**Flower Classification**

**Dr. Chinnam May, 2018**

**Sujeet, Aniket, Nilesh, Dominic**

**Abstract :**

Convolutional Neural Networks (CNNs) and Clustering are among the more popular techniques used in image recognition and they have been significant in the context of evolution of techniques for image classification. CNNs, as a working definition can be defined as multiple layers of neurons for processing more complex features at deeper layers of the network. They are different in terms of their architecture and behavior than the traditional feed forward neural networks that make them more efficient at image classification. We attempt to apply CNN and autoencoder techniques to a dataset of around 4200 images of five distinct flower species. The challenges for training include some of the images being obstructed by objects, a non-standard dimension specification of imager in terms of width and length, thereby the image edge identification during training resulting in errors. We present comparisons and performances of different architectures in Appendix A, which helped us understand and draw inferences based on various parameters in different networks. Because Alexnet and GoogleNet have been absolutely optimized and considered state of the art architectures of their respective times, variations have not given us any significant differences and hence are neglected. The structured experiments on autoencoders , however provided some interesting insights and also highlighted the limitations in terms of the accuracy autoencoders can achieve.

**Literature Review/Motivation:**

The real pioneering publication, the paper, titled “ImageNet Classification with Deep Convolutional Networks”, has been cited a total of 6,184 times and is widely regarded as one of the most influential publications in the field. In the paper, the group discussed the architecture of the network (which was called AlexNet). They used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories. It was trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories. It used ReLU for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function). It used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions and Implemented dropout layers in order to combat the problem of overfitting to the training data. It trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay. [(source)](https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html)

In the paper titled “Visualizing and Understanding Convolutional Neural Networks”, Zeiler and Fergus begin by discussing the idea that this renewed interest in CNNs is due to the accessibility of large training sets and increased computational power with the usage of GPUs. That’s what a model created in 2014 (weren’t the winners of ILSVRC 2014) best utilized with its 7.3% error rate. Karen Simonyan and Andrew Zisserman of the University of Oxford created a 19 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2The features of the ZF Net model were: conv layers back to back have an effective receptive field of 7x7.As the spatial size of the input volumes at each layer decrease (result of the conv and pool layers), the depth of the volumes increase due to the increased number of filters as you go down the network. The number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth. It worked well on both image classification and localization tasks. The authors used a form of localization as regression (see page 10 of the [paper](http://arxiv.org/pdf/1409.1556v6.pdf) for all details). It used ReLU layers after each conv layer and trained with batch gradient descent.[(source)](https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html)

**Clustering**

Clustering analysis performs a vital role in the scientific research and commercial application. K-means algorithm is an extensively used partition method in clustering techniques. As the dataset’s scale escalates rapidly, it is difficult to use K-means to deal with massively huge data [1]. A parallel strategy that incorporated into clustering method and a K-mean algorithm are suggested. For improving the efficiency of K-mean, dynamic load balance is familiarized. The disadvantages behind the algorithm are the cost of time calculation when the number of cluster is taken excessive [2].

The K-means method has been exposed to be effective in generating good clustering outcomes for many realistic applications. K-means method is well known for its comparatively naive implementation and decent outcomes [3]. However, a direct algorithm of k-means method requires time related to the consequence of the number of credentials and number of clusters per iteration.

The pixels standards are organized row by row, beginning from the lowest raw toward the highest raw. In case of palette driven formats, the listed pixel values represent the color index, while in other cases the blue, green and red components values are grouped to construct the color value [4].

In this phase the color contents of the image are scrutinized to evaluate the dominant color of the background pixels and the target cells [5]. One of the main complications in this effort is the color variability of both the background and the cells pixels. This variability is due to different kinds of paints used to enrich the visual appearance of the blood test samples [6].

To handle this task, the well-known clustering algorithm, called k-means algorithm, was exploited to allocate the pixel’s color around two dominant colors [7]. One of these centroids will be very familiar to the dominant color of the background pixels [8].

