INTERNATIONAL HELLENIC UNIVERSITY

Indoor and outdoor view recognition using deep convolutional neural networks

Author: Christos CHEITAS
Supervisor: Prof. Konstantinos DIAMANTARAS

Department of Information and Electronic Engineering
MSc in Web Intelligence
October 2020

Table of contents

1. Introduction

2. Convolutional neural networks components

3. Methodology

4. Keras to Tensorflow.js

5. Class Activation Mapping (Grad-CAM)

6. Web application

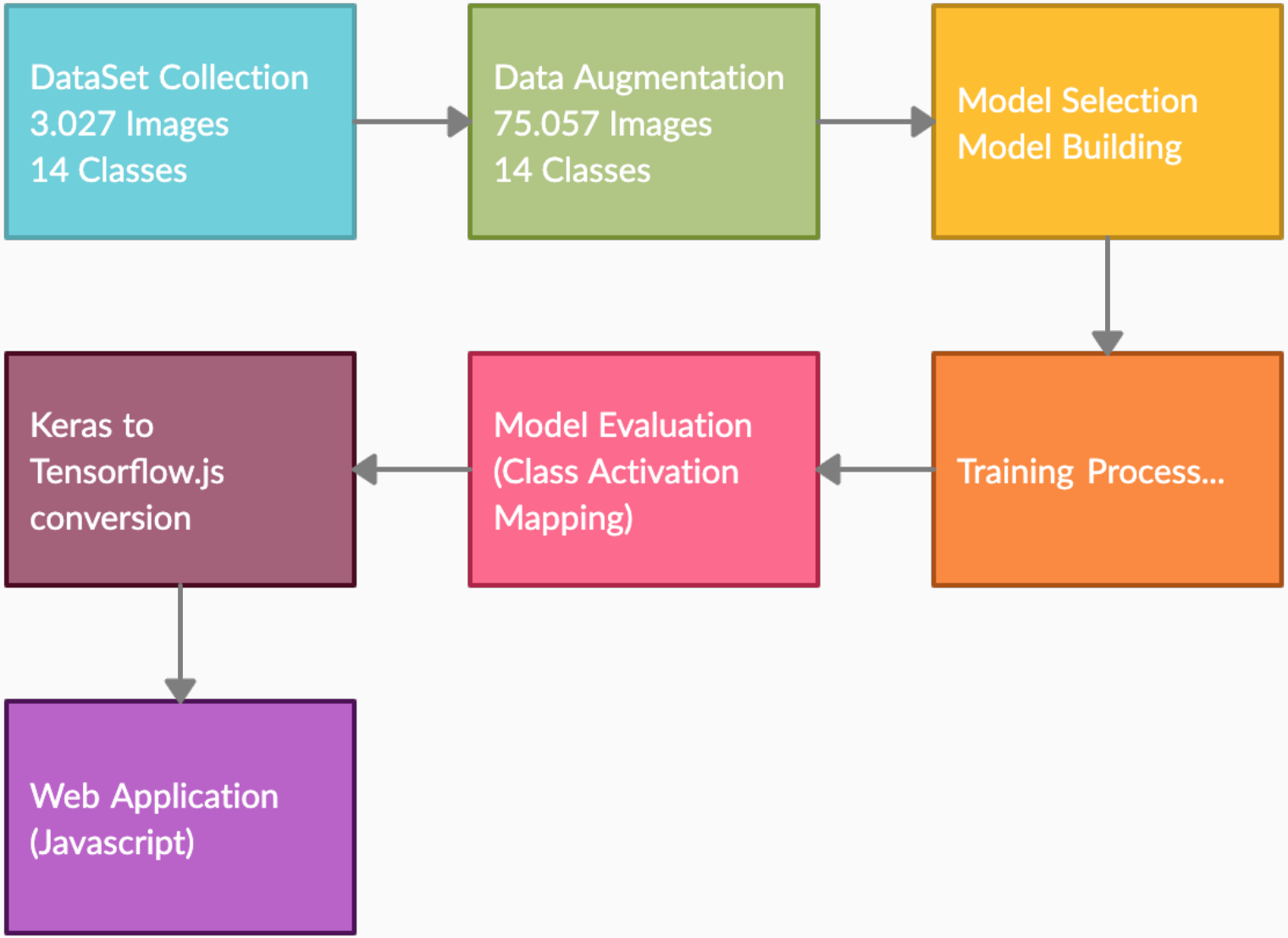
Introduction

Purpose

The purpose of this work is to present the way to create an end-to-end machine learning solution starting with model creation and ending with a live web application.

We present an approach of classification of these images based on light weight Convolutional Neural Networks

End-to-end ML Solution



Convolutional neural networks components

Convolutional neural networks

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers.

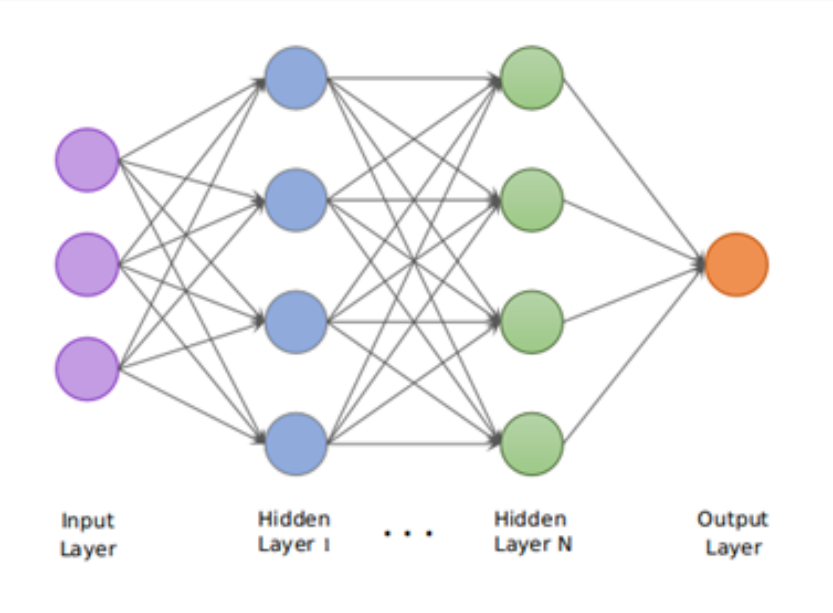


Figure 1: Convolutional Neural Network | An abstract view

The Neuron

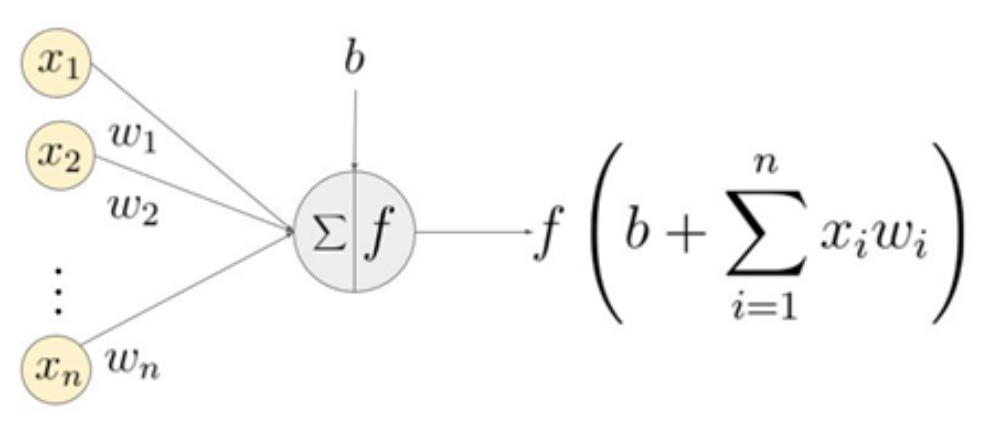


Figure 2: The Neuron

CNN architectures

There are several **CNN architectures**, such as:

- AlexNet [5]
- GoogLeNet [9]
- VGG Net [7]
- ResNet [3]
- NASNet [14]
- MobileNet [4]
- EfficientNet [10] many more.

CNN components

CNNs are composed of various “components”.

A component equals at least one layer in the CNN.

Convolutional neural network layer types mainly include three basic types

- **Convolutional layer**,
- **pooling layer**
- and **fully-connected layer** [2].

In addition to the base layers can also be used the **Dropout** and **Flatten layers**.

Convolutional Neural Network

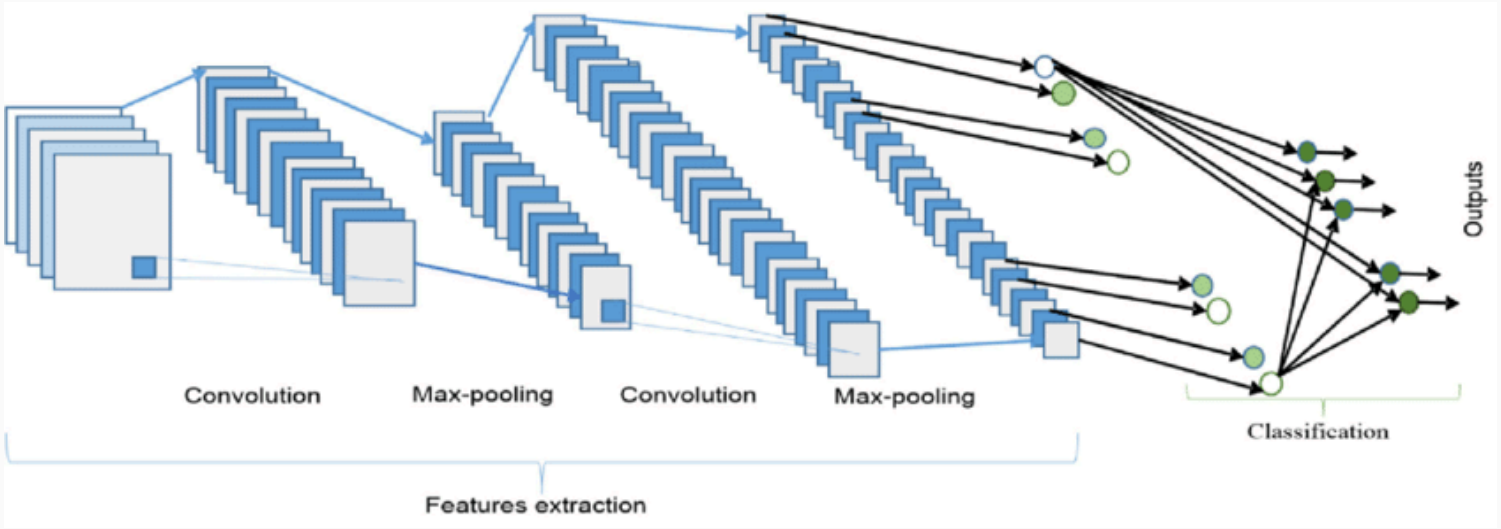


Figure 3: Convolutional Neural Network

Convolutional Layer

A convolution is the simple application of a filter to an input that results in an activation.

Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

Convolution

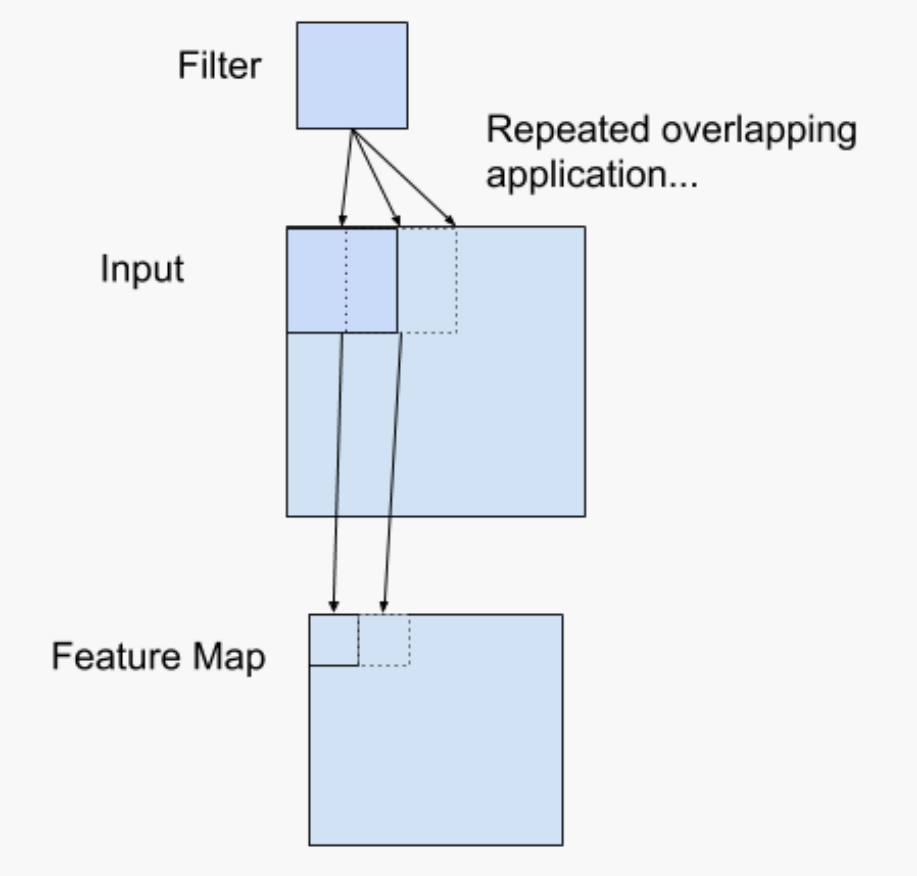
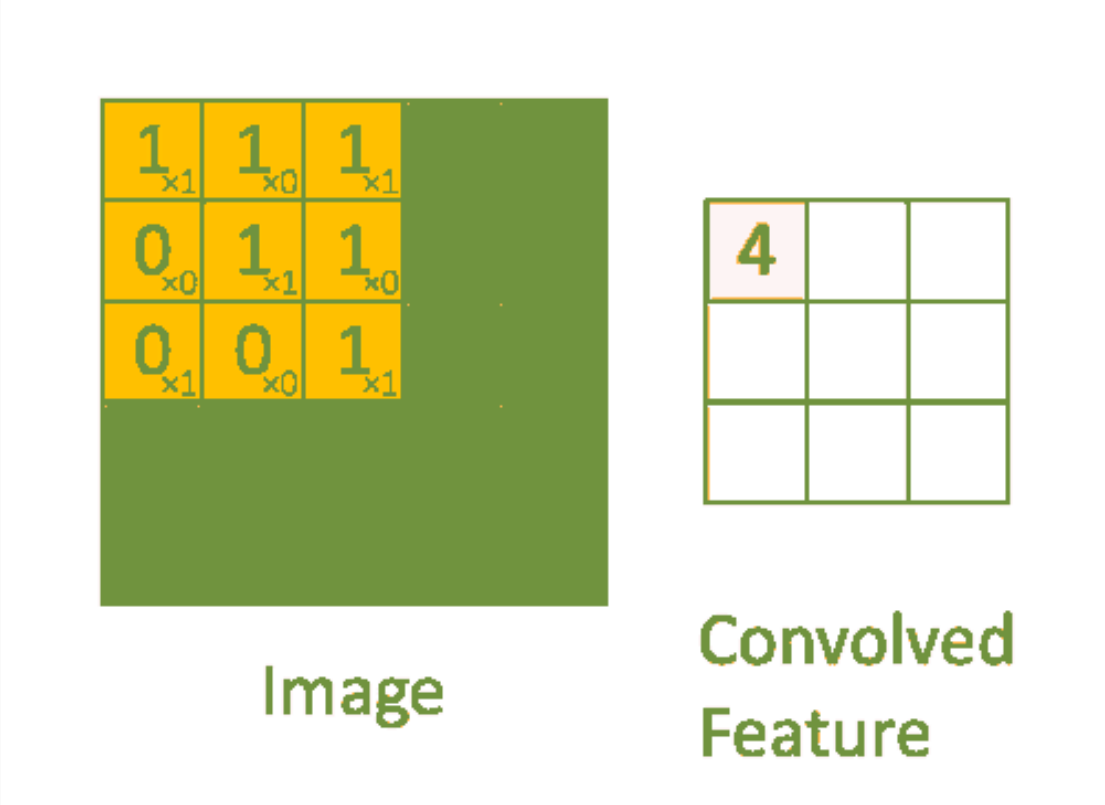
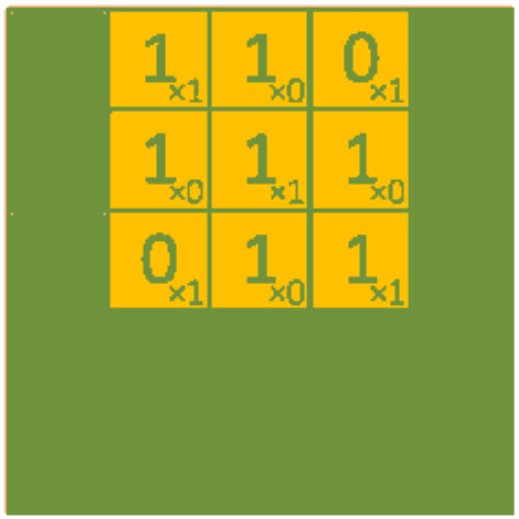


