Facial Keypoints Detection using Convolutional Neural Networks

Ayman Suliman, Idris Sahil, Chunyang Han

KTH Royal Institute of Technology

Abstract. The aim of this project was to predict facial keypoints using Convolutional Neural Network and investigate the effect of using dropout, data augmentation and early stopping. These methods affect the over-fitting of the network, thus applying them should make the system less complex. After that, we used the validation error to choose the optimal learning rate. The results shows that using only data augmentation gives the best validation loss. Quick testing showed that using Nesterov's momentum gave the best result, and using no learning rate decay also gave the best result by large. Therefore, we kept these two variables constant and only varied the learning rate. With data augmentation and a learning rate of 0.42, a final validation error of 0.00088 was achieved.

1 Introduction and Background

Facial point detection is an important aspect in a wide variety of fields, such as tracking faces in video, analyzing facial expressions and detecting dysmorphic facial signs for medical diagnosis. The development of deep learning theory allows us to push the performance of facial recognition. The key factors of such an improvement are deep models, most often Convolutional Neural Networks (CNNs), big amount of training data and modern graphic processing units (GPUs). However, CNNs are broadly used in other tasks beyond image analysis such as video analysis, drug discovery and speech recognition. Detecting facial key points is a difficult problem due to many different factors that can affect how the keypoints are placed such as posture, angle, illumination and simply the fact that every face is uniquely different from another in size and shape.

In this project we use the same model of convolutional networks to predict keypoint positions on face images, but we alter the use of methods that prevent over-fitting such as dropout, data augmentation and early stopping. This will help us analyze how well the model performs with different levels of complexity when tested with the same dataset.

1.1 Related work

The dataset used was a part of a competition from the website Kaggle, where 175 teams tried to achieve as low error as possible. Some of these teams provided thorough explanations to their methods such as Nouri [7], Yuki [5] and Yumi [6]. Their methods were used as an aid to build our own model in order to achieve our aims.

2 Theory

2.1 Convolutional Neural Networks

Convolutional Neural Networks are a type of deep feed forward neural network that is often used for analyzing visual images. The Convolutional Neural Network structure is showed in figure 1.

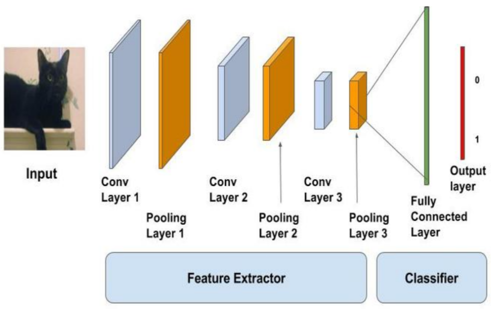


Fig. 1: Convolutional Neural Network

The first part consists of convolutional and max-pooling layers which act as the feature extractor. The second part consists of the fully connected layer which performs non-linear transformations of the extracted features and acts as the classifier.

In the convolution step the data is taken in and feature maps are created from it. This is illustrated in figure 2, where an image of a cat has been converted into pixels (represented by the squares) then a feature map is created from that image [13].

In the pooling step which is illustrated in figure 3, we select a region of the data, then take the maximum value of that region. The maximum value will then become the new value for the entire region. In this way, we reduce the spatial size. This reduces the number of parameters, hence computation is reduced. The convolutional layer and the pooling layer together make up the hidden layer of a convolutional neural network. The more hidden layers a convolutional neural network has the more complex it becomes, however too many hidden layers can lead to a model that is too complex and might result in over-fitting.

The fully connected layer is the neural network where all the nodes are fully connected. Then in the last step we get our output.

In our case, facial pictures will be used as input. In the convolutional step a map with our features (keypoints) will be created. We will then use max pooling in the pooling step and finally have our predicted facial keypoints as output.

