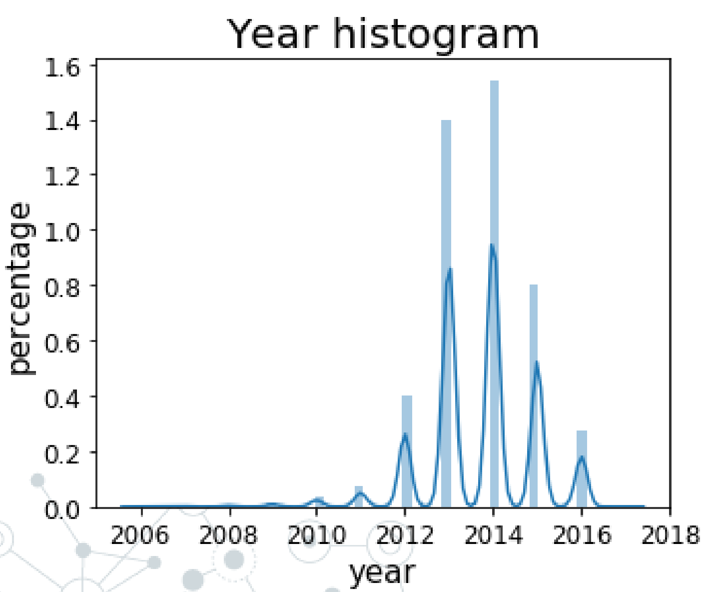
318 unique cars, each has **16** images taken from 16 different angles.

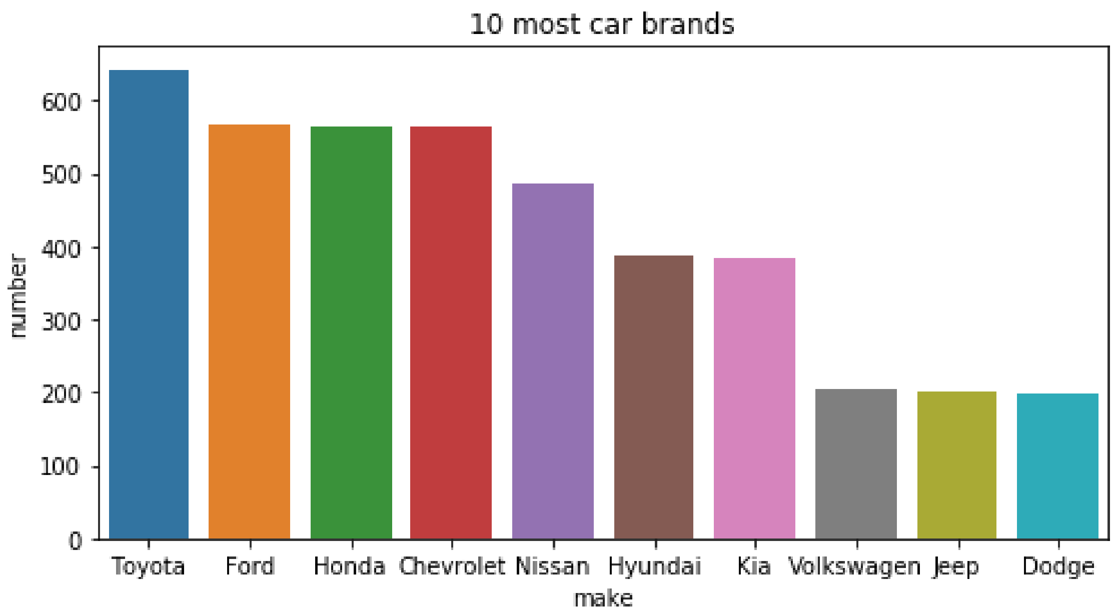
Data leakage alert! Images of a same car with different angles should NOT be split into different pools (train/validation/test)



Meta data

	id	year	make	model	trim1	trim2
0	0004d4463b50	2014.0	Acura	TL	TL	w/SE
1	00087a6bd4dc	2014.0	Acura	RLX	RLX	w/Tech
2	000aa097d423	2012.0	Mazda	MAZDA6	MAZDA6	i Sport
3	000f19f6e7d4	2016.0	Chevrolet	Camaro	Camaro	SS
4	00144e887ae9	2015.0	Acura	TLX	TLX	SH-AWD V6 w/Advance Pkg



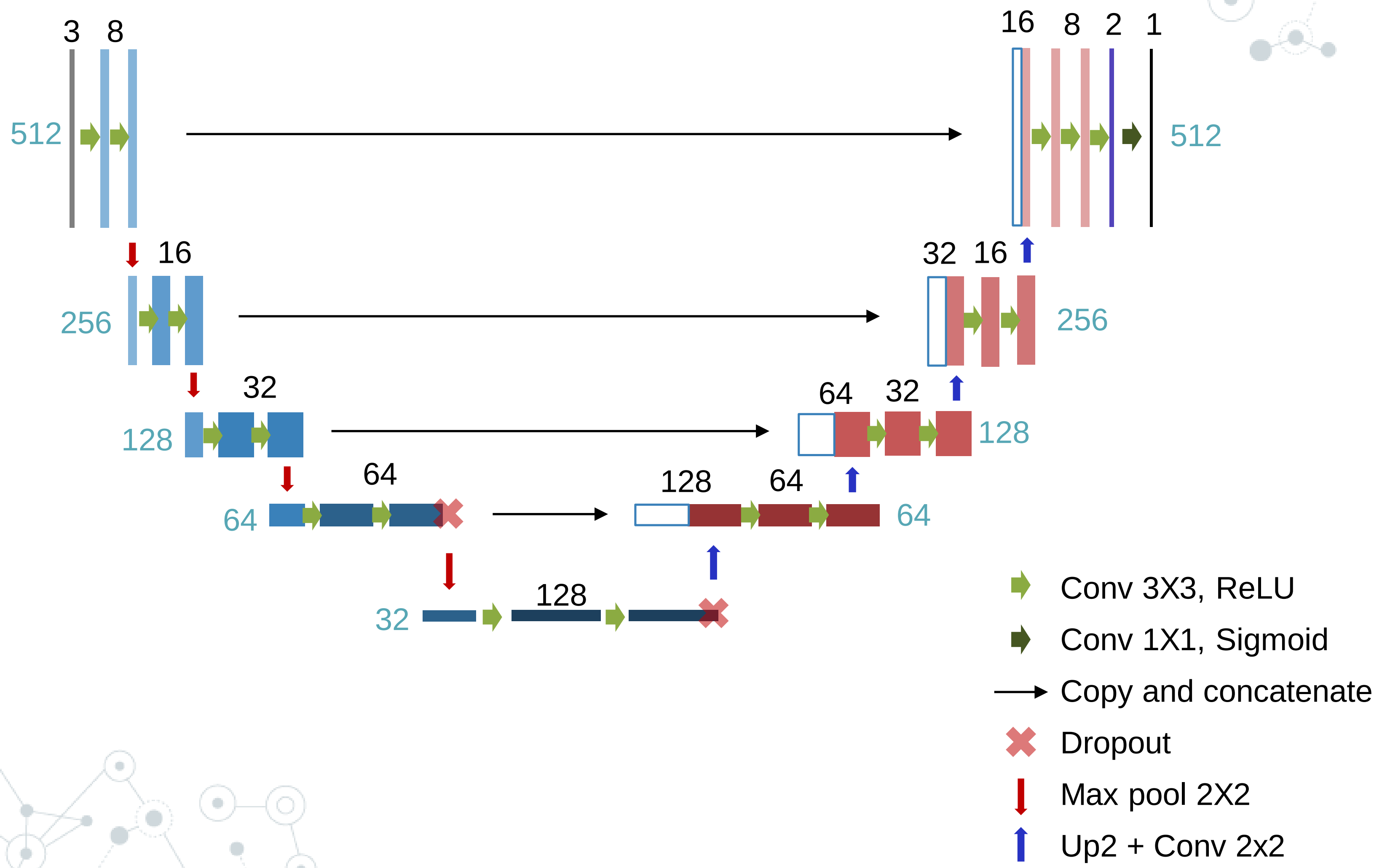


2.

U-net

Powerful end-to-end CNN used for image segmentation

U-net architecture



Reference: Olaf et al., [arXiv:1505.04597]

3.

Data preprocessing & generation

Image resize, train/val/test split and data generator

Resize the image & train/val/test split

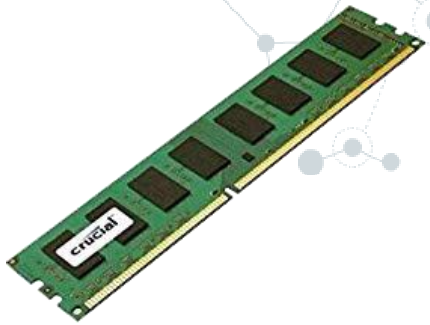
```
dic_Im_names = {'train': Im_trains,
                'val': Im_vals,
                'test': Im_tests}
```

Contains all unique car names as a list.

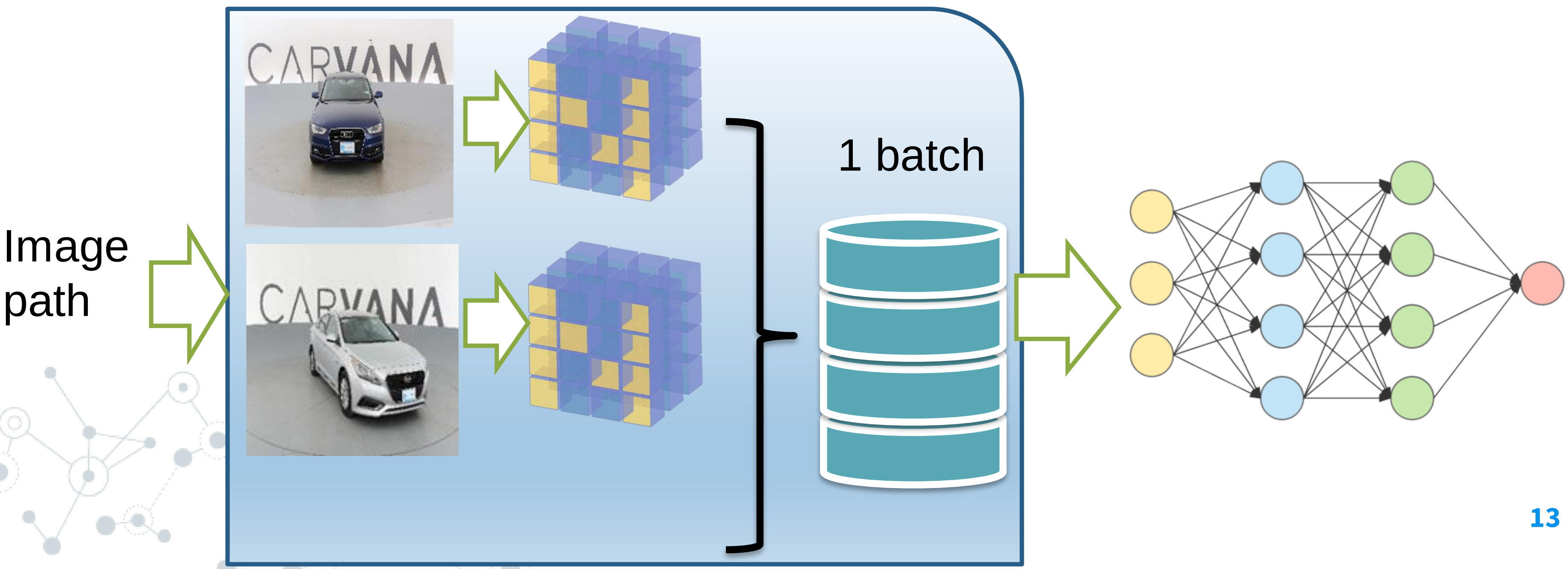
Data generator

```
print('X_train_raw shape is ', np.shape(X_train_raw))
print('y_train_raw shape is ', np.shape(y_train_raw))
print('X_train_raw size is ', X_train_raw.nbytes/1024/1024, 'Mb')
```

```
X_train_raw shape is  (3040, 128, 128, 3)
y_train_raw shape is  (3040, 128, 128)
X_train_raw size is  142.5 Mb
```



