# CS 412 Project: Why we use Convolutional Neural Networks

Vineet Patel, Carlos McNulty

May 2019

## The shortfalls of traditional Neural networks

The beauty of neural networks (also known as multi-layer perceptron) was that it would, given enough layers and node, be able, over a period of time, to detect those meaningful features. Neural networks had the ability to learning features that showed themselves in a non-linear fashion in high dimensions, something a human would never pick up on. Yet nerual networks have a major drawback: their computational intensity. Having multiple layers and multiple nodes in each layer, with each node connect to the nodes of the layer before it and ahead of it, a full connected networks introduces thousands of computations for a single forward propagation. And then, on top of that ,you have back propagation (using gradient descent), all preformed hundreds of times over the data-set.

A prime example that showcases the short fall of Neural networks is image classification. Under the traditional neural network, the image would be consider an input with hundreds of variables (one of each pixel). Those hundreds of variables would be feed into hundreds of input nodes, each of which would be fully connected with the next hidden layer. The effectiveness of the Neural network, as with any machine learning model, is the ability to pick up features. For images, the features are not simply the individual value of pixels, but all the groups that the pixels forms . Features in an images are formed from patterns that neighboring pixels form. While there is a decent probability that the traditional neural network will pick on these features, there is no guarantee, not to mention the extremely long computation time that will be required.

# Introduction of the Convolutional layer

The introduction of the convolutional layer helps offset the flaws of a traditional neural network. Input into a convolutional layer is a 2D array of data. As you can tell, this makes it more suitable for image classification from the get-go. Convolutional layers are composed of feature maps (a mapping for features extract at each location). A unit in the feature map is typically calculated from multiple inputs. The key here is that these multiple inputs all have the property that they are right next to each other. The feature map is constructed by using a *filter* (other terminology used are kernels, feature detector) which is typically a matrix of $n \times x$ dimensions. The matrix is filled with weights and goes through the input field (which is a two-dimensional array) performing the dot product on the local region (of size $n \times n$. The output matrix of these dot products is the feature map (also known as the convolutional feature).

The reason why the Convolutional layer is so important is that it is able to extract features from the original image, by considering pixels group together spatially. The basic convolutional layer described above allows to us to "extract elementary visual features such as oriented edges, end-points, corners", such that those those features can be "combined by the subsequent layers in order to detect higher-order features" [1]

Another important part of the convolutional neural network is the pooling layer. The goal of the pooling layer is two fold. The first is to reduce the size of the input. So having a pooling size of 2 $times2$ would

# Implementing a Convolutional Neural network

The interface of a CNN is not that different from a basic implementation of a neural network. Once of the most widely used Python packages for implementing CNNs is Keras [2] which uses TensorFlow [3] as a backend. GPU support was enabled to speed up the process. In general, for a basic model, it takes about 200+ ms to do one forward propagation (along with back propagation) from the input of one image. Keras allows one to make sequential models (where layers are stacked on top of each other), though it is possible to make parrellel models (where data goes through multiple layers at the same time in different channels). Throughout this report we will be use the sequential model, which seems to work best for having 1 GPU.

# Dataset

For our dataset, we should 5 categories of dogs: dalmaions, geraman sheperds, huskies, hyena dogs, and newfoudlands. There are a total of 4000 images, 3000 of which are part of the training set, with the other remaining in the test set. The images were acquired through ImageNet [4]. ImageNet provided the URLs of sites hosting images and we wrote a script to download them. Issues that had to addressed while collecting the images included sites not responding, error codes, and sites providing files of the wrong mime type. Some sites that did not have the requested file would not return an error code, but still return a custom 404 jpeg image. This was problematic and couldn't be handled with a script. We had to manually remove every one of these images from the dataset to ensure that they were not included in the final training and testing sets.

The size of the dataset would also be problematic. Our original script would download images and organize them into a folder hierarchy that could be used by Keras to load images in batches and automatically discover the classes. We needed something that could allow us to load the entire dataset, so we stored the dataset in a Hierarchical Data Format (HDF) file using the h5py package. This proved be a much more efficient way to access the dataset.

