Modified Digits: A Visual Add/Multiply with Machine Learning

Victoria Madge
victoria.madge@mail.mcgill.ca
260789644

Fares El Tin
fares.eltin@mail.mcgill.ca
260786755

Alan An
chengyun.an@mail.mcgill.ca
260683828

***Abstract*—Computer vision tasks, such as Optical Character Recognition using image-based data, present a difficult pattern recognition problem, but have become easier to solve with the advancements in complex classifiers such as Convolutional Neural Networks. This project uses three different classifiers to classify a modified MNIST dataset into 40 unique classes. Logistic Regression, Neural Networks, and Convolutional Neural Networks were used on a training dataset of 50 000 images, and were tested on 10 000 images. The Convolutional Neural Network yielded the best performance, with a testing accuracy of 97.35% on Kaggle under the team name "MAE". The code for all classifiers considered in this project can be found on the Conference Management Toolkit as a supplementary attachment.**

***Index Terms*—Optical Character Recognition, Logistic Regression, Neural Networks, Convolutional Neural Networks**

I. Introduction

Optical Character Recognition (OCR) has improved throughout the years thanks to the machine learning applications in pattern recognition [1]. OCR can include image-based datasets of natural scene characters (letters, digits, etc.), handwritten characters, and more. One popular database of handwritten digits, called MNIST, is a reliable database for learning pattern recognition techniques on real world data, whilst minimizing time spent on data pre-processing [2].

The purpose of this project is to develop a machine learning algorithm to automatically classify images consisting of two digits and an arithmetic operation. The dataset provided for this project is based on the MNIST dataset, with slight modifications to include more information in each image. Specifically, each image has two digits (from 0-9) and one letter denoting the operation, which could be oriented at 0°, 45° left and 45° right. Letters including "a" (or "A"), and "m" (or "M") denote "add" and "multiply" operations, respectively. The classifiers for this task output the result of the mathematical operation between the two digits in the image. For example, if an image contains "A", "1", and "5", the class corresponding to the instance would be 6, since "1" and "5" are added together. As such, forty unique classes are possible: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 24, 25, 27, 28, 30, 32, 35, 36, 40, 42, 45, 48, 49, 54, 56, 63, 64, 72, 81]. A training dataset of 50 000 images was provided with corresponding class labels. The testing set includes 10 000 images. The authors uploaded their predictions to Kaggle (https://www.kaggle.com/c/comp551-language-classification). In Kaggle, the leaderboard was calculated with approximately 40% of the prediction results, leaving 60% withheld for performance evaluation purposes.

This report summarizes various classifiers used to predict the mathematical operation of the two digits in each image, where the operation was denoted by the letter in the image. The report will first discuss related work in OCR and will then elaborate on the data pre-processing, feature design and selection methods employed. The following classifiers: Logistic Regression, Neural Network, and Convolutional Neural Network will then be discussed in terms of their respective methodologies, training, validation and testing employed.

II. Related Work

The following section will survey three related works corresponding to character classification from image data using Logistic Regression, Neural Networks, and Convolutional Neural Networks in their respective methods.

Although Logistic Regression as a lone classifier does not work well for complex pattern recognition tasks [3], Yin et al. integrated Logistic Regression into a novel distance metric learning algorithm for their character classification model [4]. After using novel feature selection methodology, the authors grouped the text characters of a given handwritten dataset into clusters and learned the weights and threshold simultaneously using Logistic Regression. The authors performed a 70:30 train-test split on 466 texts of labeled data from the ICDAR 2011 training database. The approach performed with a precision of 93.49%, a recall of 84.21% and an f-measure of 88.61%, exceeding performances of other classifiers. Integrating Logistic Regression to learn hyper-parameters for a more complex classifier may be useful for this task.

