Project: Follow Me - Thomas Hatting

1. Fully Connected Layer vs. Fully Convolutional Network

In a fully connected layer, the 4D tensor, which represents data propagated through the network, is flattened to a 2D tensor. In consequence, a fully connected layer does not preserve spatial information. This type of network is good at object classification. For example, detecting ‘cats’ or ‘dogs’ within a large batch of animal pictures. However, this type of network is less effective at detecting objects embedding within other objects or the same object located in different corners of an image. An example of a fully connected layer is shown in figure 1.

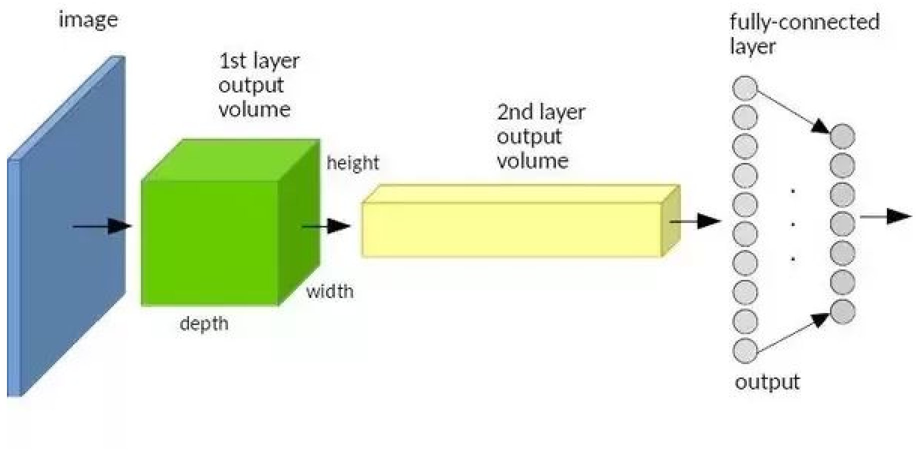

Figure 1: A fully connected layer (Quora 2016a)

To preserve spatial information, a fully convolutional network can be used instead. The network contains a series of encoding and decoding layers separated by a 1x1 convolution in the middle. The 1x1 convolution is a convolution with kernel size of 1 and stride of 1. The output of a 1x1 convolution is a 4D tensor (ie. output remains a 4D tensor instead of being flattened to a 2D tensor).

A fully convolutional network is well suited for segmenting objects in an image (semantic segmentation and scene understanding). In addition, another benefit is that the network does not care about the size of the input image unlike a network with a fully-connected layer. This is because the decoding layers in the network upscale the output picture to the same size as the input picture. An example of a fully convolutional network is shown in figure 2.

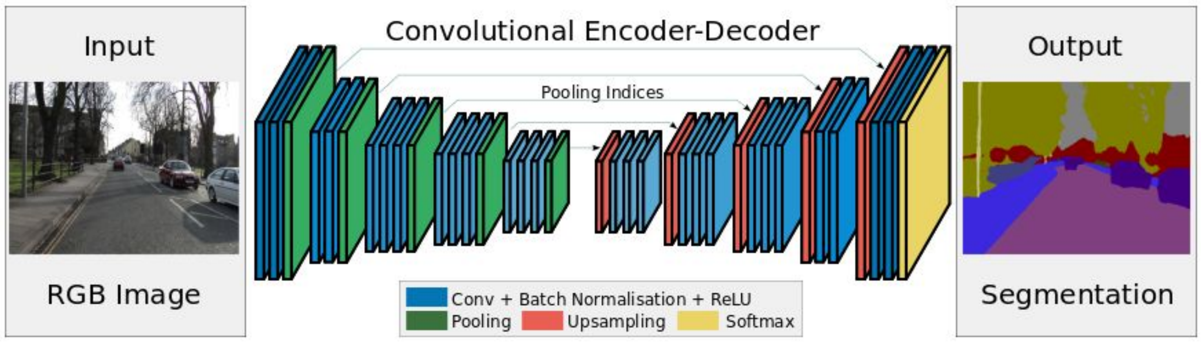

Figure 2: A fully convolutional network (Kendall et al. 2017)

2. The Network Architecture in the Project

The network architecture in this project is a fully convolutional network (FCN) consisting of a series of encoders and decoders separated by a 1x1 convolution. The network architecture is shown in figure 3.

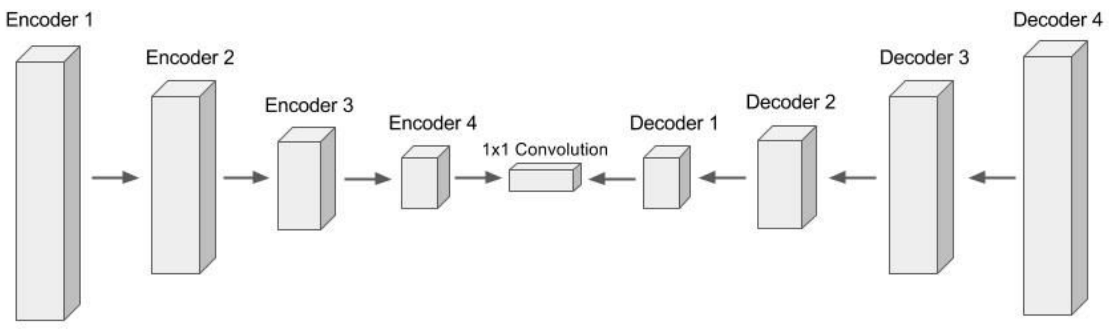

Figure 3: Network architecture with 4 encoders and 4 decoders separated by a 1x1 convolution

In all, 4 encoding layers are used to extract information.

Each encoder layer consists of two separable convolutions, with kernel_size = 3 and strides = 1, in order to extract details, followed by a third separable convolution with kernel_size = 3 and strides = 2 to downsample the image. Also, a batch normalization is performed after each convolution.

In the middle, a 1 x 1 convolution preserves the spatial information (the output is a 4D tensor rather than a 2D tensor as in a fully connected layer).

In all, 4 decoding layers are used to upsample to image back to the original size.

Each decoder layer carries out upsampling using the bilinear upsampling technique and also extract information from previous layers using the layer concatenation technique. This is followed by two separable convolutions to extract further details.

In the enclosed Jupyter Notebook, *model_training.ipynb*, each encoder layer is defined by function *encoder_block()* and each decoder layer by function *decoder_block()*.

Later on in the report, the configurable parameters are discussed and how they relate to the network. For example, the parameter, *num_epoch,* is the number of times the entire data set is propagated through the network architecture. Also, the parameter, *batch_size*, is number of samples propagated through the network architecture at a time. See section 4 below for more information.

3. The Various Techniques and Concepts in the Network Layers

In the fully convolutional network of this project, following techniques are used to carry out the semantic segmentation:

- separable convolutions
- batch normalization
- bilinear upsampling
- layer concatenation

The techniques are described below including reasons for using them.

3.1 Separable Convolutions

Separable convolutions consist of a convolution performed over each channel of an input layer followed by a 1x1 convolution taking the output channels from the previous step and combining them into an output layer. The benefit of the technique is to reduce the number of parameters and thus help to improve performance. Also, the technique reduces overfitting to some extent as there are a lower number of parameters (Pröve 2017).

In the enclosed Jupyter Notebook, *model_training.ipynb*, the separable convolutions are contained in the function *separable_conv2d_batchnorm()*.

3.2 Batch Normalization

Batch normalization is based on the idea that, instead of just normalizing the inputs to the network, we normalize the input to each layer within the network. It is called "batch" normalization because each layer's inputs are normalized using the mean and variance of the values in the current batch (Ioffe and Szegedy 2015).

The advantage of batch normalization:

a) **Faster training times** – Each iteration is slower due to extra calculations. However, the system converges more quickly, therefore training times should be faster in the long run.

b) **Learning rates can be set higher** – Batch normalization allows higher learning rates, which increases the speed at which networks train.

c) **Network simplification** – It is easier to build and faster to train a deeper neural network using batch normalization.

d) **Adds regularization** – Batch normalization adds some noise to the network. At times, batch normalization has been shown to work as well as the dropout technique (where portions of data flowing through the network are dropped).

In the enclosed Jupyter Notebook, *model_training.ipynb*, batch normalization is contained in the function *separable_conv2d_batchnorm()*.

3.3 Bilinear Upsampling and Layer Concatenation

Transposed convolutions are often used to upsample the input and form a core part of fully convolutional networks (Pröve 2017). However, there are a number of ways to carry out upsampling. In this project, bilinear upsampling is used instead.

Bilinear upsampling is a resampling technique that uses the weighted average of the four nearest known pixels, located diagonally to a given pixel, in order to estimate the intensity of a new pixel.

The benefit of bilinear upsampling is that the technique speeds up performance. However, the technique does not contribute as a learnable layer like the transposed convolutions. The method is prone to lose some details.

In convolutional networks, skip connections are used to obtain information details from previous layers during the decoding process. One way to do this is through layer addition. Another and somewhat simpler technique is to use layer concatenation. In this technique, the upsampled layer and a layer, with more spatial information than the upsampled one, are concatenated.

The benefit of using layer concatenation vis-à-vis simply adding the layers is that the depth of the layers do not have to match up. This allows for a simpler implementation. However, layer concatenation may not provide finer spatial details. It is therefore useful to add a few regular or separable convolution layers afterwards to obtain more details.

The above mentioned techniques are summarized in figure 4 below:

	Technique	**Explanation**	**Benefits**
1.	Separable Convolutions	Separable convolutions comprise of a convolution performed over each channel of an input layer and followed by a 1x1 convolution taking the output channels from the previous step and combining them into an output layer.	-Reduction in number of parameters and can thus help to improve performance - Reduction in overfitting as there are a lower number of parameters
2.	Batch Normalization	Batch normalization is based on the idea that, instead of just normalizing the inputs to the network, we normalize the input to each layer within the network.	- Faster training times - Learning rates set higher - Network simplification - Adds regularization
3.	Bilinear Upsampling	Bilinear upsampling is a resampling technique that utilizes the weighted average of four nearest known pixels, located diagonally to a given pixel, in order to estimate the intensity value of a new pixel.	- Speeds up performance However, the method does not contribute as a learnable layer just like transposed convolutions and is prone to lose some details
4.	Layer Concatenation	In layer concatenation, the upsampled layer and a layer, with more spatial information than the upsampled one, are concatenated.	- Depth of the layers do not have to match up However, layer concatenation may not provide finer spatial details.

Figure 4: Techniques used in this network

In the enclosed Jupyter Notebook, *model_training.ipynb*, bilinear upsampling and layer concatenation are defined by functions *bilinear_upsample()* and *layers.concatenate()* respectively.

4. The Parameters Chosen for the Neural Network

Prior to carrying out a simulation, a number of parameters should be configured. These are called hyperparameters and describe high-level properties such as complexity and learning rate (Quora 2016b). In this project, following hyperparameters have been chosen:

- **learning_rate** - affects the speed at which the algorithm reaches optimal values
- **batch_size** - the number of samples propagated through the network at a time
- **num_epoch** - number of times the entire dataset gets propagated through the network
- **steps_per_epoch** - number of batches of training images that go through the network in each epoch
- **validation_steps** - number of batches of validation images that go through the network in each epoch
- **workers** - maximum number of processes to spin up

In the subsections below, the results of the simulations are described including choice of hyperparameters.

4.1 Simulation Using the Pre-included Training and Validation Data

The simulation was first run on my laptop with following hyperparameters in a 3-layer network (3 encoders + 3 decoders):

learning_rate = 0.05
batch_size = 32
num_epoch = 2
steps_per_epoch = 200
validation_steps = 50
workers = 2

The initial value of learning rate was estimated to 0.05, based on values used in the lab on deep neural networks done previously in the deep learning part of the course. The value is relatively low. However, it is an advantage to use a lower value because, even though the simulation might take a bit longer, the simulation can reach a lower optimal value.

This resulted in a final grade score of 16.4 %.

Consequently, the network architecture was modified to contain 4 layers (4 encoders + 4 decoders) to extract more information. Using the same parameters as above, the final grade score came to 21.2%. The training curves for this simulation are shown in figure 5.

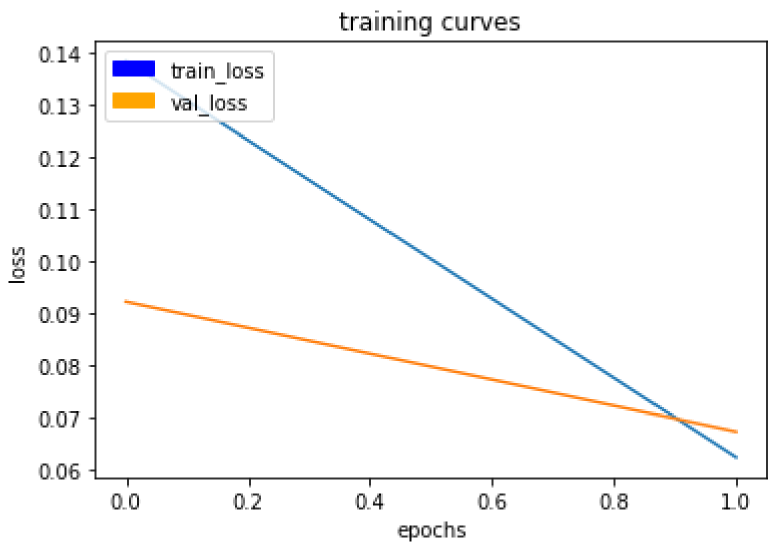

Figure 5: Initial training and validation loss for the 4-layer network architecture (learning_rate=0.05, batch_size=32, num_epoch = 2, steps_per_epoch = 200, validation_steps = 50, workers = 2)

It can be seen that loss for training and validation decreases to 0.065 and 0.072 respectively.

The simulations were run on my laptop and took between 2-4 hours each, depending on the number of layers. As the simulations took a relative long time, it was decided to run them instead on a “p2.xlarge” instance in the AWS cloud. In particular, to see what happens if the batch size was increased to 64 images and the number of epochs increased to 5 or higher. Following parameters were used during the simulation in the cloud:

learning_rate = 0.05

batch_size = 64

num_epoch = 10

steps_per_epoch = 200

validation_steps = 50

workers = 2

This resulted in a final grade score of about 30%.

The filter sizes were then changed in the network. Until now, filter size had been increased from layer to layer (32, 64, 128 etc.). Instead, the filter size was set to 32 for each layer. Also, in each encoder, two extra separable convolutions were added prior to the downsampling. This was done to help extract spatial details (Kendall et al. 2017).

In this simulation, following data was used:

learning_rate = 0.01

batch_size = 32

num_epoch = 12

steps_per_epoch = 400

validation_steps = 100

workers = 2

The batch size was changed from 64 to 32. It did not seem to make a big difference whether or not the size was set to 32 or 64. So it was decided to go back to 32 in order to speed up the simulation. Furthermore, the learning rate was lowered from 0.05 to 0.01, as lowering the learning rate can help the system to reach a lower minimum.

The parameters steps_per_epoch and validation_steps were doubled to include a larger number of images. Moreover, the parameter num_epoch was increased slightly from 10 to 12. Finally, the training data was changed to include a new set released by Udacity on 6th of October, which had been updated with more images. The training curves are shown in figure 6. The loss for training and validation converged to a value of about 0.05.

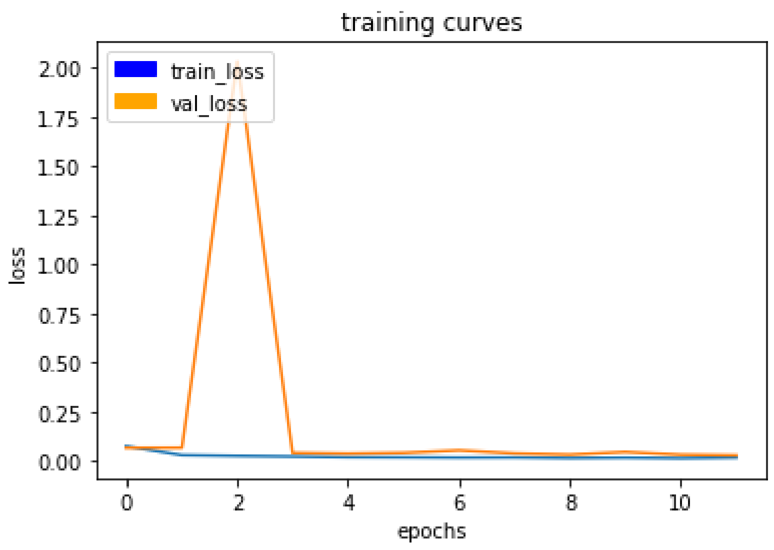

Figure 6: Training and validation loss for the 4-layer network architecture (4+4) with extra separable convolutions in encoders & decoders and calculated in AWS (learning_rate=0.01, batch_size=32, num_epoch = 12, steps_per_epoch = 400, validation_steps = 100, workers = 2)

The simulation resulted in a final grade score of 46.5%.

In the simulation, a number of issues were identified:

There seemed to be a problem with the score: *quad on patrol and the target is not visible*. The score for this particular metric was very low (and had been so in previous simulations). Also, there was quite a large number of false negatives for score: *detect the target from far away*.

See below:

```
# Scores for while the quad is following behind the target.
number of validation samples intersection over the union evaluated on 542
average intersection over union for background is 0.9950205880051157
average intersection over union for other people is 0.39982536283100467
average intersection over union for the hero is 0.9105342824888485
number true positives: 539, number false positives: 0, number false negatives: 0

# Scores for images while the quad is on patrol and the target is not visible
number of validation samples intersection over the union evaluated on 270
average intersection over union for background is 0.98990758533291
average intersection over union for other people is 0.8192873170445109
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 40, number false negatives: 0
```

```
# Scores measures how well the neural network can detect the target from far away
number of validation samples intersection over the union evaluated on 322
average intersection over union for background is 0.9968897307729778
average intersection over union for other people is 0.49855819081124514
average intersection over union for the hero is 0.28139146198309783
number true positives: 149, number false positives: 1, number false negatives: 152

# And the final grade score is
0.465405739045
```

In the next section, an attempt were made to deal with these issues using own generated training and validation data.

4.2 Simulation Using Own Generated Training and Validation Data

The simulation was also run with own training and validation data. Previously, it was shown that the pre-included training and validation data led to low scores for the "quad on patrol and the target not visible".

The data was generated in the following way:

a) In the Unity simulator, a cluster of spawn points was created followed by a large number of hero path points zigzagging in and out of this cluster. The recording was turned on and spawning activated, and the drone was manually manoeuvred to generate a large number of images of the target in the crowd with lots of people.

b) During the same recording session, the target was moved to an area with no people and made to zigzag along a path while the drone was manually manoeuvred to generate images of the target. Again the aim was to generate lots of images of the target.

c) Finally during the same recording session, the drone was sent on a standard patrol to generate a number of images of the environment at different angles without the target being visible.

The steps a) to c) were repeated for both the training and validation data. Also, the sample evaluation data that was pre-included in the simulation was used (ie. same evaluation data as in previous simulations). In this simulation, following data was used:

learning_rate = 0.01
batch_size = 32
num_epoch = 12
steps_per_epoch = 400
validation_steps = 100
workers = 2

The loss curves are shown in figure 7:

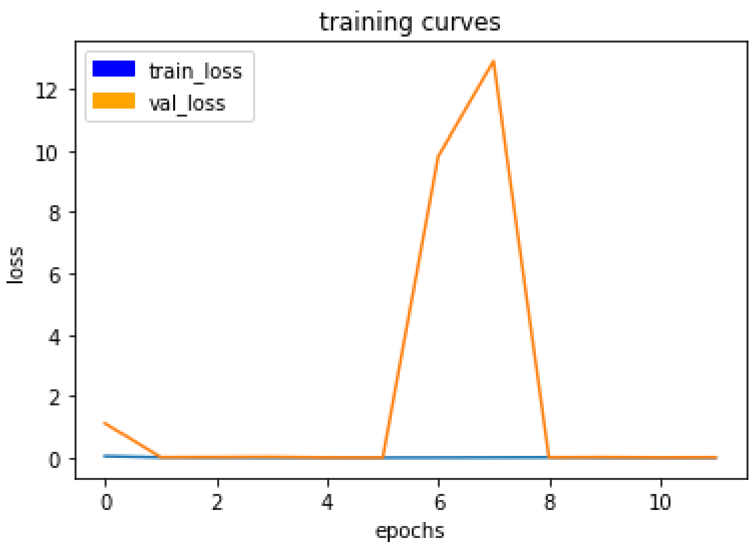

Figure 7: Training and validation loss curves for the 4-layer network architecture (4+4) with extra separable convolutions in encoders and decoders. The simulation was carried out in AWS with own training and validation data (learning_rate=0.01, batch_size=32, num_epoch = 12, steps_per_epoch = 400, validation_steps = 100, workers = 2)

The scores are given below:

```
# Scores for while the quad is following behind the target.
number of validation samples intersection over the union evaluated on 542
average intersection over union for background is 0.9807907212721322
average intersection over union for other people is 0.06089699895118974
average intersection over union for the hero is 0.7764673548694798
number true positives: 539, number false positives: 1, number false negatives: 0

# Scores for images while the quad is on patrol and the target is not visible
number of validation samples intersection over the union evaluated on 270
average intersection over union for background is 0.9643282802495906
average intersection over union for other people is 0.11226134390116015
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 183, number false negatives: 0

# Score measures how well the neural network can detect the target from far away
number of validation samples intersection over the union evaluated on 322
average intersection over union for background is 0.9904349522979717
average intersection over union for other people is 0.051106374745951935
average intersection over union for the hero is 0.14915441475197405
number true positives: 145, number false positives: 8, number false negatives: 156

# And the final grade score is
0.306746749235
```

The final grade score was about 30%. As can be seen, the score for "*quad on patrol and target not visible*" was still inadequate.

There could be a number of reasons for this. The number of generated training and validation images might not have been enough. Perhaps, more variation in angles at which images are taken of the target should be included. Also, more footage from the standard patrol could be included. Finally, further experimentation could be made with the network architecture.

5. Final Results and Limitations to the Neural Network

All in all, the best result was achieved using the pre-included training and validation data in the project using following hyperparameters: learning_rate=0.01, batch_size=32, num_epoch = 12, steps_per_epoch = 400, validation_steps = 100, workers = 2. **The final grade score for this simulation was 46.5%**. In consequence, I have submitted the Jupyter Notebook and the model and weights file for this particular simulation.

This particular final grade score was based on following 3 scores:

Scores while the quad is following behind the target:
number true positives: 539
number false positives: 0
number false negatives: 0

Scores for images while the quad is on patrol and the target is not visible:
number true positives: 0
number false positives: 40
number false negatives: 0

Scores for how well the neural network can detect the target from far away
number true positives: 149
number false positives: 1
number false negatives: 152

It can be seen that the chosen model and data appear to show limitations with regards to score: <u>while the quad is on patrol and the target is not visible</u>.

The model and data would probably not work well for following another object (such as a dog, cat, car, etc.). It would be necessary to generate an entirely new set of training and validation images based on the new object. For example, a car has a very different shape compared to a human. It would be necessary to generate a large number of images while patrolling above the car at different angles. Also, it would be necessary to generate images of the car amongst other cars in the traffic. Furthermore, it would be necessary to include images of the car viewed from far away. This new data should be propagated through the network architecture (described in section 2) with new adjustments to the hyperparameters.

6. Future Enhancements

The accuracy of the neural network may be improved by adding more layers to the architecture. For example, max pooling could be added to the end of each encoding layer. The benefit of max pooling is to reduce the size of the input to the next layer and allow the network to focus on the most important elements. This might improve the accuracy (Kendall et al. 2017).

Also, the network achieved a low final grade score (about 30%) for own generated training and validation data. More variation in the training data could be generated especially to deal with the low score for the "*quad being on patrol and the target not visible"*.

The network could be used to follow another type of object (dog, cat, car etc.). It would be necessary to include these objects somehow in the Unity simulator environment and generate new training and validation data.

Another enhancements could also be to use the network for street address identification. For example, the drone could be used to deliver a parcel to a particular building in the simulator environment (Kendall et al. 2017).

References

Ioffe, S. and Szegedy, C (2015) "Batch Normalization: Accelerating Deep Network Training by Reducing Covariate Shift". Available online: https://arxiv.org/pdf/1502.03167.pdf

Kendall, A., Badrinarayanan,V. and Cipolla,R. (2017) "SegNet: A Deep Convolutional Encoder Decoder Architecture for Robust Semantic Pixel-Wise Labelling". University of Cambridge. Available online: http://mi.eng.cam.ac.uk/projects/segnet/

Nielsen, M. (2017) "Neural Networks and Deep Learning". Available online: http://neuralnetworksanddeeplearning.com/

Ozhiganov, I. (2016) "Object Detection Using Fully Convolutional Neural Networks". Available online: http://rnd.azoft.com/object-detection-fully-convolutional-neural-networks/

Pröve, P. (2017) “An Introduction to Different Types of Convolutions in Deep Learning” Available online:
https://medium.com/towards-data-science/types-of-convolutions-in-deep-learning-717013397f4d

Quora (2016a) “How is a convolutional neural network able to learn invariant features?”. Available online:
https://www.quora.com/How-is-a-convolutional-neural-network-able-to-learn-invariant-features

Quora (2016b) “What are hyperparameters in machine learning?”. Available online
https://www.quora.com/What-are-hyperparameters-in-machine-learning