Large data size: use a data generator to grab and train the batch on the fly.



4.

Implementing & tweaking a U-net

Tackling input size & neural network hyperparameters

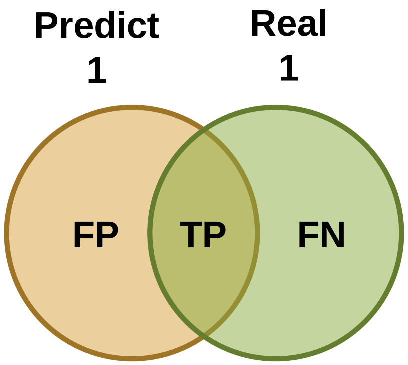
Loss function & metrics

Loss function:
Binary cross entropy = $-(y\log(p)+(1-y)\log(1-p))$

Evaluation metric:
dice coefficient

$$DSC = \frac{2|X\cap Y|}{|X|+|Y|} = \frac{2*y*round(p)}{y+round(p)}$$

$$= \frac{2TP}{2TP+FP+FN}.$$

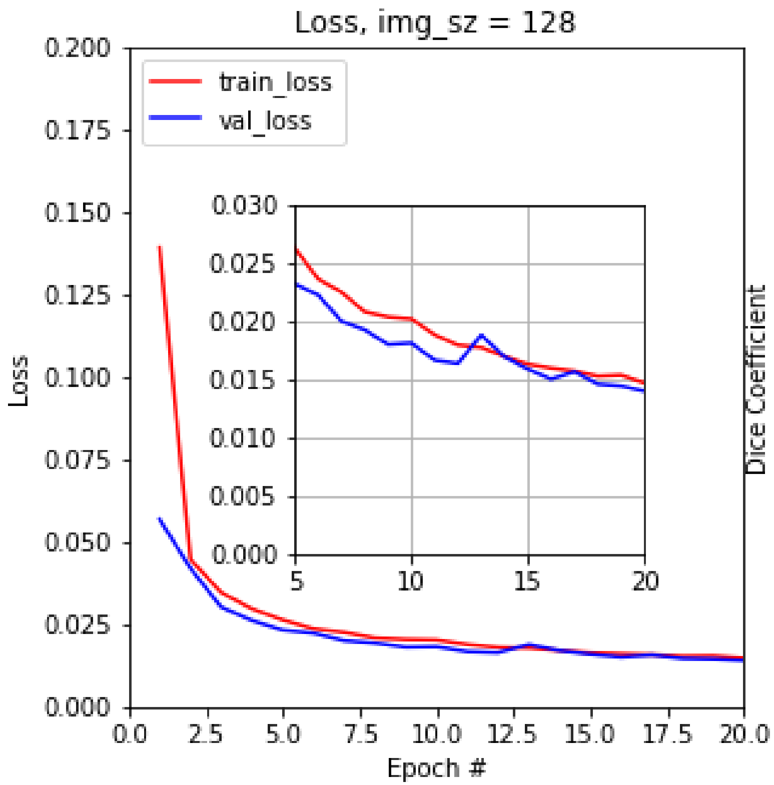


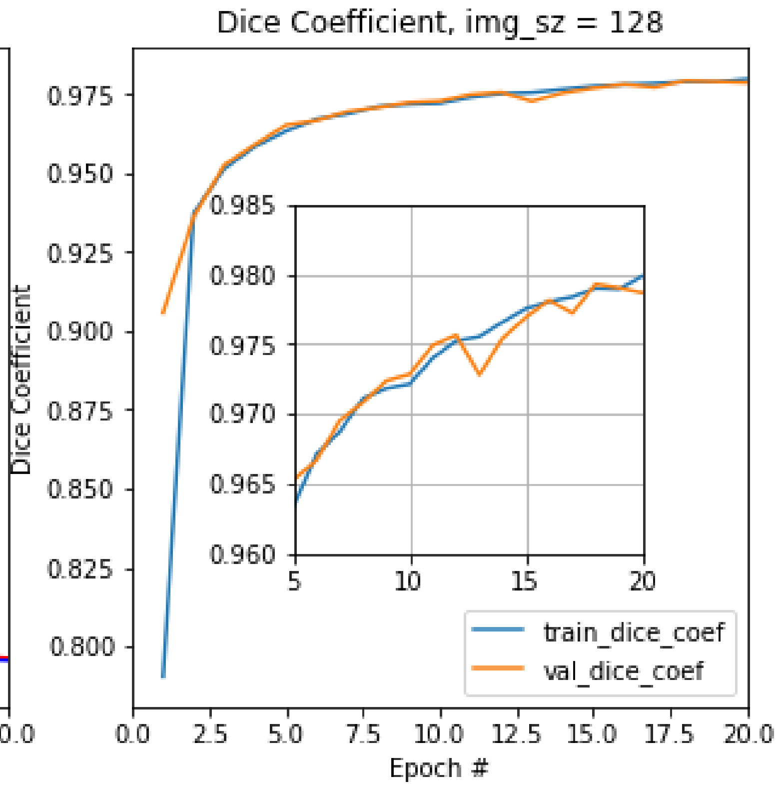
$$\text{IoU} = \frac{TP}{TP+FP+FN}$$

Compile and train the model

```
Total params: 120,965
Trainable params: 120,965
Non-trainable params: 0
```

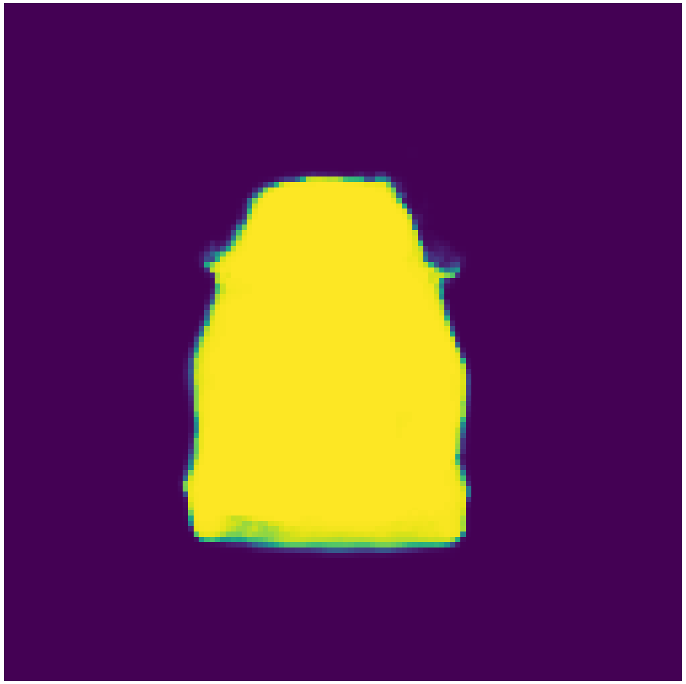
```
model.compile(optimizer=Adam(1e-3), loss='binary_crossentropy', metrics=[dice_coef])
history = model.fit(x = X_train, y = y_train,
                    validation_data=(X_val, y_val),
                    epochs = 30, batch_size = 16)
```





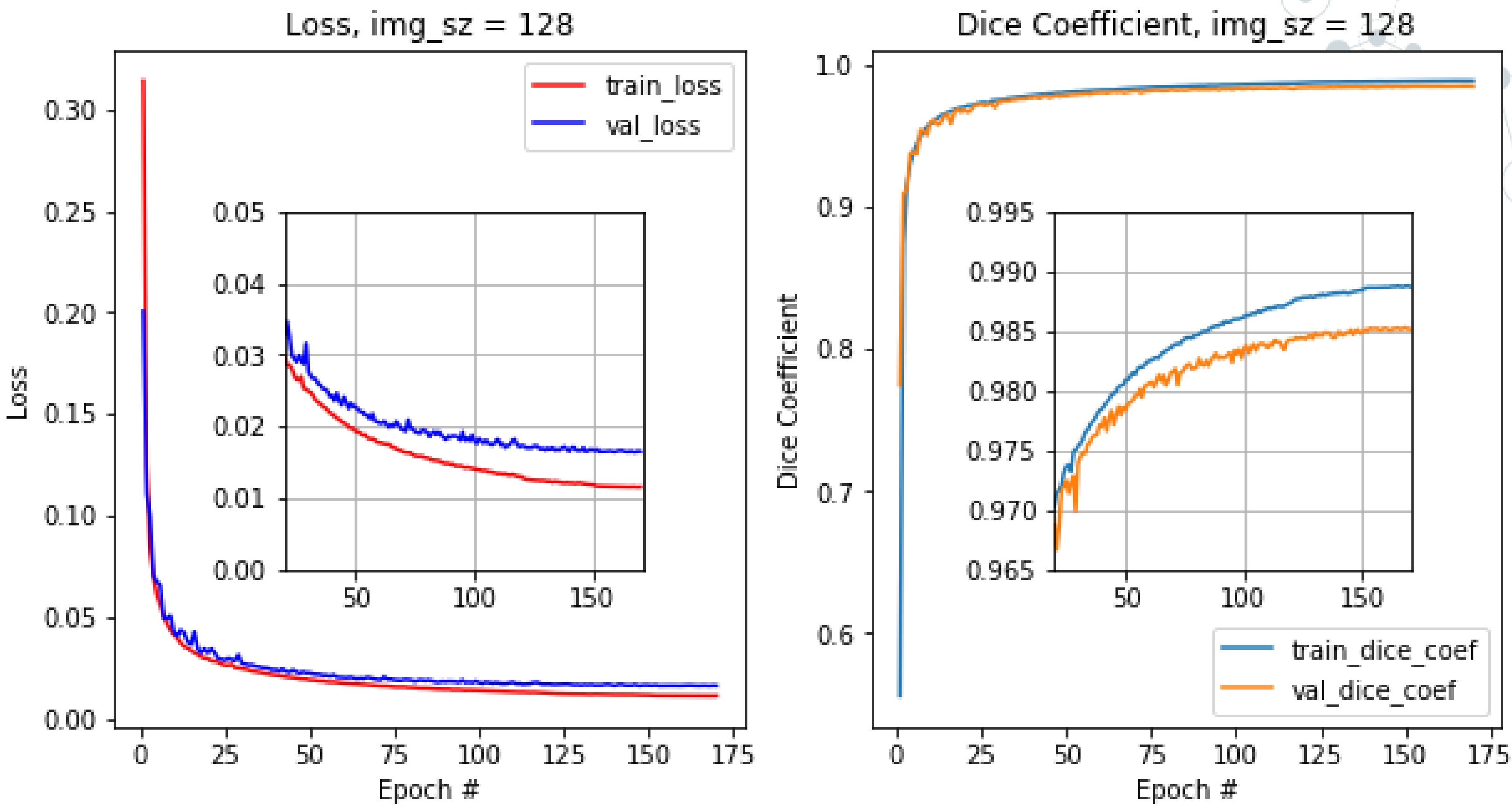

```
1024/1024 [==============================] - 54s 53ms/step
Test loss =  0.0167651942756
Test dice_coef =  0.976771984249
```