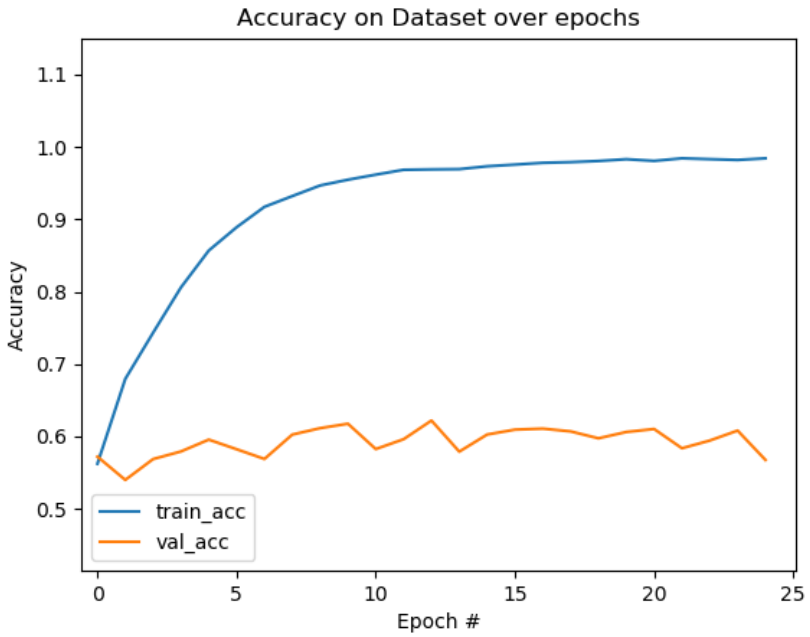
There are instances where an image contains multiple dogs (of the same category), which almost certainly leads to a more challenging dataset. We choose to have 5 categories of dogs to test out how well CNNs really were at feature extraction for images. While the dataset is still a classification problem related to images, but in this instance, a good classifier would need to pick up features pertaining to the actual dogs. As it will be stated later, this problem turned out to be a challenging one.

# Image pre-processing

Image preprocessing is incredibly important, as we want the model to pick on features in the image, not image qualities like aspect ratio or size. Luckily Keras had a easy to use pre-processing model. For all convolutional models, we set a standard size. We also allowed for random horizontal flipping of images when training. We include a sheer range and a zoom range.

# Initial Accuracy

The basic model that we set as the baseline at 1 convolutional layer, followed by a max pooling layer, followed by a dense layer (an normal fully connect neural network layer), all leading to the output layer. Initially, the convolutional layer had 32 filters, each with a size of $3 \times 3$. The pooling layer has a pool size of $2 \times 2$, and uses the Max pooling method. The dense layer had a 128 internal nodes. For all layers, except the last layer, the activation function has been "relu". The last layer is a dense layer with 5 nodes, one of each category. The first measurement we took was training and test accuracy over epochs:

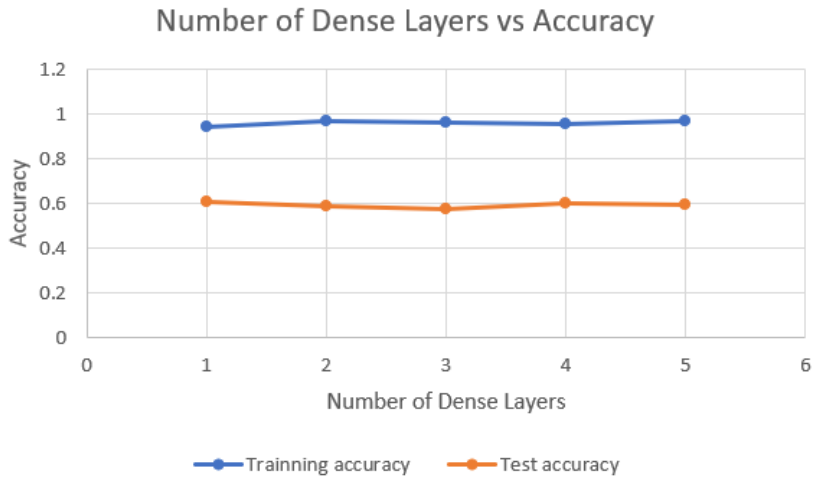



As you see, the training accuracy eventually levels off after 10 epochs. But the test accuracy (labeled val_acc) does not improve along with training accuracy. This is an example of over-fitting, where the model has failed to general features, rather it has successfully picked up on the noise of the training set. It's not to say that the model is completely inaccurate. Since there are 5 categories, randomly guessing categories would give an accuracy of 20%, so in a sense 60% accuracy is better than nothing. But we will try to improve this accuracy and try to prevent over-fitting.