Garris et al. use a Neural Network-based approach to recognize handwritten letters and digits in documentation from a publicly available dataset from the National Institute of Standards and Technology (NIST) [5]. The dataset comprises of handwritten documentation which undergoes rigorous data pre-processing to isolate each character as a representation of 1024 binary pixel values. A Karhunen-Loéve transform is then applied for feature selection and to reduce dimensionality and noise. A network of size 128x128x10 was used to classify digits, and networks of size 128x128x26 was used to classify letters. The total accuracy on all networks was 96.3%. The task of splitting the letters and digits into separate neural networks proves to be an interesting approach for an OCR problem,

and could be considered for this project. However, it is noted that the number of classes Garris et al. considered included 26 from the alphabet, and 10 from digits; the problem for this project has 12 classes (2 letters and ten digits), if the letters and digits were to be separated.

Cireşan et al. proposed a Graphics Processing Unit (GPU) implementation of a Convolutional Neural Network for object classification and handwritten digit recognition from NORM, CIFAR10, and MNIST datasets, respectively [6]. Their approach utilizes convolution and max pooling layers, with millions of feature map sizes. The weights were randomly initialized with respect to a uniform random distribution. Neurons utilized a hyperbolic tangent (i.e. tanh) activation function. The CNN with the lowest validation error was used on the test set, yielding an error of 0.35% for MNIST experiments, an error of 2.53% for NORB experiments, and an error of 19.31% for CIFAR10 experiments. This architecture deems worthy of testing since similar datasets (MNIST versus modified MNIST) are used. Although, given computation time, the feature map size may be reduced for this task.

III. Problem Representation

As previously mentioned, the purpose of this project is classify image-based data into forty unique classes, which are the result of a mathematical operation from each image. The dataset comprises of 64x64x1 images containing two digits and one letter, either "a" or "A" for an add operation, and "m" or "M" for a multiply operation. Figure 1 depicts the class imbalance present in the training set provided. It is assumed that the training data (50 000 images) and testing data (10 000 images) have been stratified, such that the class distributions are identical.

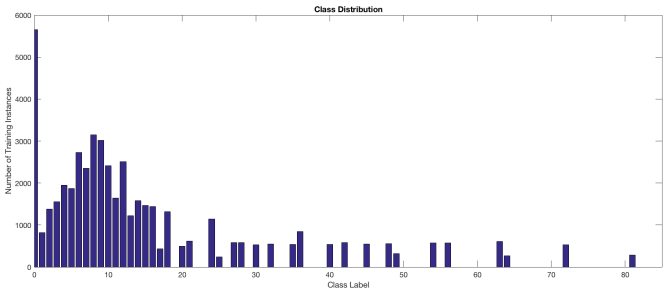

Fig. 1: Class distribution

From Figure 1, it is noted that there is a class imbalance present, with the majority class being 0 (around 11%). Although class imbalance was not managed due to time constraints, other data pre-processing methods will be discussed in this section, followed by the feature design and selection methods employed.

A. Data Pre-Processing

The dataset provided contained images with noise present, which had the potential of hindering digit and letter recognition from the classifiers. To yield better accuracy, the images were passed through a pre-processing pipeline to remove unwanted information (i.e. noise).The images were first converted to binary by thresholding to conserve the character shape. The objects in the binarized images were then individually labeled, and the three largest objects were retained, assuming they were the digits of interest. Figure 2 depicts the result of passing an image through the pre-processing pipeline. Figure 2.1 depicts the original image in greyscale, Figure 2.2 depicts the binarized image, Figure 2.3 depicts the binarized image with noisy objects removed.

Fig. 2: Image pre-processing pipeline: (1) original greyscale image; (2) binarized image; (3) binarized image with smaller objects (i.e. noise) removed;

Any further data pre-processing was performed for the purpose of feature extraction, and will be described in the next subsection.

B. Feature Design and Selection

Feature design methods employed followed Due Trier et al.'s survey of various feature extraction methods for character recognition [1], and are described below.

1) Raw pixel values from image data: The raw pixel images were considered as features for use with the Convolutional Neural Network since the first few layers within the architecture produced adequate feature extraction when running backpropagation [7]. These pixels values are the greyscale values taken directly from the given image data, but they have been normalized for use in the classifier. The total number of features in this feature set is equivalent to the number of pixels in an image, which amounts to 4096 features.

2) Pre-processed image data: The pre-processed data, as described in Section 2, were considered for features in all classifiers. Since the images were binarized (i.e. pixel values were either 0 or 1), no normalization was necessary before running the data through the network classifiers. The total number of features in this feature set is equivalent to the number of pixels in an image, which amounts to 4096 features.