The implemented steps could be summarized as follows:

\* Use of K-means clustering for segmentation

\* Convert Image from RGB Color space to L\*a\*b color space The L\*a\*b\* space consists of a luminosity layer 'L\*', chromaticity-layer 'a\*' and 'b\*'. All of the color information is in the 'a\*' and 'b\*' layers.

\* Classify the colors in a\*b\* colorspace using K means clustering.

\* Since the image has 3 colors create 3 clusters.

\* Measure the distance using Euclidean Distance Metric.

\* Label every pixel in the image using results from K means

\* Create a blank cell array to store the results of clustering

\* Extract the features from the segmented image

\* Convert to grayscale if image is RGB, Evaluate the area

\* Load All The Features , Load (‘Training\_Data.mat’) , Put the test features into variable ‘test’

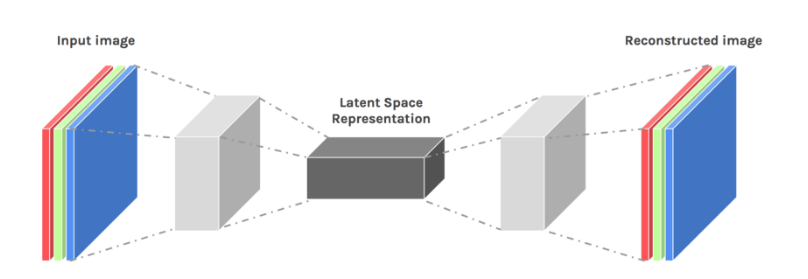
\* Display and Visualize Results, Evaluate Accuracy ,Recognized Area % = 82.87

**Autoencoders**

Auto encoders are artificial neural networks capable of learning efficient representation of the input data called coding without any supervision. This means that training data sets are unlabeled. These coding typically have much lower dimensionality than the input data, making auto encoders useful for dimensionality reduction. More importantly auto encoder’s acts as a powerful feature detectors and they can be used for unsupervised pre-training of deep neural networks. They are capable of randomly generating new data that looks very similar to the training data; this is called as a generative model. For example, in our case we trained auto encoder on picture of five different type of flowers and it would then be able to generate or detect new flowers.

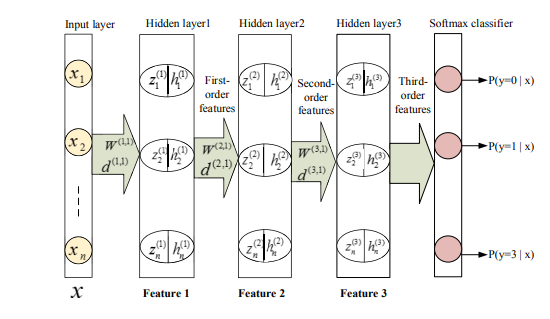
Autoencoders work simply by copying their inputs to their outputs. This may sound like a trivial task but we will see that constraining the network in various ways can make it rather difficult. For instance, we can limit the size of the internal representation or we can add noise to the inputs and train the network to recover the original inputs. These constraints prevent the auto encoder from trivially copying the inputs directly to the outputs, which forces it to learn efficient ways of representing the data. In short, codings are byproducts of the auto encoders which attempt to learn the identity function under some constraints.

Auto encoders (AE) are a family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.



Based on image reconstruction, we presented a novel unsupervised deep learning framework for image recognition (IR). The proposed method takes class label information from training samples into account in the deep learning procedure and can automatically discover the underlying nonlinear manifold structures. The trained auto encoder extracts characteristic features from corrupted/clean flower images and reconstructs the corresponding similar flower images. The reconstruction is realized by a so-called “bottleneck” neural network that learns to map flower images into a low-dimensional vector and reconstruct the respective corresponding flower images from the mapping vectors.

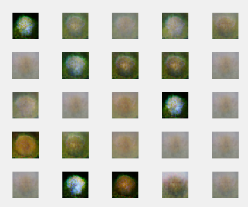
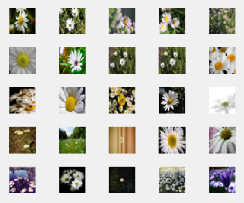
A stacked autoencoder is composed of a plurality of sparse autoencoders. The output of the previous autoencoder is the input of the next autoencoder. The weight of each sparse autoencoder can be obtained by using unlabeled training samples. For the classification task, combining all hidden layers to form a stacked autoencoder and a soft classifier layer that can classify the flowers as desired. The structure of stacked auto encoder with 3 hidden layers and a softmax layer is shown below.