An important thing to note is the time it takes to run the model on the training data. A single step takes at least 200ms, thus going through the entire training set takes around 500s, with at least 5 epochs done. All of this leads to A total runtime of 30+ minutes for a single model. Thus we were limited by the runtime of the model when it came to tuning the model with different parameters.

# Adding More dense layers

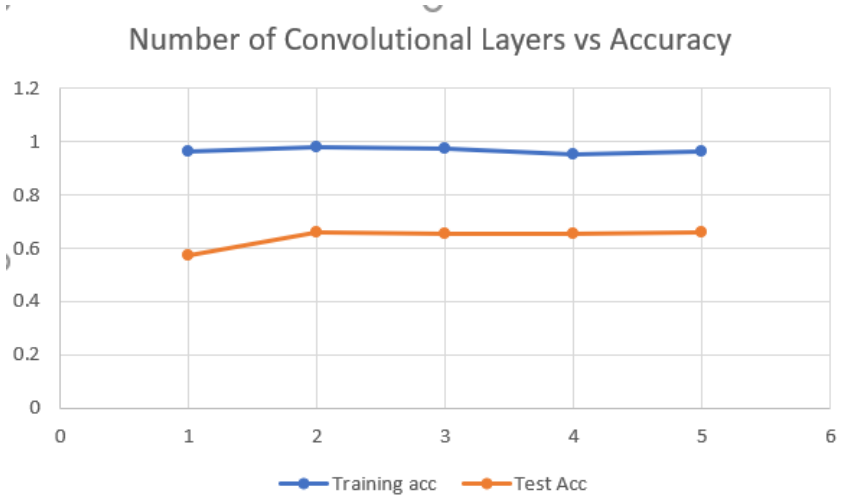
Before we start adding more convolutional layers, we will first see if adding more dense layers will improve the accuracy (of the test set). Each dense layer we add has 128 internal nodes. We took the accuracy values after 10 epochs.




Although the range is rather limited, it is evident that adding more dense layers does not improve test/validation accuracy. The over-fitting still exists.

# Adding Convolutional layers

We decided to fix the number of dense layers at 2 (once again, each dense layer had 128 nodes). Every convolutional layer we added had 64 filters, each with a kernel size of $3 \times 3$. After each convolutional layer, we add a pooling layer (max pooling with pooling size of $3 \times 3$).




It seems that just like dense layers, adding more convolutional layers does not lead to a drastic improvement in test accuracy. Though it is important to note that the overall accuracy of the model did improve, as by having two convolutional layers, we got a test accuracy of 65% which is an improvement of the previous 60%. But, yet the problem of overfitting remains.

# AlexNet implementation

After adding Convolutional layers and normal dense layers, we decided to implemented AlexNet. AlexNet is a famouse convolutional neural network, designed by Alex Krizhevsky, who published an article (along with others) about it [5]. Alex was a winner of the ImageNet Large Scale Visual Recognition Challenge. AlexNet achieved remarkable low error rates (at that time) by having 5 convolutional layers and 3 fully connected layers. In order to prevent overrfitting they used the regularization method dropout. Dropout essentially randomly selects neuron to be ignored during training. This prevents the weights associated with neurons from becoming too dependent on the features in the training set. At the time, AlexNet was one of the largest convolutional neural networks, with "s 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax" [5].

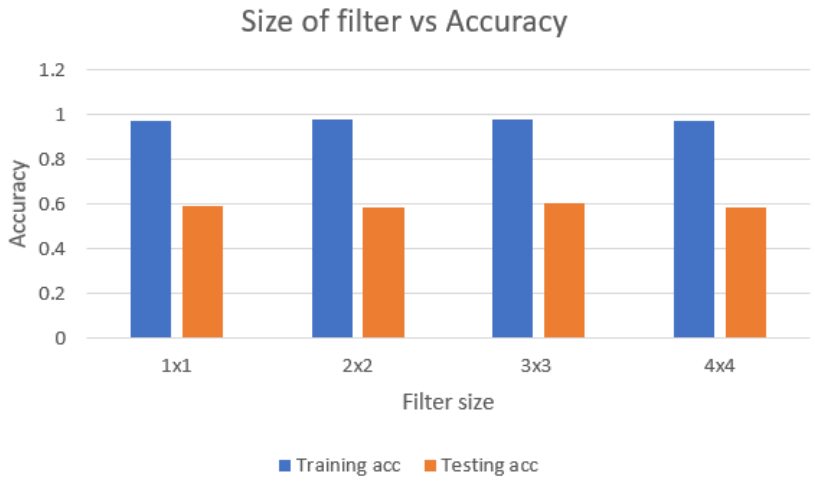
We replicated the AlexNet model in Keras. We added all the layers with the right configuration (for example the input into the first convoluational layer is of size $224 \times 224$). The only difference is that our model is sequential, while the original AlexNet had parallel convolutional layers in the begining.