3) Zernike moments: Zernike moments are projections of an image onto a space defined by orthogonal polynomials $V_{nm}(x, y)$ described in Equation 1 [1].

$$V_{nm}(x, y) = R_{nm}(x, y)e^{jmtan^{-1}(y/x)} \quad (1)$$

where $j = \sqrt{-1}$, n ≥ 0, m ≤ |n| and $R_{nm}(x, y)$ is given by Equation 2 [1].

$$R_{nm}(x,y) = \sum_{s=0}^{(n-|m|)/2} \frac{(-1)^s (x^2+y^2)^{n/2-s}(n-s)!}{s!(\frac{n+|m|}{2}-s)!(\frac{n-|m|}{2}-s)!} \quad (2)$$

The Zernike moment is therefore given by Equation 3 [1].

$$A_{nm}(x,y) = \frac{n+1}{\pi} \sum_x \sum_y f(x,y)[V_{nm}(x,y)]* \quad (3)$$

where $x^2 + y^2 \leq 1$ and * is the complex conjugate operator.

Zernike moments were calculated from the pre-processed data after going through a thinning process, which infinitely erodes the objects in the binarized images to a single pixel representation whilst preserving the object shape [1]. Each character was then saved as its own image, and the magnitudes of the Zernike moments were calculated from this. The overall feature subset comprises of 25 rotationally invariant Zernike moments per character, yielding 75 features total, which is a smaller subset compared to raw or binarized and can save on computation time.

For feature selection, the feature set which yielded the best accuracy with each classifier used in this task was selected as such. More information on which features were used on which classifier can be found in both Sections IV and V.

IV. Algorithm Selection and Implementation

This section describes the machine learning classifiers employed for this project. Logistic Regression, Neural Networks, and Convolutional Neural Networks will be discussed in terms of their respective implementation, any hyper-parameter optimization employed, and train-test splits utilized.

A. Logistic Regression

In order to obtain a baseline value accuracy expected when classifying this dataset, a Logistic Regression linear classifier was implemented using the Scikit-learn package in Python [8]. The library function employs a one-vs-rest scheme in multi-class problems, such as this one, with L2 regularization. The linear solver was executed with a maximum limit of 100 iterations. The Logistic Regression classifier was trained on all three feature subsets, with a 75:25 train-test split; The set with the highest validation accuracy was selected to make predictions on Kaggle. The validation and testing accuracies will be discussed in Section V.

B. Neural Network

A Neural Network classifier provides more accurate classification of complex tasks [5] compared to Logistic Regression, and hence was considered for this problem. The Neural Network for this task was implemented from scratch using Python, including numpy libraries. Since the forty classes ranged from 0 to 81, one-hot encoding was implemented for both input and output layers of the NN. Inputs to the NN were normalized. Weights were initialized using a random Gaussian distribution from -0.5 to 0.5. Each neuron in the hidden layers used a sigmoid activation function (see Equation 4).

$$f(z) = \frac{1}{1+e^{-z}} \quad (4)$$

where z is a vector of the neuronal input values multiplied by their corresponding weights. The number of hidden nodes and layers was determined using cross-validation, and resulted in an architecture comprised of 90 neurons and 3 hidden layers. Weights were updated using backpropagation with a learning rate of 0.001, determined through cross validation.

Backpropagation is one of the most important components in an artificial neural network. It was implemented with the stochastic gradient descent procedure described in the course notes [9] to minimize the sum-squared loss function. In the implementation, a helper function was created to calculate the matrix of deltas. The vectors of deltas for the neurons in every layer, except the input layer, were calculated using Equations 5 and 6 [9] and were stored in matrix format as the Neural Network class attribute. Since the vector of deltas for the last layer was calculated first and appended first to the class attribute matrix, it was apparent that the matrix was in reverse order.