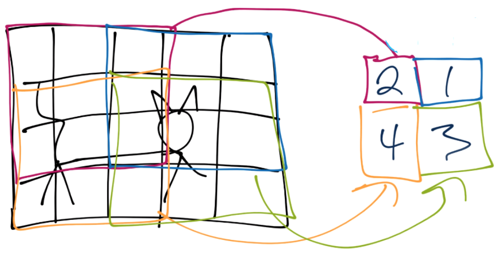


Fig. 2: Convolutional step with cat image into converted feature map

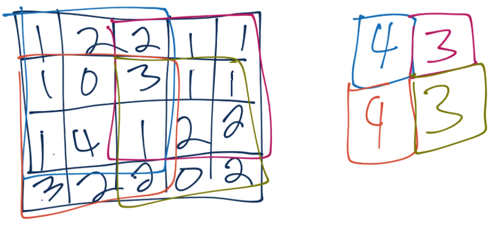


Fig. 3: Pooling step for the convolutional neural network

2.2 Loss function

The root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed. The RMSE represents the sample standard deviation of the differences between predicted values and observed values. RMSE is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the squared error, thus larger errors have a disproportionately large effect on RMSE which means RMSE is sensitive to outliers. In our case, we set up a loss function $f(x)$ of RMSE in formula 1.

$$RMSE: \quad f(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(x_i;\theta))^2} \tag{1}$$

where
- $\hat{y}_i(x_i;\theta)$ predicts the label y_i.
- (x_i, y_i) represents our ground-truth training data and labels.
- θ represents the parameters we have to learn by exploiting ground-truth training data.

2.3 Activation function

An artificial neuron calculates a weighted sum of its input, adds a bias and then decides whether the neuron should be activated or not [12]. An activation function helps the neuron decide whether it should be activated or not.

In Keras, we implement the (leaky) rectified linear unit activation (ReLU) which is defined as

$$f(x) = \max(x, \alpha x) \tag{2}$$

where $\alpha \in [0, 1]$. In our case, $\alpha = 0$ was chosen. In simpler terms, ReLU turns every negative x to output zero and every positive to x.

2.4 Dropout

In order to prevent over-fitting the training data, a method called dropout is used. Dropout is a regularization technique in neural networks. It works by ignoring neurons with a certain probability during the training phase, shown in figure 4 which results in a reduced network after the training and in the end reducing the possibility for each neuron to over-fit.

However, dropout roughly doubles the number of iterations required to converge, but also reduces the training time for each epoch due to the reduced number of nodes [8].

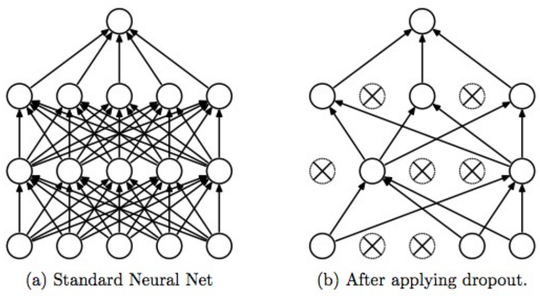


Fig. 4: Visualization of what Dropout does to the neurons.

2.5 Early Stopping

Another way to prevent over-fitting is by the use of the early stopping method. Early stopping prevents the model from over-fitting to the training dataset when too many epochs are used. When that happens, the error in the validation dataset starts to increase instead of decrease, which is a hint that the training dataset is over-fit. As we can see in figure 5. [10]. The procedure will then need to be shut down prematurely to avoid getting a larger validation error.

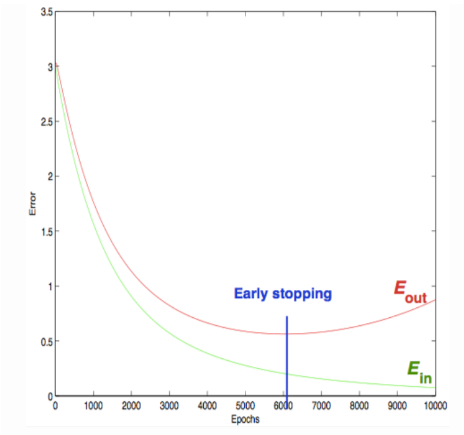


Fig. 5: An example of correct usage of Early Stopping

3 Approach

3.1 Dataset

The data set used to train our models is from Kaggle, provided by Dr. Yoshua Bengio of the University of Montreal.

Each predicted keypoint is specified by an x,y coordinate that points to the pixel number of the 96x96 black and white image. There are in total 15 keypoints, 5 for left eye, 5 for right eye, 1 for nose, and 4 for mouth. However, some images in the training data have some missing keypoints, which means some elements of the face have in total more keypoints in the dataset than other keypoints. In average around 2000 images are considered ”high quality” with all keypoints, while the other 5000 are considered ”low quality” and only have four keypoints. Figure 6 shows examples of these labeled images. There are 7049 training images and 1783 test images, and in total 27124 keypoints to predict in the test set [4].