The results of the AlexNet were better, reaching a testing accuracy of 70%,

the highest yet. A key feature that AlexNet had was having a larger input size (for the images) which may have made it easier for the convolutional layers to pick up on features. It also had significantly higher number of internal nodes in the dense layers (4096 for each vs 128 for our baseline). The only downside to all of this is that AlexNet toke much longer to run (rought 5 times as longer). Overall, one can see why AlexNet revolutionized computer vision in 2013, as it shows the potential for deep convolutional neural networks.

## Other hyper-parameters

We also tried changing a couple of the hyper-parameters in the convolutional layer itself. The first hyperparameter we change was the size of the filter. As a remind, a filter, in the context of convolutional neural networks, it s matrix of weights which may be used to transform the input in order to see if the input resembles a feature. The standard filter size seems to be $3 \times 3$. We keep the square shape of the filter, but gave it different sizes: $1 \times 1, 2 \times 2, 3 \times 3, ....$



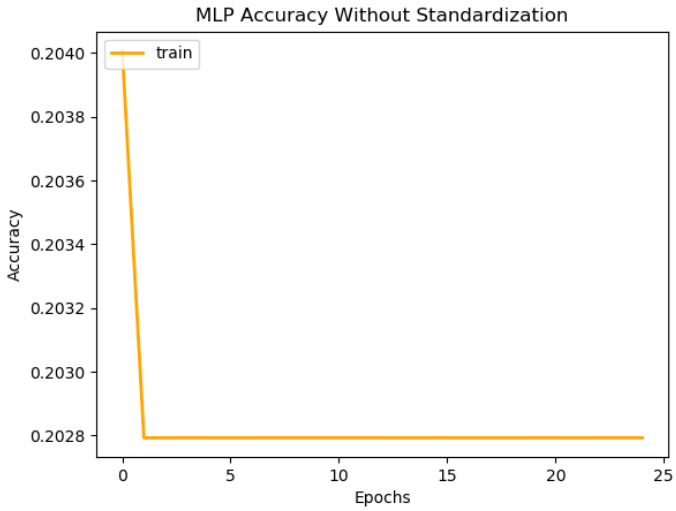

It seems that changing the filter size did not lead to a change in accuracy. The reason behind this may be that the features that the convolution layers pick up (using the filters) do not seems to be that influential when it comes to training on the training data. Thus the size of the filter did not make that much of a difference. This may reflect more apon the image dataset rather than the model itself. The second parameter we tried changing was the input image size. This may be more related to image preprocessing, but it is still influential, as the greater the size, the greater the potential of the convolutional layer to pick up on features.

# Comparsions with a traditional Neural Network (Multilayer perceptron)

Next we decided to see if the convolutional layer even really help. So we decided to use the dataset through a more traditional nerual network.

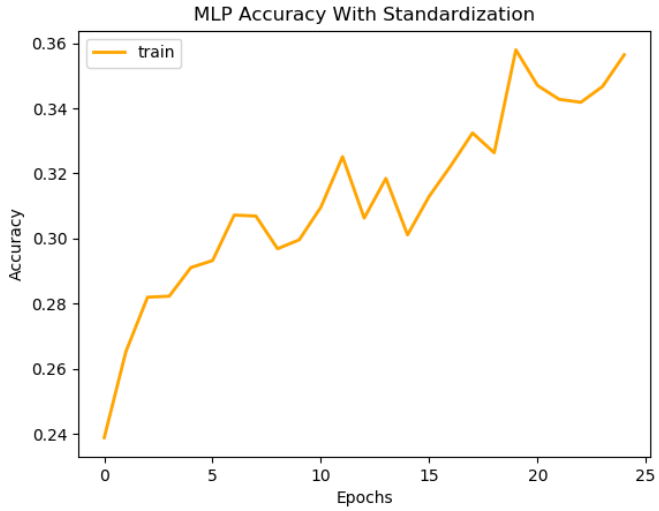
## Image Pre-processing

In order to run on a more flexible range of models, we applied a different image pre-processing approach this time.




MLP model accuracy without standardization.

The first step in pre-processing the images in the dataset was to convert the images to grayscale. After the images were converted to grayscale we standardized the features of the dataset by removing the mean and scaling to unit variance. The training accuracy for the model before applying standardization is 0.21, and after applying standardization is 0.34, as can be seen in the figures.



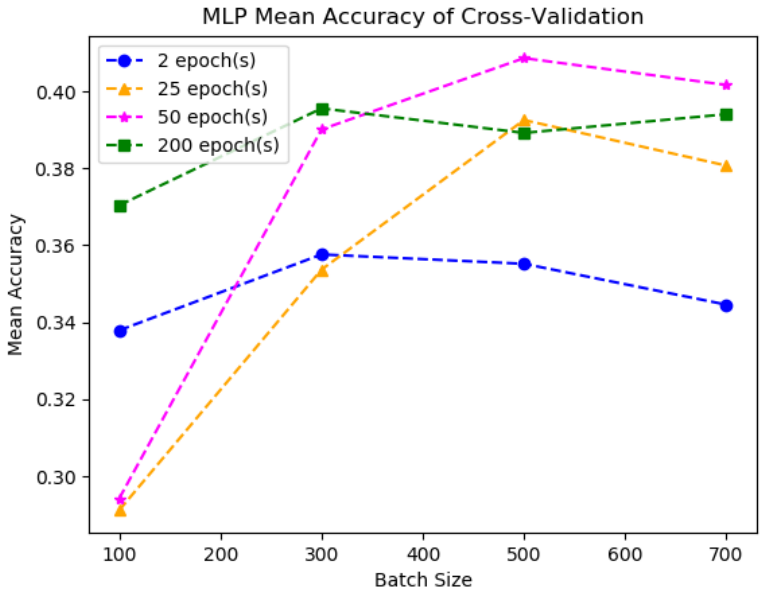

MLP model with standardization.

## Initial Accuracy

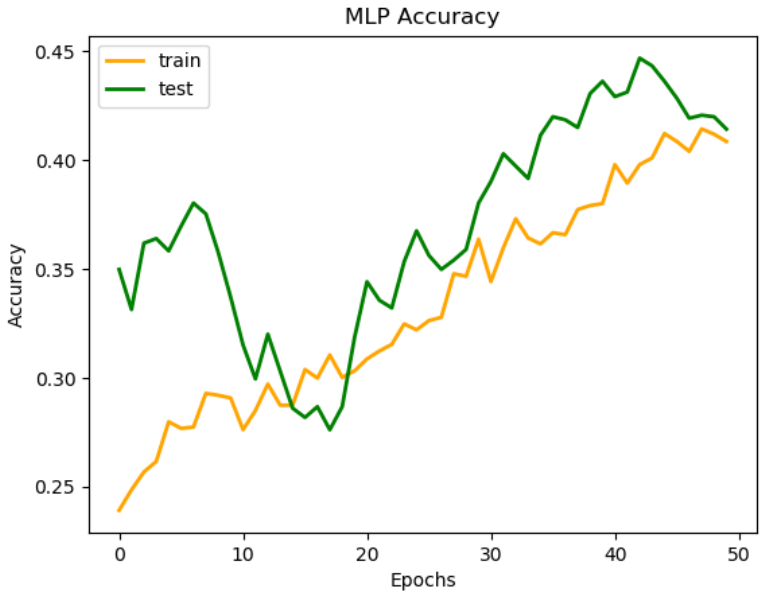
## Hyper-parameters

### Grid Search

To improve the performance of the model we had to perform some hyper-paramater tuning using grid search to find optimal parameter values for the number of epochs and batch sizes. We searched over epochs {2, 25, 50, 200}, and batch sizes {100, 300, 500, 700}, using 10-fold cross-validation as our performance metric. The data from the search shows that a model with a batch size of 500 and 50 epochs with accuracy 0.409 had the lowest cross-validation error. This is nearly double the training accuracy of our model before tuning the hyper-parameters. The accuracy of the models generally increased until the number of epochs reached 200. This may be because the model is training for a greater number of epochs, so as it repeatedly attempts to reduce training error, it begins to start learning noise. It is also clear that increasing the batch size past 500 results in an increase in cross-validation error. Evidence supports that models using large batches tend to converge to sharp minima, which leads to poor generalization [9]. On the basis of this research and the data that we've collected, we'll look no further, and choose a model with 50 epochs and a batch size of 500.




