COMP551-001 Kaggle Competition: Identifying Largest Handwritten Digit In An Image

Kaggle Team: LearnItAll

Anand Kamat
McGill ID: 260773313
Email: anand.kamat@mail.mcgill.ca

Dipanjan Dutta
McGill ID: 260779557
Email: dipanjan.dutta@mail.mcgill.ca

Xinyue Yu
McGill ID: 260638508
Email: xinyue.yu@mail.mcgill.ca

***Index Terms*—Classification, CNN, NN, SVM, Logistic Regression, KNN, Bounding Box, Contour Extraction.**

I. Introduction

In this competition, we are given a variant of the MNIST dataset where there are multiple handwritten digits in an image. The problem statement asks to identify the largest digit in every image. The largest digit in every image is the digit that has the highest area. The approach we took for solving the problem is as follows :

1) We took logistic regression and linear Support Vector Machines as the baseline learner.
2) A fully connected feed forward neural network. This has been coded by hand without support of direct neural network libraries.
3) A k-Nearest neighbor algorithm.
4) A Support Vector Machine with RBF kernel.
5) A convolutional neural network.

We created the above models and tested on validation data, that we created by splitting the training dataset 80-20. On getting the best hyper parameters for the above algorithms, we found out the convolutional neural networks performed best on the validation data. Thus we trained the CNN model on the whole dataset and predicted on the test data. The best accuracy we got was 93.76%.

II. The Dataset

The MNIST dataset is a database of handwritten digits that is widely used in the field of machine learning to familiarise with pattern recognition and learning techniques. The digits in this dataset have been size-normalized and centered.
The dataset provided in the Kaggle competition is a modified version of the MNIST dataset. Here, each image is of size 64x64 and has multiple digits, each one scaled to 40/60/80/100/120 percent of their original sizes. These images also have the digits rotated in a random orientation and every image has some noise in it. There are 50000 images for training, each image label is the largest digit (in terms of area) in the image, and 10000 images, unlabeled, as test data.
The dataset is a flat table of 4096 features, one column for every pixel in the image. Each pixel value is between 0 (black) and 255 (white).

Figures 1 and 2 show some sample image representations of the MNIST and the modified MNIST dataset respectively.

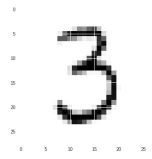

Fig. 1. MNIST Data Sample

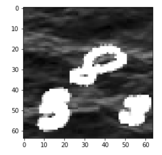

Fig. 2. Modified MNIST Data Sample

III. Feature Design

The image data was provided in the form of an 64x64 dimension array. We took several different approaches to preprocessing the dataset before training the models which have been presented the following sections.

A. Noise Reduction

All the images were presented with a background noise upon which the numbers were placed. It made sense to remove this noise from the background since the new image would only contain varying values where characters were present. This would significantly improve the accuracy of the classifiers. Several methods were used to filter out the noise in the data. The following filters and the smoothened image is shown below:

Gaussian Filter is one of the most common filters used to smooth out the noise in the image. Here, filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them to produce the output array. The openCV package provides direct methods to apply various filters including the Gaussian filter.

Median Filter is another tool used to smoothen images. The median filter run through each element of the image and replace each pixel with the median of its neighboring pixels (located in a square neighborhood around the evaluated pixel).

Pixel intensity threshold was also applied to clear the noise from the image. A noticeable trait in all images was that the characters had a uniform pixel value significantly higher than the rest of the image. Simply clearing all pixels with a value less than the threshold should clear the noise. We set the threshold to 240 and the results were very encouraging. We decided to proceed with the dataset processed using this approach.

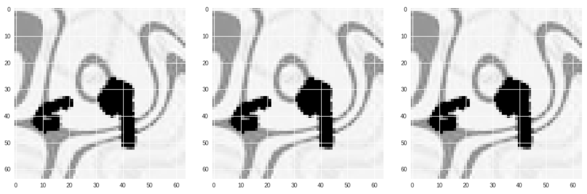

Fig. 3. From the left, Images without a filter, median filter and Gaussian filter

B. *Extracting the largest image*

Although after smoothing the images created clearer and more distinguishable images, the classifiers were still expected to distinguish the different images on top of recognizing the handwritten digit. Linear classifiers such as SVM and Logistic Regression would fail to perform as their simple architecture would fail make this distinction efficiently. We can expect non-linear classifiers such as Neural Networks (NN) or Convolutional Neural Nets (CNN) to be able to handle such complex tasks with more hidden layers but providing a simplified dataset would naturally ease and improve the training. Hence, it was decided that extracting the character that was largest in size and remodeling the dataset would be the logical next step. We had two significant approaches to processing this.

Manually Recognize Biggest Image: The methods involved extracting the largest image by finding the mask of the image and finding the areas rectangles encompassing the digits. The algorithm used simple straightforward steps using the basic functions provided in *ndimage* package. This method proved to be fairly successful in finding the largest image, however there were certain cases where the character extracted was the largest character in the image. This may be due to the fact the rectangle it assumed wasn't as closely bound to the image as needed. For this reason we attempted to use openCV to extract the largest character.

Extracting using OpenCV: Using OpenCV, the process became a lot more efficient. The pre-existing methods to find contours creates a tight bound box around the digits. The dataset created using this method showed no errors extraction based on the size of the characters. There however, was another issue with the dataset. There were several cases where either the characters were graphed on top of each other or were in a form which couldn't be seen as a single digit. One such example is observable at the 80th entry in the training set, shown in the figure. The labels for such examples were set to match the character which was clearly visible as a number. To handle this complication, we dropped these samples from the training dataset to improve accuracy but not the testing dataset.

The next phase to pre-processing was to resize the image to a 28x28 array. This creates a standard size necessary to train the classifiers. Padding was one such method to resize in order to not disrupt the scale of the image. The final step before training the classifiers was to flatten the images to one dimension arrays.

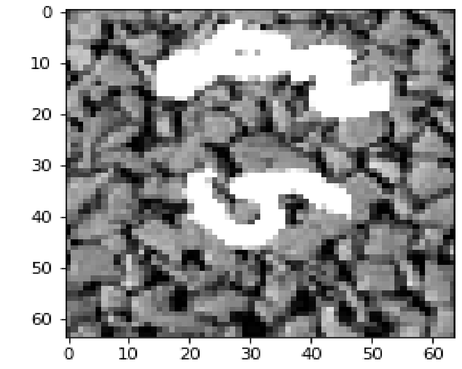

Fig. 4. Indistinguishable image

IV. Algorithms

A. *Logistic Regression*

The central mathematical concept that underlies logistic regression is the $logit$ the natural logarithm of an odds ratio. The simplest logistic regression is of the form :

$$logit(Y) = naturallog(odds) = ln\left(\frac{\pi}{1-\pi}\right) = \alpha + \beta X \quad (1)$$

$$\pi = Probability(Y = outcome|X = x) = \frac{e^{\alpha+\beta X}}{1+e^{\alpha+\beta X}} \quad (2)$$

where π is the probability of the outcome of interest or event, α is the Y intercept, β the regression coefficient, and e = 2.71828 is the base of the system of natural logarithms. According to Equation 1, the relationship between logit(Y) and X is linear. Yet, according to Equation 2, the relationship between the probability of Y and X is nonlinear. For this reason, the natural log transformation of the odds in Equation 1 is necessary to make the relationship between a categorical outcome variable and its predictor(s) linear.

B. *Support Vector Machines*

The support vector machines (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995) require the solution of the following optimization problem:

$$\min_{\mathbf{w},b,\xi} \ \frac{1}{2}w^T w + C\sum_{i=1}^{n} \xi_i \quad (3)$$

$$\text{subject to } \ y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Support Vector Machines find a linear separating hyperplane with the maximal margin in the higher dimension space. C is the cost or penalty term for misclassification. The kernel function $K(x_i, y_i) = \phi(x_i)^T\phi(x_j)$ is used maps the input to a different dimension. The two types of kernel functions we used are :

- Linear kernel : $K(x_i, x_j) = x_i^T x_j$
- Radial Basis Function (RBF) kernel : $\exp(-\gamma \left\| x_i - x_j \right\|^2), \gamma > 0$

C. K-Nearest Neighbors

The k-nearest neighbor classifier is commonly based on the Euclidean distance between a test sample and the specified training samples. Let x_i be the input sample with p features $(x_{i1}, x_{i2}, \dots, x_{ip})$. The Euclidean distance between samples x_i and x_j $(j = 1, 2, \dots, n)$ is defined as :

$$(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2} \tag{4}$$

Classification typically involves partitioning samples into training and testing categories. During the training process, we use only the true class ω of each training sample to train the classifier, while during testing we predict the class $\hat{\omega}$ of each test sample. It warrants noting that kNN is a "supervised" classification method in that it uses the class labels of the training data.

D. Feed Forward Neural Network

Artificial Neural Network (ANN) is inspired by biological neuron has the capability of recognizing patterns effectively. Among all the neural networks, feed forward neural networks with back propagation has the highest rate of convergence. The neural network model can be described as a series of functional transformation. Activations are defined as:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{5}$$

where j = 1, 2, ...,M and the superscript (1) indicates the corresponding parameters are in the first layer of the network. Each of the activation is then transformed using a differential, non-linear activation function h(.) to give

$$z_j = h(a_j) \tag{6}$$

These quantities correspond to the outputs of the basis functions, that in the context of neural networks are called hidden units. The activation functions mainly used are sigmoid, tanh and relu. Example of the sigmoid activation function is shown:

$$y_k = \sigma(a_k) \tag{7}$$

$$\sigma(a_k) = 1/(1 + \exp{(-a)}) \tag{8}$$

Backpropagation is a method used to calculate gradient that is needed to calculate the weights used in the network. Backpropagation is very common in deep neural networks. The gradient calculated is usually that of an error function (loss function).

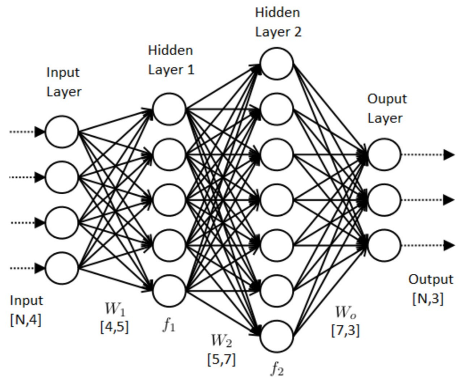

Fig. 5. Feed Forward Neural Net Architecture

E. Convolutional Neural Networks

The problem with regular feed forward neural networks is that it does not scale well with images. For example, an $200x200$ RGB image would have $120,000$ weights for the neurons in the first layer, which can lead to overfitting and enormous space and time complexity.
Convolutional Neural Networks (also called CNN or $ConvNet$) consider the fact that the input is an image and they implement the architecture in a more sensible way. In particular, unlike a regular neural network, the layers of a $ConvNet$ have neurons arranged in 3 dimensions: width, height, depth ($depth$ refers to the depth of the activation volume, not the depth of the entire network). We usually use four main types of layers to build a $ConvNet$ architecture:

- Convolution Layer
- Pooling Layer
- Dropout Layer
- Fully-Connected Layer

1) Convolution layer: When dealing with images, it is impractical to connect neurons to all neurons in the previous layer. Instead, each neuron is connected to only a local region of the input volume. The spatial extent of this connectivity is a hyper parameter called the receptive field of the neuron (or the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. The connections are local in space (along width and height), but always full along the entire depth of the input volume.

- Spatial arrangement : The depth of the output volume is a hyper parameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- The stride is the number of pixels by which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3

or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
- Sometimes it will be convenient to pad the input volume with zeros around the border. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

2) Pooling Layer: It is common to periodically insert a Pooling layer in-between successive convolution layers in a $ConvNet$ architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

3) Dropout Layer: Because a fully connected layer is prone to overfitting, we use dropout technique to regularize. At each training stage, individual nodes are either "dropped out" of the net with probability $1 - p$ or kept with probability p, so that the network is smaller in size; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights. By training all the nodes with all the training data, dropout reduces overfitting, improves training time and makes the model more robust.

4) Fully connected layer: This layer basically takes an input volume (whatever the output is of the convolution or $ReLU$ or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. The way this fully connected layer works is that it looks at the output of the previous layer and determines which features most correlate to a particular class.

V. Methodology

A careful and systematic approach was adopted to make sure important steps were not overlooked. Once we had the pre-processed data generated we moved towards training the classifiers. Before training the dataset was split into a training set with 80% as training set and 20% as the validation set by randomly sampling samples from the pre-processed dataset. The following classifiers were trained and cross-validated on these datasets.

Linear SVM: The hyper parameter for the Support Vector Machine (SVM) classifier is *penalty*. We tried various values of penalty such as 0, 0.01, 0.1, 1, 10, 100 to train the model and tested the performance on the validation set. The best validation accuracy was 91.5455% when the penalty was 10.

Logistic Regression: We aimed at optimizing the classifier using two hyperparamters: penalty and tolerance for stopping criteria. We trained the model using L1 and L2 regularization. The value of tolerance were selected as to be 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10. The cross-validation gave the best performance with a tolerance of 0.01 with a corresponding accuracy of 73.582%.

Neural Network (NN): The feed-forward neural network was hand-coded with varying hyperparamters for cross validation. The hyperparameters we attempted to optimize for neural nets were activation functions, learning rate and hidden layer sizes. Due to the computational limitation we limited our hidden layer size to 100. We found the best validation accuracy 89.78 when the hidden layer size was 100, activation function used was RELU and the invscaling learning rate. The gradient was calculated on the mean-squared error function for the backpropagation.

K-Nearest Neighbors: K-Nearest Neighbors had just one significant hyperparameter, k specifying the number of nearest neighbors to consider. Being a complex no-linear classifier we limited the k values to 5 values: 1, 4, 7, 10 and 12. The best performance of 89.208% was recorded when k was 10. The model began over-fitting after this.

SVM with RBF Kernel: We also trained the SVM using a RBF Kernel. The kernel transforms the data to a simpler form that the linear SVM classifier is more efficient at classifying. The hyperparameter is the same as the linear case, penalty. The best performance of 92.56% was recorded when the penalty was 10.

Random Forest Classifier:

We also were prepared to test the Random Forest Classifier. However, the classifier was very resource demanding while also needed extensive hyperparameter tuning due to the large number of hyperparameters for the model. Due to these factors, we decided to opt out this classifier.

Convolutional Neural Net (CNN): We did an extensive implementation of various architectures for CNNs. We created CNN using $keras$ framework in Python. We first tried with two layers with two convolutions, a max pooling layer and a dropout layer each. We varied the number of output filters in each convolution from 3 to 64. However, we fixed the kernel size and stride length as 3 and 3 respectively. The activation for the convolution was rectified linear unit (ReLU). The max pooling layer pool size is set to (2,2) with the stride size same as the pool size. We experimented with the dropout rate of dropout layer, with values from 0.25 to 0.5. Finally we added a flattening layer and a fully connected layer of 2000 nodes and a dropout layer with dropout rate of 0.5. Finally we added a softmax layer to classify into one of the ten digit classes. We then ran the architectures with batch sizes of 50,100,200 and 500 with 2,3,5,10 and 20 epochs. However, the best accuracy we found was close to 88% Thus, in order to improve the accuracy, we created a variant of the Convolutional Neural Network inspired by the VGG16 model introduced by Simonyan and Zisserman in their 2014 paper, *Very Deep Convolutional Networks for Large Scale Image Recognition*. The final network is as follows :

- Network Layer 1
  - Convolution layer : 64 output filters, kernel size 3x3, strides of 3 pixels, ReLU activation.

- Convolution layer : 64 output filters, kernel size 3x3, strides of 3 pixels, ReLU activation.
    - Max Pooling layer : Pool size 2x2, stride length of 2
    - Dropout layer : Dropout rate of 0.25
- Network Layer 2
    - Convolution layer : 128 output filters, kernel size 3x3, strides of 3 pixels, ReLU activation.
    - Convolution layer : 128 output filters, kernel size 3x3, strides of 3 pixels, ReLU activation.
    - Max Pooling layer : Pool size 2x2, stride length of 2
    - Dropout layer : Dropout rate of 0.25
- Flattening layer
- Fully connected layer : 2000 nodes, ReLU activation
- Dropout layer : Dropout rate 0.25
- Fully connected layer : 2000 nodes, ReLU activation
- Dropout layer : Dropout rate 0.25
- Fully connected layer : 10 nodes, softmax activation.

We ran this model for epochs with sizes 2,3,5 and 10 epochs with batch sizes of 50,100,200 and 500.

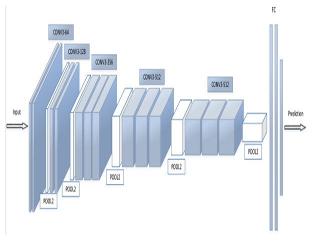

Fig. 6. VGG CNN Model Architecture

Following the preprocessing described in the feature selection section we also attempted to boost accuracy using a concatenated dataset consisting of the pre-processed original MNIST dataset and the modified-MNIST dataset provided for this project. We believed the collaborative dataset would train the model a lot better and gain better results. Testing this theory proved unsuccessful as the MNIST data had all the images of the same scaled size and orientation whereas the modified-MNIST dataset had enlarged, rotated and scaled images. Which is why we didn't see any improvement in performance. In fact the new concatenated dataset showed a validation accuracy of 92% whereas the pre-processed modified-MNIST dataset showed an accuracy score of 93.7 %. The reason for this drop in performance can be explained by the larger number of MNIST data samples as compared to the modified MNIST which skewed the predictions for the MNIST dataset.

VI. Results

The project the gave promising results predicting the handwritten digits. We inferred that the right pre-processing technique factored in significantly in the performance of the models. We observed the first pre-processing method to give a maximum performance accuracy of 88.7% using a deep CNN model. This was due to the fact that while extracting the largest digit, the algorithm makes a few errors which causes the accuracy to plateau at 88.7%. Upon using openCV we observed a huge spike in the accuracy. The efficient extraction algorithm showed an accuracy of 93.7% on the same CNN model. While training the dataset, we began with linear classifiers. The following are the results of 2 such linear classifiers on varying hyperparameters: SVM (Linear Kernel) and Logistic Regression. The row in bold is the hyperparameter with the best result.

TABLE I
Performance of the SVM Classifier

Penalty	Accuracy
0.01	0.67835935
0.1	0.72849438
1	**0.836344**
10	0.817349365
100	0.79474397

TABLE II
Performance of the Logistic Regression Classifier

Tolerance	Accuracy
L1	**0.7079**
L2	0.7041

TABLE III
Performance of the Logistic Regression Classifier

Penalty	Accuracy
1	0.7350090854
4	0.89067232
7	**0.8937644**
10	0.892085605
12	0.891984656

Training the non-linear models was the next phase. We chose classifiers such as the feed forward neural net and K-Nearest Neighbor classifier. The non-linear models perform significantly better than the linear models. The ability to learn complex functions by transforming the data to a simpler dimension is the reason the feed forward classifier is able to converge to a better estimate. Adding hidden layers further improves the accuracy. The knn algorithm although computationally very demanding is able to model the dataset quite efficiently and converges to a very good approximate. However, both the feed forward neural net and the knn model start to overfit when they hyperparameters (number of hidden layers for Neural net and K for knn) are set to be high.

In an attempt to improve upon the linear classifiers we created a SVM model using a RBF (Radial Basis Function) Kernel. The RBF kernel enables the model to transform the data before the training phase. We observe an significant improvement over the SVM model with a linear kernel.

TABLE IV
PERFORMANCE OF THE NEURAL NET CLASSIFIER WITH 100 HIDDEN LAYERS

Activation	Learning	Accuracy
identity	constant	0.741066021
identity	invscaling	0.742378357
identity	adaptive	0.740965072
logistic	constant	0.756612154
logistic	invscaling	0.87007874
logistic	adaptive	0.870583485
tanh	constant	0.869876842
tanh	invscaling	0.86634363
tanh	adaptive	0.86725217
relu	constant	0.878457501
relu	**invscaling**	**0.886230567**
relu	adaptive	0.874318595

TABLE V
PERFORMANCE OF THE KNN CLASSIFIER

K	Accuracy
1	0.869977791
4	0.89067232
7	**0.89349889**
10	0.892085605
12	0.891984656

TABLE VI
PERFORMANCE OF THE SVM CLASSIFIER WITH RBF KERNEL

Penalty	Accuracy
0.01	0.756612154
0.1	0.8463557439
1	0.906319402382
10	**0.92560064607**
100	0.9162123965

The CNN model inspired from the VGG network was the best performing with the high accuracy of 93.7% on Kaggle test data when we ran for 5 epochs with batch size of 100. CNN works really well on image data and is able to find patterns such as borders, shapes and pixel distribution very well. It hence, makes sense to create a deeper CNN network which showed significant improvement over all other models explained in this paper.

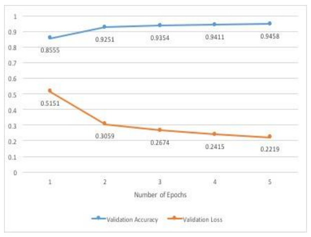

Fig. 7. CNN Model Converging at each Epoch

VII. DISCUSSION

The methodology adopted was very effective in producing efficient results however there is scope for significant improvement. Some of the ways our methodology became a success are:

- Efficient pre-processing was an important part of the project which, as proven turned out to be a very important factor.
- Analyzing a wide range of classifiers (linear and non-linear) to provide a elaborate analysis
- Further improving the poor performing linear classifiers by using kernel functions (such as RBF kernel)
- Using a VGG scheme on a CNN model which is very efficient to analyze image data giving very accurate predictions

Despite a adopting a solid straight approach, there are ways our methodology can be improved upon:

- Time was a big factor slowing us down. The models were implemented very simply and without using GPUs. This is one area we can improve upon.
- The accuracy as observed on Kaggle wasn't as good as we had expected. It could have been significantly improved if we tried even more architectures variations of the CNN network.

Although the results provided by our model was efficient, there is plenty of scope of improvement. In the future we would like to attempt to experiment with various CNN architectures and hope to improve the accuracy of the model. The success of the model encourages its application to real world issues where CNN networks enable recognizing the patterns in images. We hope to extend this project and make it better.

STATEMENT OF CONTRIBUTION

The project was a combined effort of all the members. The work done by any member was supervised by the others and provided with positive feedback or constructive criticism if needed. Dipanjan spearheaded the report and supervised any editing required to it. The task of writing of the report was divided amongst all three members. Anand handled the general overview of the programming component of the project with the help of Selina and Dipanjan. He handled the preprocessing of the data and defining the classifiers required for the project and writing the source code for the feed forward neural network. He also worked on optimizing the CNN model to improve the accuracy of the model on the test set along with Dipanjan. He also wrote several sections of the report. Selina took responsibility of tuning the hyperparamters for the every classifiers with data pre-processed in different ways. She also participated in the report writing process. The project was truly a collaborative effort and every team member has actively participated in every step of the project.

References

[1] Sushmita Das, Aleena Swetapadma and Chinmoy Panigrahi *Building Occupancy Detection Using Feed Forward Back-Propagation Neural Networks*, International Conference on Computational Intelligence and Networks, 2017.

[2] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer Publications, 2006

[3] https://techblog.viasat.com/using-artificial-neural-networks-to-analyze-trends-of-a-global-aircraft-fleet/

[4] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, *A Practical Guide to Support Vector Classification*, Department of Computer Science, National Taiwan University, Taipei 106, Taiwan

[5] Chao-Ying Joanne Peng, Kuk Lida Lee, Gary M. Ingersoll, *An Introduction to Logistic Regression Analysis and Reporting*

[6] http://cs231n.github.io/convolutional-networks/

[7] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[8] https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

[9] https://keras.io