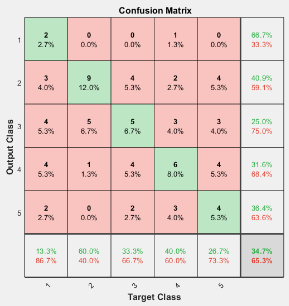
In this paper, we use different number of sparse auto encoders to extract features, and a softmax classifier to output classification result. With Matlab we created a stacked autoencoder model, like the one below, for classifying 5 different types of flowers: daisy, dandelions, roses, sunflowers and tulips. We alter the size and number of the hidden layers. We experiment with different pixel size images. We train on different size data sets. For example, the autoencoder below inputs 30x30x3 pixel size images which equals 2700 inputs. The inputs pass through 4 hidden layers, size 2000, 500, 100, and 20 before going to a softmax layer where the resulting “image” is classified into one of the five classes.



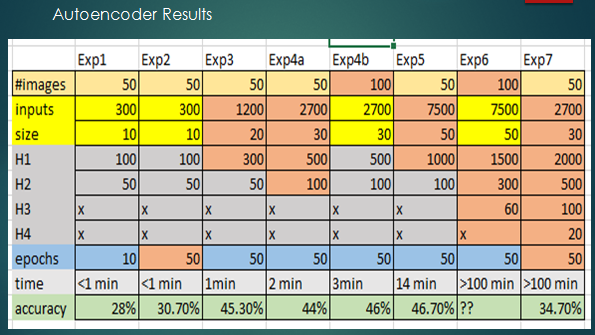
We explore the “effects” of the first layer autoencoder on the original image by reconstructing original image afterwards. The resulting image shows us how well or poorly the autoencoder maintained image features. The reconstruction of those images is often blurry and of lower quality. This is a consequence of the compression during which we have lost some information. However, with higher pixel images, it does well in some of the images, as seen below.



After the first autoencoder, the compressed image then passes through another autoencoder and is then classified. The accuracy achieved in this example was 34.7% as seen in the confusion matrix below.



Although we had high hopes for the previous example since it had 4 hidden layers and 30x30 pixel size, it only had 50 images in the data set to work with. Also, the additional hidden layers did not seem to help. Conducting multiple experiments, with careful tuning of parameters, as seen below, we can do better achieving nearly 50% classification accuracy.

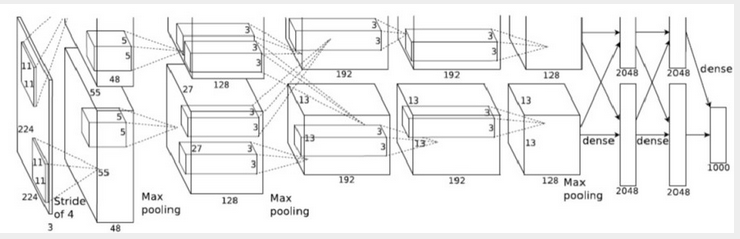


From the above results it can be seen that, larger pixel size image, more epochs, and larger data sets leads to improved classification accuracy. Adding the number of hidden layer and changing the size of hidden layer didn’t have any effect on the accuracy.