$$\delta_{N+H+1} \leftarrow o(1-o)(y-o) \quad (5)$$

$$\delta_h \leftarrow o_h(1-o_h) w_{N+H+1,h} \delta_{N+H+1} \quad (6)$$

In every backpropagation iteration, the helper function was called to update the matrix of deltas. The matrix was subsequently used in reverse order (i.e. the correct order) to generate the updated matrix of weights. Equation 7 [9] was then applied. This step was repeated for every layer vector (except the input layer), weights' matrix and deltas vector.

$$w_{h,i} \leftarrow w_{h,i} + \alpha_{h,i} \delta_h x_{h,i} \quad (7)$$

The NN was trained on the Zernike moments feature set (with an 80:20 train-test split) for 300 epochs. The termination, or number of epochs, was also determined using cross-validation. All validation and testing accuracies can be found in Section V.

C. Convolutional Neural Network

Image recognition tasks in machine learning favor Convolutional Neural Networks for their ability to learn from complex structured (2D) datasets [7]. The Convolutional Neural Network (cNN) implemented for this project used the Keras deep learning package for Python (https://github.com/fchollet/keras) as the front end and TensorFlow (https://www.tensorflow.org/) as the back end. One main advantage of this software stack was that Keras, a high-level neural networks API, enables fast experimentation while TensorFlow is optimized for rapid computation and is Google Machine Learning Engine compatible. The architecture for the cNN followed a similar architecture to LeCun et al. in [2]. Layers included an input layer, followed by

two convolutional layers, and a max pooling layer with a pool size of 2x2. The two convolutional layers and max pooling layer were repeated four times. The last max pooling layer was followed by one flatten layer, and three dense layers each with 200 neurons. To avoid over-training, one dropout layer with a probability of 0.2 was added after the first max pooling layer, another one added after the second max pooling layer and a final dropout layer with a probability of 0.5 was added after the last dense layer, followed by the final output layer. The architecture for the cNN can be found in Appendix A. From the Appendix, it is noted that an increasing number of feature maps were used in the deeper convolutional layers. The rationale behind this particular design was that the first two convolutional layers were used to look for simple edges while the deeper convolutional layers were used to look for complex shapes, such as circles and curves [10].

To maximize the accuracy, two different sets of data were used to train the exact same cNN model, namely, pre-processed binarized pixel values and normalized raw pixel values (see Table III, and the model with the higher validation accuracy was used for predictions. The cNNs were trained with a Google Cloud Machine Engine with one GPU (Nvidia Telsa k80) for 200 epochs and a batch size of 128. The Validation and testing results (80:20 split) for the two cNN trained will be further discussed in Section V.

V. Validation and Testing

Since the Kaggle performance results are calculated by dividing the number of correct predictions over the total number of test instances, the validation accuracy was modeled in the same manner.

Table I depicts the validation results from running the Logistic Regression classifier on the raw pixel values, the pre-processed data, and the Zernike moments, on 1000 training samples, respectfully. From the table, it was concluded that Zernike moments yielded the best validation accuracy, and this feature subset was used to train the classifier on all samples using a 75:25 train-test split to make predictions on the test set.

TABLE I: Comparing validation accuracies of the three feature subsets using Logistic Regression

Feature Set (1000 samples)	*Validation Accuracy (%)*
Raw pixel values	5.2
Pre-processed values	7.6
Zernike moments	10.4

The Neural Network implemented from scratch was validated for its implementation by training the same data (i.e. a reduced training set of 10 000 using Zernike moments) on the Scikit-learn MLP package, with all hyper-parameters set to the same values. Table II depicts the comparison of validation accuracies for the two classifiers. From the table, it can be concluded that the NN was implemented correctly since the validation accuracies were similar.

TABLE II: Comparing validation accuracies of the Neural Network implemented from scratch to the Scikit-learn package equivalent

Method	*Validation Accuracy (%)*
NN (from scratch)	11.75
NN (scikit-learn)	11.95

While training the Neural Network, the number of nodes and layers, learning rate for backpropagation and number of iterations (i.e. epochs) were determined by cross-validation. Figure 3 depicts the validation accuracy results from cross-validation determination of the number of hidden layers. Note that cross-validation for learning rate, number of neurons per layer, and number of epochs did not show variation in validation accuracies, therefore the hyper-parameters were justified using other resources [8] [11]. From the Figure, it was concluded that 6 hidden layers produced the highest accuracy whilst preventing overfitting.