Make a prediction



Use a larger U-net

```
Total params: 485,957
Trainable params: 485,957
Non-trainable params: 0
```



Increase the image size

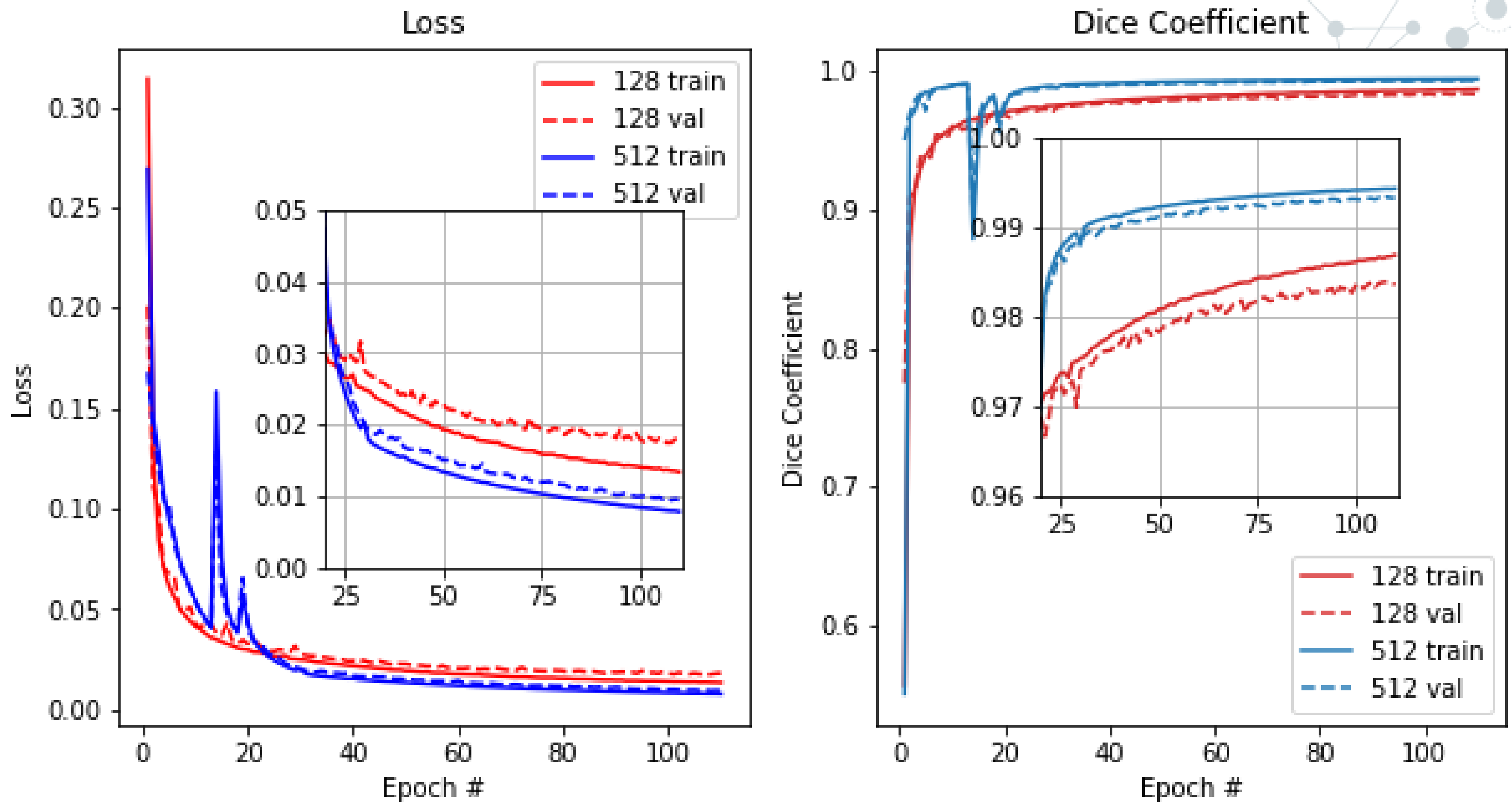


Image size = 512

```
score = model.evaluate_generator(test_generator)
print ("Test loss = ", score[0])
print ("Test dice_coef = ", score[1])
```

Test loss = 0.00861717724183
Test dice_coef = 0.993911107071

5.

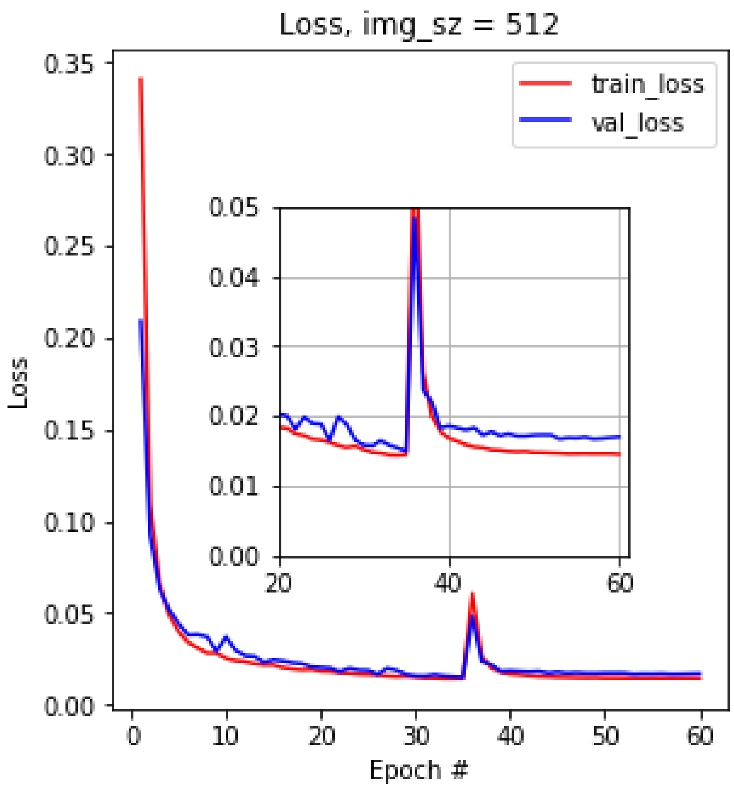
Data augmentation

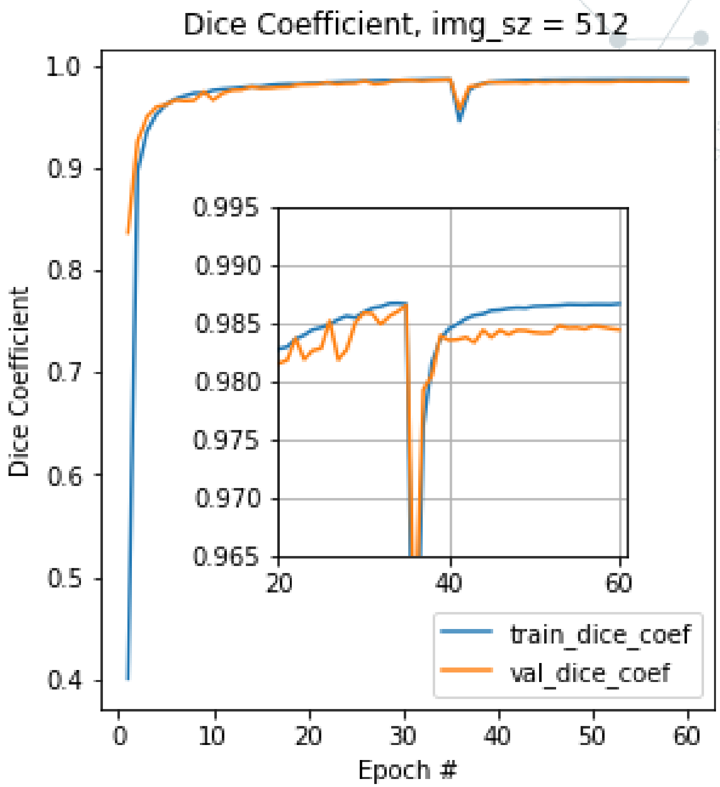
Generate more training data to train a larger U-net

Random rotation & horizontal flip

- Random rotation (<25 degree)
- 50% chance to be flipped horizontally.

Set mode='symmetric' to eliminate the black triangles.

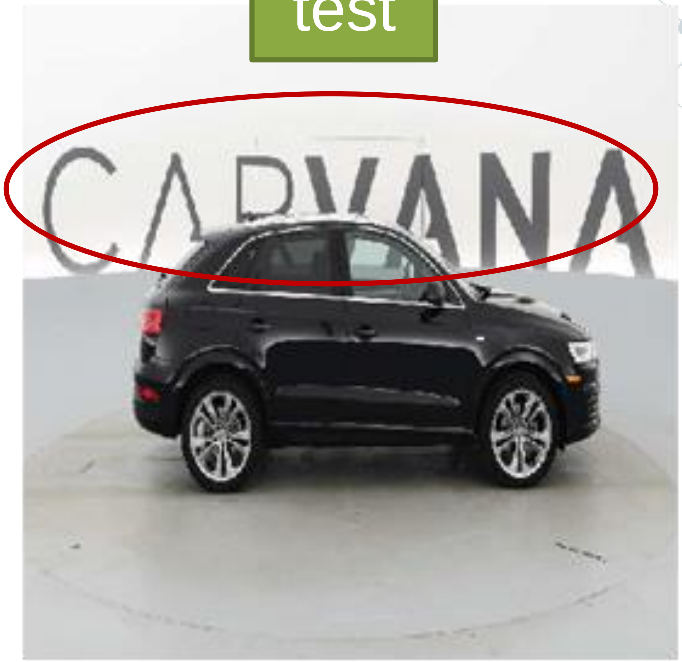




Performance drops after implementing augmentation, indicating that **model with data augmentation underfits the data.**

train

test



Training images w/o augmentation share the same studio background with that of the test images. It will be easier for nn to learn these simple features and make a better prediction.

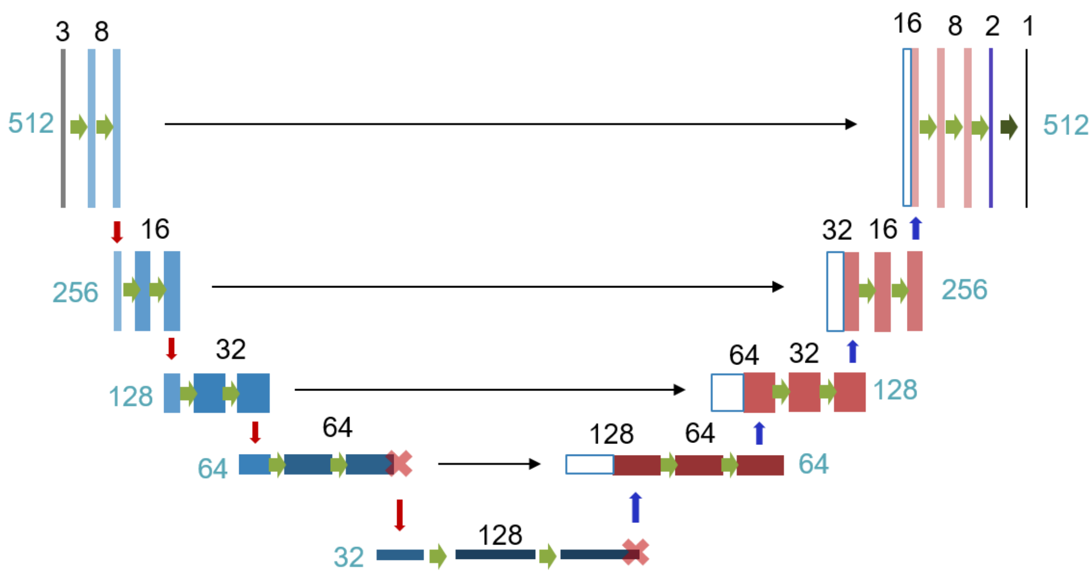
Possible solution: use **batch norm** to reduce the “covariance” of different layers in nn.

6.

Results analyses & visualization

Model evaluation, results analyses and feature map visualization

Model summary



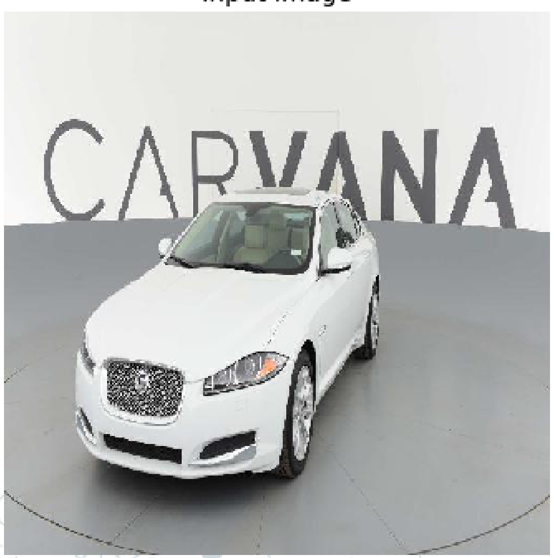
Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 512, 512, 3)	0	
conv2d_49 (Conv2D)	(None, 512, 512, 8)	224	input_3[0][0]
conv2d_50 (Conv2D)	(None, 512, 512, 8)	584	conv2d_49[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 256, 256, 8)	0	conv2d_50[0][0]
conv2d_51 (Conv2D)	(None, 256, 256, 16)	1168	max_pooling2d_9[0][0]
conv2d_52 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_51[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_52[0][0]
conv2d_53 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d_10[0][0]
conv2d_54 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_53[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_54[0][0]
conv2d_55 (Conv2D)	(None, 64, 64, 64)	18496	max_pooling2d_11[0][0]
conv2d_56 (Conv2D)	(None, 64, 64, 64)	36928	conv2d_55[0][0]
dropout_5 (Dropout)	(None, 64, 64, 64)	0	conv2d_56[0][0]
max_pooling2d_12 (MaxPooling2D)	(None, 32, 32, 64)	0	dropout_5[0][0]
conv2d_57 (Conv2D)	(None, 32, 32, 128)	73856	max_pooling2d_12[0][0]
conv2d_58 (Conv2D)	(None, 32, 32, 128)	147584	conv2d_57[0][0]
dropout_6 (Dropout)	(None, 32, 32, 128)	0	conv2d_58[0][0]
up_sampling2d_9 (UpSampling2D)	(None, 64, 64, 128)	0	dropout_6[0][0]
conv2d_59 (Conv2D)	(None, 64, 64, 64)	32832	up_sampling2d_9[0][0]
concatenate_9 (Concatenate)	(None, 64, 64, 128)	0	dropout_5[0][0] conv2d_59[0][0]
conv2d_60 (Conv2D)	(None, 64, 64, 64)	73792	concatenate_9[0][0]
conv2d_61 (Conv2D)	(None, 64, 64, 64)	36928	conv2d_60[0][0]
up_sampling2d_10 (UpSampling2D)	(None, 128, 128, 64)	0	conv2d_61[0][0]
conv2d_62 (Conv2D)	(None, 128, 128, 32)	8224	up_sampling2d_10[0][0]
concatenate_10 (Concatenate)	(None, 128, 128, 64)	0	conv2d_54[0][0] conv2d_62[0][0]
conv2d_63 (Conv2D)	(None, 128, 128, 32)	18464	concatenate_10[0][0]
conv2d_64 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_63[0][0]
up_sampling2d_11 (UpSampling2D)	(None, 256, 256, 32)	0	conv2d_64[0][0]
conv2d_65 (Conv2D)	(None, 256, 256, 16)	2064	up_sampling2d_11[0][0]
concatenate_11 (Concatenate)	(None, 256, 256, 32)	0	conv2d_52[0][0] conv2d_65[0][0]
conv2d_66 (Conv2D)	(None, 256, 256, 16)	4624	concatenate_11[0][0]
conv2d_67 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_66[0][0]
up_sampling2d_12 (UpSampling2D)	(None, 512, 512, 16)	0	conv2d_67[0][0]
conv2d_68 (Conv2D)	(None, 512, 512, 8)	520	up_sampling2d_12[0][0]
concatenate_12 (Concatenate)	(None, 512, 512, 16)	0	conv2d_50[0][0] conv2d_68[0][0]
conv2d_69 (Conv2D)	(None, 512, 512, 8)	1160	concatenate_12[0][0]
conv2d_70 (Conv2D)	(None, 512, 512, 8)	584	conv2d_69[0][0]
conv2d_71 (Conv2D)	(None, 512, 512, 2)	146	conv2d_70[0][0]
conv2d_72 (Conv2D)	(None, 512, 512, 1)	3	conv2d_71[0][0]

Total params: 485,957
Trainable params: 485,957
Non-trainable params: 0

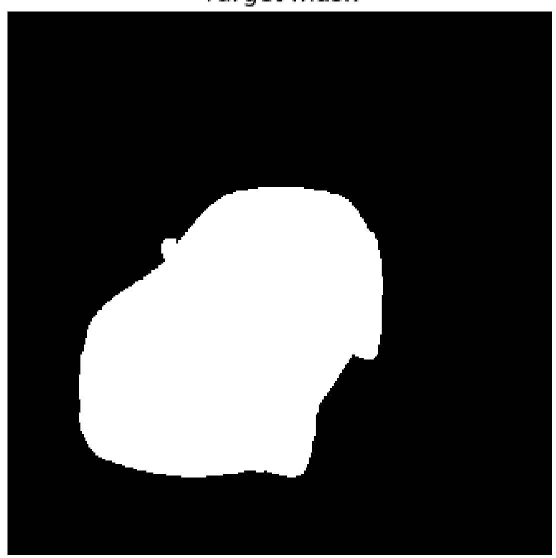
Model evaluation: Good predictions

	Im_names_test	scores
113	2267f4aa0d2c_02.jpg	0.997615
114	2267f4aa0d2c_03.jpg	0.997280
502	9ab2a45de8c7_07.jpg	0.997027
127	2267f4aa0d2c_16.jpg	0.996976
126	2267f4aa0d2c_15.jpg	0.996976

Input image



Target mask



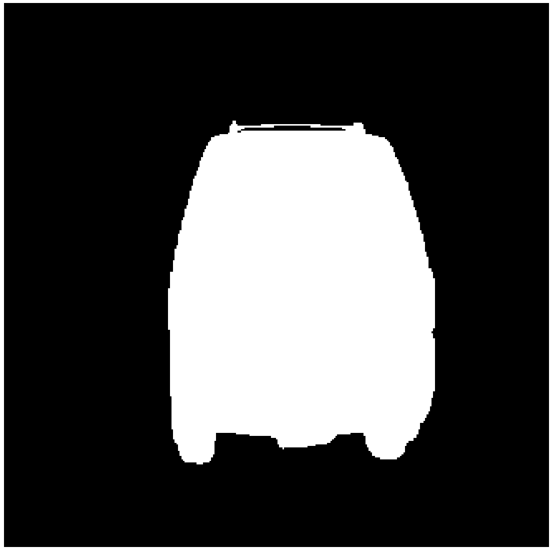
Predicted mask

Bad predictions

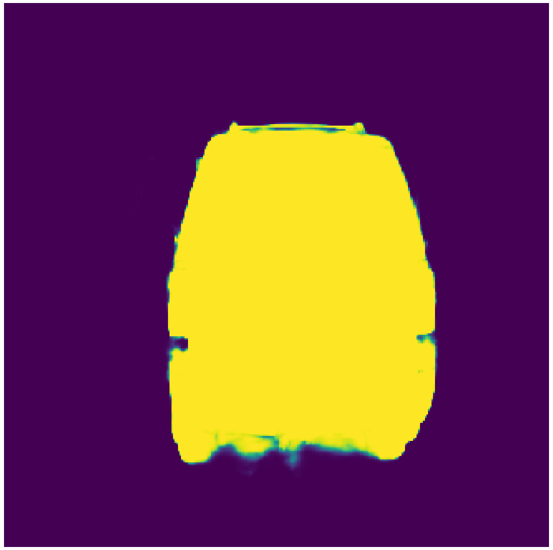
	Im_names_test	scores
104	1390696b70b6_09.jpg	0.982636
256	ae296a20fdd9_01.jpg	0.983955
103	1390696b70b6_08.jpg	0.984998
471	61060ada97c9_08.jpg	0.985197
920	28109f18d9d4_09.jpg	0.985368