Figure 4: Convolution

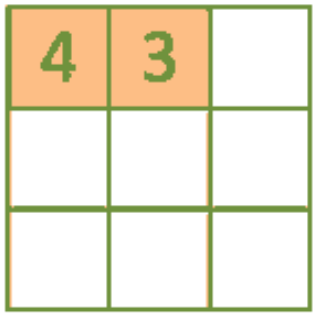
Image(5x5x1) Filter(3x3) 1



Image(5x5x1) Filter(3x3) 2

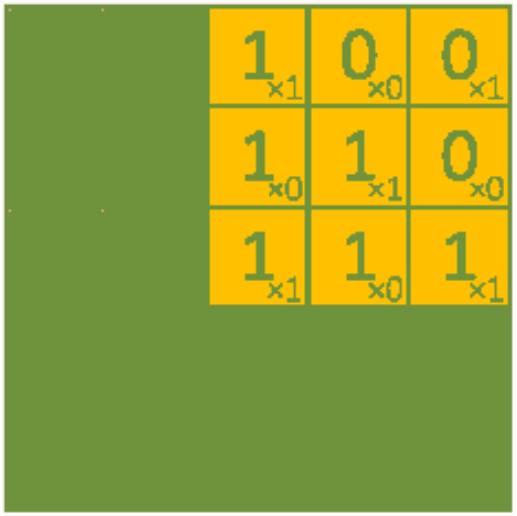


Image



Convolved Feature

Image(5x5x1) Filter(3x3) 3

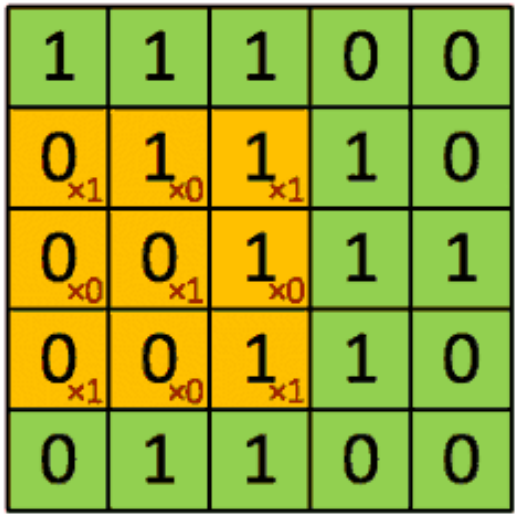


Image

4	3	4

Convolved Feature

Image(5x5x1) Filter(3x3) 4

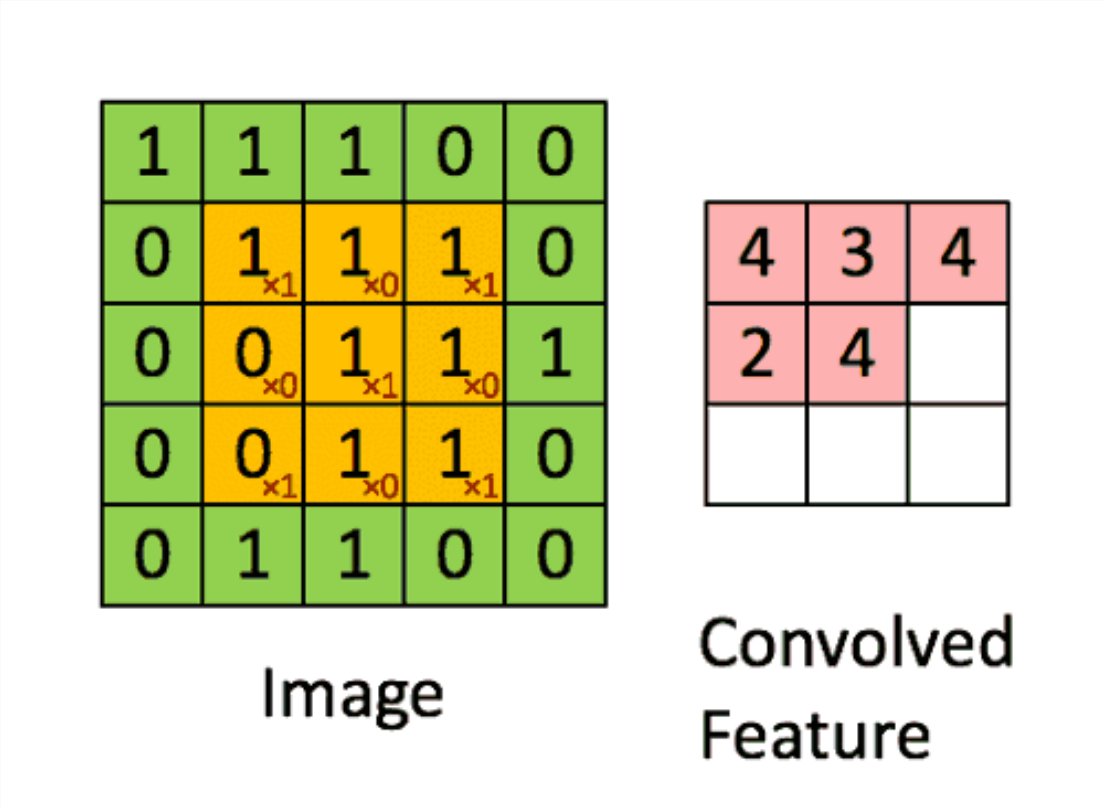


Image

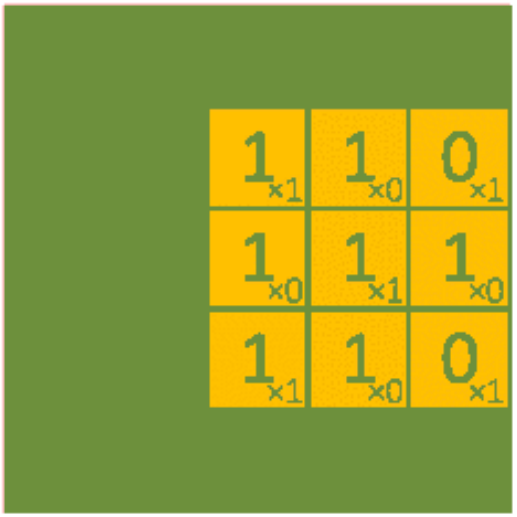
4	3	4
2		

Convolved Feature

Image(5x5x1) Filter(3x3) 5



Image(5x5x1) Filter(3x3) 6

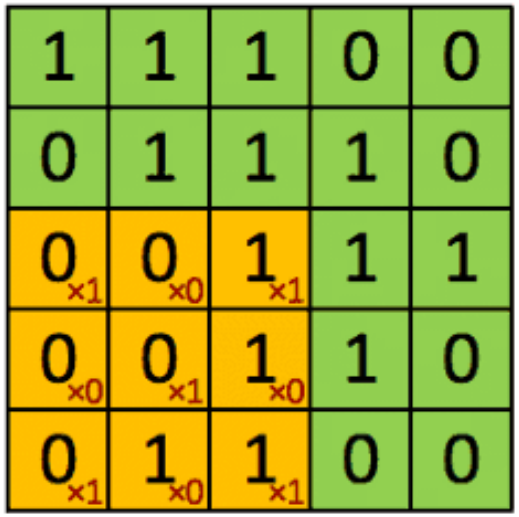


Image

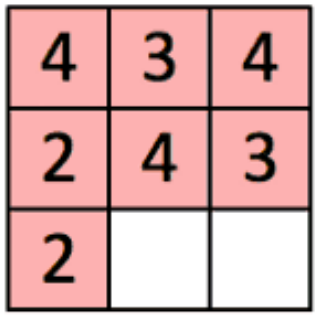
4	3	4
2	4	3

Convolved Feature

Image(5x5x1) Filter(3x3) 7



Image

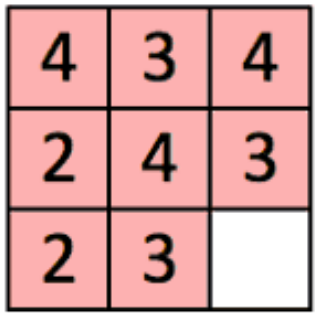


Convolved Feature

Image(5x5x1) Filter(3x3) 8



Image



Convolved Feature

Image(5x5x1) Filter(3x3) 9



Image

4	3	4
2	4	3
2	3	4

Convolved Feature

RGB Image - Depth 3

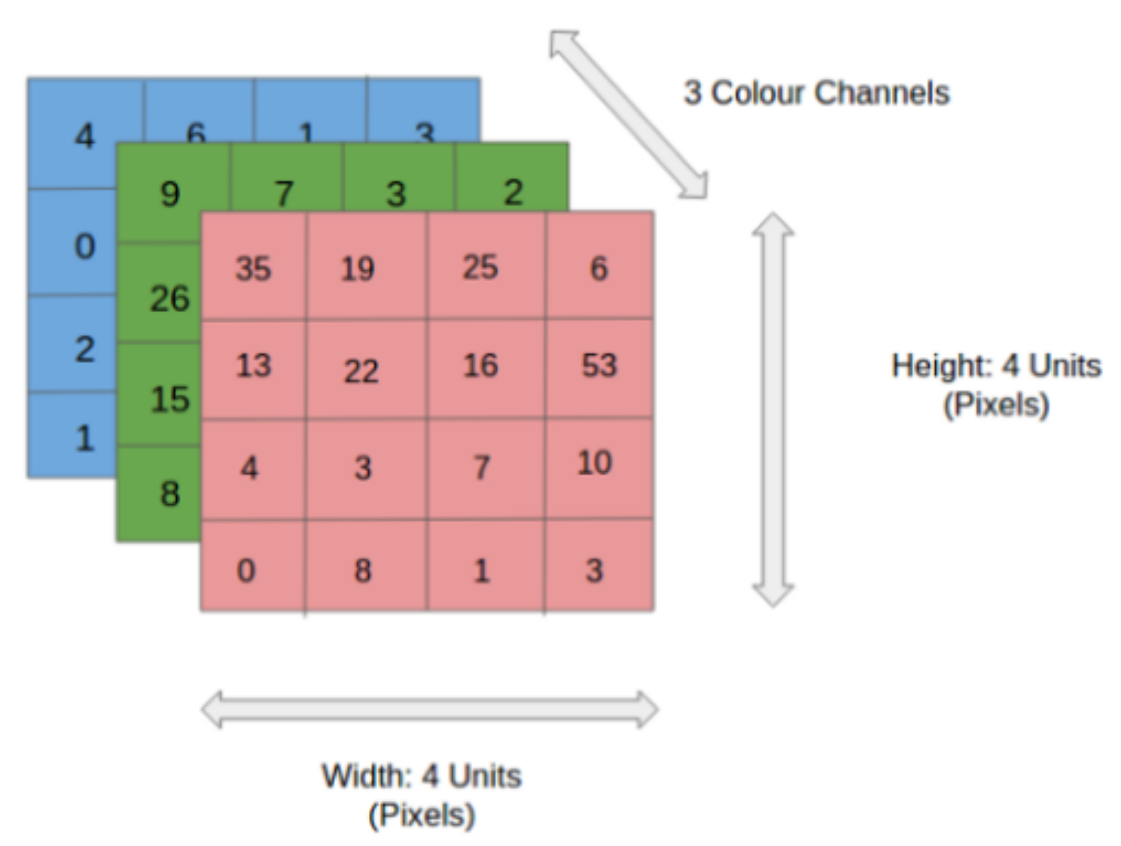
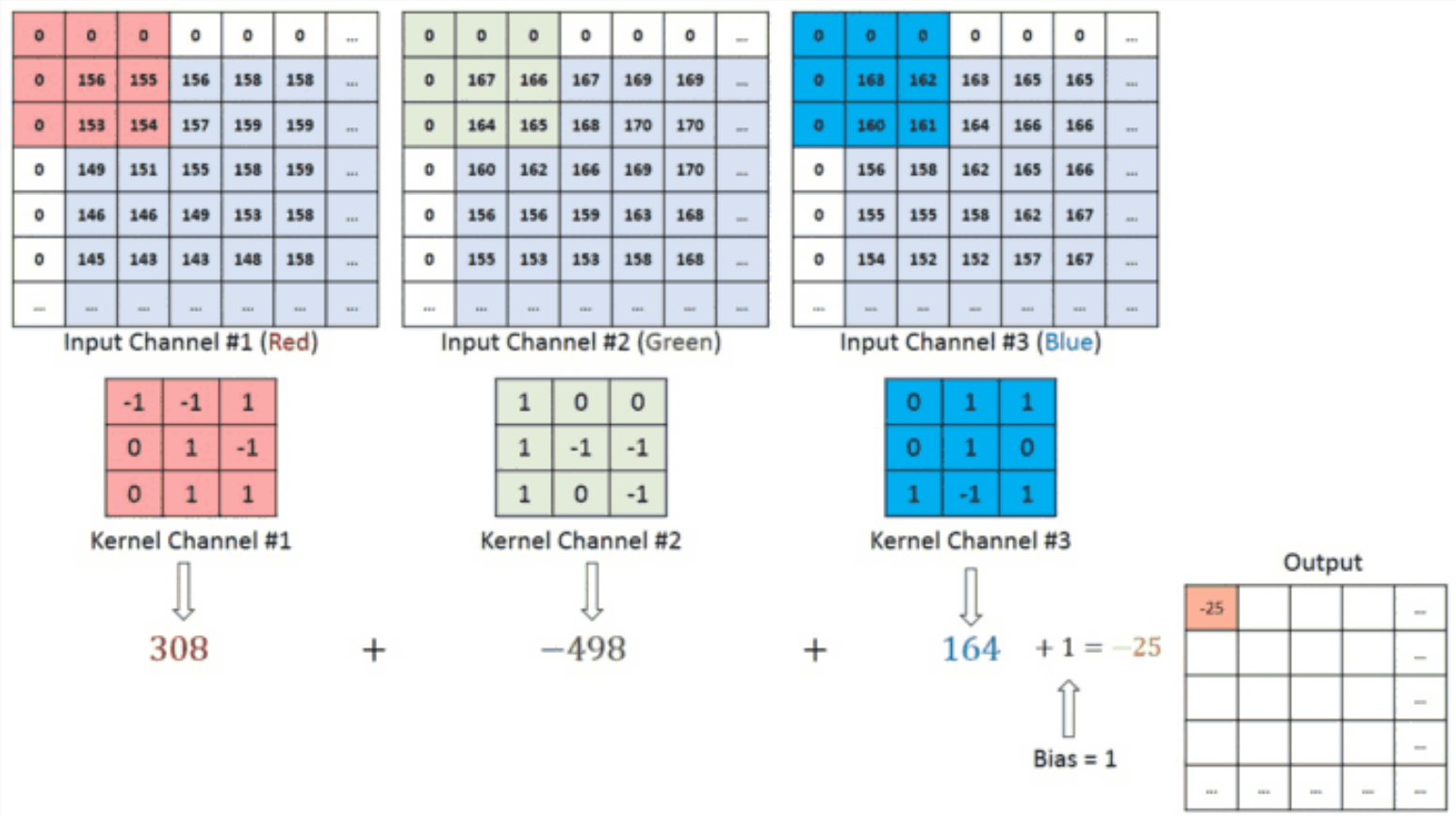
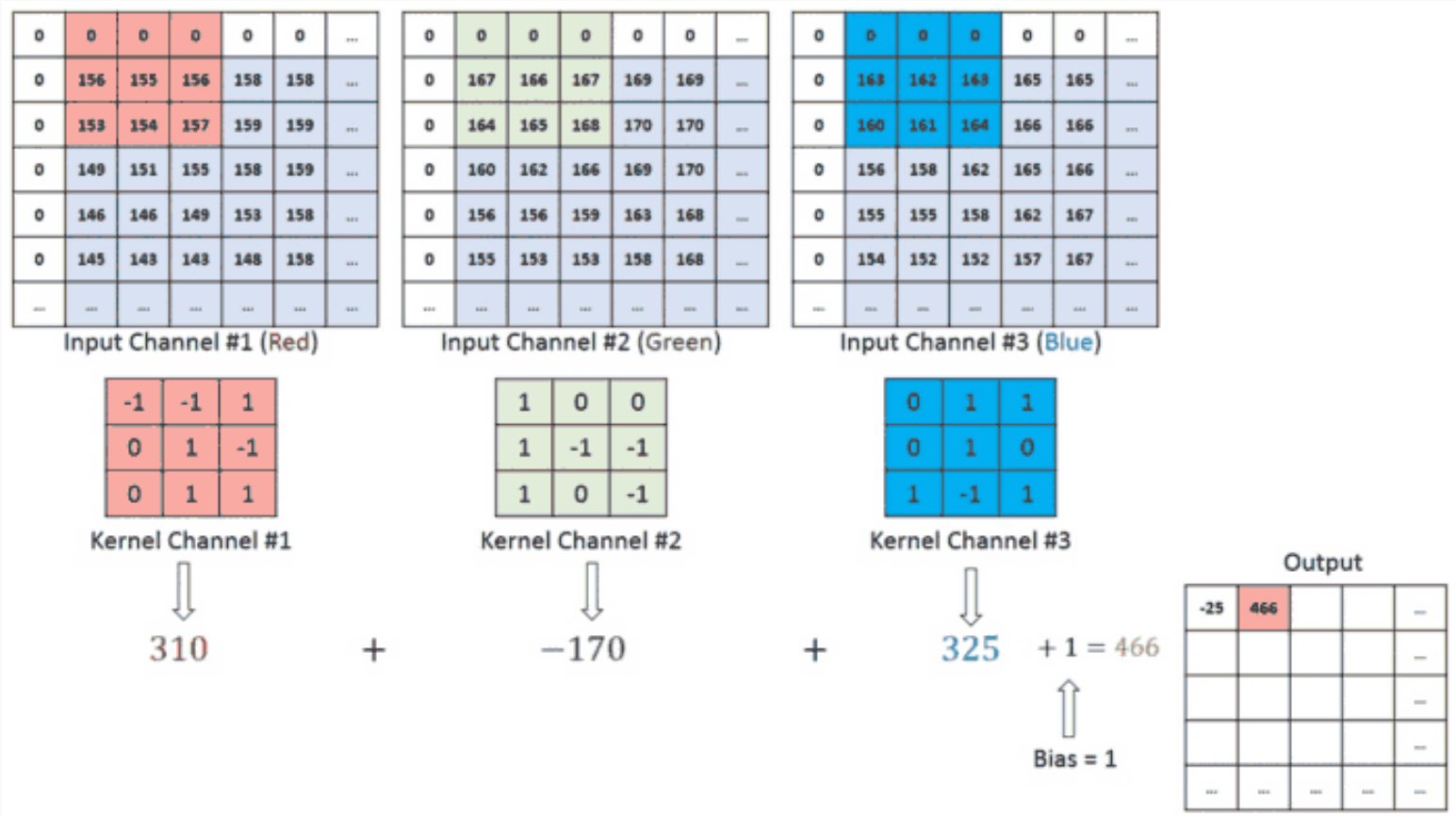


Figure 5: 3 Channels

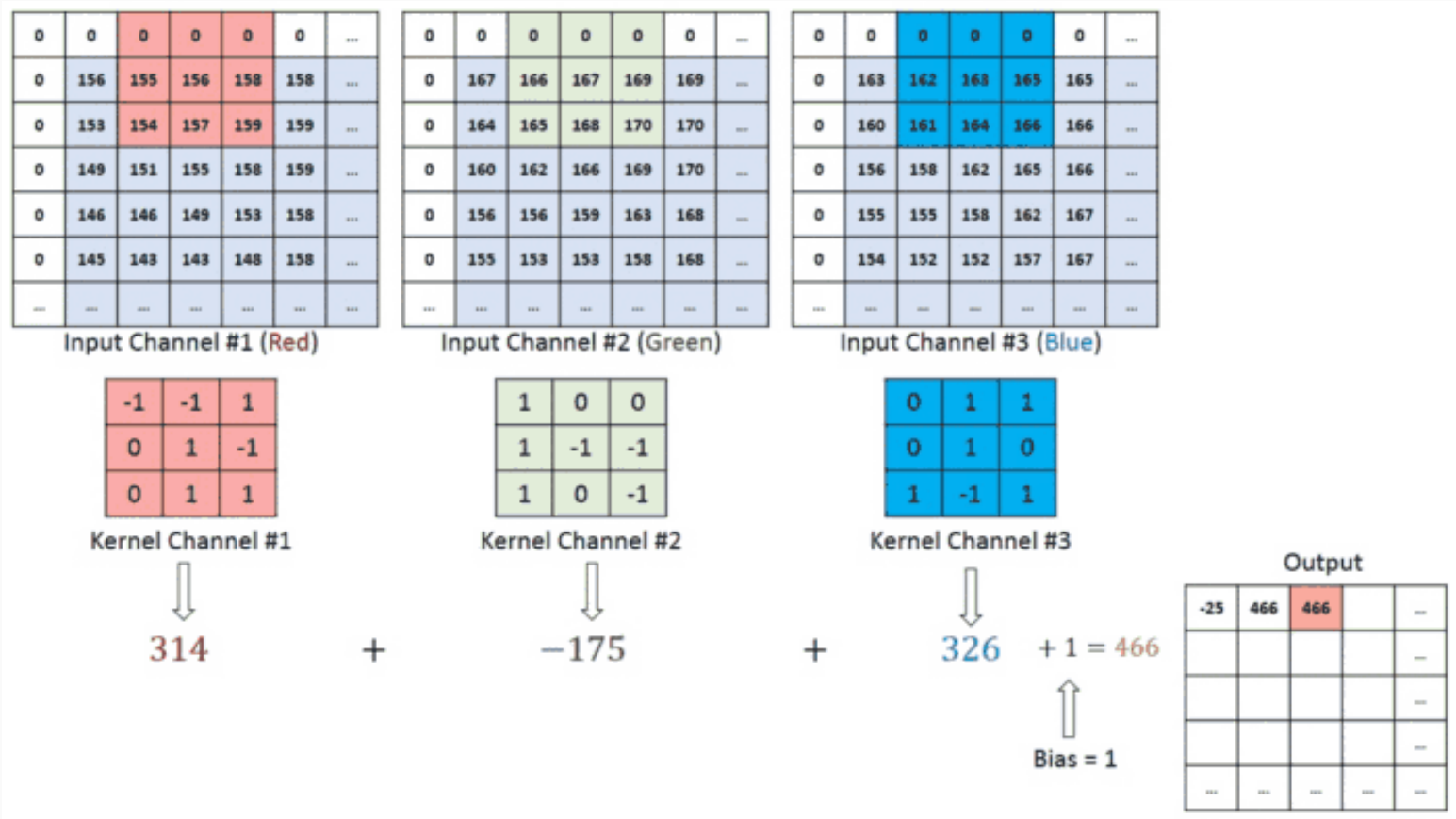
Image(MxNx3) Filter(3x3x3) 1



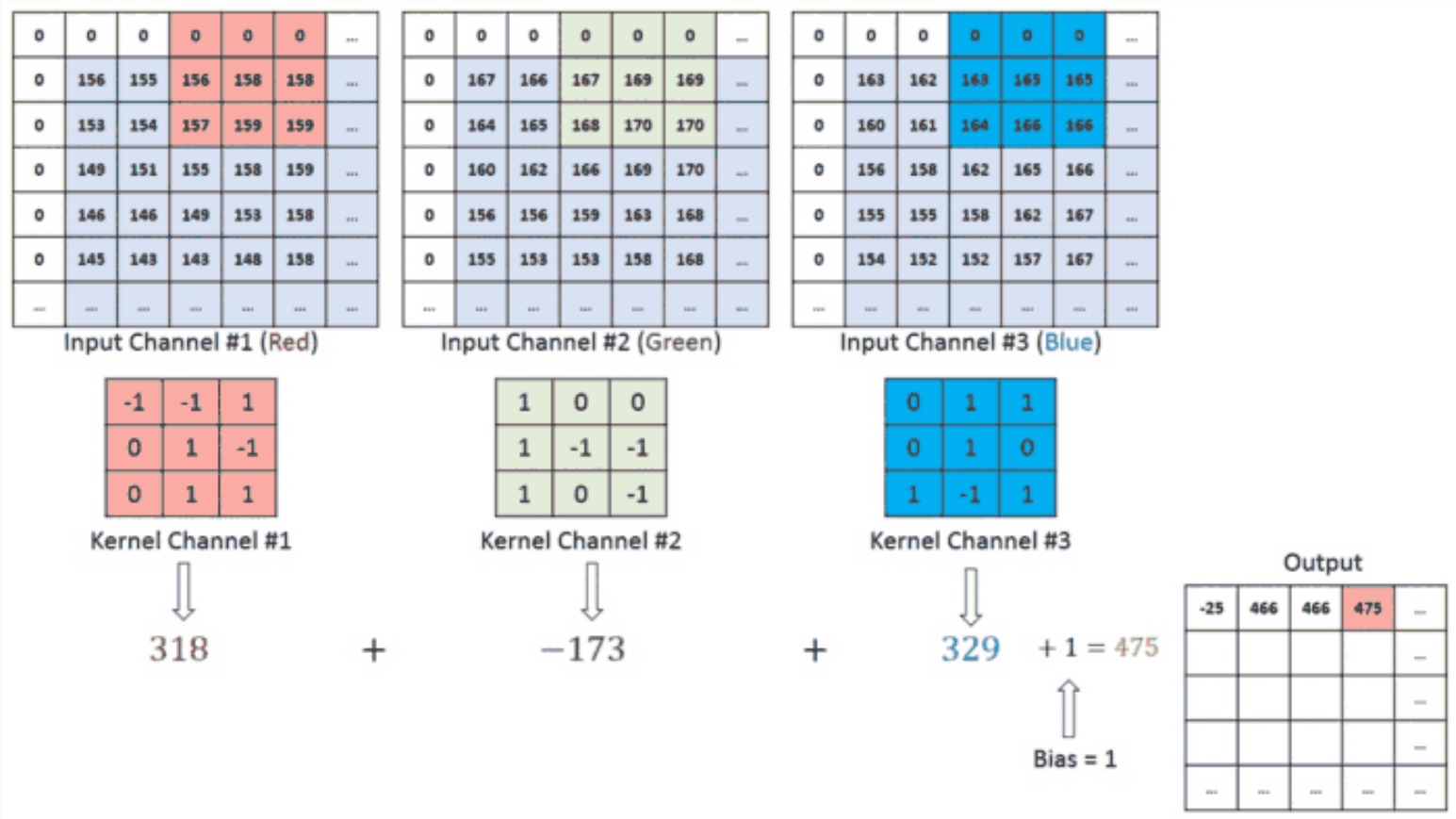
Image(MxNx3) Filter(3x3x3) 2



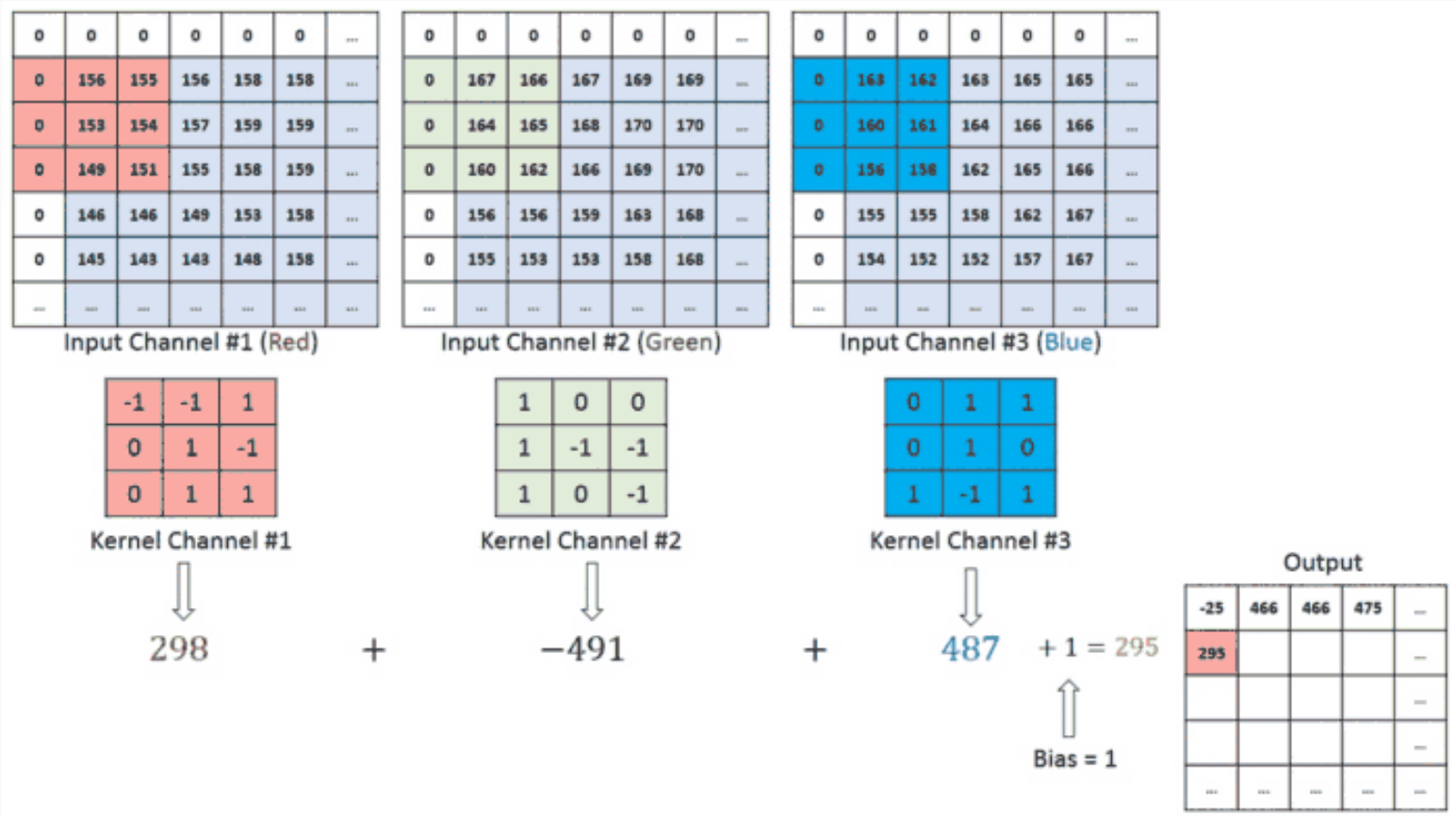
Image(MxNx3) Filter(3x3x3) 3



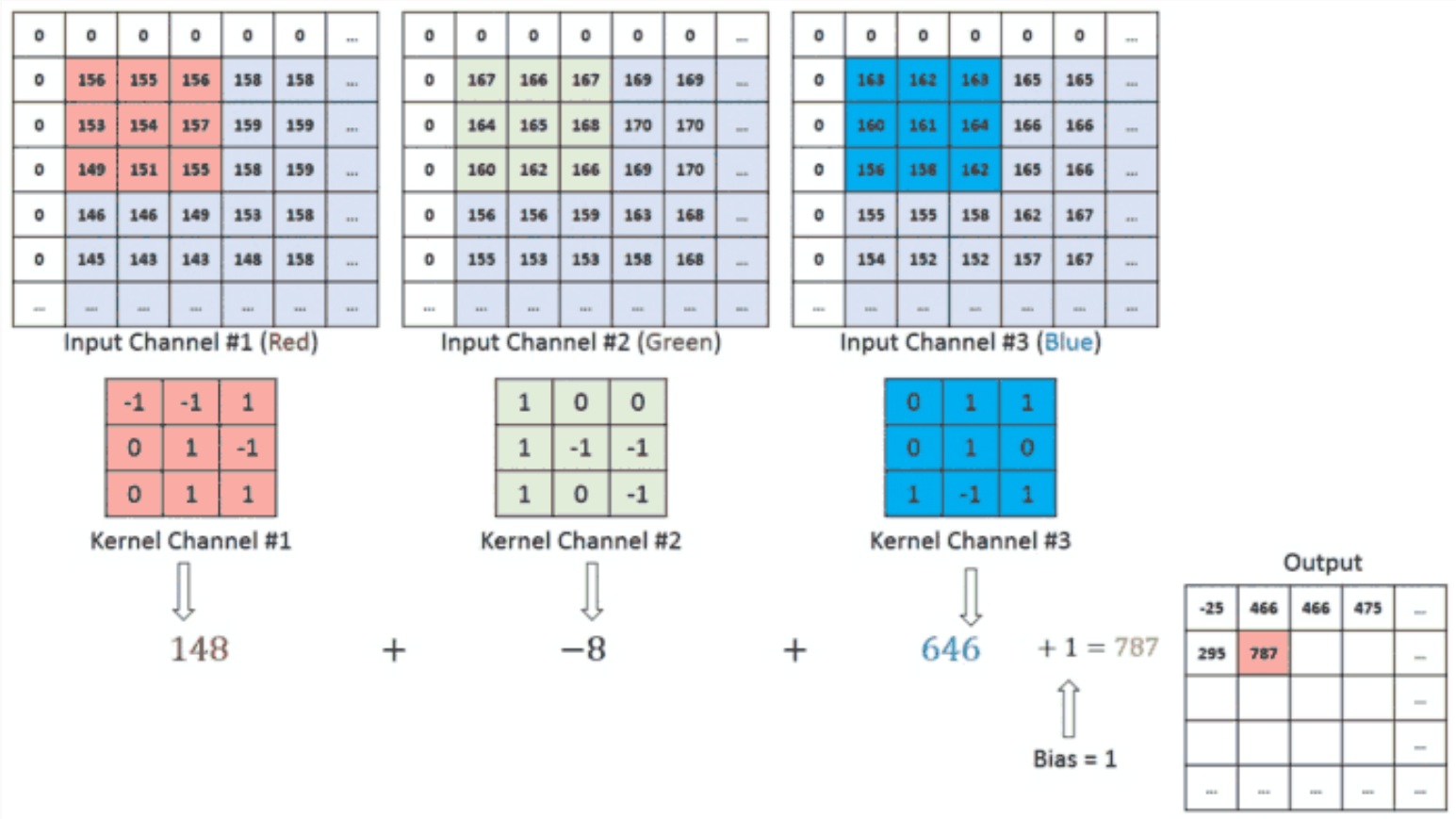
Image(MxNx3) Filter(3x3x3) 4



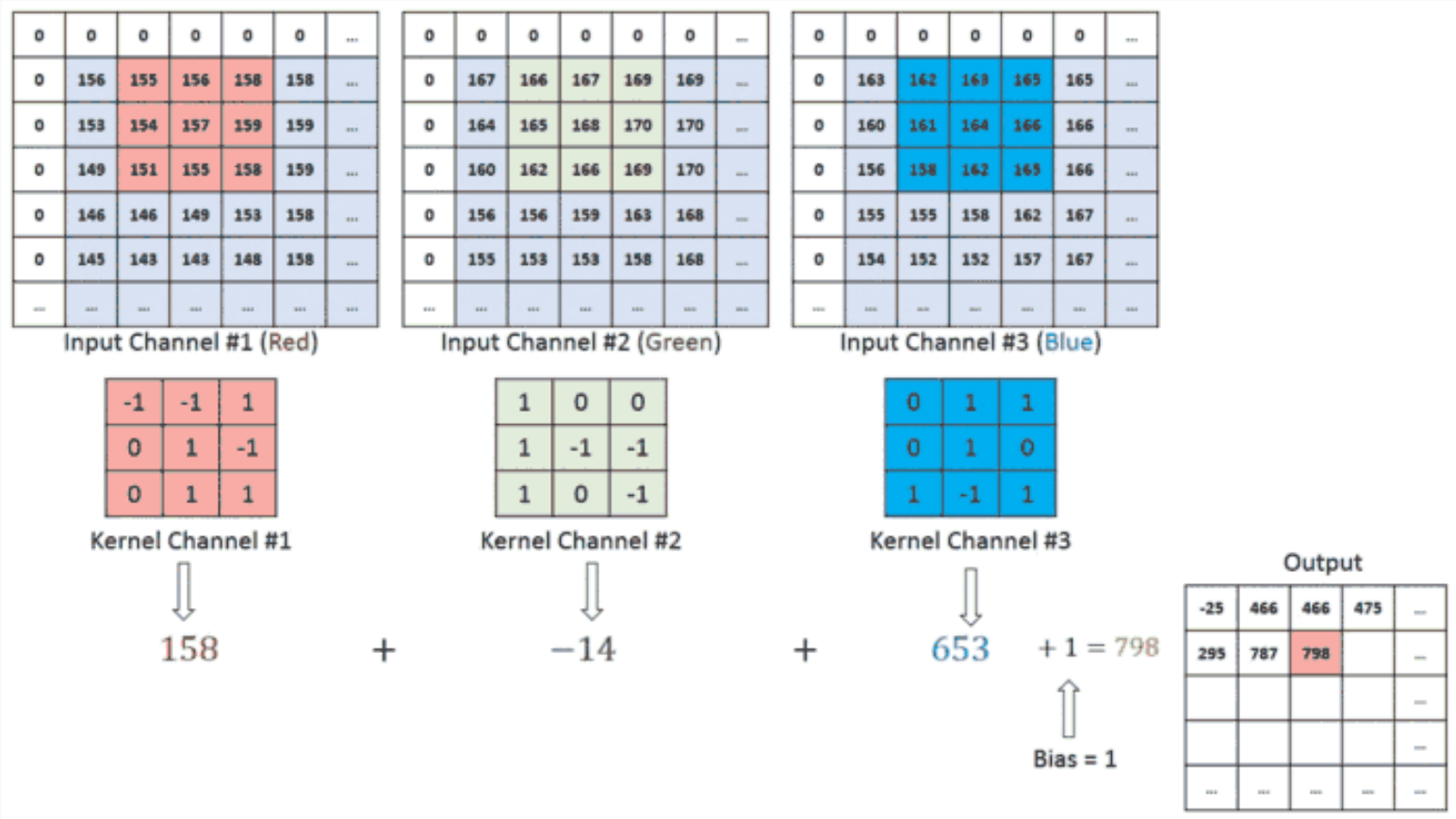
Image(MxNx3) Filter(3x3x3) 5



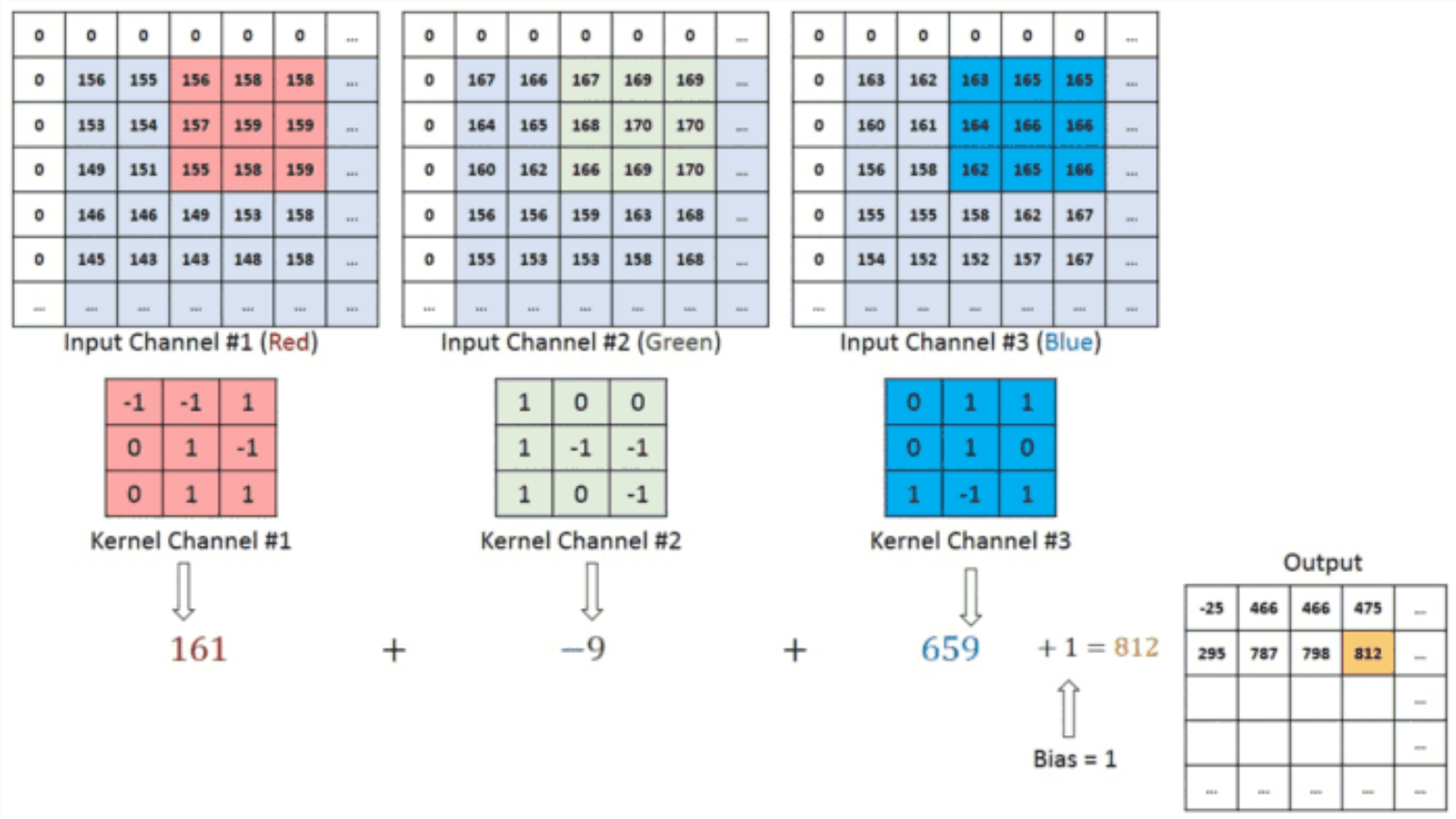
Image(MxNx3) Filter(3x3x3) 6



Image(MxNx3) Filter(3x3x3) 7



Image(MxNx3) Filter(3x3x3) 8



Movement of the kernel

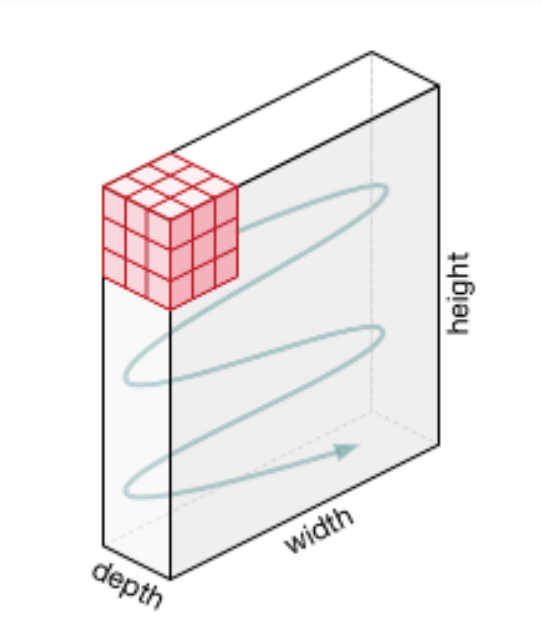


Figure 6: Movement of the kernel

What is a filter?

For example, below is a hand crafted 3×3 element filter for detecting vertical lines:

0.0, 1.0, 0.0
0.0, 1.0, 0.0
0.0, 1.0, 0.0

Applying this filter to an image will result in a feature map that only contains vertical lines.

A Convolutional Layer

- **Multiple Filters**: learn multiple features in parallel for a given input.
- **Multiple Channels**: Color images have multiple channels, such as red, green, and blue. That means that a single image provided as input to the model is, in fact, three images
- **Multiple Layers**: The filters that operate directly on the raw pixel values will learn to extract low-level features, such as lines. The filters that operate on the output of the first line layers may extract features that are combinations of lower-level features, such as features that comprise multiple lines to express shapes.

Example of 2D Convolutional Layer 1/7

```
# define input data
data = [[0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0]]
data = asarray(data)
data = data.reshape(1, 8, 8, 1)
```

4D shape [samples, rows, columns, channels] = [1, 8, 8, 1] in this case

Example of 2D Convolutional Layer 2/7

```
# create model
model = Sequential()
model.add(Conv2D(1,
                 (3,3),
                 input\_shape=(8, 8, 1)))
```

- We will define the Conv2D with a single filter.
- The filter will be two-dimensional and square with the shape 3×3.
- The layer will expect input samples to have the shape [columns, rows, channels] or [8,8,1].

Example of 2D Convolutional Layer 3/7

We will define a vertical line detector filter to detect the single vertical line in our input data.

```
# define a vertical line detector
detector = [[[[0]],[[1]],[[0]]],
           [[[0]],[[1]],[[0]]],
           [[[0]],[[1]],[[0]]]]
weights = [asarray(detector), asarray([0.0])]

# store the weights in the model
model.set_weights(weights)
```

Example of 2D Convolutional Layer 4/7

Apply the filter to the input image

```
# apply filter to input data
pred = model.predict(data)
```

Example of 2D Convolutional Layer 5/7

$$\begin{matrix} 0, 1, 0 \\ 0, 1, 0 \\ 0, 1, 0 \end{matrix} \quad . \quad \begin{matrix} 0, 0, 0 \\ 0, 0, 0 \\ 0, 0, 0 \end{matrix} \quad = \quad 0$$

The filter is moved along one by one column to the left. the feature is not detected.

$$\begin{matrix} 0, 1, 0 \\ 0, 1, 0 \\ 0, 1, 0 \end{matrix} \quad . \quad \begin{matrix} 0, 0, 1 \\ 0, 0, 1 \\ 0, 0, 1 \end{matrix} \quad = \quad 0$$

One more move to the left to the next column and the feature is detected for the first time.

$$\begin{matrix} 0, 1, 0 \\ 0, 1, 0 \\ 0, 1, 0 \end{matrix} \quad . \quad \begin{matrix} 0, 1, 1 \\ 0, 1, 1 \\ 0, 1, 1 \end{matrix} \quad = \quad 3$$

and the process is repeated

Example of 2D Convolutional Layer 6/7

This process is repeated until the edge of the filter rests against the edge or final column of the input image. This gives the last element in the first full row of the feature map.

[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]

Example of 2D Convolutional Layer 7/7

The feature map is a **6×6** matrix, **smaller than the 8×8 input** image because of the limitations of how the filter can be applied to the input image.

[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]

Pooling Layer

- It can reduce the dimensions of the feature maps,
- increase the robustness of feature extraction.
- It is usually placed between two Convolutional layers.
- The typical pooling operations are average pooling [11] and max pooling. [1].

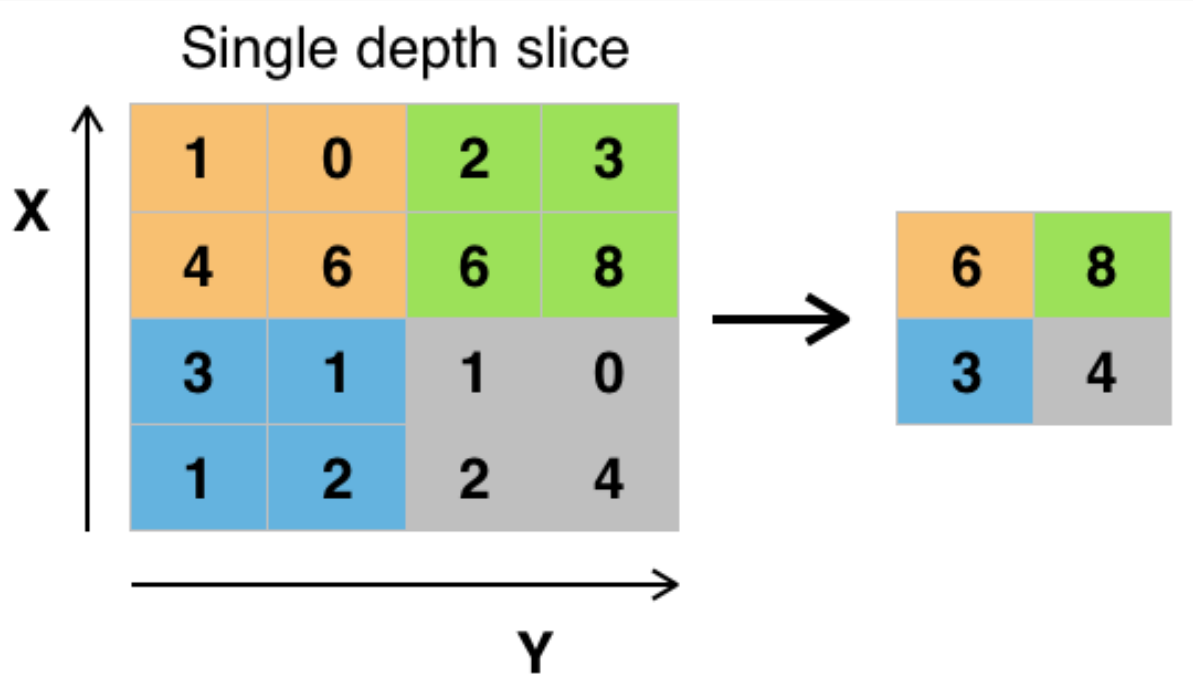


Figure 7: Max-Pooling

Fully-connected (Dense) Layer

Take the results of the convolution/pooling process and use them to classify the image into a label (in a simple classification example).

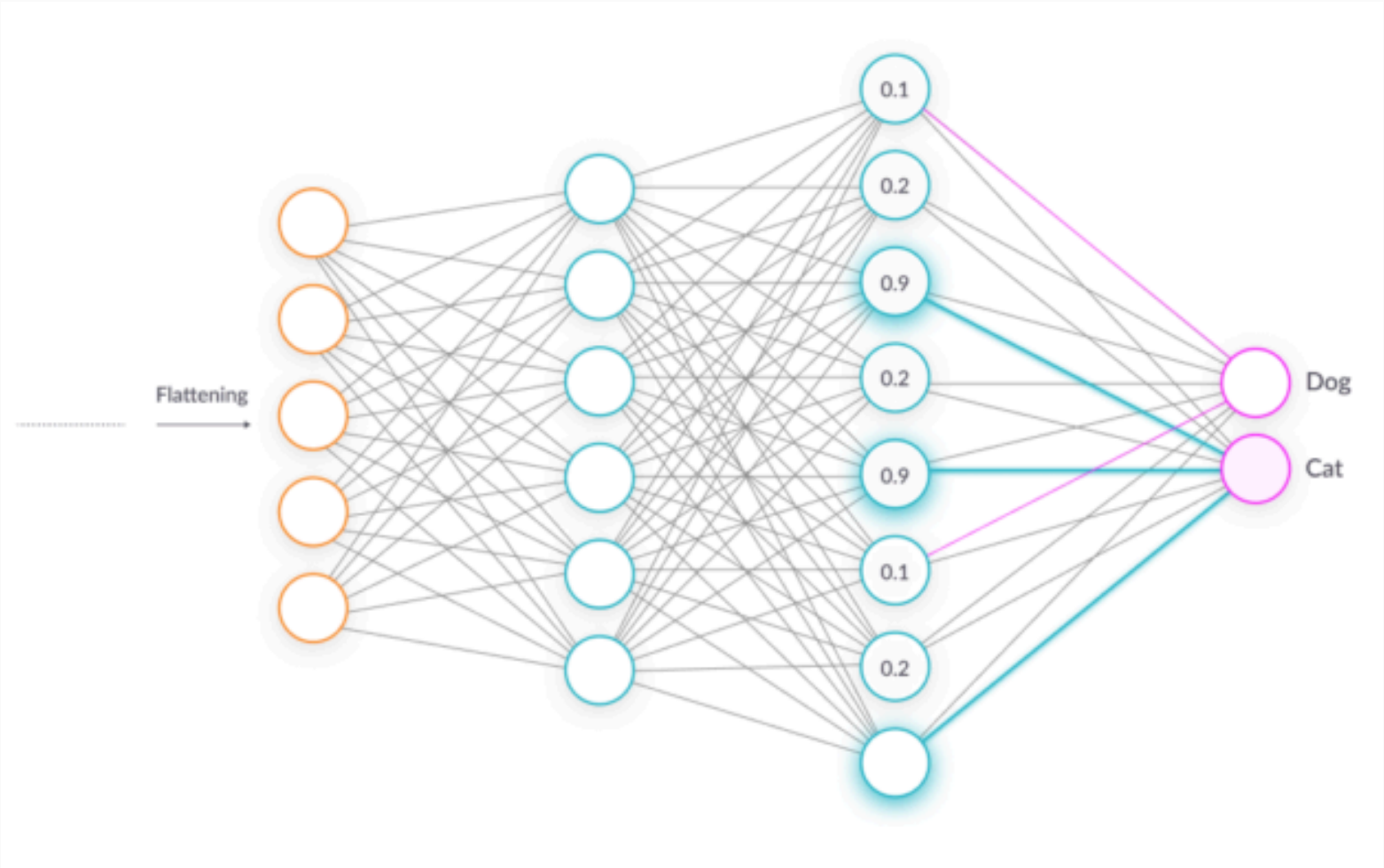


Figure 8: Fully Connected Layer

Dropout Layer

- Prevent the NN from overfitting during the training.
- Dropout keeping a neuron active with some probability p and setting it to 0 otherwise.
- This forces the network to not learn redundant information.

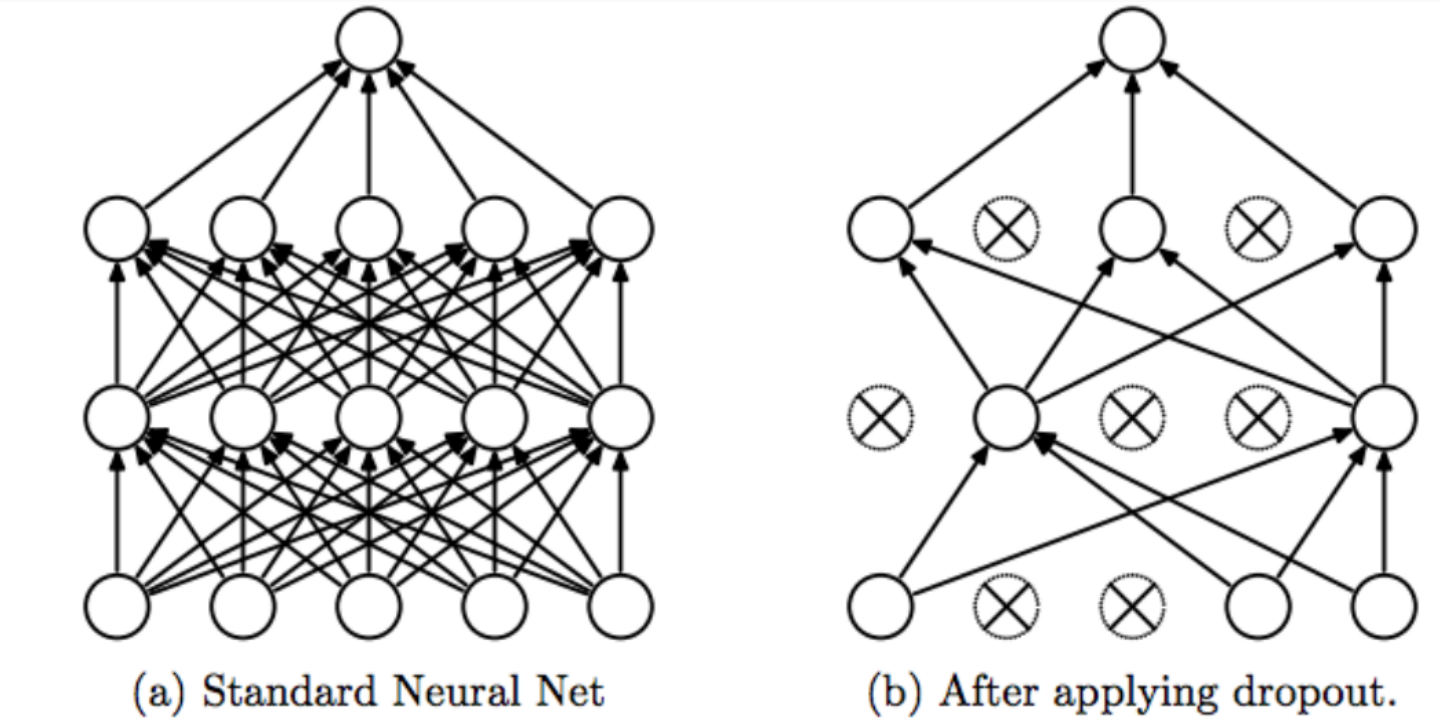


Figure 9: Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014 [8]

Flatten Layer

In between the convolutional layer and the fully connected layer.

Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier (Dense layer).

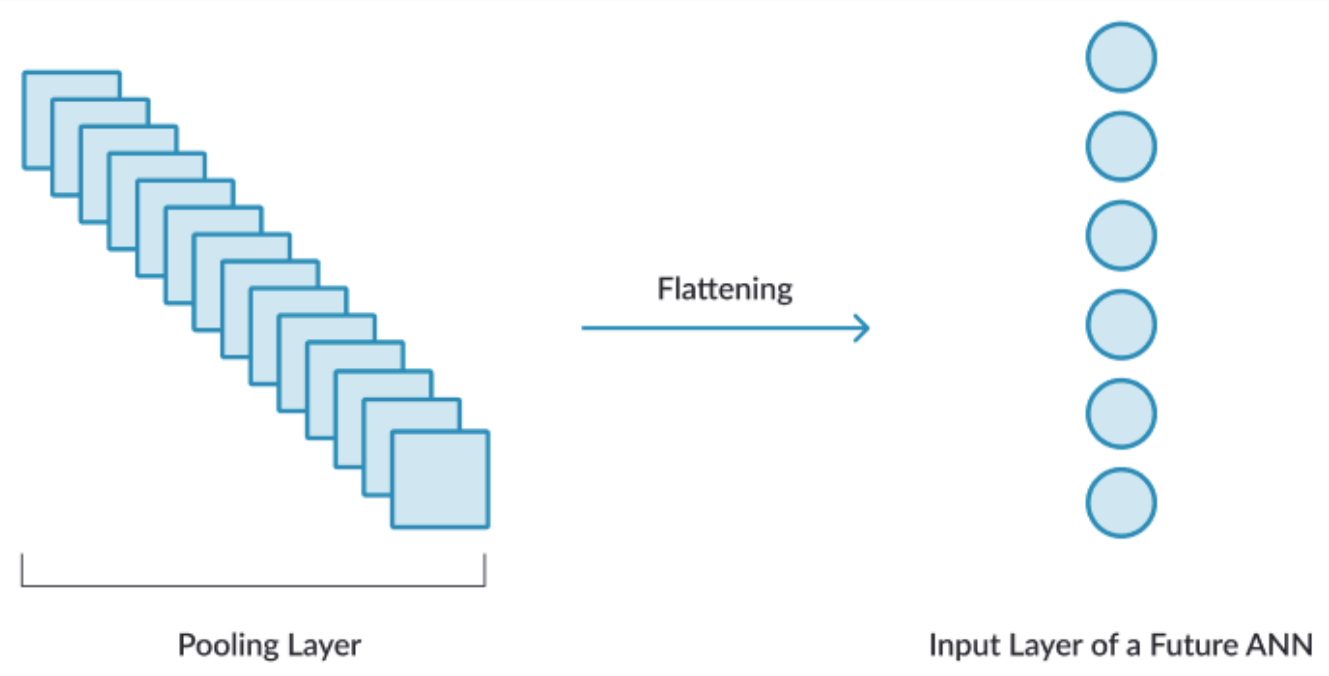


Figure 10: Flatten Layer

Methodology

Requirements

- **Development** was made with python 3.7 in PyCharm IDE.
- **TensorFlow** is an end-to-end open source platform for machine learning
- **Keras** is a high-level neural networks API, written in Python and capable of running on top of TensorFlow
- **TensorFlow.js** is an open-source library you can use to define, train, and run machine learning models entirely in the browser, using Javascript and a high-level layers API
- **pandas, numpy, sklearn, iter-tools, shutil**
- **System Details** MacBook Pro CPU: 2,3GHz Intel Core i7 (4 Cores) RAM: 16 GB1600MHz DDR3

Dataset collection 1/2

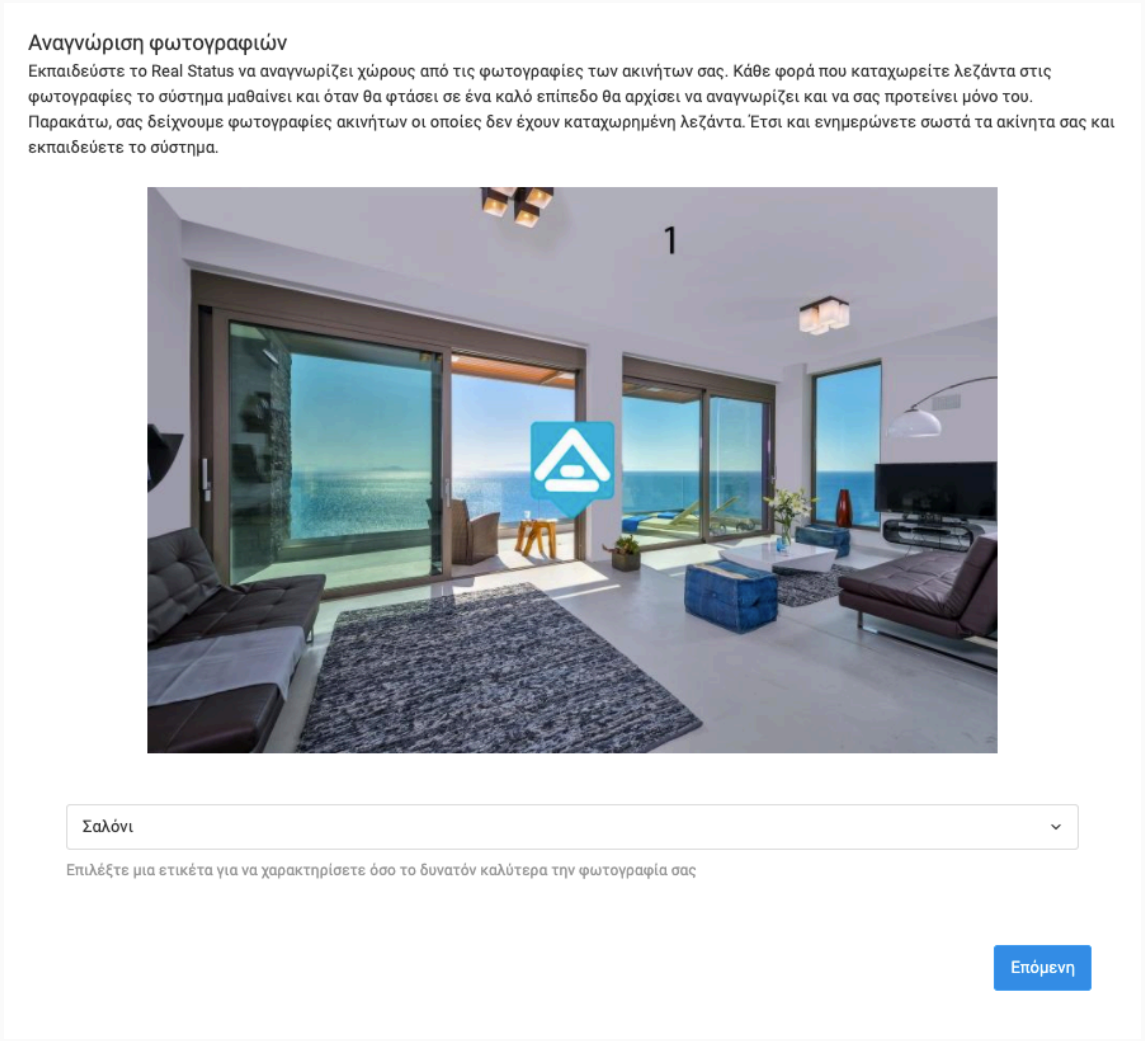


Figure 11: *https://cloud.realstatus.net/demo*

Dataset collection 2/2

The initial dataset we managed to classify was very small.
With this **small** and **unballanced** dataset the process of machine learning would be impossible. And the reason will be overfitting

Class	Total Images
Kitchen	510
Living room	276
Bedroom	394
Bathroom	218
WC	166
Exterior of the house	86
Pool	121
View from balcony	180
Elevator	89
Top view	135
Balcony	298
Hallway	160
View	252
hasHuman	141
Total:	3027

“Why small datasets lead to overfitting?”

The goal of a machine learning model is to generalize patterns in training data so that you can correctly predict new data that has never been presented to the model.
Overfitting occurs when a model adjusts excessively to the training data, seeing patterns that do not exist, and consequently performing poorly in predicting new data:

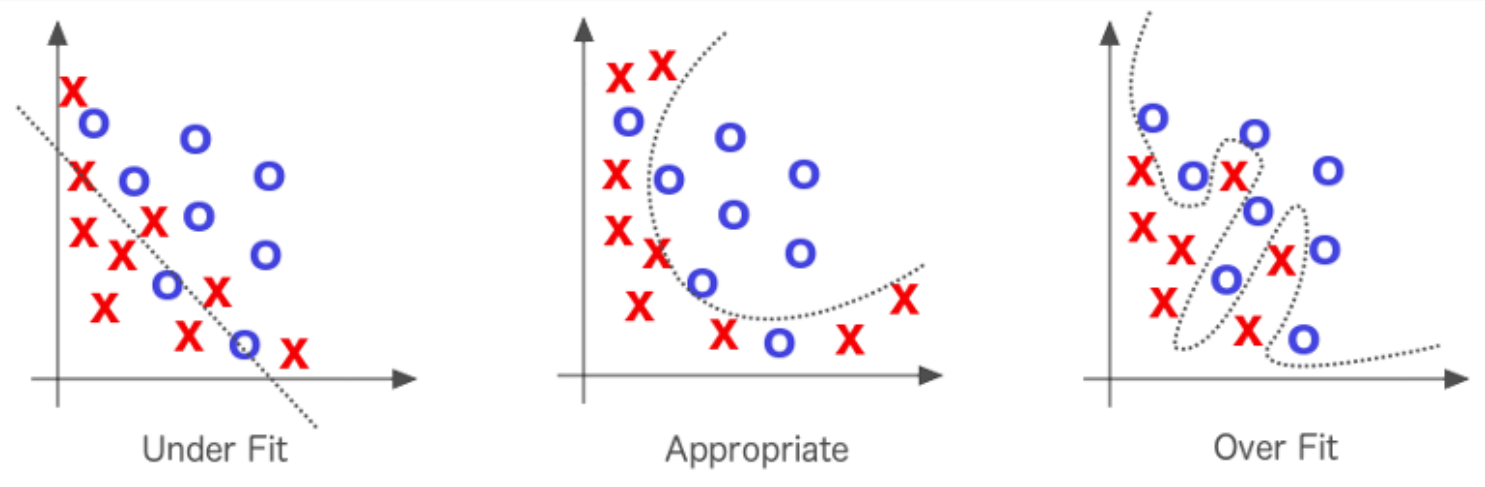


Figure 12: overfitting. Source: https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets

The fewer samples for training, the more models can fit our data.

In an extreme example (a), for just one training point, any model will be able to "explain" it, however simple or complex the model may be. As we get to have more samples (b, c), fewer models are able to explain them:

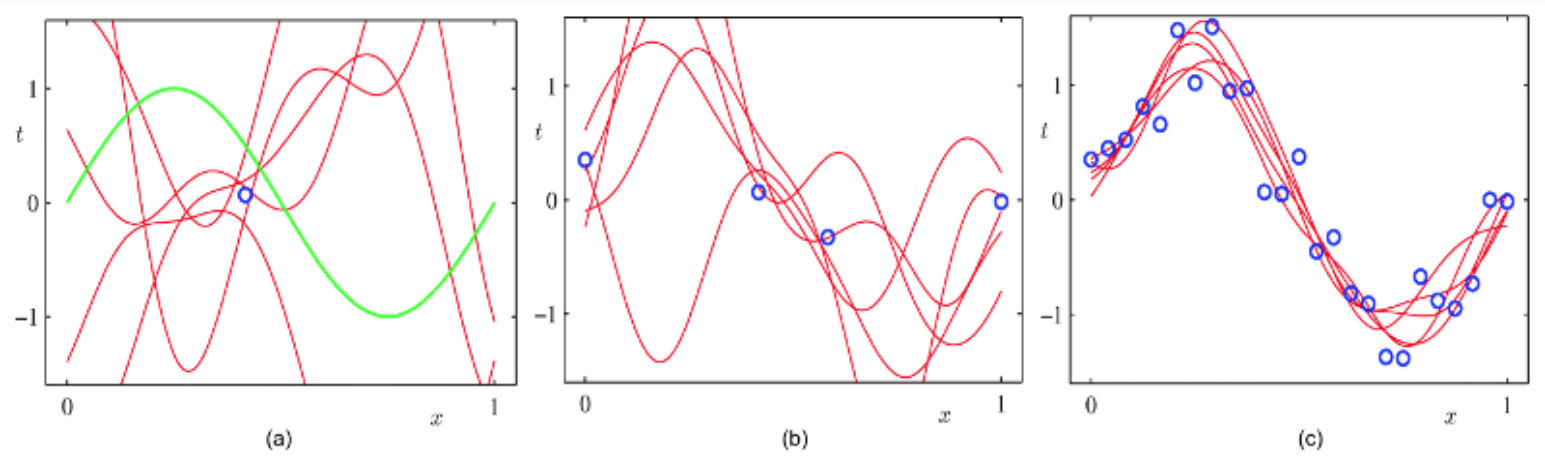


Figure 13: overfitting. Source: https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets

Pre-Processing and Data-Augmentation

Data augmentation is a regularization technique where we produce more images from the training data provided with random jitter, crop, rotate, reflect, scaling etc.

For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right.

Exchangeable image file format

(officially Exif, according to JEIDA/JEITA/CIPA specifications)

"Is a standard that specifies the formats for images, sound, and ancillary tags used by digital cameras (including smartphones), scanners and other systems handling image and sound files recorded by digital cameras" [12].

Tag	Value
Manufacturer	CASIO
Model	QV-4000
Orientation (rotation)	top-left [8 possible values[24]]
Software	Ver1.01
Date and time	2003:08:11 16:45:32
YCbCr positioning	centered
Compression	JPEG compression
X resolution	72.00
Y resolution	72.00
Resolution unit	Inch
Exposure time	1/659 s
F-number	f/4.0
Exposure program	Normal program
Exif version	Exif version 2.1
Date and time (original)	2003:08:11 16:45:32
Date and time (digitized)	2003:08:11 16:45:32
Components configuration	Y Cb Cr –

Orientation - Rotation

Note that one would only expect the four orientation settings to be possible with a digital camera. The other four settings 2 - 4 - 5 - 7 would imply that the resulting image was flipped horizontally before recording (e.g, taking a photo in a mirror).

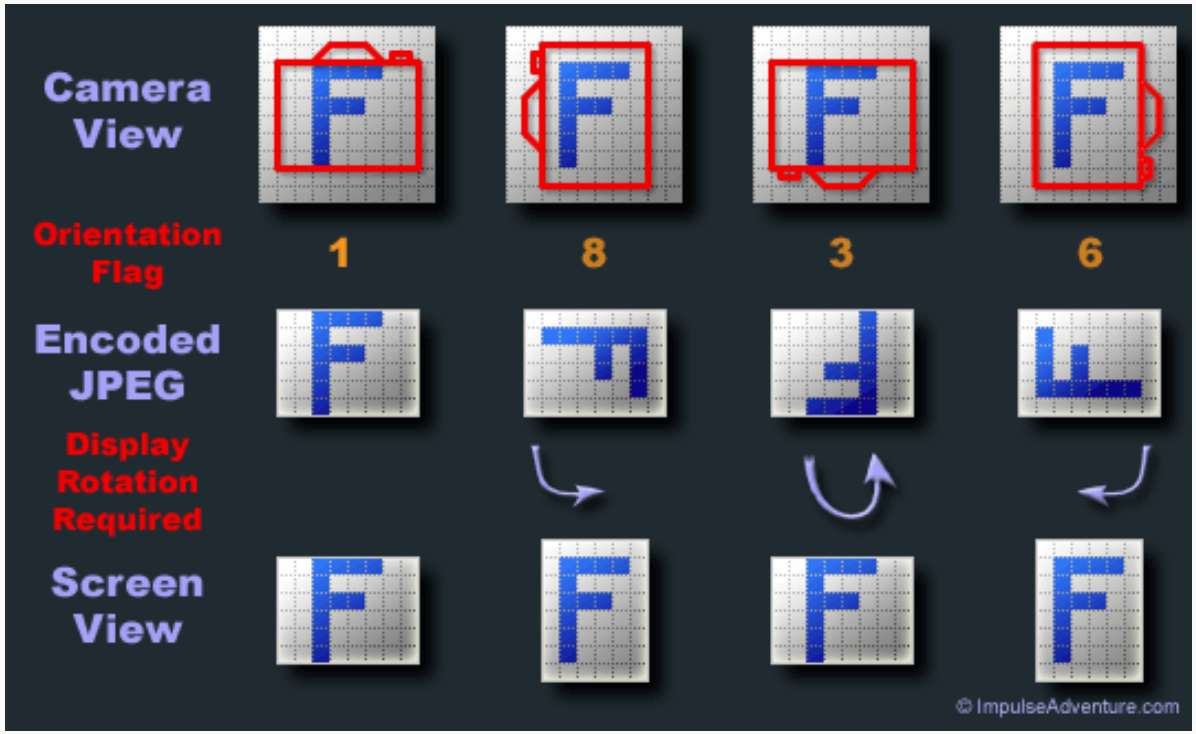


Figure 14: https://www.impulseadventure.com/photo/exif-orientation.html

The augmented dataset

After the data augmentation we expand the training dataset, as we can see in the following table.

Class	Total Images
Kitchen	5600
Living room	5551
Bedroom	5946
Bathroom	5259
WC	5001
Exterior of the house	5151
Pool	4851
View from balcony	5421
Elevator	5331
Top view	5411
Balcony	5991
Hallway	4821
View	5071
hasHuman	5651
Total:	75057

Model Selection

As we can see MobileNet, MobileNetV2, NASNetMobile and EfficientNetB0 has the fewest parameters and as a result the smaller size.

Model	Size	Parameters
Xception	88 MB	22,910,480
VGG16	528 MB	138,357,544
VGG19	549 MB	143,667,240
ResNet50	98 MB	25,636,712
ResNet101	171 MB	44,707,176
ResNet152	232 MB	60,419,944
ResNet50V2	98 MB	25,613,800
ResNet101V2	171 MB	44,675,560
ResNet152V2	232 MB	60,380,648
InceptionV3	92 MB	23,851,784
InceptionResNetV2	215 MB	55,873,736
MobileNet	16 MB	4,253,864
MobileNetV2	14 MB	3,538,984
DenseNet121	33 MB	8,062,504
DenseNet169	57 MB	14,307,880
DenseNet201	80 MB	20,242,984
NASNetMobile	23 MB	5,326,716
NASNetLarge	343 MB	88,949,818
EfficientNetB0	29 MB	5,330,571
EfficientNetB1	31 MB	7,856,239
EfficientNetB2	36 MB	9,177,569
EfficientNetB3	48 MB	12,320,535
EfficientNetB4	75 MB	19,466,823

Why?

- Reduced training time
- small size and speed makes it ideal for web deployment

realstatus.gr/labs/thesis/

Indoor and outdoor view recognition using deep convolutional neural networks

The model has been trained in the following 14 classes:
Bedroom, Kitchen, Living room, WC, Bathroom, Hallway, View (Empty Room), Elevator,
Balcony, Exterior of the house, Pool, View from balcony, Top view (Architectural design), Has Human (Existence of a human figure)

Select Image(s)

Selected model:

MobileNet (Freeze 23 Layers)

Results

Model MobileNetFreeze23Layers load time: 13.545 sec(s).

ihu.gr

Elements Console Sources Network Performance Memory Application Security Lighthouse

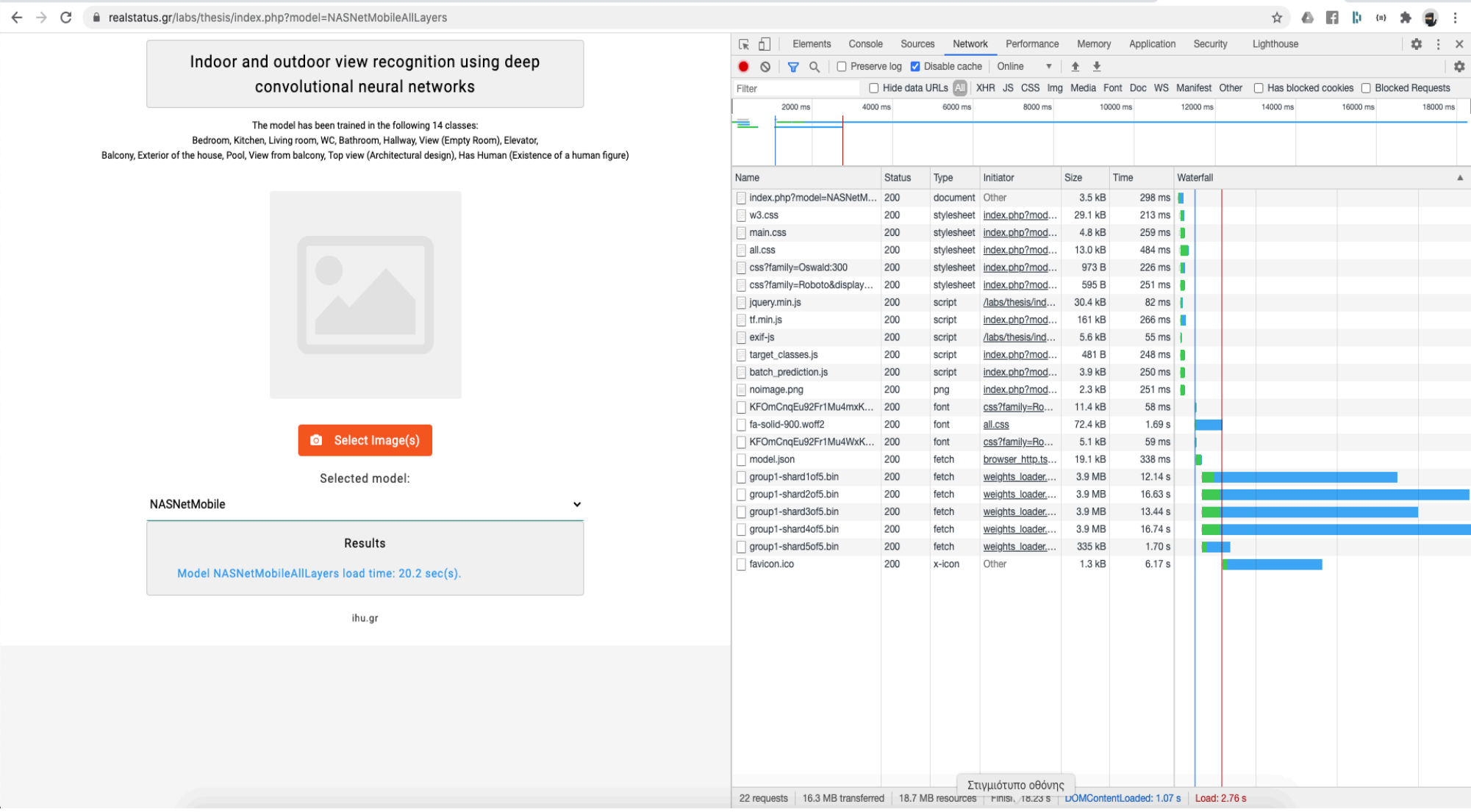
Preserve log | Disable cache | Online

Filter | Hide data URLs | All | XHR JS CSS Img Media Font Doc WS Manifest Other | Has blocked cookies | Blocked Requests

1000 ms 2000 ms 3000 ms 4000 ms 5000 ms 6000 ms 7000 ms 8000 ms 9000 ms 10000 ms 11000 ms 12000 ms 13000 ms 14000 ms 1500

Name	Status	Type	Initiator	Size	Time	Waterfall
thesis/	200	document	Other	3.5 kB	236 ms	
w3.css	200	stylesheet	(index)	29.1 kB	270 ms	
main.css	200	stylesheet	(index)	4.8 kB	271 ms	
all.css	200	stylesheet	(index)	13.0 kB	352 ms	
css?family=Oswald:300	200	stylesheet	(index)	973 B	123 ms	
css?family=Roboto&display…	200	stylesheet	(index)	595 B	303 ms	
jquery.min.js	200	script	(index)	30.8 kB	315 ms	
tf.min.js	200	script	(index)	161 kB	515 ms	
exif-js	200	script	(index)	5.6 kB	55 ms	
target_classes.js	200	script	(index)	481 B	249 ms	
batch_prediction.js	200	script	(index)	3.9 kB	249 ms	
noimage.png	200	png	(index)	2.3 kB	249 ms	
KFOmCnqEu92Fr1Mu4mxK…	200	font	css?family=Ro…	11.4 kB	60 ms	
fa-solid-900.woff2	200	font	all.css	72.4 kB	194 ms	
KFOmCnqEu92Fr1Mu4WxK…	200	font	css?family=Ro…	5.1 kB	63 ms	
model.json	200	fetch	browser_http.ts…	3.3 kB	608 ms	
favicon.ico	200	x-icon	Other	1.3 kB	597 ms	
group1-shard1of4.bin	200	fetch	weights_loader.…	3.9 MB	10.14 s	
group1-shard2of4.bin	200	fetch	weights_loader.…	3.9 MB	11.15 s	
group1-shard3of4.bin	200	fetch	weights_loader.…	3.9 MB	12.07 s	
group1-shard4of4.bin	200	fetch	weights_loader.…	363 kB	2.09 s	

21 requests | 12.4 MB transferred | 14.1 MB resources | Finish: 14.15 s | DOMContentLoaded: 1.40 s | Load: 1.46 s


realstatus.gr/labs/thesis/index.php?model=NASNetMobileAllLayers
Indoor and outdoor view recognition using deep convolutional neural networks
The model has been trained in the following 14 classes:
Bedroom, Kitchen, Living room, WC, Bathroom, Hallway, View (Empty Room), Elevator,
Balcony, Exterior of the house, Pool, View from balcony, Top view (Architectural design), Has Human (Existence of a human figure)
Select Image(s)
Selected model:
NASNetMobile
Results
Model NASNetMobileAllLayers load time: 20.2 sec(s).
ihu.gr
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Preserve log
Disable cache
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
Has blocked cookies
Blocked Requests
2000 ms
4000 ms
6000 ms
8000 ms
10000 ms
12000 ms
14000 ms
16000 ms
18000 ms
Name
Status
Type
Initiator
Size
Time
Waterfall
index.php?model=NASNetM... 200 document Other 3.5 kB 298 ms
w3.css 200 stylesheet index.php?mod... 29.1 kB 213 ms
main.css 200 stylesheet index.php?mod... 4.8 kB 259 ms
all.css 200 stylesheet index.php?mod... 13.0 kB 484 ms
css?family=Oswald:300 200 stylesheet index.php?mod... 973 B 226 ms
css?family=Roboto&display... 200 stylesheet index.php?mod... 595 B 251 ms
jquery.min.js 200 script /labs/thesis/ind... 30.4 kB 82 ms
tf.min.js 200 script index.php?mod... 161 kB 266 ms
exif-js 200 script /labs/thesis/ind... 5.6 kB 55 ms
target_classes.js 200 script index.php?mod... 481 B 248 ms
batch_prediction.js 200 script index.php?mod... 3.9 kB 250 ms
noimage.png 200 png index.php?mod... 2.3 kB 251 ms
KFOmCnqEu92Fr1Mu4mxK... 200 font css?family=Ro... 11.4 kB 58 ms
fa-solid-900.woff2 200 font all.css 72.4 kB 1.69 s
KFOmCnqEu92Fr1Mu4WxK... 200 font css?family=Ro... 5.1 kB 59 ms
model.json 200 fetch browser_http.ts... 19.1 kB 338 ms
group1-shard1of5.bin 200 fetch weights_loader.... 3.9 MB 12.14 s
group1-shard2of5.bin 200 fetch weights_loader.... 3.9 MB 16.63 s
group1-shard3of5.bin 200 fetch weights_loader.... 3.9 MB 13.44 s
group1-shard4of5.bin 200 fetch weights_loader.... 3.9 MB 16.74 s
group1-shard5of5.bin 200 fetch weights_loader.... 335 kB 1.70 s
favicon.ico 200 x-icon Other 1.3 kB 6.17 s
Στιγμιότυπο οθόνης
22 requests
16.3 MB transferred
18.7 MB resources
DOMContentLoaded: 1.07 s
Load: 2.76 s

Training process

Prepare model (Transfer learning - Fine Tuning)

1. Pre-trained MobileNet model weights (1000 Classes)
2. Exclude the last 5 layers of the above model (Dense 1000 classes)
3. Create a new dense layer for predictions (14 Classes)

Compile

Configures the model for training

Callbacks

A callback is an object that can perform actions at various stages of training

fit

Trains the model for a fixed number of epochs

Training process / Prepare model

```
mobile = MobileNet()
x = mobile.layers[-6].output
x = Dropout(0.25)(x)
predictions = Dense(14,activation='softmax')(x)
model = Model(inputs=mobile.input,
              outputs=predictions)
for layer in model.layers[:-23]:
    layer.trainable = False
model.compile(...)
model.fit(...)
```

Training process / Compile

```
model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=[categorical_accuracy,
                       top_2_accuracy,
                       top_3_accuracy])
```

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Stochastic gradient descent

Gradient descent

- Batch Gradient descent
- Stochastic Gradient descent

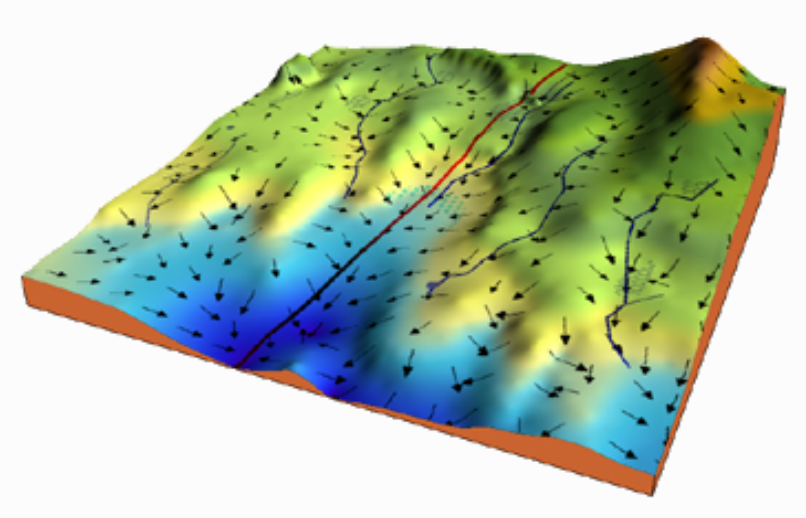


Figure 16: 3-dimensional graph of a cost function

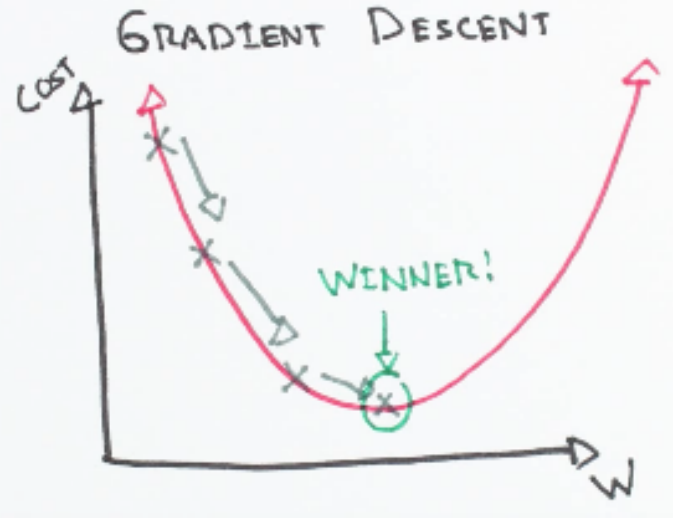


Figure 17: Gradient descent step

Usage of callbacks: A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

- **ModelCheckpoint**: The training process is set up to monitor validation accuracy. This helps to ensure that the best version of the model gets saved i.e. if the model starts to overfit it does not get saved.
- **ReduceLROnPlateau**: Reduce learning rate when a metric has stopped improving.
- **CSVLogger**: Streams epoch results to a CSV file.