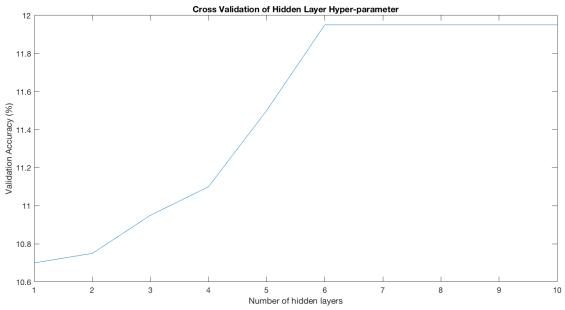

Fig. 3: Validation accuracies across cross-validated determination of hidden layers

Once a reliable Convolutional Neural Network architecture was selected, the cNN underwent training with several hundred epochs in order to determine the appropriate amount of iterations necessary to best predict the test data, based off of a high validation accuracy, but without overfitting to the training data. Figure 4 shows the validation accuracy versus number of epochs.

TABLE III: Comparison of all algorithms implemented for text classification

Method	*Validation Accuracy (%)*	*Test Accuracy (%)*
Logistic Regression	17.000	17.624
Neural Network	13.6	12.3
cNN (binarized data)	93.01	N\A
cNN (raw data)	97.14	**97.35**

Table III depicts the validation and testing accuracies (produced from the entire dataset) of the three classifiers considered for this image classification task. From the table, it can be seen that the cNN, using raw data, performed the best with a testing accuracy of 97.35%. Since the Kaggle leaderboard is only based off of 40% of the prediction results, it is assumed

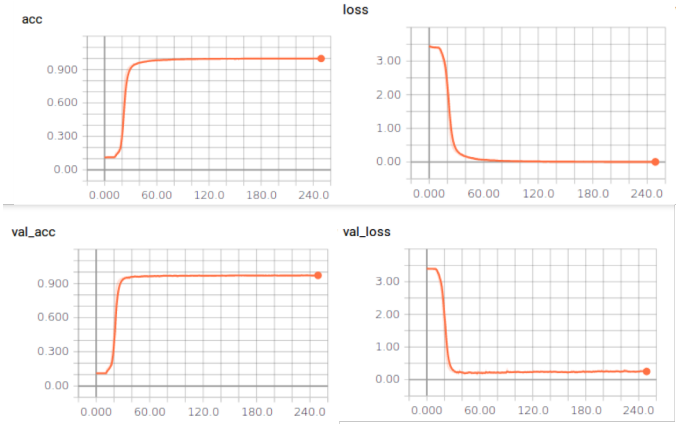

Fig. 4: Training curves for cNN with increasing number of epochs; (top left) training accuracy (top accuracy 99.74%); (top right) training loss; (bottom left) validation accuracy; (bottom right) validation loss

that the remaining 60% is a stratified sample of the entire testing dataset, such that the actual prediction results yield relatively the same accuracy.

VI. Discussion

The performance of each of the classifiers can be analyzed in terms of data-preprocessing, features designed and selected, chosen hyper-parameters (including architecture), optimization techniques, as well as the hardware used to train the classifiers. Each classifiers implemented will be discussed below, in terms of these analysis criteria.

A. Logistic Regression

The Logistic Regression classifier, although not fit for a complex pattern recognition problem [3], performed well above the baseline performance on Kaggle (5.714%) with an accuracy of 17.624%. The "success" of this performance can be attributed to the feature design criteria - using Zernike moments. However, further data pre-processing to account for class imbalance using Synthetic Minority Over-sampling Technique (SMOTE), or under-sampling of the majority class [12], as well as further feature extraction and a more objective feature selection criterion, could have improved the accuracy of this model and the NN and cNN models.