MLP accuracies for models of varying epochs and batch sizes.




MLP accuracies for model.

## Final Accuracy for the MLP Model

The testing accuracy of the model is 0.41. This is far from ideal, but nearly double the training accuracy of our model before standardizing the images, and using grid search

to optimize the hyper-parameters. The final testing and training accuracy can be seen in the last figure. This data shows no signs of overfitting. This is a challenging dataset. The convolutional neural network was able to achieve a superior testing accuracy than our multilayer perceptron classifier.

# Region-Based CNNs

While image classification is an meaningful problem, often times we want to do more than classify images. Sometimes we want to detected objects within the images themselves. Object detection requires not only classification, it also requires finding the specific location of the object within an image. Convolutional neural networks does not natrually lead itself to this problem for multiple reasons. One reasons is that the output is not fixed in size, in a sense that there may be multiple objects to be detected within an image. THe objects may be in different location, with different sizes.

One potential solution solution would be to have a slide-window, where the model would try to classify at every instance within the window. But this is obviously an inefficient solution. This 2014, researchers at the University of California Berkeley (Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik) creates an extension to the Convolutional Neural Network model: Regions with CNN features (R-CNNs) [6]. Their R-CNNs model consisted of three parts.

The first part is category-independent region proposals. In essence they generate region proposals within the image, and then those regions are feed into the convolutional layers (which is the second part). The region proposal that they used was selective search, which looks at the image structure to generate sample regions [7]. Once a region has been proposed, it is feed into a convolutional layer which will try to extract out features. The convolutional layer is standard as far as CNNs, with nothing out of the ordinary. There third part is a class-specific linear SVM used to score how well the features line up with the overall class. Each extracted feature vector, originating from the convolutional layer(s), goes into an SVM trained for each class.

This approach needs to levels of training. The convolutional layer needs to be trained to extract the right features effectively , and there needs to be training on the linear SVMs. Their model leads to substantially higher detection accuracy than other prominent models for that time (2010).

The biggest drawback to their initial approach was the amount of time it took to both train the model and run the model on images. The model would have to classify multiple regions per image, and depending on the region proposal algorithm that could be a substantial amount (the standard selective search prodcues up to 2000 regions).

Also another problem was that selective search (the region proposal algorithm) was not selft-learning.

So the main research (Ross Girshick) decided to build a faster R-CNNs architecture, known as Fast R-CNN [8]. The biggest improvement that Fast R-CNN is that instead of forwarding the regions proposed into the convolutional layer, the model instead processes the image with all the convolutional layers and max pooling layers, and then it uses the convolutional feature map to draw up regions. This means that the image goes through the convolutional layer only once (vs 2000), which leads to significant improvements in performance..

# Conclusions

Overall, it is clear that Convolutional Neural networks are effective. In our testing, we achieved a much higher training accuracy with CNNs. We also achieved a higher testing accuracy with CNNs, though there still remained a sizable gap between between testing accuracy and training accuracy, indicating a case of over-fitting. In our experimentation, we found that changing the hyper-parameters of the CNNs could lead to minor improvements, but nothing noticeable. The highest accuracy we were able to achieve was with the AlexNet implementation, and in our analysis AlexNet's biggest strength of how large of input data points it allowed for. Overall, we feel that in order to improve our model, it may be better for us to simply increase the size of our training set. Often times, a good solution for over-fitting is to simply to increase the amount of data being trained on, with the hope that the model will be better at generalizing features. It is clear that in the realm of image processing, Convolutional neural networks are quite powerful.

# References


[1] Yann LeCun, Leon Buttou, Yoshua Bengio, and Patrick Haffiner *Gradient-Based Learning Applied to Document Recognition*. Proc. of the IEEE, November 1998.

[2] "Keras: The Python Deep Learning Library." *Home - Keras Documentation*, keras.io

[3] "TensorFlow." *TensorFlow*, www.tensorflow.org.

[4] "ImageNet." *ImageNet*, www.image-net.org/.

[5] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik *Rich feature hierarchies for accurate object detection and semantic segmentation*

[7] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M Smeulders *Selective Search for Object Recognition* Technical Report 2012, submitted to IJCV

[8] Ross Girshick *Fast R-CNN* Mircosoft Research

[9] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*