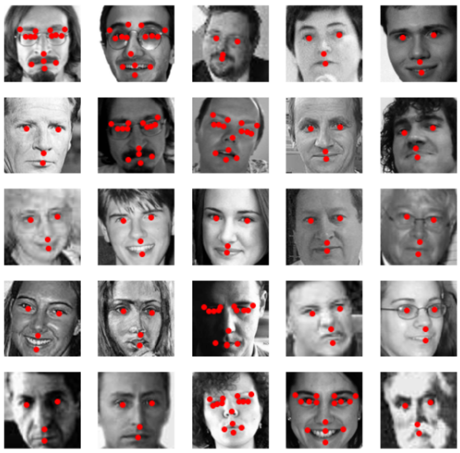


Fig. 6: Examples of the labeled faces in the dataset, showing some faces with all 15 keypoints and some faces with only 4 keypoints.

3.2 Evaluation method

In order to measure the error of the testing dataset, we need an evaluation method that determines how good the placements of the keypoints are. Since the images are 96x96, there are 9216 different placements for the keypoints. Therefore, looking at the accuracy of the keypoint placements will not be very efficient. Hence, we need to look at the error of the keypoint placements. The error in this case is the distance between the placed keypoint and the ground-truth label. This distance is calculated with RMSE from equation 1.

In order to minimize the loss function (objective function) in equation 1, we use Stochastic Gradient Descent (SGD). It's a stochastic approximation of the gradient descent optimization. This method iterates to find the minima or maxima of the loss function $f(x)$, where x is the parameter that has to be found in order to minimize $f(x)$.

In SGD, the true gradient of $f(x)$ is approximated by the following update step at epoch t:

$$x^{(t+1)} := x^{(t)} - \eta_t \nabla_x f(x^{(t)}), \tag{3}$$

where η_t is the learning rate and $\nabla_x f(x^{(t)})$ is the gradient. However, simply adjusting the learning rate is very difficult to optimize since a too small value can cause very slow convergence, and too large may cause oscillations. In order to fix this, momentum is introduced:

$$v^{(t+1)} = \gamma v^{(t)} + \eta \nabla_x f(x^{(t)}) \tag{4}$$

$$x^{(t+1)} = x^{(t)} - v^{(t+1)} \tag{5}$$

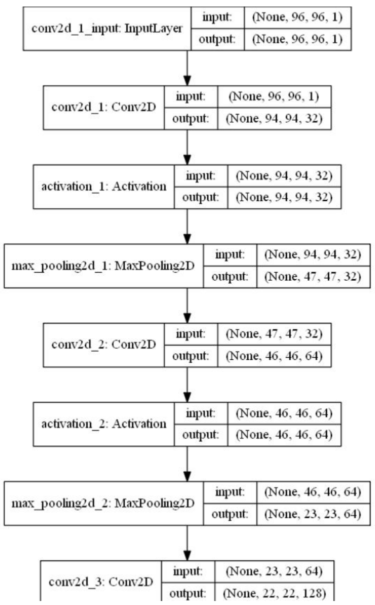
where γ is the momentum term, typically set to 0.9. It helps accelerate SGD in the appropriate direction and dampens the oscillations of the default SGD, which results in a faster convergence on average.

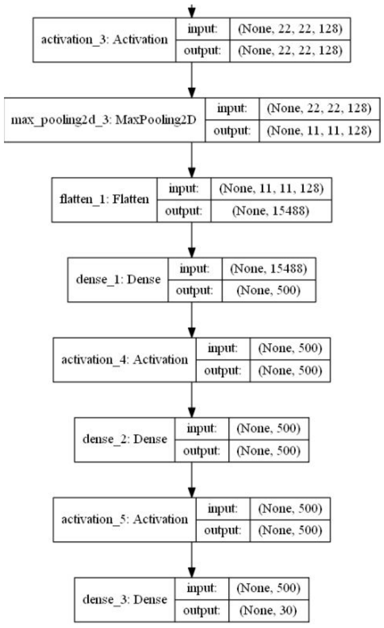
Another method of calculating momentum is by using the Nesterov method. It works by first taking a normal gradient jump from $x^{(t)}$ to $y^{(t+1)}$ but then it moves slightly towards the previous value of $y^{(t)}$ to correct itself [11]. The update function now looks like this:

$$v^{(t+1)} = \gamma v^{(t)} + \eta \nabla_x f(x^{(t)} + \gamma v^{(t)}) \tag{6}$$

3.3 Network Architecture

The chosen CNN model for the project consists in total of three convolutional layers, three maxpooling layers and three fully connected layers, all with ReLU activation between the layers [6]. The structure of the CNN model can be seen in figure 7.


conv2d_1_input: InputLayer
input: (None, 96, 96, 1)
output: (None, 96, 96, 1)
conv2d_1: Conv2D
input: (None, 96, 96, 1)
output: (None, 94, 94, 32)
activation_1: Activation
input: (None, 94, 94, 32)
output: (None, 94, 94, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 94, 94, 32)
output: (None, 47, 47, 32)
conv2d_2: Conv2D
input: (None, 47, 47, 32)
output: (None, 46, 46, 64)
activation_2: Activation
input: (None, 46, 46, 64)
output: (None, 46, 46, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 46, 46, 64)
output: (None, 23, 23, 64)
conv2d_3: Conv2D
input: (None, 23, 23, 64)
output: (None, 22, 22, 128)
(a) First part of CNN structure.



(b) Second part of CNN structure.

Fig. 7: The chosen CNN model structure.

3.4 Data Augmentation

One way to prevent over-fitting in the training part is to add more training data. Since the training data is limited in the database, we have to modify our available data in order to create copies that are altered in some way. This can theoretically double the available training data and further generalize your model. The chosen data augmentation method for this project is horizontal flipping, where images were translated horizontally. An example of this can be seen in figure 8, annotated with the translated ground-truth facial landmarks.

When we implement data augmentation, it's important not to forget to horizontally flip all the data points except the nose, or you'll get the case shown in figure 9.

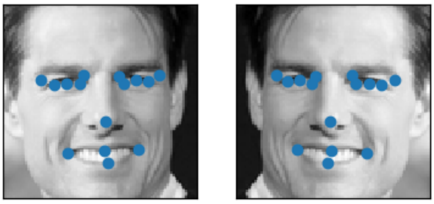


Fig. 8: Visualization of the data augmentation by flipping horizontally, faces annotated with the ground-truth facial landmarks. Left image is original image, right image is horizontally flipped image with fixed data points.

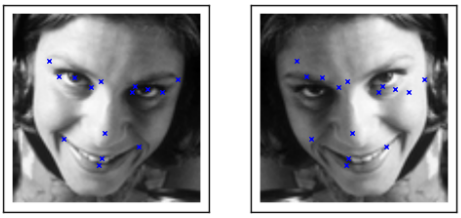


Fig. 9: Visualization of the data augmentation by flipping horizontally, faces annotated with the ground-truth facial landmarks. Left image is original image, right image is horizontally flipped image without fixed data points.

3.5 Keras

Keras is a high-level neural networks API written in Python and capable of running on top of TensorFlow. It allows for much easier application and testing of a CNN model, and results in much shorter code that does the same as a code done purely in TensorFlow. Keras will be used throughout the process to create the model and train and test the dataset [3].

4 Results

4.1 Dropout and Data Augmentation choice

These results were achieved by using the same model and activating or deactivating dropout and data augmentation, while keeping a constant learning rate at 0.1. By experimenting with dropout and data augmentation, four figures were achieved which represent the four possible cases shown in figure 10,11, 12, and 13 . We can see the values of training loss and validation loss after 1000 epochs respectively in table 1.

In the model with both dropout and data augmentation,training loss is much lower than validation loss which means the network might be overfitting; in the model with only dropout, the training and validation loss are about equal which means model is underfitting.In the best model,the training loss and validation loss should be approaching one another as training epochs continues. This is because as if training error begins to get lower than your validation error, the model begins to be overfitted. Compared to the model without any implementations, the "data augmentation only" model gave the best validation loss by only a small number.

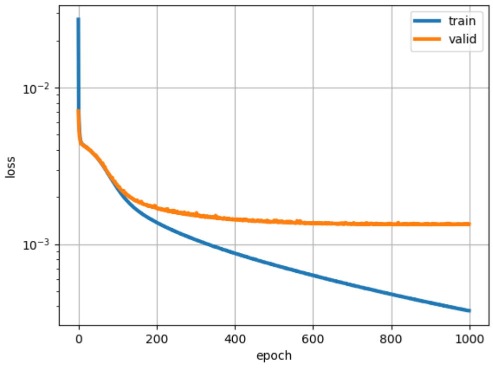


Fig. 10: Loss plot without dropout or data augmentation.

4.2 Finding optimal model parameters

After finding the best method to reduce complexity of our model, the choice of using only data augmentation was made. The model can now be kept constant

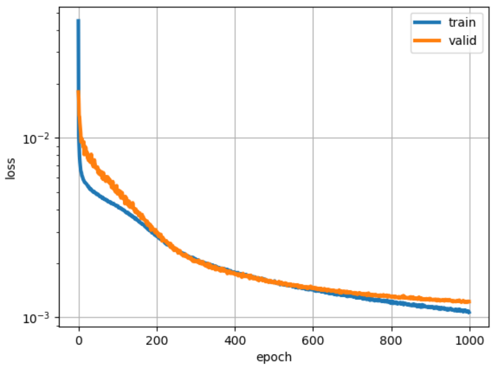


Fig. 11: Loss plot with dropout but without data augmentation.

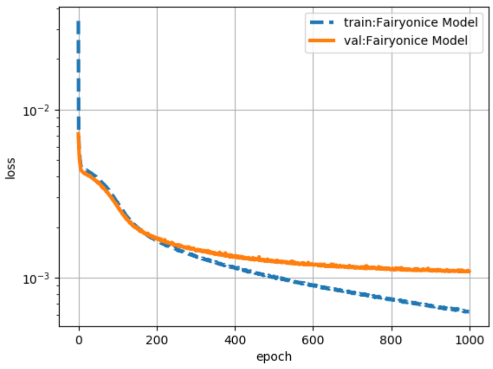


Fig. 12: Loss plot without dropout but with data augmentation.

Dropout	0	1	0	1
Data Augmentation	0	0	1	1
training loss	0.0004	0.0011	0.0007	0.0012
validation loss	0.0013	0.0012	0.0011	0.0012

Table 1: Training loss and validation loss after 1000 epochs (0 means not used,1 means used).

while searching for the optimal model parameters. This was done by running the experiment for 100 epochs and changing the learning rate to compare the validation errors of the different learning rates.

Quick testing showed that using Nesterov's momentum show in equation 6 together with momentum value of 0.9 gave the best result, and using no learn-

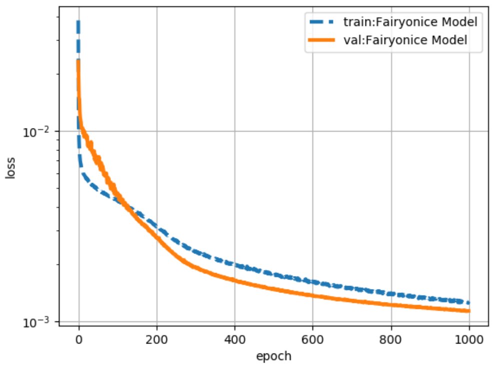


Fig. 13: Loss plot with both dropout and data augmentation.

ing rate decay also gave the best result by large. Therefore it was decided to keep these two constant and only vary the learning rate during the experiment. It's also important to know that the values varied for each run and are only approximations.

Learning rate	Validation error
0.20	0.00101
0.35	0.00097
0.40	0.00092
0.41	0.00092
0.42	0.00088
0.50	0.00105

Table 2: Different learning rates with their respective error after 100 epochs. Chosen learning rate in bold.

4.3 Testing with optimal learning rate

After the best learning rate was found from table 2, it was found that a higher learning rate can achieve a better score by only using 200 epochs. However, using more than 200 epochs will only result in an increase in validation error. This can be seen in figure 14. This increase in validation error could be stopped by using early stopping, but unfortunately it did not work for us.

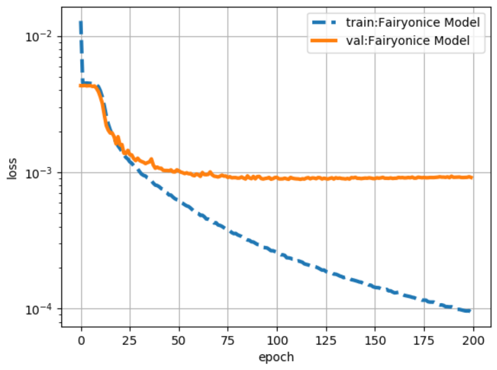


Fig. 14: Loss plot with data augmentation and a learning rate of 0.42. Final validation error at 0.00088.

5 Discussion

We can see from the results in section 4.1 that the validation losses are all quite the same. However, the training loss plots from all four models varied greatly depending on if dropout was being used or not. By adding dropout, the training loss got a larger loss, and the validation loss decreased slightly. This is mostly because using dropout requires more epochs for the model to converge, thus stopping it at 1000 epochs is too early for the system to converge.

After choosing the best model, the most optimal learning rate was found to be 0.42 with a validation error of 0.00088. But it was difficult to find a precise number due to the shuffling of data for every run. However, it's very clear that the model is overfitting the training data due to how low the training data is. Despite this flaw, it achieved a lower validation error than the previous models, with a lot lower epochs.

5.1 Future Work

One way to improve the loss values in figure 14 is by adding another method of data augmentation. One way of creating more training data is by slightly shifting the images and the key points diagonally. Adding this would double the number of available images once again and add a lot more generalization to the model.

Another way to improve the model is by separating all 15 keypoints into two or more models that do the job. This could allow the models to specialize in their own keypoints (like the left eye center) and increase the efficiency of the models.

Instead of increasing the learning rate, it's possible to lower the learning rate and increase the number of epochs. This is usually best to do since having a lower learning rate results in a more reliable training, and since the number of epochs is not limited in this assignment, it may be more optimal to do so[14].

6 Conclusion

This experiment shows that using too many methods to reduce complexity can negatively affect the outcome of the validation loss when handling facial key-points. When both data augmentation and dropout were used, the resulting loss increased compared to only using data augmentation. However, as we see in Table 1, for both data augmentation and dropout we have an improvement in validation loss compared to not using either of the methods.

There's also a big importance in choosing learning rate, however there's a big relation between learning rate and the number of epochs that want to be run. Choosing a high learning rate converges the loss very early, while choosing a low learning rate leads to slower convergence.

References

1. Hof. R. *Is artificial intelligence finally coming into its own - mit technology review.* [Online]. Available: `https://www.technologyreview.com/s/513696/deep-learning/` (2017)
2. *Tensorflow* [Online]. Available: `https://www.tensorflow.org`(2018)
3. *Keras*. Available: `https://keras.io/` (2018)
4. Kaggle. *Facial Keypoints Detection.* Available: `https://www.kaggle.com/c/facial-keypoints-detection/data` (2017)
5. Yuki. S. *Implementing Kaggle Facial Keypoints Detection with Keras.* [Online]. Available: `https://elix-tech.github.io/ja/2016/06/02/kaggle-facial-keypoints-ja.html` (2016)
6. Anonymous. *Achieving Top 23% in Kaggle's Facial Keypoints Detection with Keras + Tensorflow.* Available: `https://fairyonice.github.io/achieving-top-23-in-kaggles-facial-keypoints-dete` `html` (2018)
7. D. Nouri. *Using convolutional neural nets to detect facial keypoints tutorial* Available: `http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/` (2014)
8. Budhiraja, A. *Dropout in (Deep) Machine Learning.* Available: `https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less` (2016)
9. Srivasta, N. Hinton, G. Krizhevsky, A. Sutskever, I. Salakhutdinov, R. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Available: `http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf` (2014)
10. Y. S. Abu-Mostafa, M. Magdon-Ismail, and H-T. Lin. *Learning From Data.* Available: `www.AMLbook.com` (2012)
11. S. Bubeck. *Nesterov's Accelerated Gradient Descent.* Available: `https://blogs.princeton.edu/imabandit/2013/04/01/acceleratedgradientdescent/` (2013)
12. A. Sharma. *Understanding Activation Functions in Neural Networks.* Available: `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0` (2017)
13. V. Gupta *Image Classification using Convolutional Neural Networks in Keras* Available: `https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/` (2017)
14. P. Surmenok *Estimating an Optimal Learning Rate For a Deep Neural Network.* Available: `https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0` (2017)