B. Neural Networks

The Neural Network implemented for this project performed poorly, with a testing accuracy of 12.3%. Firstly, the discrepancies between the Scikit-learn MLP package and the implemented NN can be attributed to default parameters within the MLP package which were not integrated into the implementation of the NN. Secondly, the poor testing performance of the NN can be due to many factors. A more extensive cross-validation to determine hyper-parameters could have also improved performance (and potentially produce an accuracy above that of Logistic Regression); however this option was constrained with time since it was unable to run on the GPU available due to code adaptation restrictions. The use of optimization techniques, to speed up train time and allow for more trail-and-error could have also potentially improved performance. It was concluded from the performance of the Neural Network, that this classification task using image-based data was too complex for this type of classifier.

C. Convolutional Neural Network

The Convolutional Neural Network outperformed both the Logistic Regression and Neural Network classifiers, and surpassed the cNN baseline performance on Kaggle (53.925%) with a test accuracy of 97.35%. The performance of the cNN can be attributed to many factors including an appropriate train-test split (better results were seen using an 80:20 split versus a 90:10 split), as well as a strong architecture and hardware which was able to handle the computationally expensive training. The architecture featured several convolutional layers with increasing feature map values, allowing for less noise to be captured in the feature extraction during the first few convolutional layers [2], while deeper layers can extract further shape information. Max pooling layers in the architecture allowed for small enough images to capture the characters without cutting them out altogether. The combination of convolutional layers and max pooling layers, repeated, produced a deep enough architecture to detect the characters. The use of dropout layers to reduce overfitting also attributed to overall performance [13], as well as using ReLu activation versus sigmoid activation (ReLu yielded better accuracy). It is worth noting that although a better accuracy was expected from the processed images, the architecture was tuned on raw images, hence producing better results. Ultimately, the use of a Google Cloud Machine Engine GPU sped up training time and allowed for more trial-and-error to increase performance.

VII. Conclusion

The purpose of this task was to classify modified MNIST image-based data into forty unique classes using a baseline learner, Logistic Regression, a fully implemented Neural Network, and a Convolutional Neural Network. The Convolutional Neural Network performed the best with a testing accuracy of 97.35%. Its success can be ultimately be attributed to a strong architecture, and hardware which can handle computationally expensive training.

VIII. Statement of Contributions

Below are the project contributions from each author.
Victoria Madge: I was responsible for the majority of the report writing, as well as contributing to cNN implementation, NN hyper-parametrization, and some data pre-processing.
Fares El Tin: I was responsible for data pre-processing, feature design, Logistic Regression, NN hyper-parametrization, as well as contributing to cNN implementation.
Alan An: I was responsible for NN implementation, as well as cNN implementation. I also contributed to the report through writing subsections on cNN and NN.

With this, we hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] O. Due Trier, A. K. Jain, and T. Taxt:, "FEATURE EXTRACTION METHODS FOR CHARACTER RECOGNITION–A SURVEY," *Pattern Recognition*, vol. 29, no. 4, pp. 641-662, 1996.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2292, 1998.

[3] K. Deng, "Chapter 4 Logistic Regression as a Classifier," Carnegie Mellon University, 1998.

[4] X.-C. Yin, X. Yin, K. Huang, and H.-W. Hao, "Robust Text Detection in Natural Scene Images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 5, pp. 970-983, 2014.

[5] M. D. Garris, C. L. Wilson, and J. L. Blue, "Neural network-based systems for handprint OCR applications," *IEEE Trans. Image Process.*, vol. 7, no. 8, pp. 1097-1112, 1998.

[6] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," IJCAI Int. Jt. Conf. Artif. Intell., pp. 1237-1242, 2011.

[7] Y. LeCun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time-Series." 2013.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *JMLR*, vol. 12, pp. 2825-2830. 2011.

[9] J. Pineau, "COMP 551 - Applied Machine Learning Lecture 14: Neural Networks," 2017.

[10] J. Brownlee, "Object Recognition with Convolutional Neural Networks in the Keras Deep Learning Library - Machine Learning Mastery", *Machine Learning Mastery*, 2017. [Online]. Available: https://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/.

[11] "How to choose the number of hidden layers and nodes in a feedforward neural network?", *Stats.stackexchange.com*, 2017. [Online]. Available: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

[12] J. Brownlee, "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset - Machine Learning Mastery", *Machine Learning Mastery*, 2015. [Online]. Available: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929-1958, 2014.

APPENDIX A: CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE