Advanced Technologies: Level Creation through Satellite Imagery Utilising a CNN

Alexander Stephen Allman
16015338

University of the West of England

April 29, 2020

This project has the capability to take a satellite image, break it down into sections and classify the sections. The results of the classification are then outputted to JSON and are used to construct a 3D level in the popular engine Unity.

1 Introduction

As deep learning has evolved over the recent years, academics have explored what problems deep learning can be used to solve. These can vary drastically from automatic caption generation for news images (Yansong Fen, 2012) to translating text real time (Good, 2015). This project will specifically focus on classifying satellite images and, from the result of the classification, build a 3D representation of the image within the game engine Unity. To help do this Tensorflow and Keras will be leveraged to build and train the CNN model used.

2 Related Work

Image classification in deep learning has been a common use case. Whether the focus be at more general improvements to image classification such as the exploration of image degradation and its effects on classification (Yanting Pei, 2019); or related to a more specific problem such as trying new types of neural networks to help ease the shortcomings of previously standard methods whilst classifying satellite imagery (Emmanuel Maggiori, 2017; M.A Popov, 2017). Alex Net is one of the original pioneers of now frequently adopted methodology for CNN training and architecture. Alex Net was one the first to leverage multiple GPU's for training the CNN, utilising ReLU activation functions and using method like Dropout to reduce overfitting (Alex Krizhevsky, 2017). The model used within this project will take great inspiration from Alex Net in its architectural structure.

3 Theory

3.1 CNN

A Convolution neural network is a specialised type of deep neural network used for Computer Vision or imagery analysis. This network primarily revolves around taking different types of small square matrices, often referred to as kernels and striding them across the input image matrix to output a resultant matrix (Michelucci, 2019). This method of striding a kernel across an image is known as a convolution and is where the networks namesake is derived from. In general the size of the kernels are significantly smaller than than the input image; where a common size for a kernel would be (3x3) or (5x5), whereas, the input image is (ImageWidth x ImageHeight x ColourChannels) (Michelucci, 2019). The CNN is built from many layers each contain these kernel convolutions, feeding the result of the convolution to the next layer forward. The different types of layers leveraged in this project are as follows.

Convolution: Convolution layers take a three-dimensional input (image with three colour channels) and strides multiple kernels over the input image, with the typical size being (3x3) or (5x5) (Michelucci, 2019). The weights that each kernel is constructed from are parameters which are trained by the neural network to achieve more accurate predictions. The result of each stride is the dot product of the input image matrix and the kernel matrix; the result of which is then passed through to an activation function to get the final result for that convolution. A representation of a convolutional layer can be seen in Figure 1.

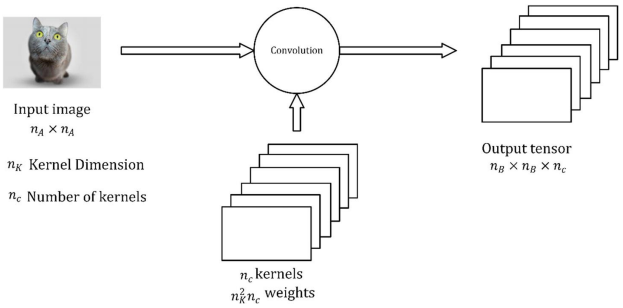

Figure 1: *Convolutional layer representation (Michelucci, 2019).*

Pooling: Pooling layers follow on from Convolutional layers. They take the tensor outputted from the Convolution layer as input and outputs a tensor constructed from the input with the pooling applied to it (Michelucci, 2019). The main aim of pooling is to is to down sample the tensor received from a convolution layer. This is done to so that large or important structural elements of the image are kept but the finer details such as their precise position in the input is discarded. This allows for a more consistent feature map to be built from many different images. Pooling layers stride over the image similarly to the convolution layer, however, the breadth of the stride is directly based upon the kernel size, whereas, in convolution layers they stride across one row or column at a time. There are multiple types of pooling but the mostly commonly used is Max pooling (Michelucci, 2019). This is where the max value is taken from each convolution of the image and is represented in Eq (1) where the the input matrix is $Matrix_{input} = \left[\begin{smallmatrix} 1 & 4 \\ 3 & 11 \end{smallmatrix}\right]$.

$$Result = max(1, 4, 3, 11) = 11 \tag{1}$$

Fully Connected Layer: The Fully Connected layer is the final layer in a CNN. These layers as the name suggests have a complete connection to all activations from the previous layer. Since the previous layers represent high-level features from the input image, the Fully Connected layer uses these features for classifying the input into various classes based upon the training dataset (Karn, 2016). To ensure that the sum of output probabilities of the fully connected layer sums to 1, a Softmax activation function is used.

Softmax is described in Eq (2), where it's performed on n numbers $x_1 ... x_n$:

$$Softmax : f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2}$$

3.2 Activation Functions

Activation functions are used to determine at what point a resultant output will fire an activation of a neuron in the neural network. Different types of activation functions will determine the boundaries of which the activation will fire, as the input can range anywhere from $-\infty$ to ∞, and to what intensity the activation fires at (Sharma, 2017). Due to the projects NN being categorical and not binary, step functions will be avoided. This is due to step functions always returning 0 or 1, meaning if there is more than one category in a classification, all will fire at 1 intensity. Meaning that multiple categories will share the same result. Instead a combination of Sigmoid and Rectified Linear Unit functions will be utilised.

The Sigmoid function is useful as it bounds our outputs between 0 and 1 and due to its nonlinear nature, allows stacking of layers. The main negative with this function is the vanishing gradient problem. As x_j approaches the bounds (0,1); the gradient gets smaller and smaller, leading to the weighting not being updated significantly or at all in the backwards pass of Backpropagation. It can also be quite costly as neurons with sigmoid activations will fire below zero, meaning near all neurons with a sigmoid activation function will fire each epoch resulting in slower training times. An example of the sigmoid activation function can be seen in Figure 2 and below:

$$Sigmoid : f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

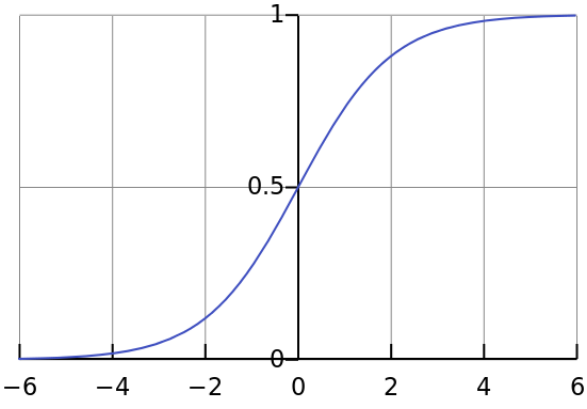

Figure 2: *Sigmoid Activation (Sharma, 2017).*

To help fix the vanishing gradient problem and the performance issue of sigmoid, Rectified Linear Unit will be utilised. Whilst ReLu's upper bound is not clamped to one which could result in blow up of the activation. It can be stacked like sigmoid but where ReLu is superior, however, is that its gradient is constant. This results in ReLu not suffering from the Vanishing Gradient problem like Sigmoid does. It is also more efficient than Sigmoid as anything below zero doesn't fire an activation, resulting in sparser activations and quicker training. An example of the ReLu's activation function can be seen in Figure 3 and below:

$$ReLu : f(x) = max(0, x) \tag{4}$$

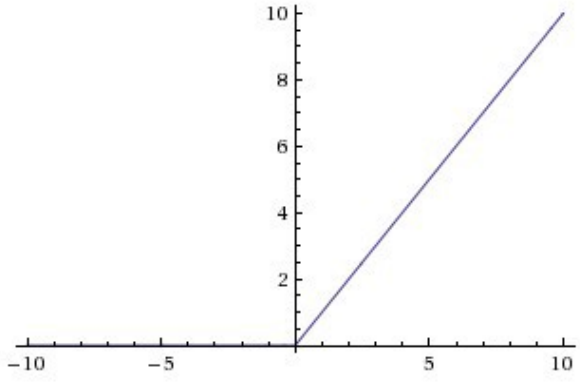

Figure 3: *ReLu Activation (Sharma, 2017).*

3.3 Backpropagation

Backpropagation is the method in which our CNN will be trained. The neural network will be a feed-forward network built up of three types of layers: input, hidden and output. Feed-forward defines how each layer will only activate nodes on the subsequent layer downstream (Makin, 2006), where each node in a layer is connected to every node in the subsequent layer, which can be seen in Figure 4.

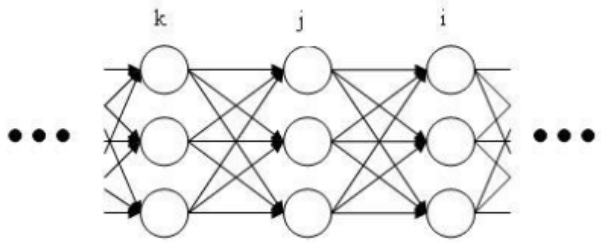

Figure 4: *Activation flows from layers k to i (Makin, 2006).*

A given epoch via this method involves two passes:

Forward Pass: The forward pass involves moving downstream the layers and calculating each neurons output by taking the weighted sum of the inputs and passing the sum through an activation function. This can be seen in Figure 5 is represented mathematically as such:

$$x_j = \sum_{k \in K_j} w_{kj} y_k \tag{5}$$

Where the output $y_j = f(x_j)$ and $f(\cdot)$ is the activation function of choice for that layer (Makin, 2006).

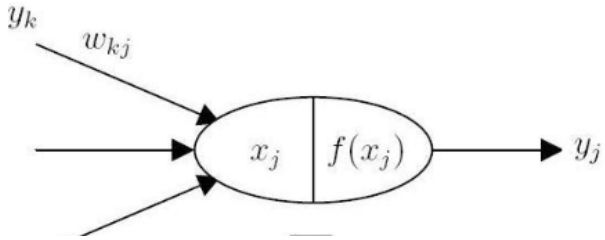

Figure 5: *A visualisation of the mathematical expression (Makin, 2006).*

Once this process has been followed for every layer of the NN a loss function is used to calculate the total error for the NN using the sum of the squared error of each output (the $\frac{1}{2}$ scale factor is for convenience as it will be cancelled out when adjusting the weighting on nodes in the backwards pass (Makin, 2006)):

$$E_{total} = \frac{1}{2} \sum_{j=1}^{J} (t_j - y_j)^2 \tag{6}$$

Where J and j are the total and current output respectively, y_j is the outputted result and t_j is the desired outputted result.

Backwards Pass: Once the total error has been calculated, weightings can be adjusted from the output upstream towards the input. The weight change for a weight connecting a node in layer k to layer j is:

$$\partial w_{kj} = -\alpha \frac{\partial E_{total}}{\partial w_{kj}} \tag{7}$$

Where α represents a learning rate where adjustments to it will be ad hoc and the negative sign represents weight changes in direction of decrease in error.

Expanding the partial derivative by chain rule yields:

$$\frac{\partial E_{total}}{\partial w_{kj}} = \frac{\partial E_{total}}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{kj}} \tag{8}$$

Where the expansion of $\frac{\partial x_j}{\partial w_{kj}}$'s partial derivative is (due to Eq (5)):

$$\frac{\partial x_j}{\partial w_{kj}} = y_k \tag{9}$$

,

The Expansion of $\frac{\partial y_j}{\partial x_j}$'s partial derivative is related to the activation function, In the case of Sigmoid where $y_j = f(x_j)$:

$$\frac{\partial y_j}{\partial x_j} = y_j(1 - y_j) \tag{10}$$

Finally the expansion of $\frac{\partial E_{total}}{\partial y_j}$'s partial derivative is in the error term, which is just Eq (6) with respect to y_k:

$$\frac{\partial E_{total}}{\partial y_j} = -(t_j - y_j) \tag{11}$$

Bringing equations (11), (10) and (9) together results in:

$$\frac{\partial E_{total}}{\partial w_{kj}} = -(t_j - y_j)y_j(1 - y_j)y_k \tag{12}$$

Equations (11) and (10) are also known as the error term and can be treated as a single quantity (Makin, 2006):

$$\delta_j = -\frac{\partial E_{total}}{\partial y_j}\frac{\partial y_j}{\partial x_j} \tag{13}$$

However in the cases where j is a hidden layer another application of chain rule is required due to $\frac{\partial E_{total}}{\partial y_j}$ needing to take into account the errors propagated into the next i^{th} layer.

Therefore, in the case of hidden layers:

$$\frac{\partial E_{total}}{\partial w_{kj}} = -\sum_{i \in I_j}(\delta_i w_{ji})y_j(1 - y_j)y_k \tag{14}$$

4 Practice

To discover which model worked best for the given data-set, extensive architecture search was performed. To be able to test many iterations quickly, many of the components of the layers and their parameters were nested in for loops. By doing this it was easy to make many different permutations of CNN's. By leveraging Tensorboard, models could be compared easily to see which ones gave the best results to use for classification later.

The core layout of the model starts with Convolution and Max pooling layers in order to highlight the key features of the input image. This is followed by a Flatten to reduce the tensor to a single dimension. The result of the Flatten is then passed into Dense layers of reducing size. The purpose of these dense layers are to take the features from the previous layers and combine them into a variety of attributes which will be used to help with classification. The final Dense layer will be of the same size as the total categories and Softmaxed to return the classification probabilities of the input image in relation to the data-set categories.

5 Evaluation

Tensorboard was leveraged in this project in order to visually compare the performance of many different models. It plots the accuracy, loss, validation accuracy and validation loss for each epoch of a model. Each models graph can then be plotted to the same set of axis for comparison. The user can then see which models performed the best from this by looking for a smooth increase of validation accuracy and a smooth decrease of validation loss. If the curve of the graphs have spikes in them and are not smooth, this can indicate overfitting. Figures 6 and 7 show the validation accuracy and loss of three models respectively over a thousand epochs. The red line is what should be aimed for when training a CNN. It has a consistent trend and only ever deviates slightly per epoch. The blue and green lines, however, have strange jumps in their values between epochs, this is a sign that something is not correct whilst training and the outputted model will not perform as desired. All three models had 2 convolutional layers with a kernel size of (7x7), followed by a MaxPooling layer of size (2x2) and finally a single dense layer. They differed however in their layer sizes. The red lined model's convolutional layers had a size of (128x128) with the dense layer size being (256x256). The blue lined model's convolutional layers were also of size (128x128) however, the dense layer was larger at (512x512). The green lined model's convolutional layers had a size of (256x256) and a dense layer size of (256x256). The red model gave the best results in the testing with some examples being in Figures 8) and 9.

Once the final model permutation was decided upon it was used to classify images larger satellite images. This was done by splitting the image into multiple segments, classifying each segment and then writing the result of the classification out to a JSON file. This JSON file was then used to build a level in the Unity engine based upon the inputted satellite image. Examples of a satellite image and its corresponding result in Unity can be seen in Figures 8 and 9.

6 Conclusion

In conclusion this project resulted in a reliable model that could be reused multiple times. The model was semi consistent at predicting the correctly category for the given segment but did trip up at times. The Unity implementation took the JSON outputted from the model and built a level from it, joining adjacent tiles where appropriate. Future iterations of this project could vastly be improved by training the models with a better data-set as a lot of the mis-classifications were where an image had multiple categories in the image. By utilising a more high-res and category specific data,

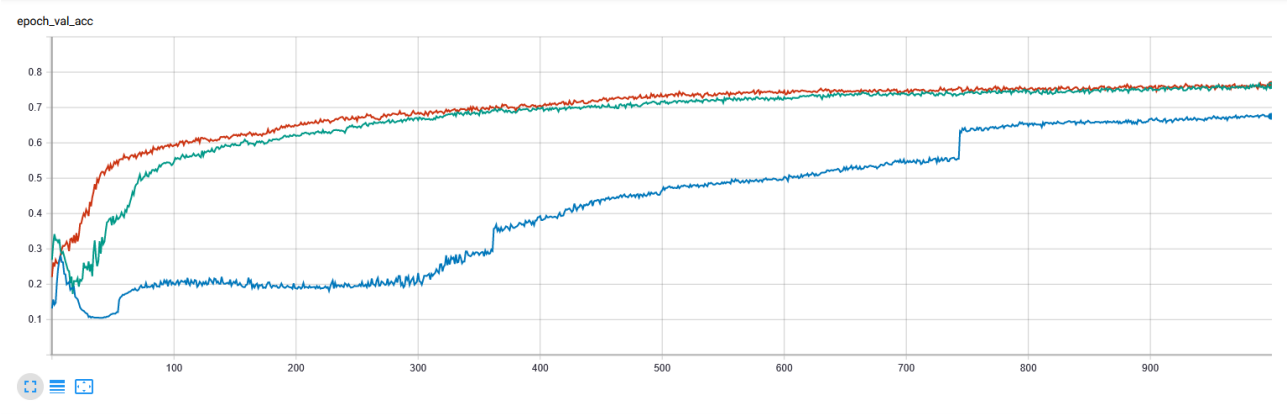

Figure 6: *Validation Accuracy of three models.*

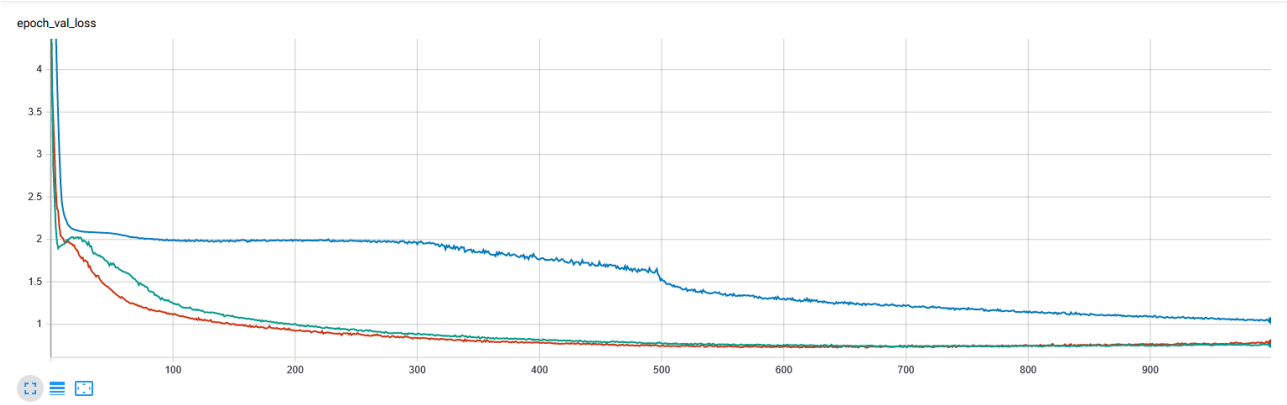

Figure 7: *Validation Loss of three models.*

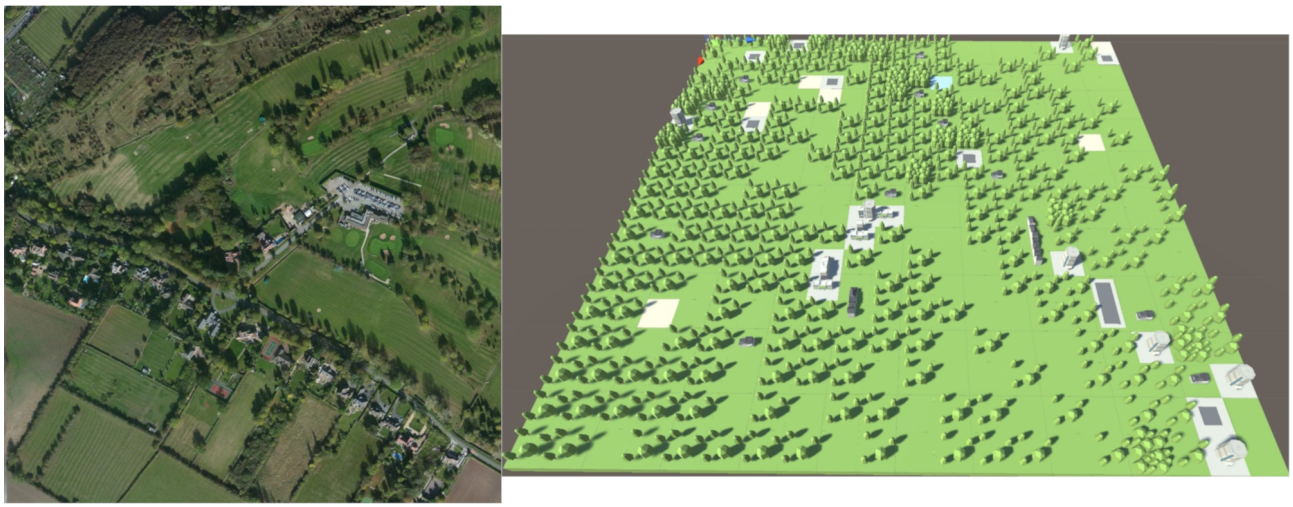

Figure 8: *Image and level built from classification side by side.*

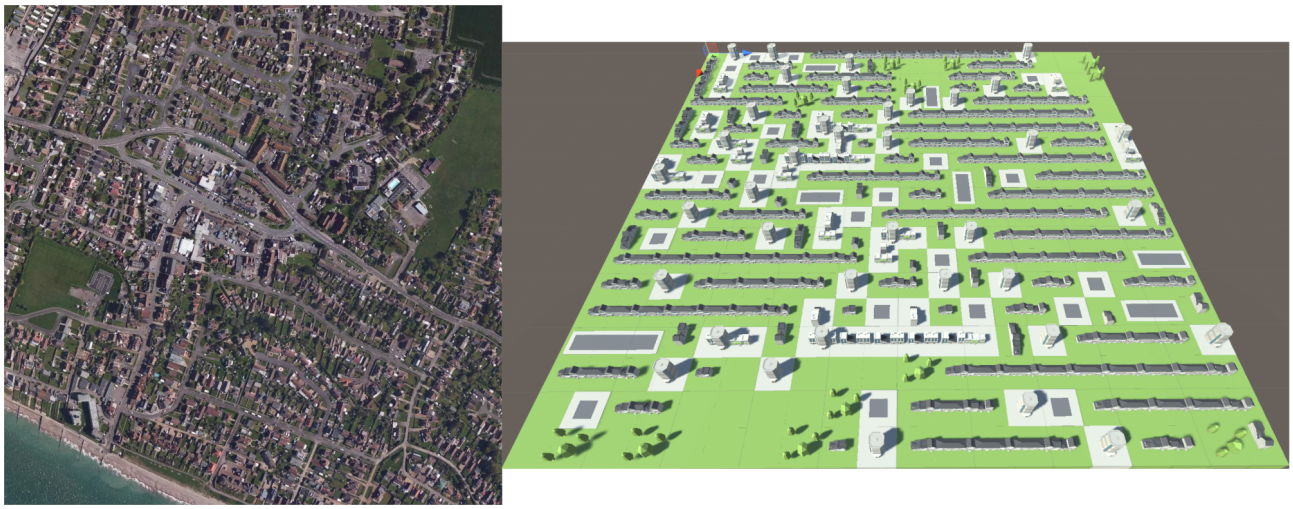

Figure 9: *Image and level built from classification side by side.*

the reliability of the classifier could be improved greatly. In addition to this, the image sizes that are being used to train the model should also be made smaller. This in turn would allow for splitting the desired satellite image into smaller segments, resulting in more detail of the Unity example as the resolution of the inputted data would be higher.

Bibliography

Alex Krizhevsky Ilya Sutskever, Geoffrey E. Hinton (2017). "ImageNet Classification with Deep Convolutional Neural Networks". In: Communications of the ACM. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Emmanuel Maggiori Guillamume Charpiat, Yuliya Tarabalka Pierre Alliez (2017). "Recurrent Neural Networks to Correct Satellite Image Classification Maps". In: *IEEE Transactions on Geoscience and Remote Sensing*. Vol. 55. 9. IEEE, pp. 4962 –4971.

Good, Otavio (2015). "How Google Translate squeezes deep learning onto a phone". In: *Google AI Blog*. Google. URL: https://ai.googleblog.com/2015/07/how-google-translate-squeezes-deep.html.

Karn, Ujjwal (2016). "An Intuitive Explanation of Convolutional Neural Networks". In: URL: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

M.A Popov Alpert, S.I & Podorvan (2017). "Satellite Image Classification Method Using the Dempster–Shafer Approach". In: *Atmospheric and Oceanic Physic*. Vol. 53. 9. Pleiades Publishing, 1112–1122.

Makin, J.G (2006). "Backpropagation". In: URL: https://inst.eecs.berkeley.edu/~cs182/sp06/notes/backprop.pdf.

Michelucci, Umberto (2019). "Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection". In: Apress. ISBN: 9781484249765.

Sharma, Avinash (2017). "Understanding Activation Functions in Neural Networks". In: *The Theory of Everything*. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

Yansong Fen, Mirella Lapata (2012). "Automatic Caption Generation for News Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35. 4. IEEE, pp. 797 –812.

Yanting Pei Yaping Huang, Qi Zou Xingyuan Zhang Song Wang (2019). "Efftects of Image Degradation and Degradation Removal to CNN-base Image Classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. IEEE, pp. 1–1.