Keras Callbacks *https://keras.io/api/callbacks/*

fit_generator: Trains the model on data generated batch-by-batch by a Python generator (or an instance of Sequence). The generator is run in parallel to the model, for efficiency.

```
callbacks_list = [checkpoint, reduce_lr, csv_logger]
history = model.fit_generator(generator=train_batches,
                              steps_per_epoch=train_steps,
                              class_weight=class_weights,
                              validation_data=valid_batches,
                              validation_steps=val_steps,
                              epochs=30,
                              verbose=2,
                              callbacks=callbacks_list)
```

Model evaluation

- Categorical Accuracy: Measures how often the model gets the prediction right.
- Categorical Crossentropy: Is a prediction error of Neural Net
- Precision: Given a class prediction from a classifier, how likely is it to be correct?
- Recall: Given a class, will the classifier be able to detect it?
- F1 Score: The harmonic mean of the recall and precision. Essentially, it punishes extreme values.

Model evaluation

1) loss > validation_loss (Underfitting)

Underfitting refers to a model that can neither model the training data nor generalize to new data.

2) loss « validation_loss (Overfitting)

This means that your model is fitting very nicely the training data but not at all the validation data, in other words it's not generalizing correctly to unseen data

3) loss == validation_loss ("Perfect" fitting)

If both values end up to be roughly the same and also if the values are converging (plot the loss over time) then chances are very high that you are doing it right

Experiment 1

Mobile Net - All Layers - batch size 10 $\approx$ >2.5h/epoch = 75h Total

Total params: 3,243,214 **Trainable**: 3,221,326 **Non-trainable**: 21,888

The last epoch

- val_loss: 0.0
- val_cat_acc: 0.9144144058227539
- val_top_2_acc: 0.9639639854431152
- val_top_3_acc: 0.9864864945411682

the best epoch (val_top_3_acc)

- val_loss: 0.0
- val_cat_acc: 0.9144144058227539
- val_top_2_acc: 0.954954981803894
- val_top_3_acc: 0.9909909963607788

Loss

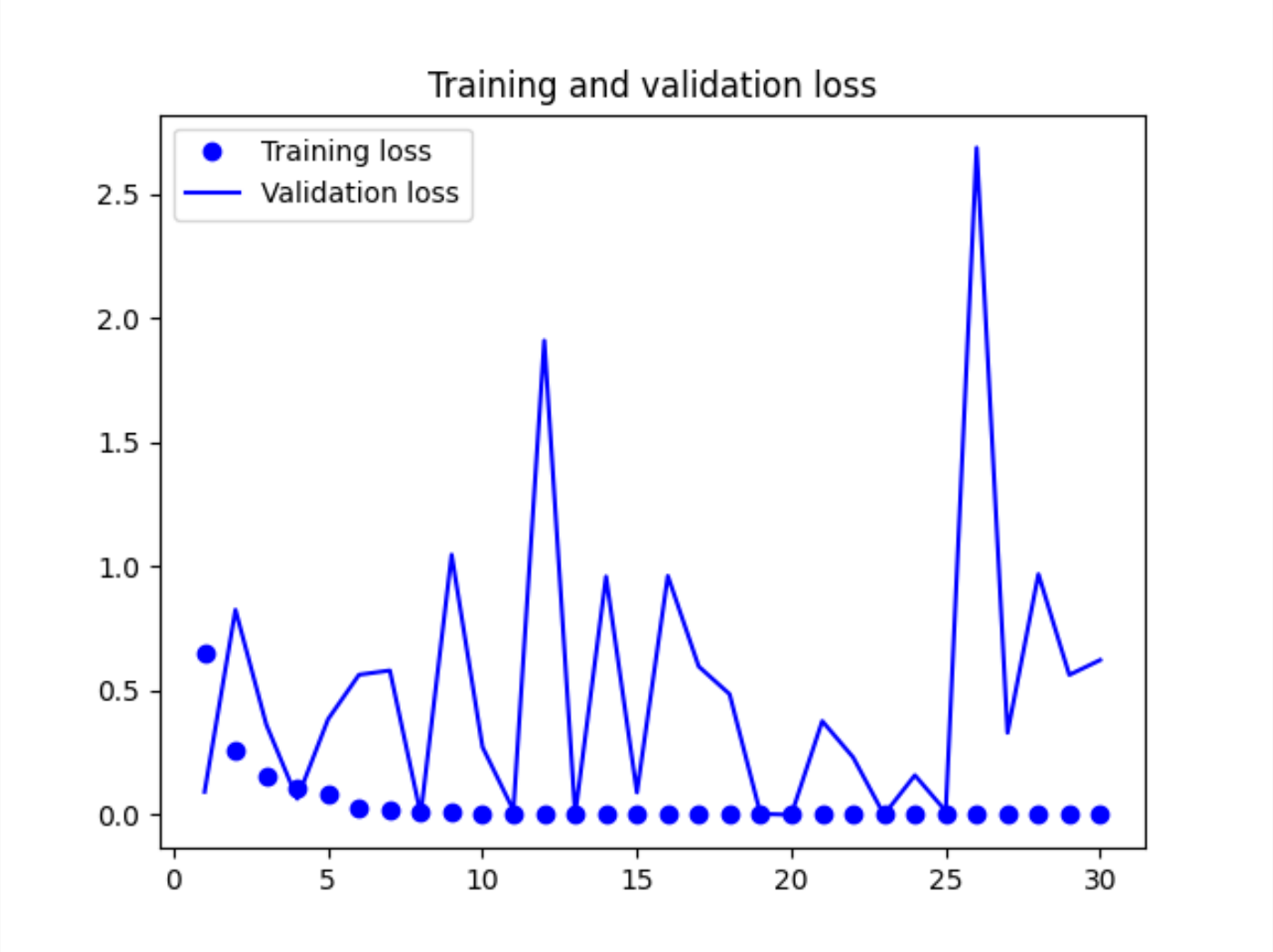


Figure 18: Mobile Net All Layers (10/batch): Loss

Categorical Accuracy

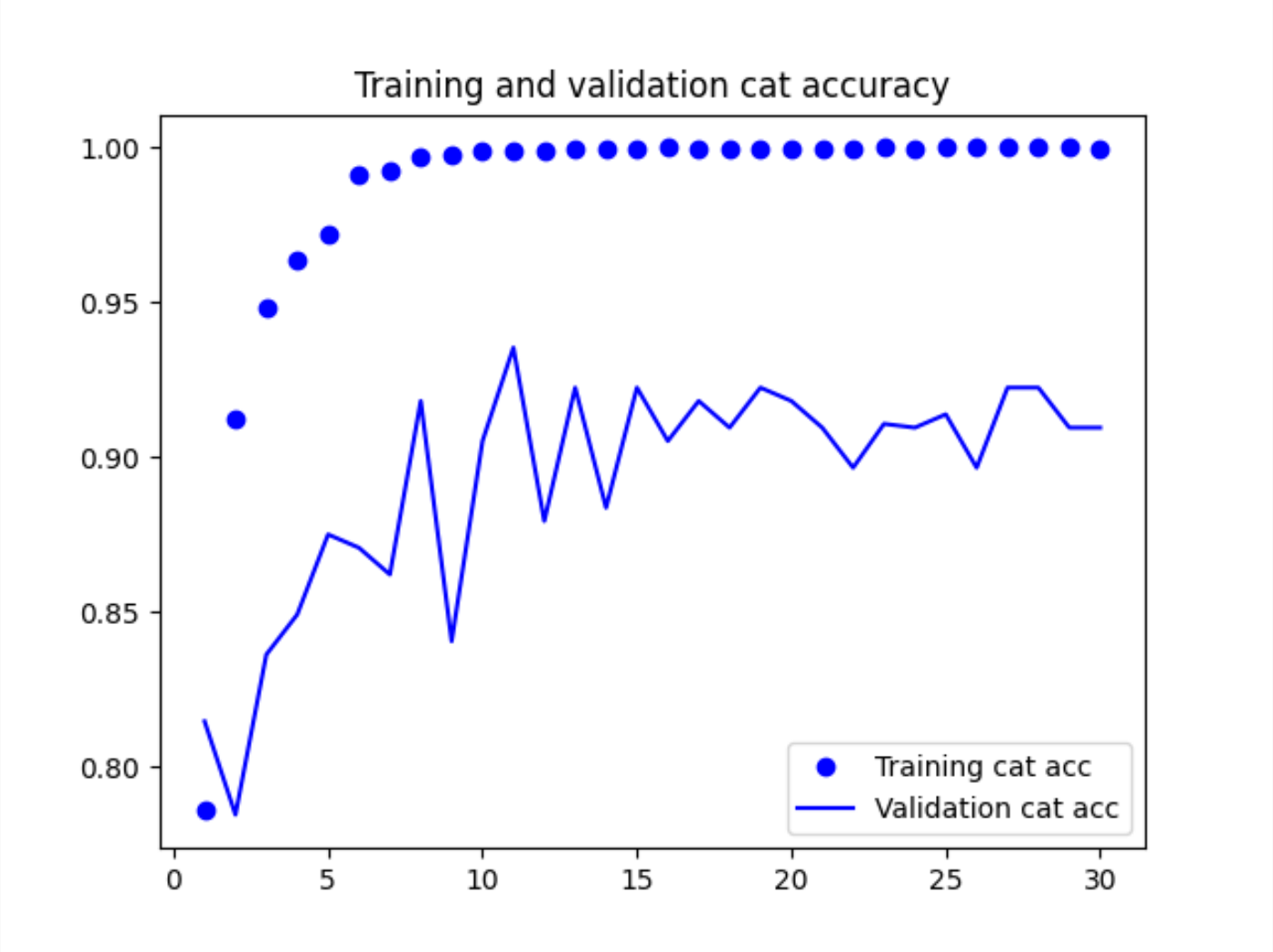


Figure 19: Mobile Net All Layers (10/batch): Cat. Acc

Top2

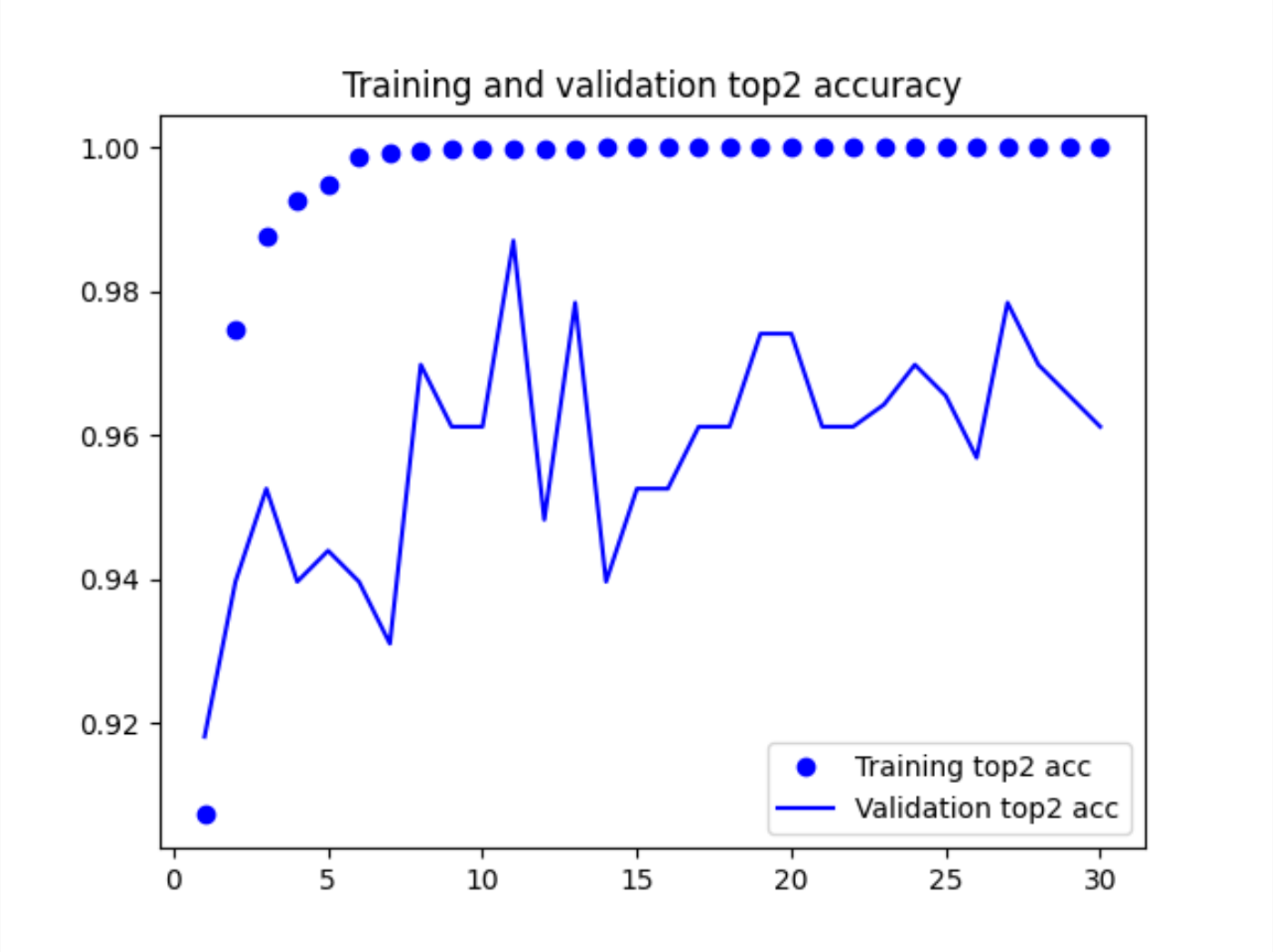


Figure 20: Mobile Net All Layers (10/batch): Top2

Top3

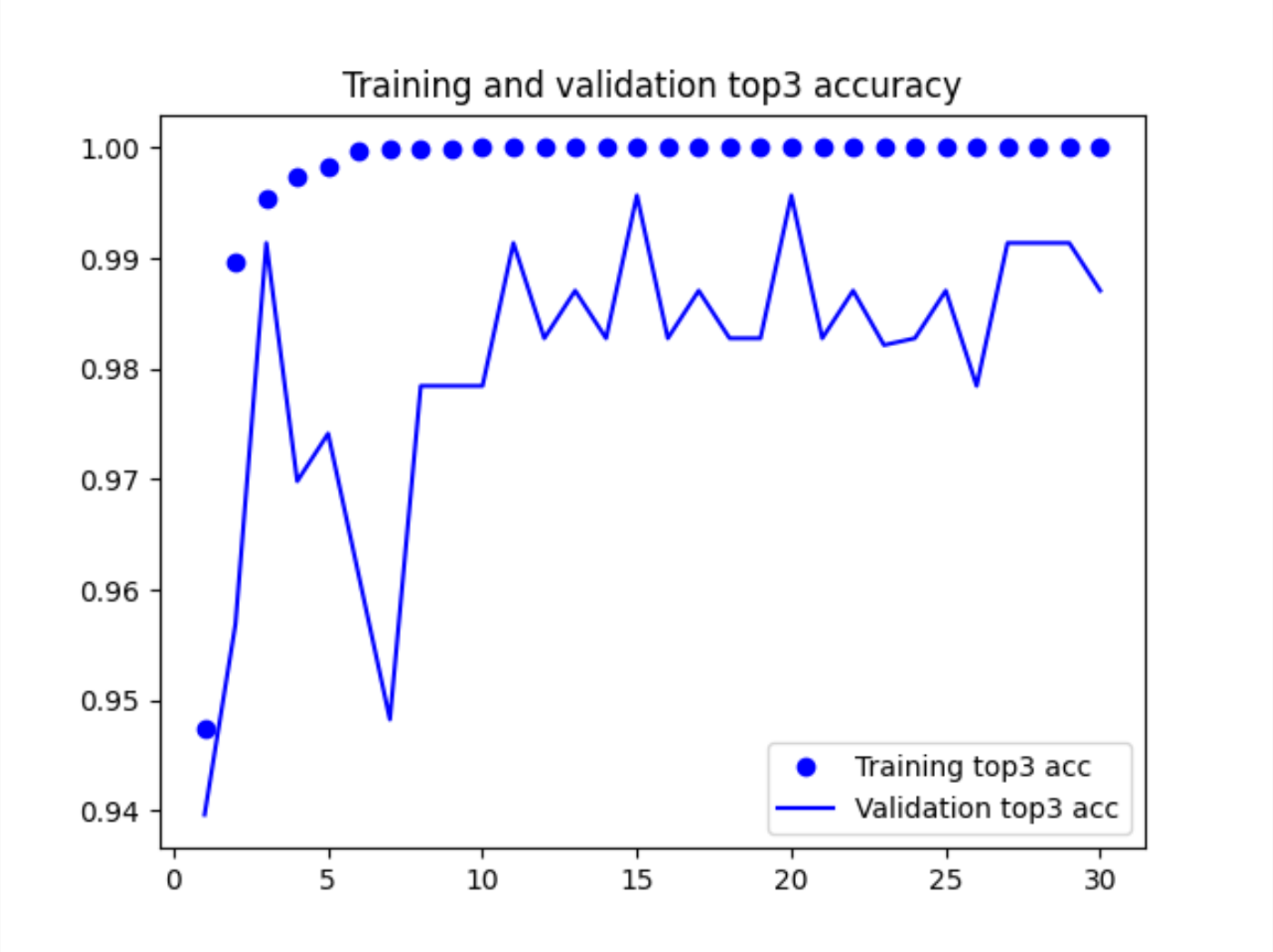
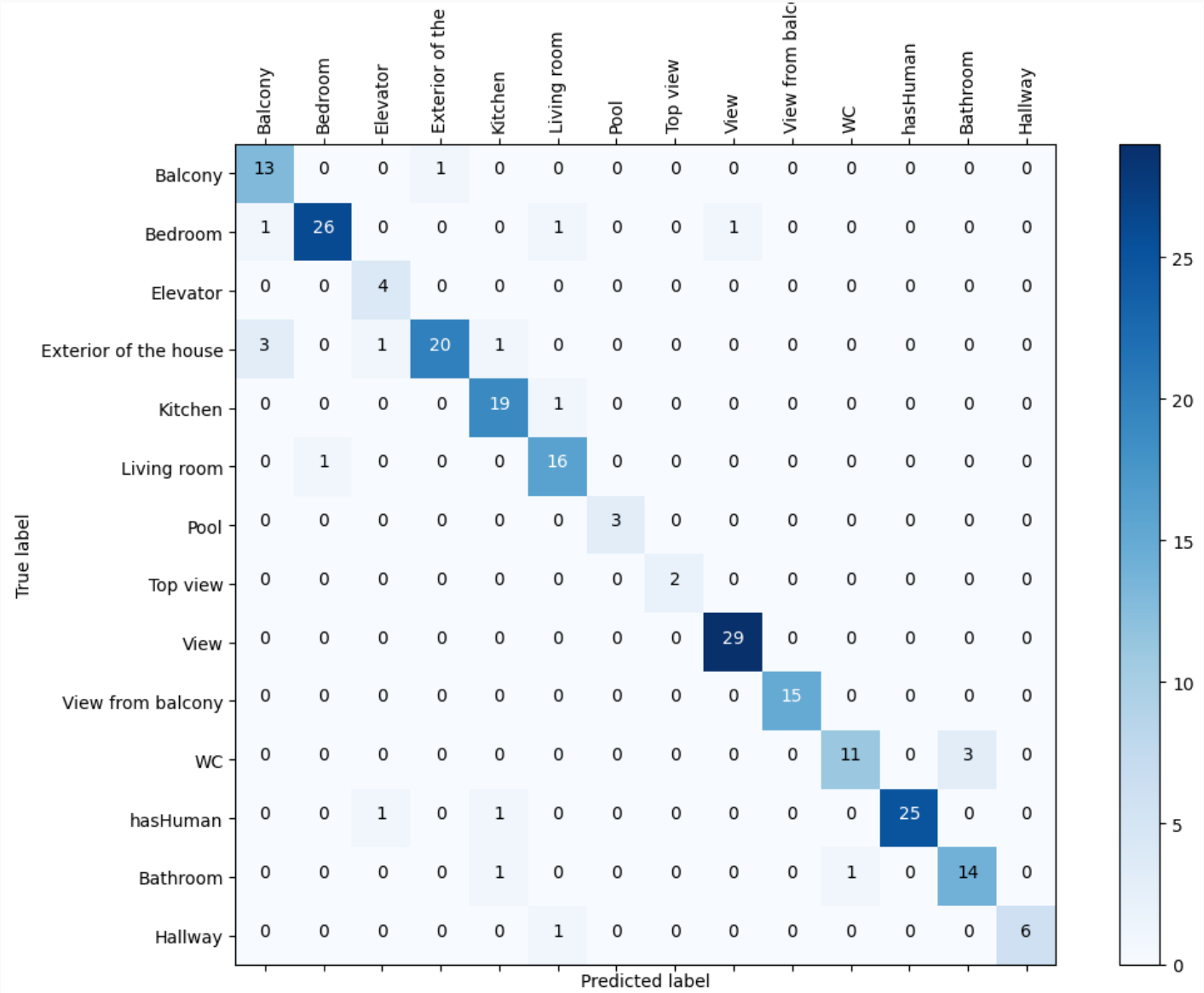


Figure 21: Mobile Net All Layers (10/batch): Top3

Confusion Matrix



Experiment 2

Mobile Net - Freeze Layers - batch size 10 ≈ <1h/epoch = 30h Total

Total params: 3,243,214 **Trainable**: 1,877,006 **Non-trainable**: 1,366,208

The last epoch

- val_loss: 0.0
- val_cat_acc: 0.883408069610595 7
- val_top_2_acc: 0.9596412777900696
- val_top_3_acc: 0.9775784611701965

the best epoch (val_top_3_acc)

- val_loss: 0.0
- val_cat_acc: 0.8923766613006592
- val_top_2_acc: 0.9596412777900696
- val_top_3_acc: 0.9820627570152283

Loss

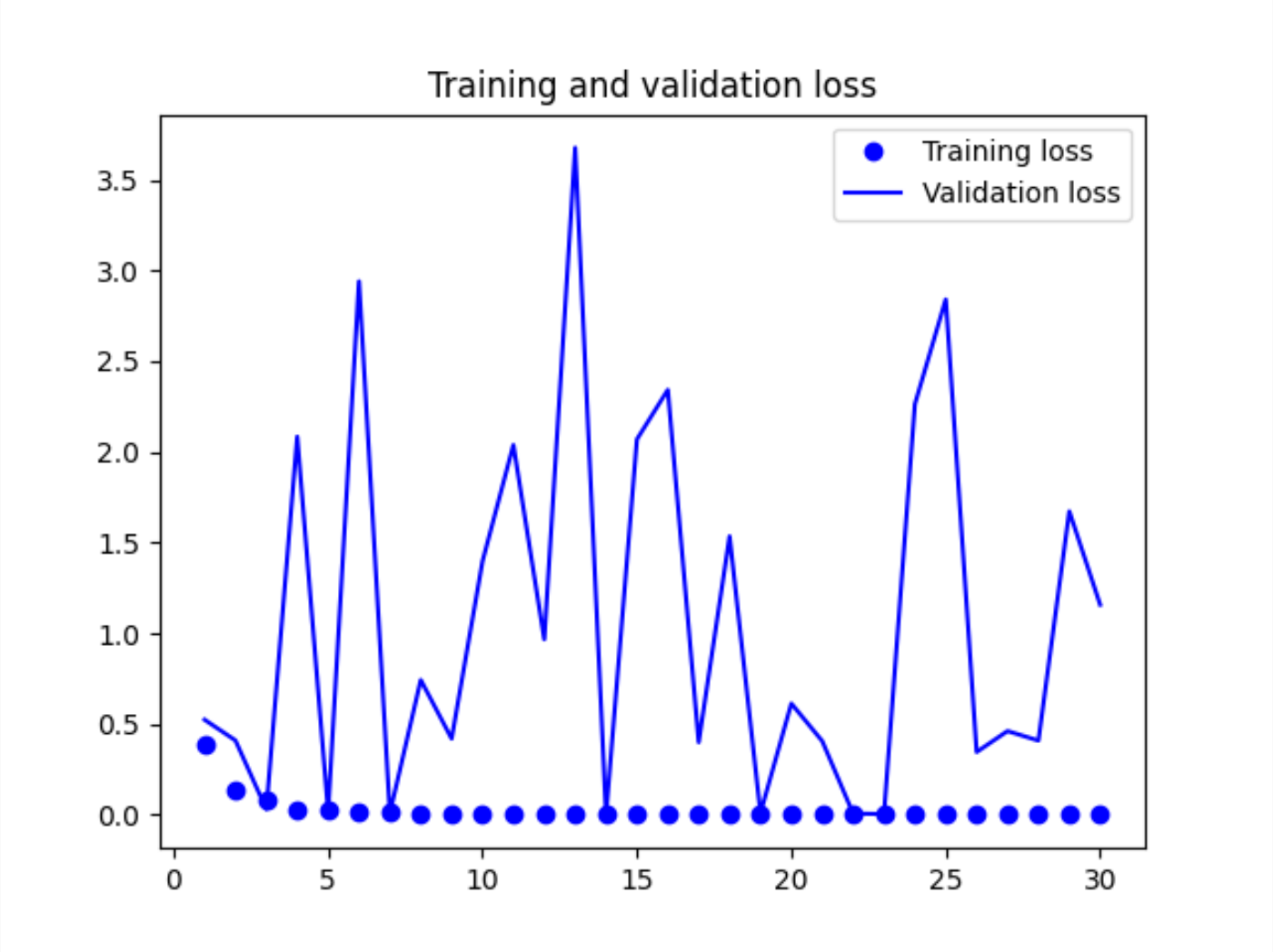


Figure 22: Mobile Net Freeze Layers (10/batch): Loss

Categorical Accuracy

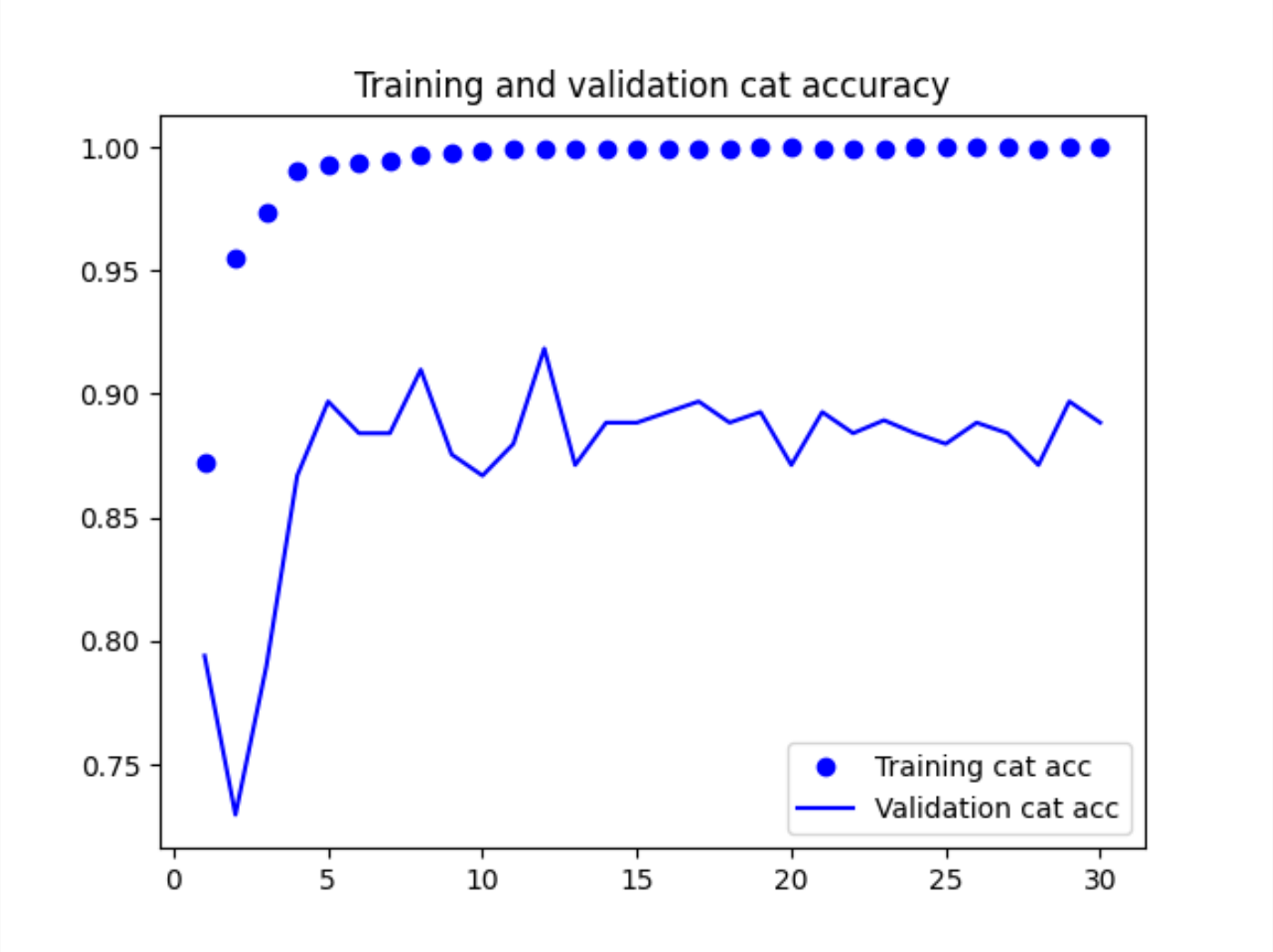


Figure 23: Mobile Net Freeze Layers (10/batch): Cat. Acc

Top2

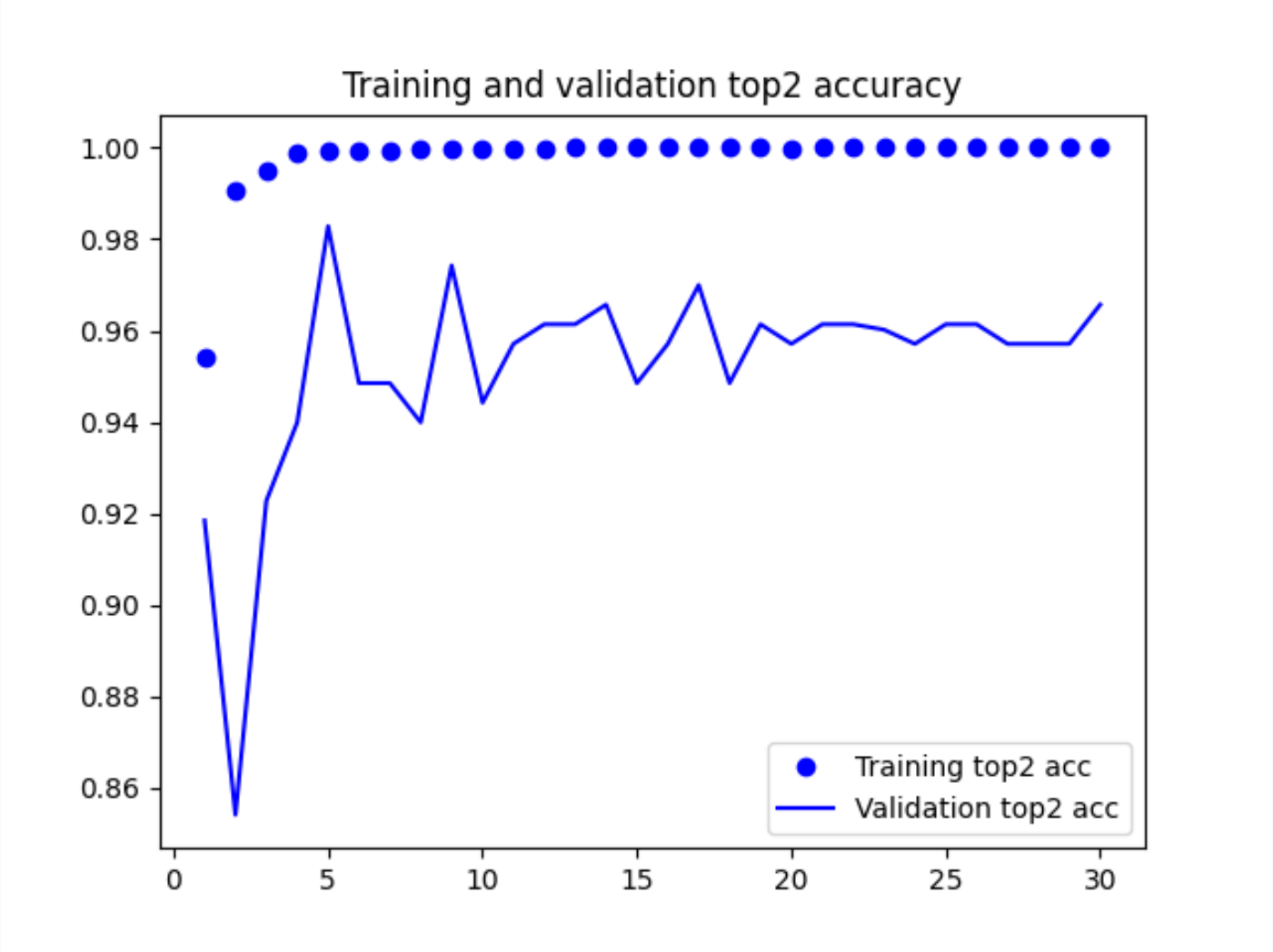


Figure 24: Mobile Net Freeze Layers (10/batch): Top2

Top3

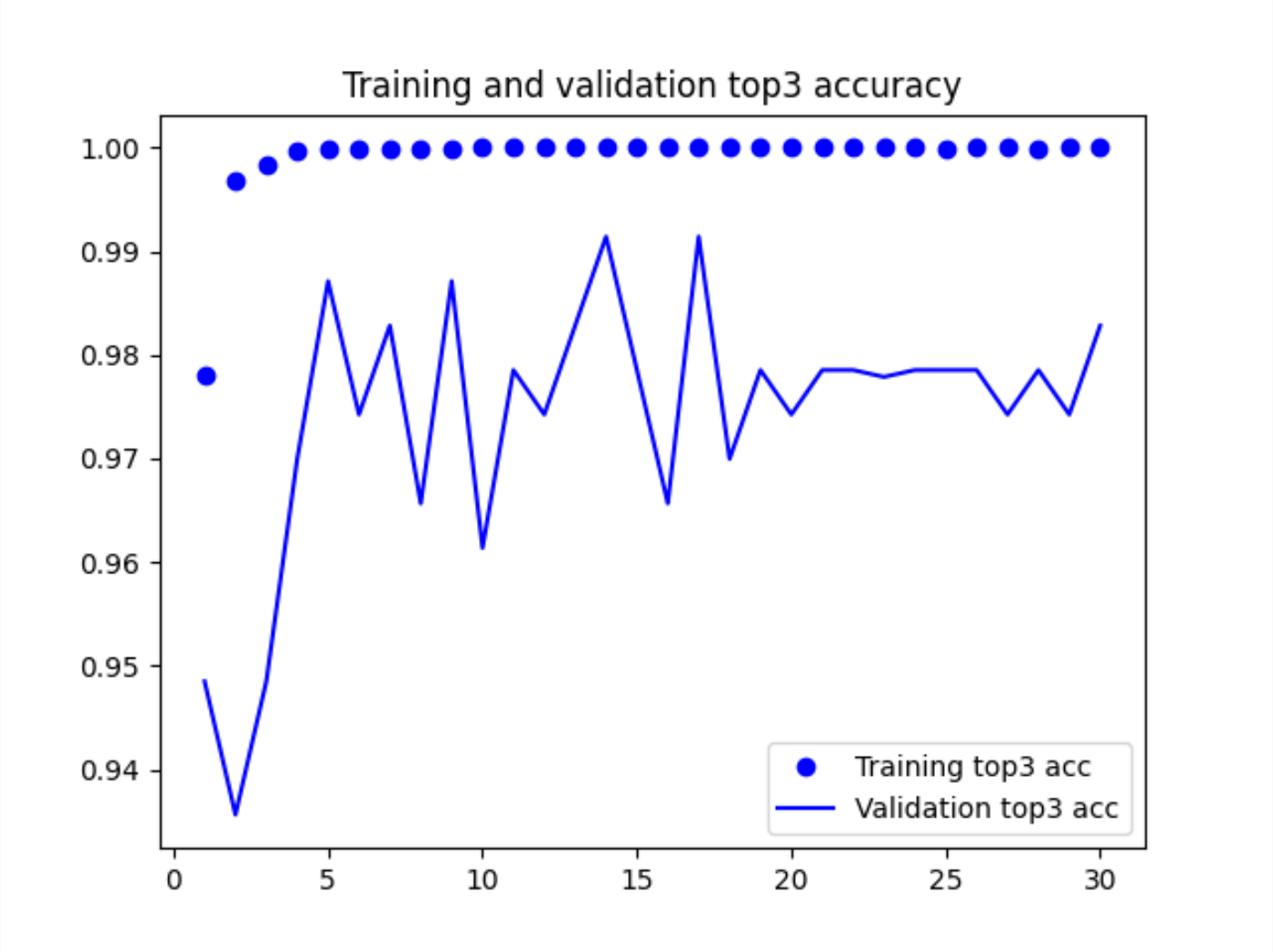
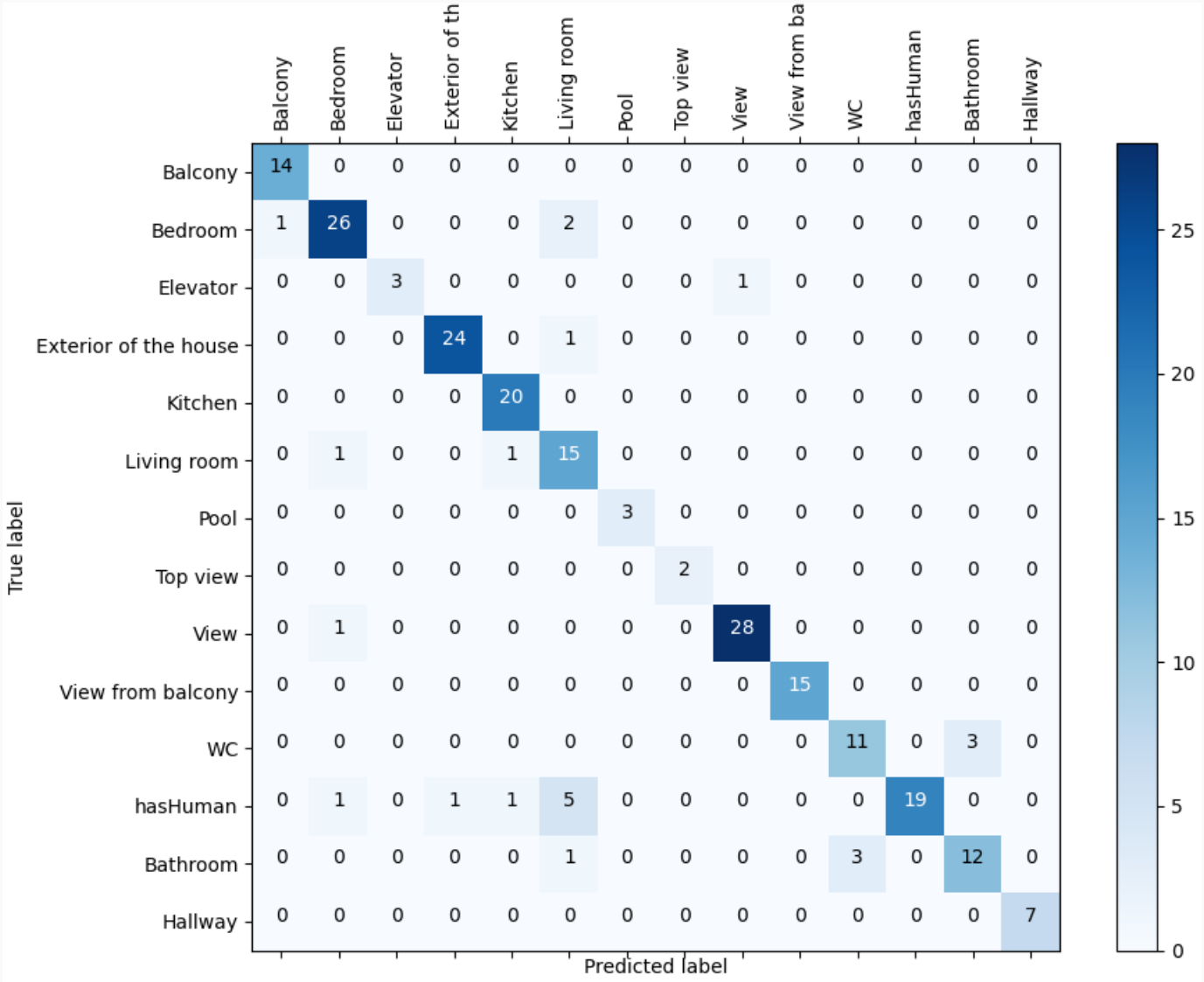


Figure 25: Mobile Net Freeze Layers (10/batch): Top3

Confusion Matrix



Precision

Precision								
class	MobileNet Freeze	MobileNet All	MobileNetV2 Freeze	MobileNetV2 All	NasNetMobile Freeze	NasNetMobile All	EfficientNetB0 Freeze	EfficientNetB0 All
Balcony	0.93	0.76	0.82	0.87	0.62	0.93	0.92	1.00
Bedroom	0.90	0.96	0.92	0.93	0.84	0.85	0.90	0.79
Elevator	1.00	0.67	0.60	1.00	1.00	1.00	1.00	0.75
Exterior of the house	0.96	0.95	1.00	0.96	0.92	0.88	0.91	0.95
Kitchen	0.91	0.86	0.76	0.91	0.65	0.90	0.62	0.90
Living room	0.62	0.84	0.75	0.84	0.44	0.70	0.86	0.71
Pool	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.75
Top view	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
View	0.97	0.97	0.93	0.91	0.91	1.00	0.90	0.96
View from balcony	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.93
WC	0.79	0.92	1.00	0.67	0.71	0.80	1.00	0.75
hasHuman	1.00	1.00	1.00	0.96	0.90	0.92	1.00	0.91
Bathroom	0.80	0.82	0.88	0.73	0.59	0.87	0.65	0.68
Hallway	1.00	1.00	0.78	0.83	0.42	0.86	0.88	0.88
accuracy								
macro avg	0.92	0.91	0.89	0.90	0.79	0.91	0.90	0.86
weighted avg	0.91	0.92	0.91	0.89	0.78	0.89	0.89	0.87

Precision: Given a class prediction from a classifier, how likely is it to be correct?

Recall

	Recall							
class	MobileNet Freeze	MobileNet All	MobileNetV2 Freeze	MobileNetV2 All	NasNetMobile Freeze	NasNetMobile All	EfficientNetB0 Freeze	EfficientNetB0 All
Balcony	1.00	0.93	1.00	0.93	0.93	0.93	0.86	1.00
Bedroom	0.90	0.90	0.76	0.90	0.72	0.79	0.90	0.93
Elevator	0.75	1.00	0.75	1.00	0.50	1.00	0.75	0.75
Exterior of the house	0.96	0.80	0.84	0.88	0.92	0.92	0.84	0.80
Kitchen	1.00	0.95	0.95	1.00	1.00	0.90	1.00	0.95
Living room	0.88	0.94	0.88	0.94	0.71	0.94	0.71	0.88
Pool	1.00	1.00	1.00	1.00	1.00	0.67	1.00	1.00
Top view	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
View	0.97	1.00	0.97	1.00	0.69	0.93	0.97	0.83
View from balcony	1.00	1.00	1.00	1.00	0.87	1.00	1.00	0.93
WC	0.79	0.79	0.86	0.71	0.36	0.86	0.50	0.64
hasHuman	0.70	0.93	0.81	0.81	0.33	0.85	0.81	0.74
Bathroom	0.75	0.88	0.94	0.69	0.81	0.81	0.81	0.81
Hallway	1.00	0.86	1.00	0.71	0.71	0.86	1.00	1.00
accuracy								
macro avg	0.91	0.93	0.91	0.90	0.75	0.89	0.87	0.88
weighted avg	0.90	0.91	0.89	0.89	0.73	0.89	0.86	0.86

Recall: Given a class, will the classifier be able to detect it?

F1-Score

	F1 Score							
class	MobileNet Freeze	MobileNet All	MobileNetV2 Freeze	MobileNetV2 All	NasNetMobile Freeze	NasNetMobile All	EfficientNet B0 Freeze	EfficientNet B0 All
Balcony	0.97	0.84	0.90	0.90	0.74	0.93	0.89	1.00
Bedroom	0.90	0.93	0.83	0.91	0.78	0.82	0.90	0.86
Elevator	0.86	0.80	0.67	1.00	0.67	1.00	0.86	0.75
Exterior of the house	0.96	0.87	0.91	0.92	0.92	0.90	0.87	0.87
Kitchen	0.95	0.90	0.84	0.95	0.78	0.90	0.77	0.93
Living room	0.73	0.89	0.81	0.89	0.55	0.80	0.77	0.79
Pool	1.00	1.00	1.00	1.00	1.00	0.80	1.00	0.86
Top view	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
View	0.97	0.98	0.95	0.95	0.78	0.96	0.93	0.89
View from balcony	1.00	1.00	1.00	1.00	0.93	1.00	1.00	0.93
WC	0.79	0.85	0.92	0.69	0.48	0.83	0.67	0.69
hasHuman	0.83	0.96	0.90	0.88	0.49	0.88	0.90	0.82
Bathroom	0.77	0.85	0.91	0.71	0.68	0.84	0.72	0.74
Hallway	1.00	0.92	0.88	0.77	0.53	0.86	0.93	0.93
accuracy	0.90	0.91	0.89	0.89	0.73	0.89	0.86	0.86
macro avg	0.91	0.91	0.89	0.90	0.74	0.89	0.87	0.86
weighted avg	0.90	0.92	0.89	0.89	0.72	0.89	0.86	0.86

F1 Score: The harmonic mean of the recall and precision. Essentially, it punishes extreme values.

Keras to Tensorflow.js conversion

Model conversion 1/3

The TensorFlow.js converter works with several different model formats:

- Saved-Model: This is the default format in which TensorFlow models are saved.
- Kerasmodel: Keras models are generally saved as an HDF5 file.
- TensorFlow Hub module: These are models that have been packaged for distribution on TensorFlow Hub, a platform for sharing and discovering models.

Model conversion 2/3

The TensorFlow.js converter has two components:

1. A command line utility that converts Keras and TensorFlow models for use inTensorFlow.js.

2. An API for loading and executing the model in the browser with TensorFlow.js.

Model conversion 3/3

A command line utility

```
$ tensorflowjs_converter --input_format=keras
                         /tmp/model.h5
                         /tmp/tfjs_model
```

Conversion with Python

```
model = load_model'(model.'h5,
                              custom_objects=dependencies)
tfjs.converters.save_keras_model(model, "tfjs")
```

Importing a Keras model into TensorFlow.js

TensorFlow.js Layers format is a directory **containing** a model.json file and a set of sharded weight files in binary format.

Load the model by providing the URL to the model.json file

```
import * as tf from '@tensorflow/tfjs';
var model = 'https://foo/model.json';
const model = await tf.loadLayersModel(model);
```

The loaded model can be immediately used to make a prediction

```
const example = tf.fromPixels(webcamElement);
const prediction = model.predict(example);
```

Supported features

TensorFlow.js Layers currently only supports Keras models using standard Keras constructs.

Models using unsupported ops or layers—e.g. custom layers, Lambda layers, custom losses, or custom metrics—cannot be automatically imported, because they depend on Python code that cannot be reliably translated into JavaScript.

That's the reason why we can not embed EfficinetNetB0 model into our application

Class Activation Mapping (Grad-CAM [6])

Introduction

Deep learning models are treated as "black box" methods, and many times we have no reasonable idea as to:

- Where the network is "looking" in the input image
- Which series of neurons activated in the forward-pass during inference/prediction
- How the network arrived at its final output

That raises an interesting question —

“how can you trust the decisions of a model if you cannot properly validate how it arrived there?”

Class Activation Mapping

Using Class Activation Mapping [13], we can visually validate where our network is looking, verifying that it is indeed looking at the correct patterns in the image and activating around those patterns.

If the network is not activating around the proper patterns/objects in the image, then we know:

- Our network hasn't properly learned the underlying patterns in our dataset
- Our training procedure needs to be revisited
- We may need to collect additional data
- And most importantly, our model is not ready for deployment.

Configuring the development environment

In order to use our Grad-CAM [6] implementation, we need to configure our system with a few software packages including:

- **TensorFlow** (2.0 recommended)
- **OpenCV** (library of Python bindings designed to solve computer vision problems)
- **imutils** (functions to make basic image processing)
- The **pyimagesearch module** (contains the Grad-CAM implementation inside the GradCAM class)
- Our **apply_gradcam.py** driver script accepts any of our sample/ images and applies our trained CNN model to (1) compute the Grad-CAM heatmap and (2) display the results in an OpenCV window.



Figure 26: Charlie Sheen as Charlie Harper (Two and a Half Men)

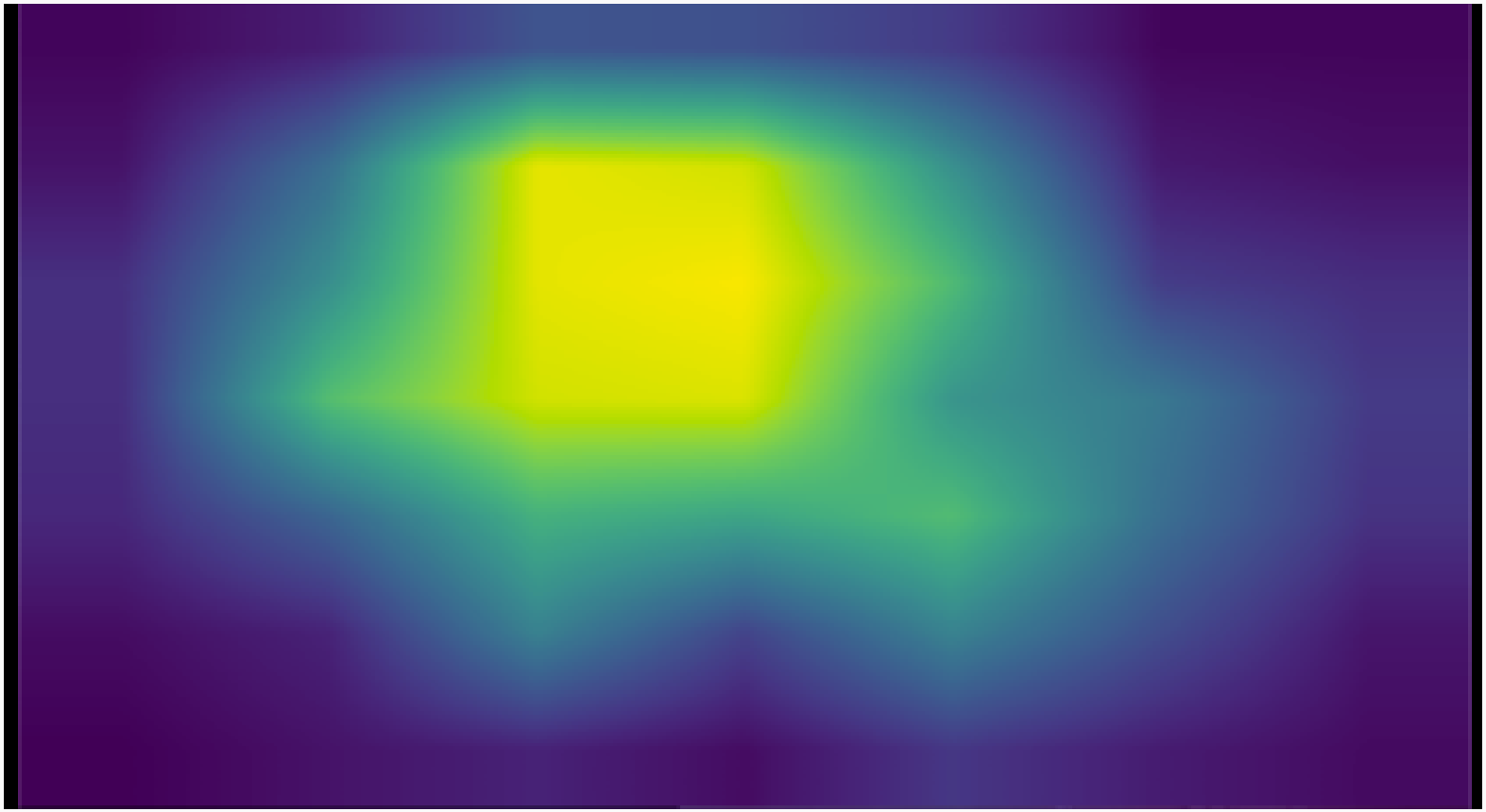
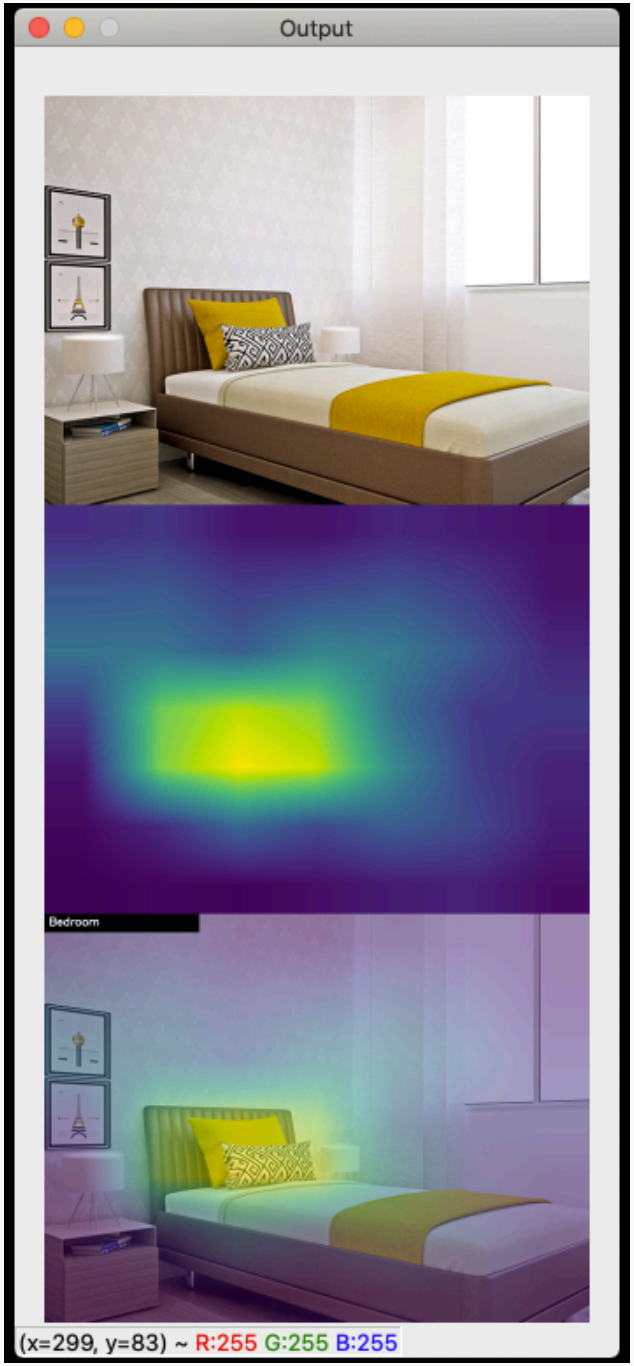


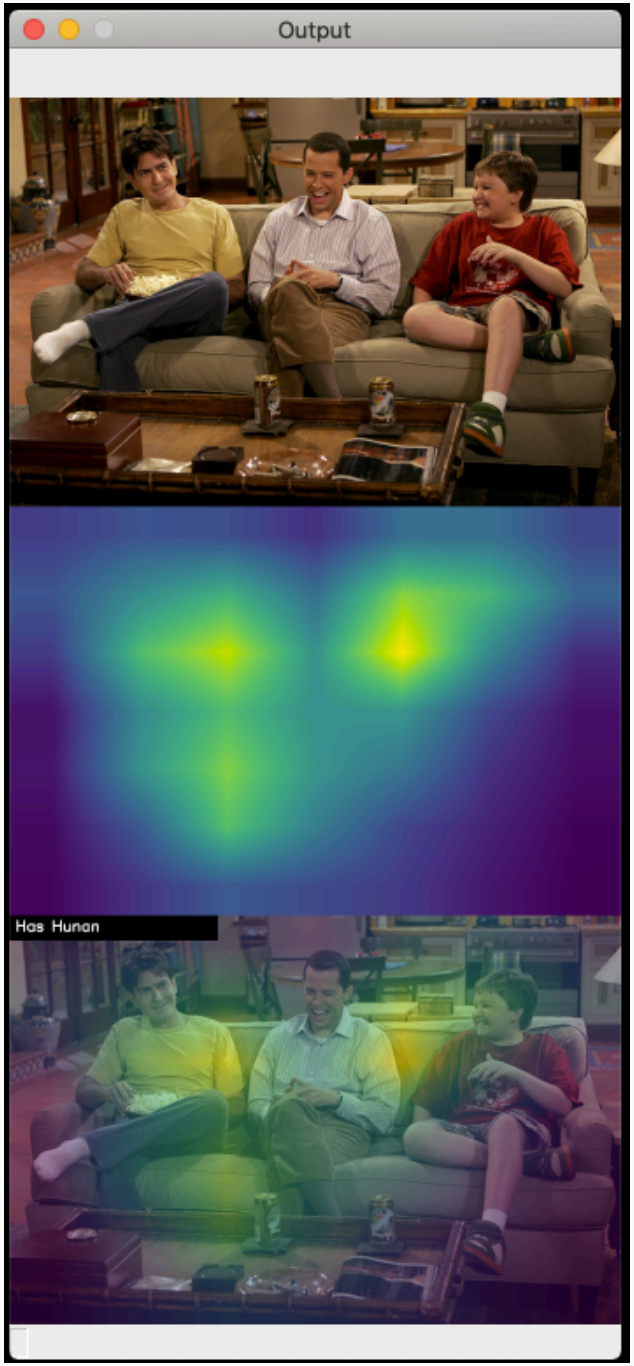
Figure 27: Activation Mapping

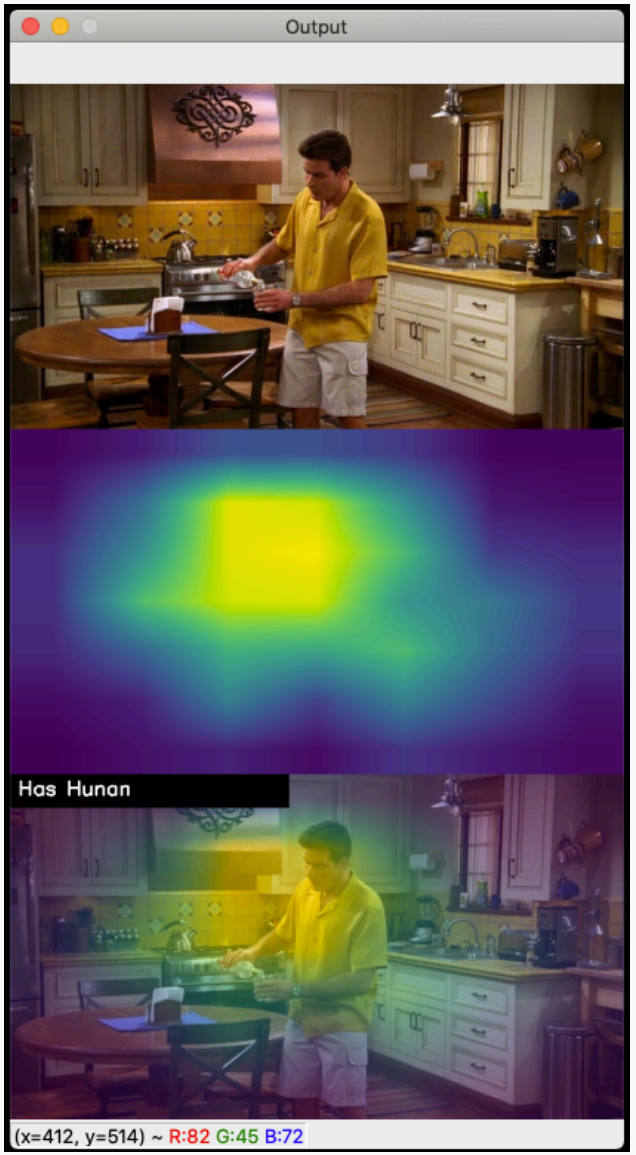


Figure 28: Output image


Output
kitchen
(x=344, y=38) ~ R:64 G:58 B:51


Output
Bedroom
(x=299, y=83) ~ R:255 G:255 B:255


Output
Has Human


Output
Has Human
(x=412, y=514) ~ R:82 G:45 B:72

Web application

Demo:

https://realstatus.gr/labs/thesis

Browser Support

- Google Chrome
- Opera

Thank you very much for your attention.
Questions?

References i


Y.-L. Boureau, J. Ponce, and Y. LeCun.
A theoretical analysis of feature pooling in visual recognition.
In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.


T. Guo, J. Dong, H. Li, and Y. Gao.
Simple convolutional neural network on image classification.
In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*, pages 721–724. IEEE, 2017.


K. He, X. Zhang, S. Ren, and J. Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

References ii



A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam.
Mobilenets: Efficient convolutional neural networks for mobile vision applications.
arXiv preprint arXiv:1704.04861, 2017.



A. Krizhevsky, I. Sutskever, and G. E. Hinton.
Imagenet classification with deep convolutional neural networks.
In *Advances in neural information processing systems*, pages 1097–1105, 2012.

References iii


R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra.
Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization.
CoRR, abs/1610.02391, 2016.


K. Simonyan and A. Zisserman.
Very deep convolutional networks for large-scale image recognition.
arXiv preprint arXiv:1409.1556, 2014.


N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov.
Dropout: a simple way to prevent neural networks from overfitting.
The journal of machine learning research, 15(1):1929–1958, 2014.

References iv


C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich.
Going deeper with convolutions.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.


M. Tan and Q. V. Le.
Efficientnet: Rethinking model scaling for convolutional neural networks.
CoRR, abs/1905.11946, 2019.


T. Wang, D. J. Wu, A. Coates, and A. Y. Ng.
End-to-end text recognition with convolutional neural networks.
In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3304–3308. IEEE, 2012.

References v


Wikipedia contributors.
Exif — Wikipedia, the free encyclopedia.
https://en.wikipedia.org/w/index.php?title=Exif&oldid=976324614, 2020.
[Online; accessed 14-September-2020].


B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba.
Learning deep features for discriminative localization.
CoRR, abs/1512.04150, 2015.


B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le.
Learning transferable architectures for scalable image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.