Input image

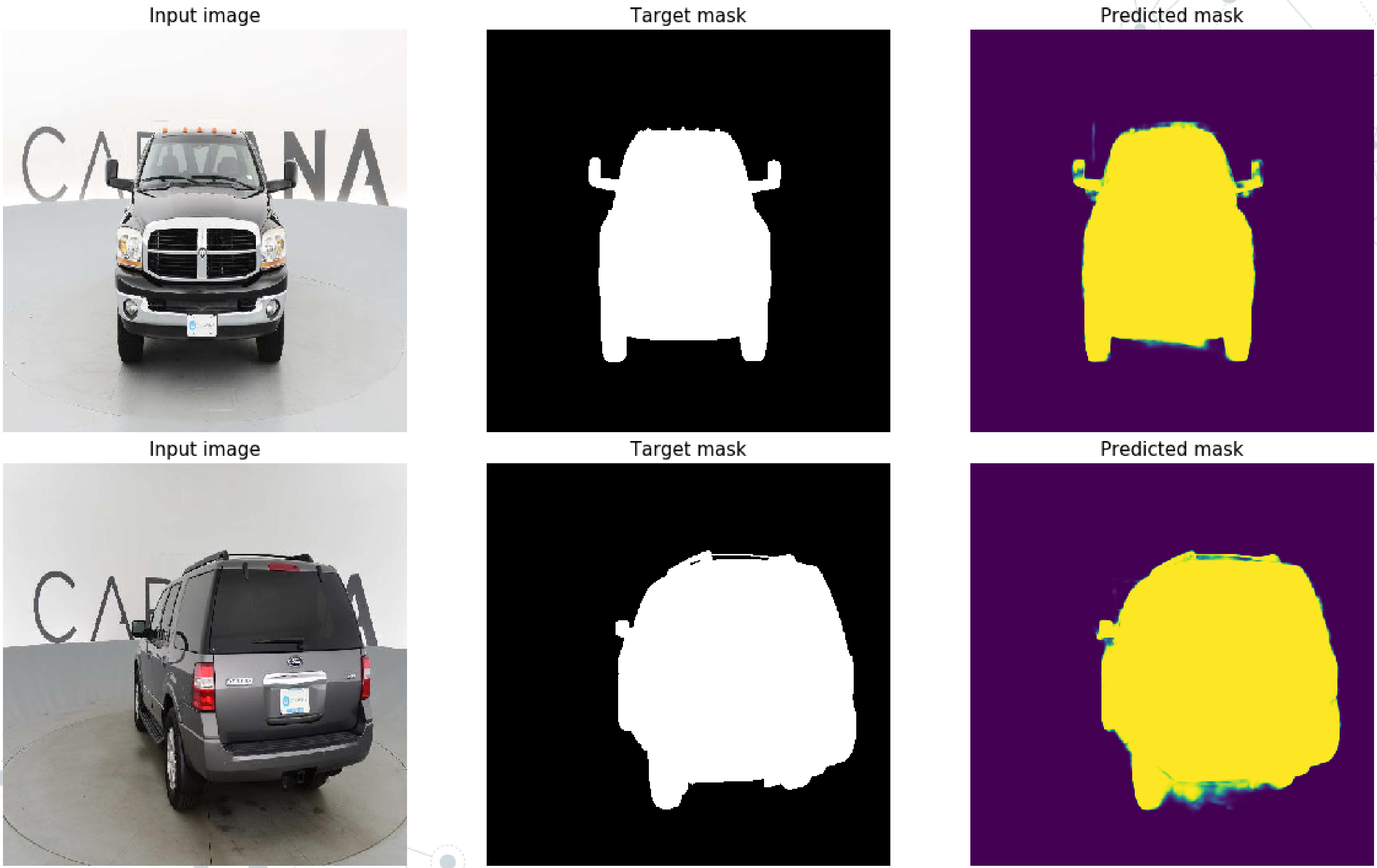
Target mask



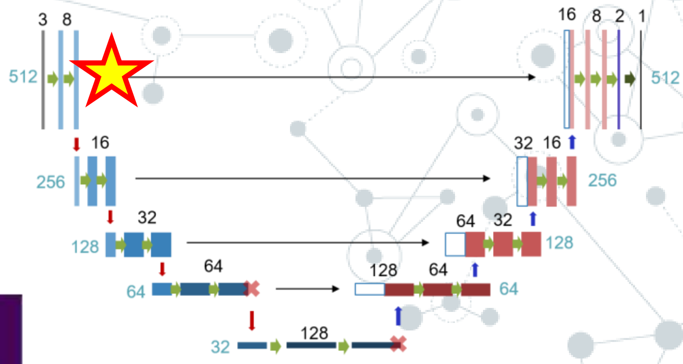
Predicted mask

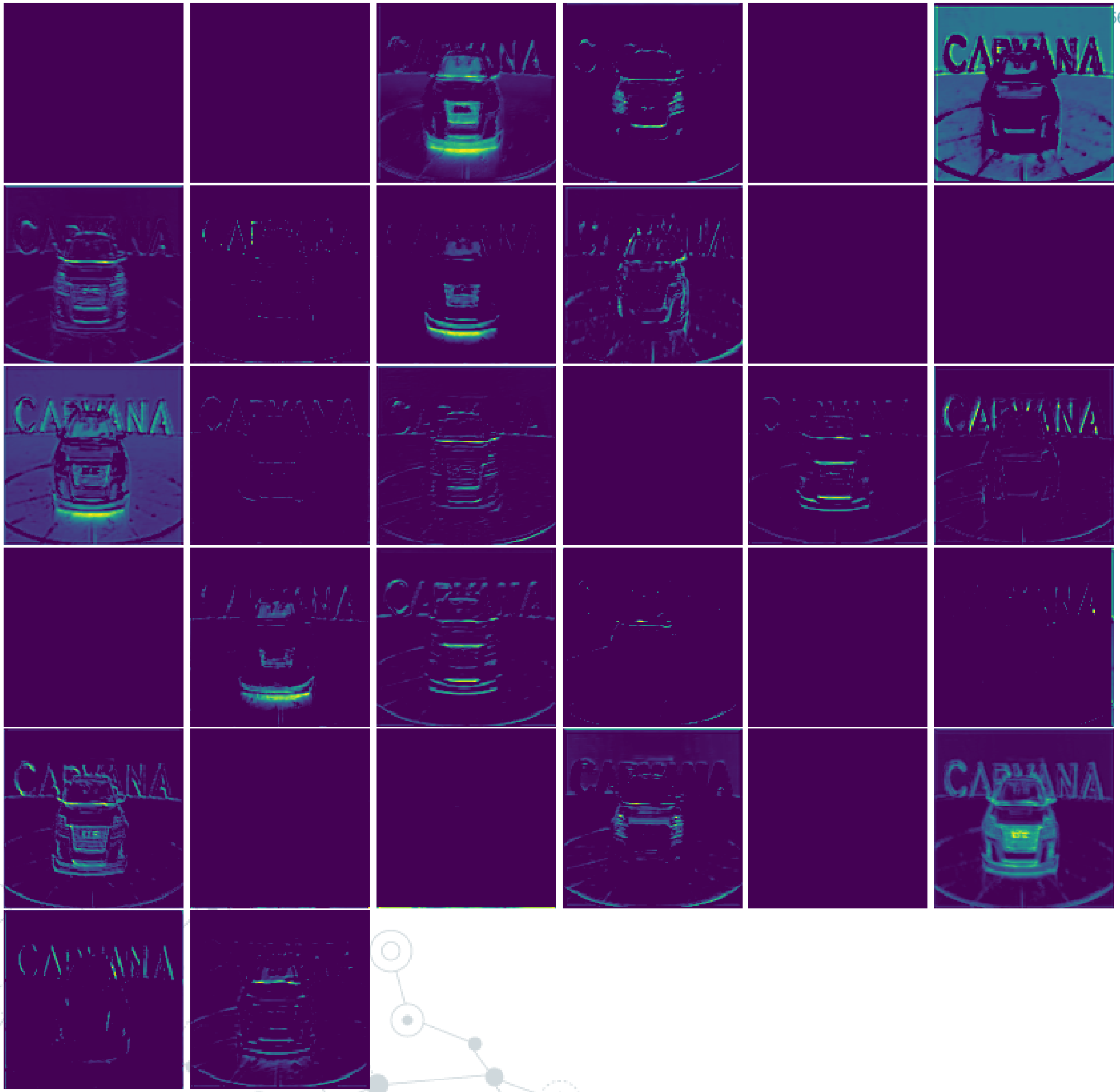


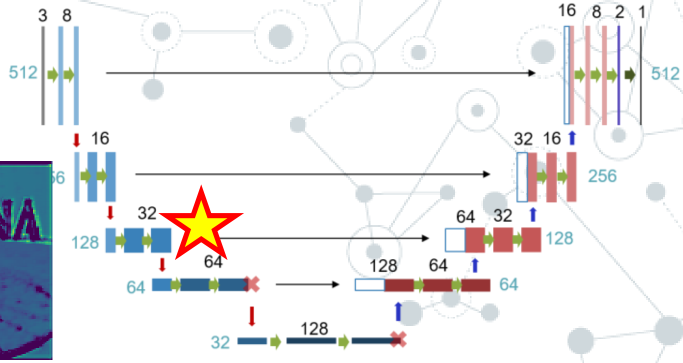
	Im_names_test	scores
256	ae296a20fdd9_01.jpg	0.983955
103	1390696b70b6_08.jpg	0.984998



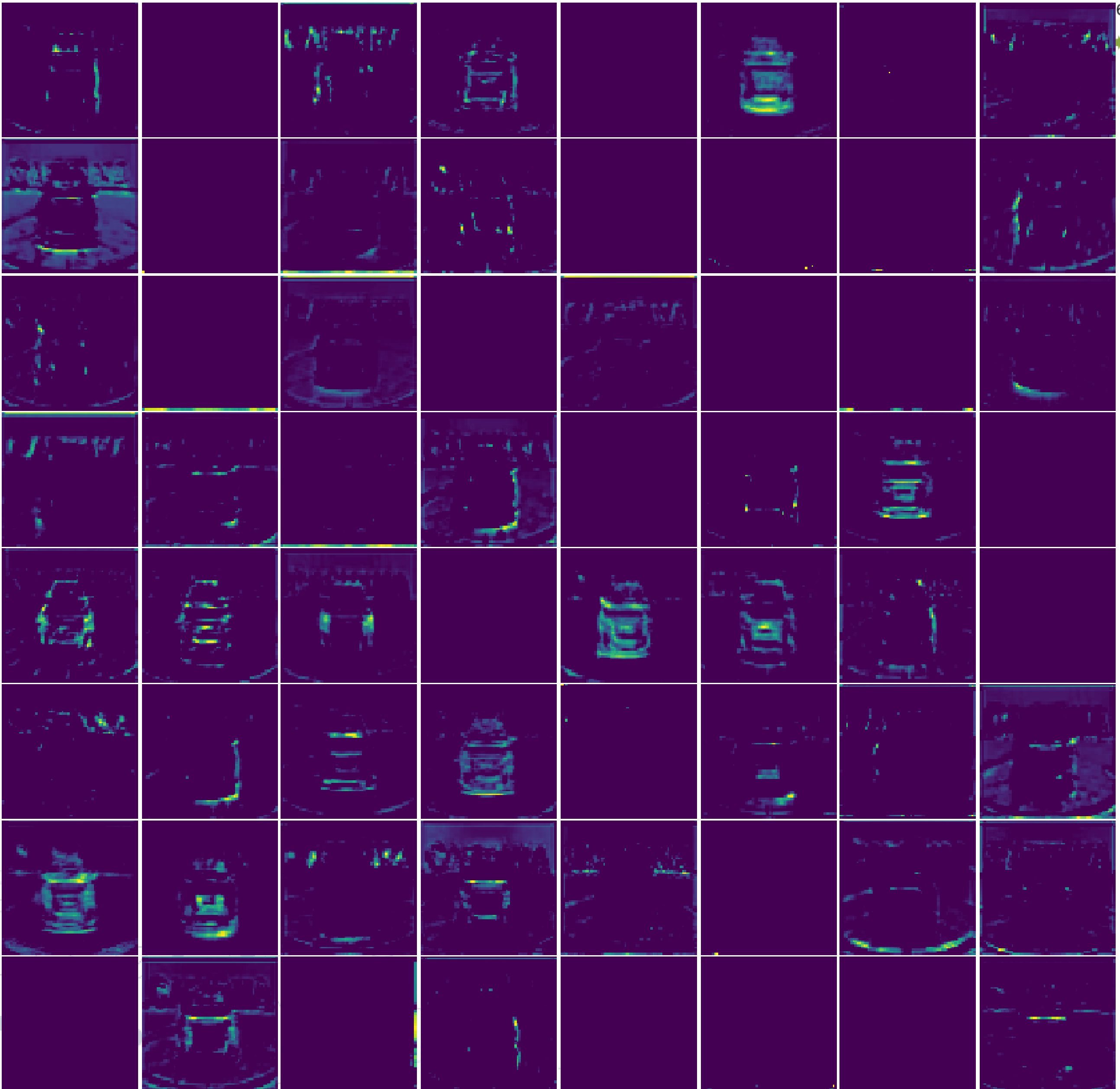
Visualization of Feature map

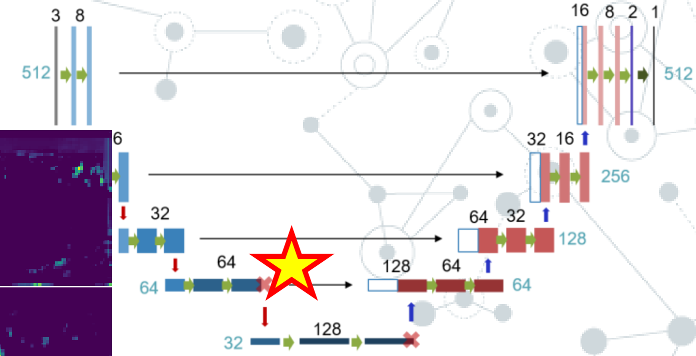




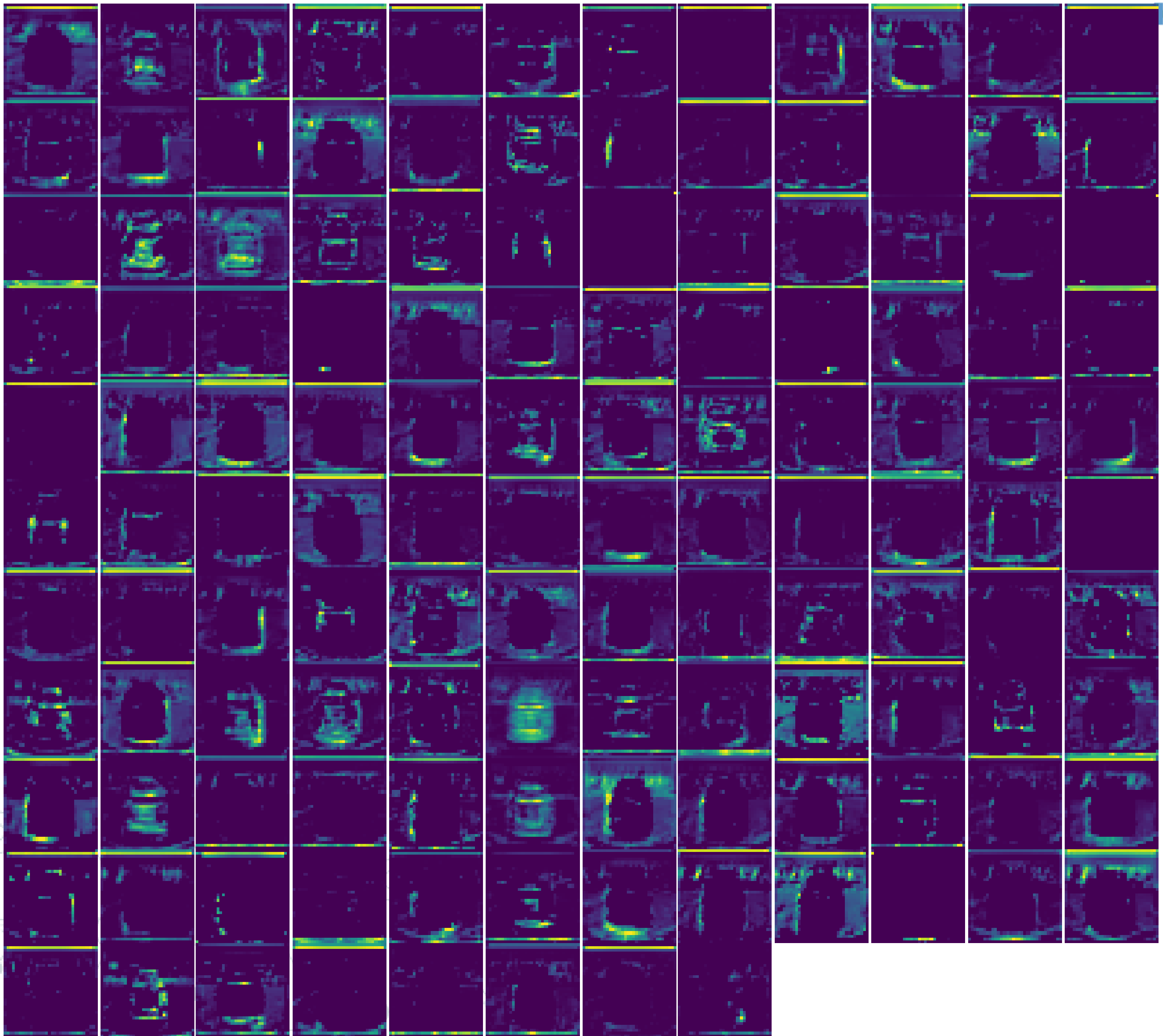


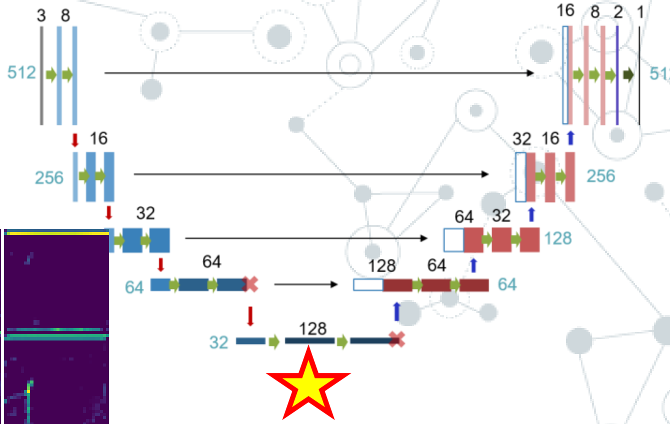
3
8
16
8
2
1
512
512
16
32
16
256
32
64
32
128
128
64
128
64
64
64
32
128



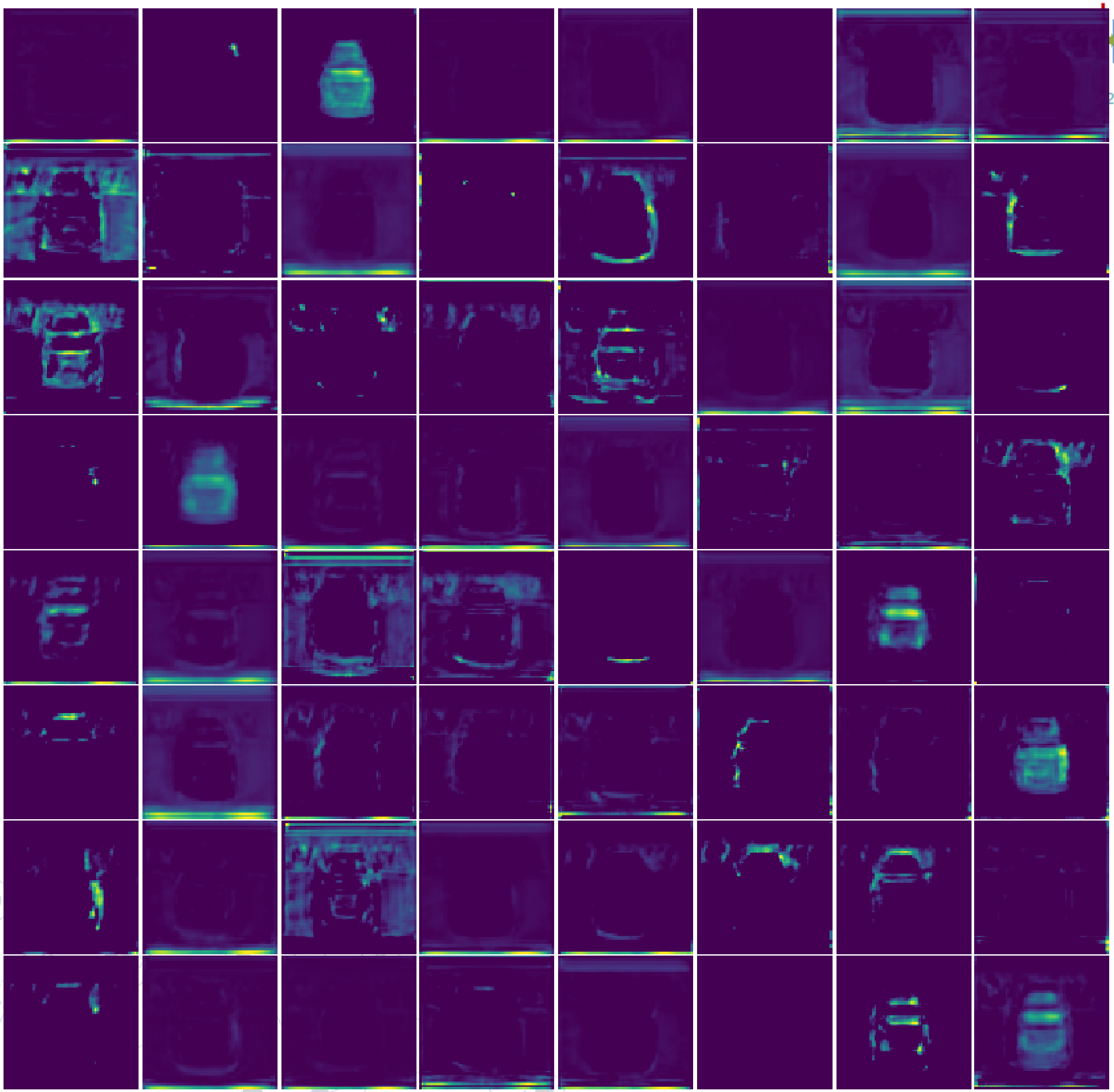


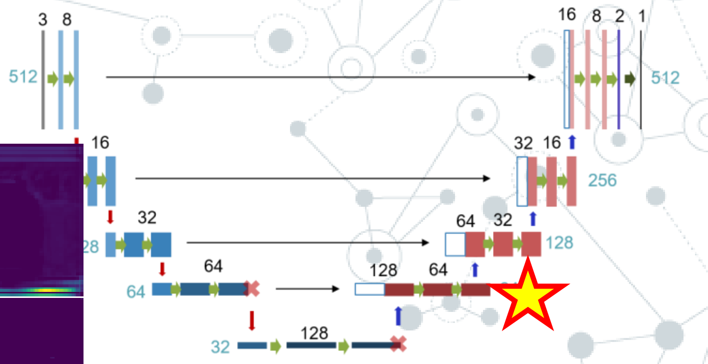
3
8
512
16
8
2
1
512
6
32
16
256
32
64
32
128
64
64
128
64
64
32
128

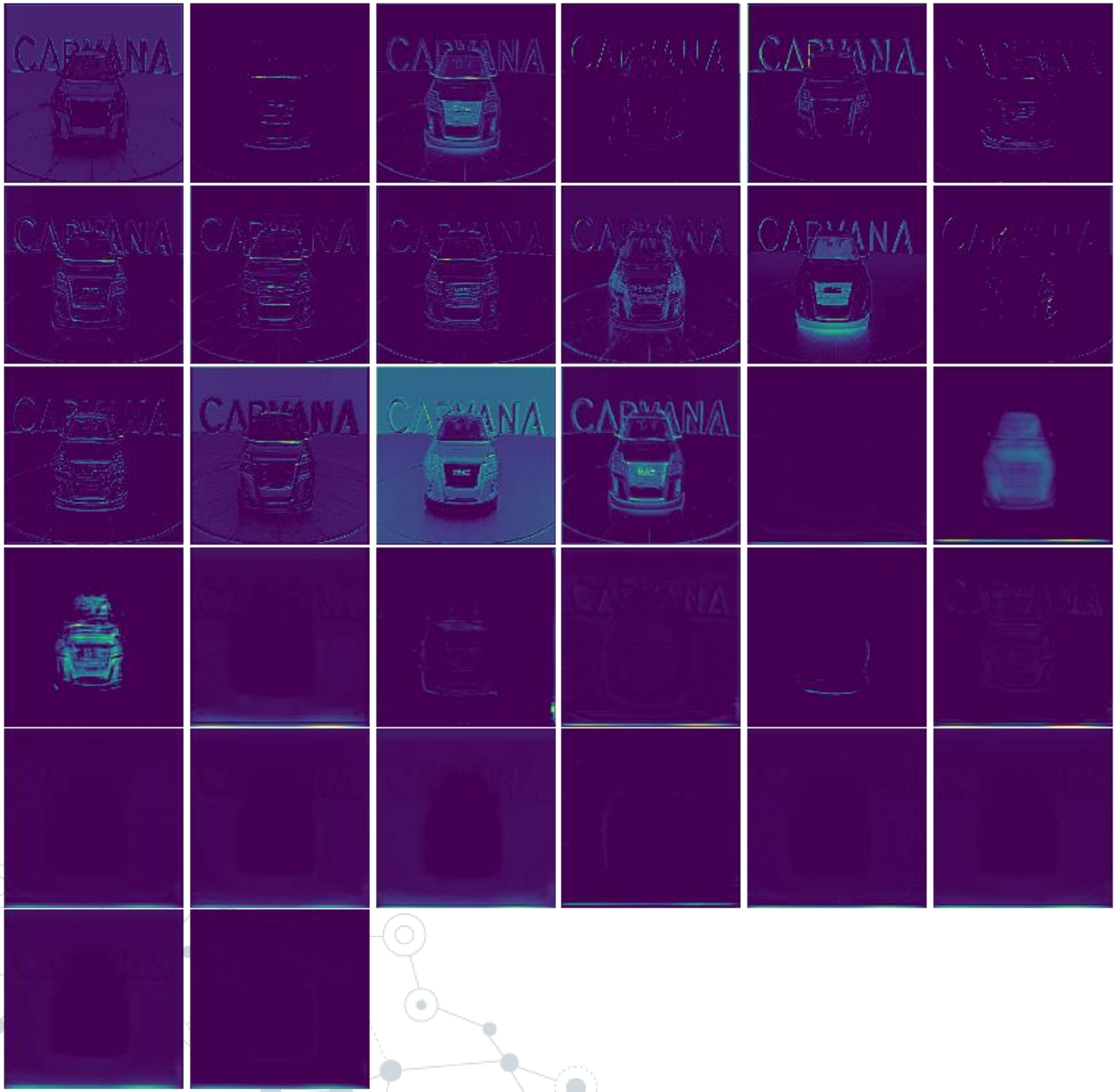


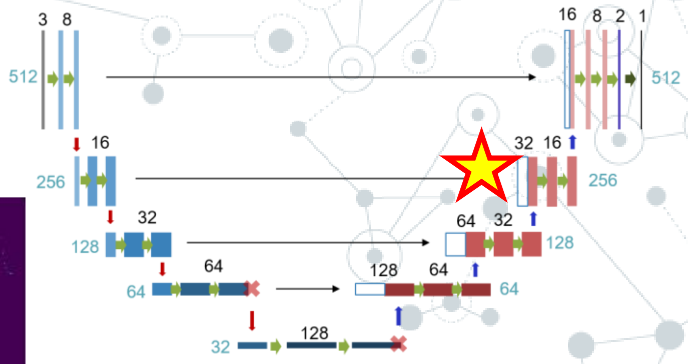


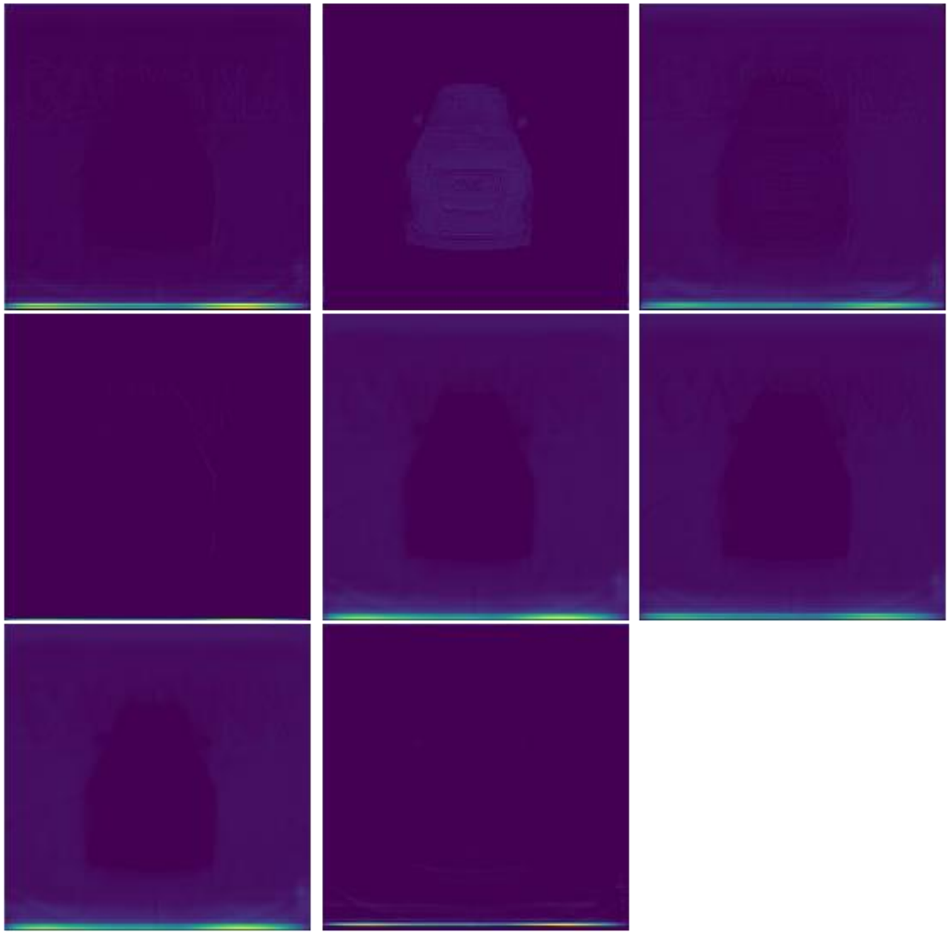
3
8
512
512
16
8
2
1
16
256
32
16
256
32
64
32
128
64
64
128
64
64
32
128

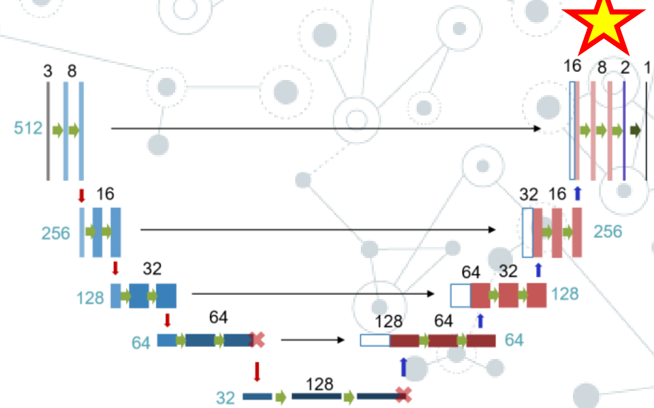



3
8
512
16
8
2
1
512
16
32
16
256
32
64
32
128
64
128
64
32
128




3
8
512
16
256
32
128
64
64
32
128
128
64
64
64
32
128
32
16
256
16
8
2
1
512





And finally…a prediction!

Here is what I learnt...

U-net

A powerful CNN architecture for semantic segmentation tasks.

Data Generator

Train NN on the fly. Practiced OOP.

Keras

Awesome API to implement CNN in a few lines of codes.

Data Augmentation

Generate enormous data to train a large NN.

Visualization

Plot results for a better analyse and evaluation.

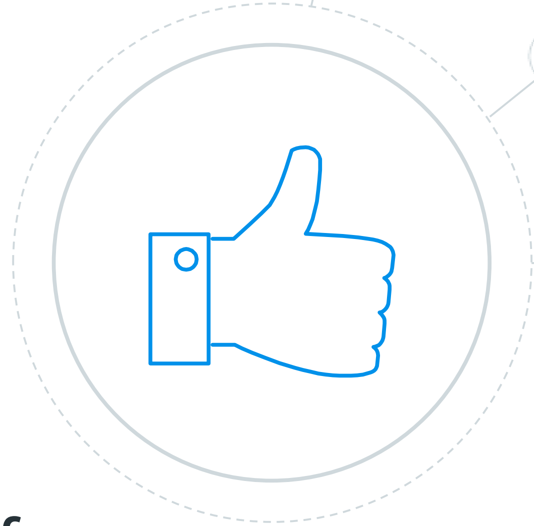
Feature map

"Look into" the deep NN and see what it is learning.

Thanks!

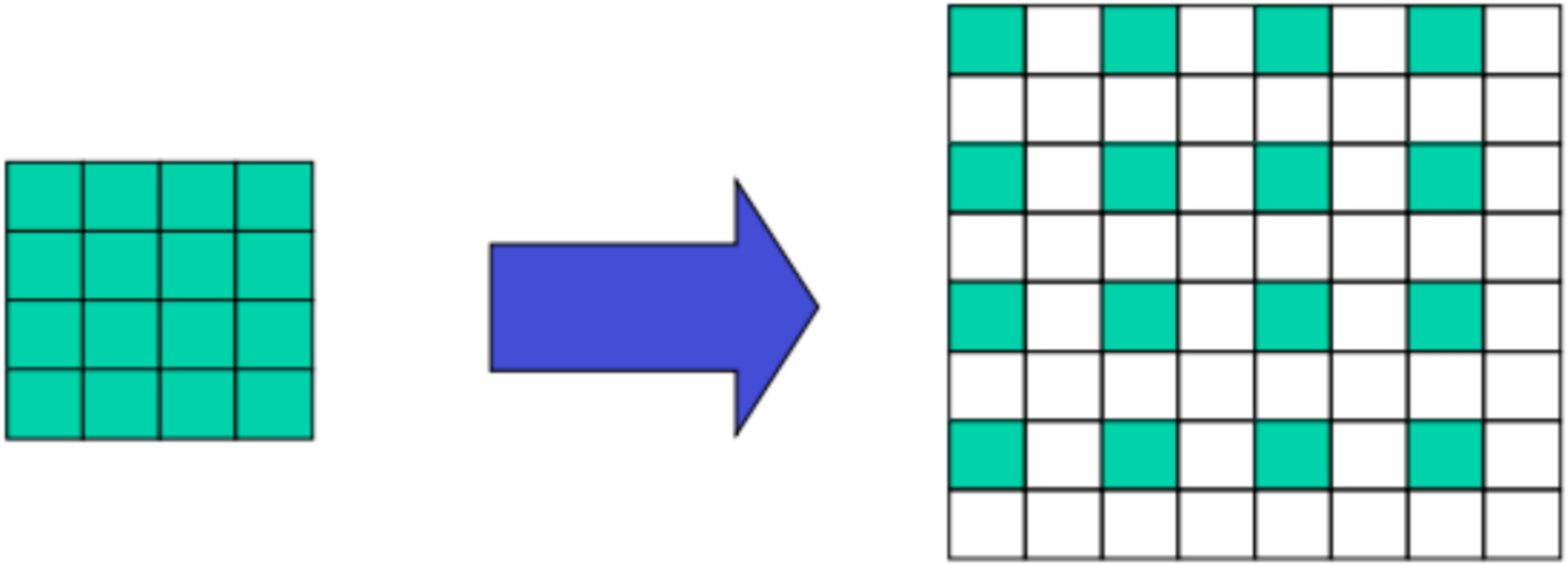
Any questions?

Appendix



Some useful stuff..

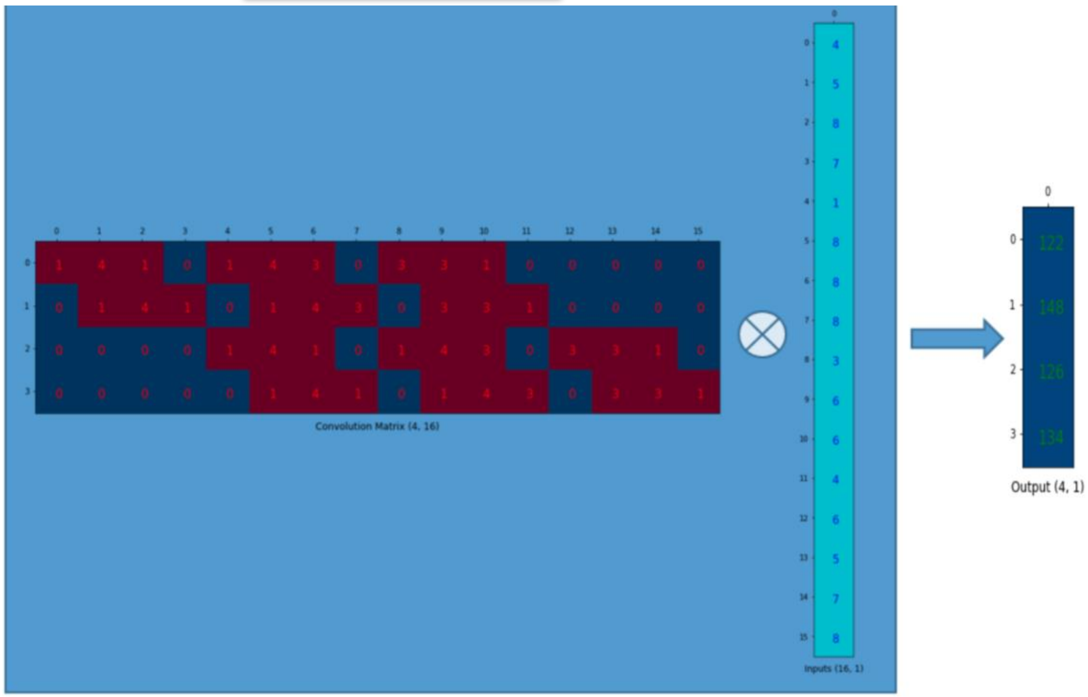
Upsampling: interpolation



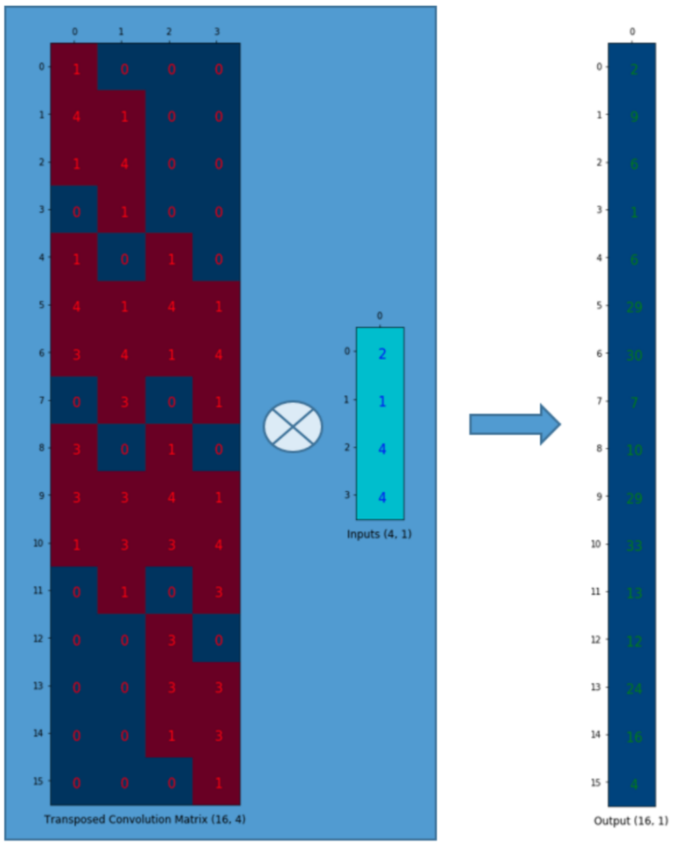
https://www.cs.toronto.edu/~guerzhoy/320/lec/upsampling.pdf

Transposed convolution

Conv



Trans Conv



Convolution By Matrix Multiplication

Source: https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

Transposed convolution