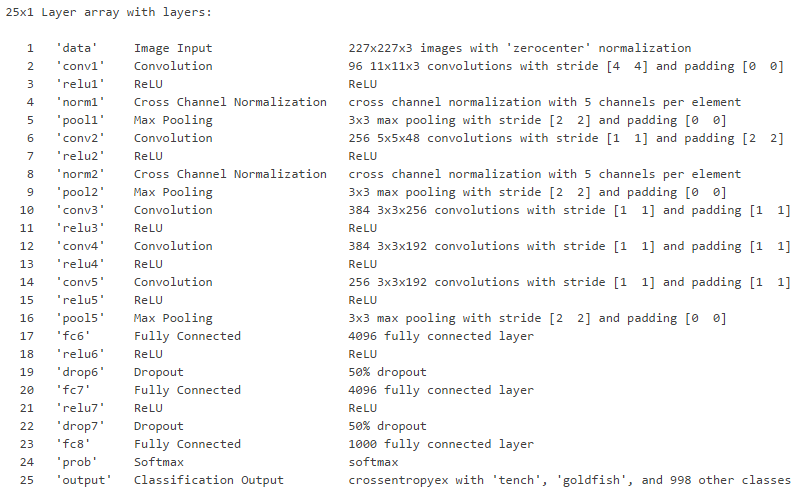
From the gathered experimental data one can conclude that useful nonlinear features can be learnt by (stacked) autoencoders only using large training sets, since these networks don’t recover true underlying regularities in data, but approximate them by complex (multi-parametric) models. Most positive effect from usage of autoencoders on small training sets is possibly connected with redundancy reduction, and not with construction of nonlinear features. Thus, some principal ways of efficiently extracting nonlinear representations of patterns from small training sets or constructing some general representations (similar to the proposed flower images descriptors which are applicable for recognizing different image categories) are required.

**Convolutional Neural Networks**

Convolutional neural networks CNN are architecturally rich models that structure powerful machine learning through dense and intense parceling of the data. We will dive deeply into the mechanics of CNN by looking closely at the Alexnet architecture (shown below). The hope is that through looking at the interior picture of a CNN we can compare it with autoencoders and appreciate the different approaches that are taken by these methods of image classification.

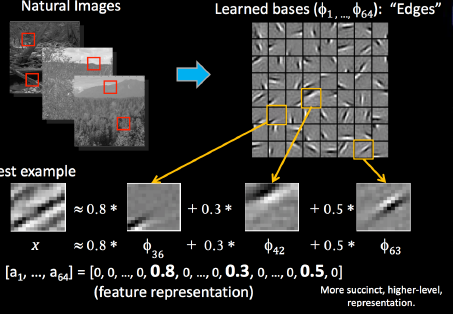


Alexnet

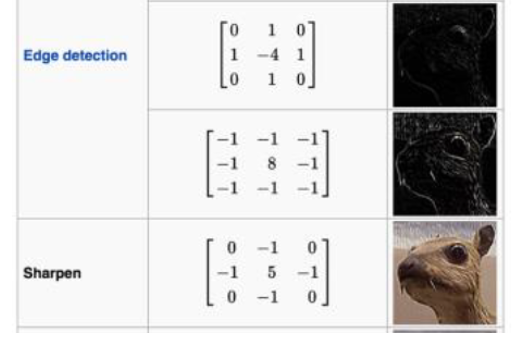


Alexnet is a pretrained convolutional neural network that takes images of pixel size 227x227x3. The image is a 3 dimensional matrix with 227 x227x3 = 154587 entries consisting of values 0 to 255. (Each pixel of the image ranges from 0 to 255 in each of the R, G, B channels).Alexnet has 25 layers which perform convolutions, ReLu, Max Pooling, cross channel normalization and finishes with fully connected layers that classify the images into categories. “The network comprises of 25 layers. There are 8 layers with learnable weights: 5 convolutional layers, and 3 fully connected layers.” Matlab documentation.

After the input layer, layer two, named ‘conv1’ is a layer that applies 96 filters to the input image. Each of these 96 filters is an 11x11x3 convolution with a stride of [4 , 4]. Which means this 3 dimensional filter slides 55 times across the top of the picture (no padding) (55\*4 = 220 pixels) + (the initial position = 5,6 , or 7 pixels down and right from the top left corner) = roughly 227 pixels across the top. The stride then moves down by 4 pixels, as indicated in the stride, and completes the next row. Therefore, each of the 96 filters results in a 55x55 pixel recomposed original image for each color channel (rgb). Each of these 96 filters is doing something unique. Each filter has 11x11x3 = 363 unique weight attributes that it learns. With 96 filters, this is 96x363= 34848 values assigned on this layer for filters. In general, filters at this early layer learn to identify edges, particular colors, etc. similar to what we see below:

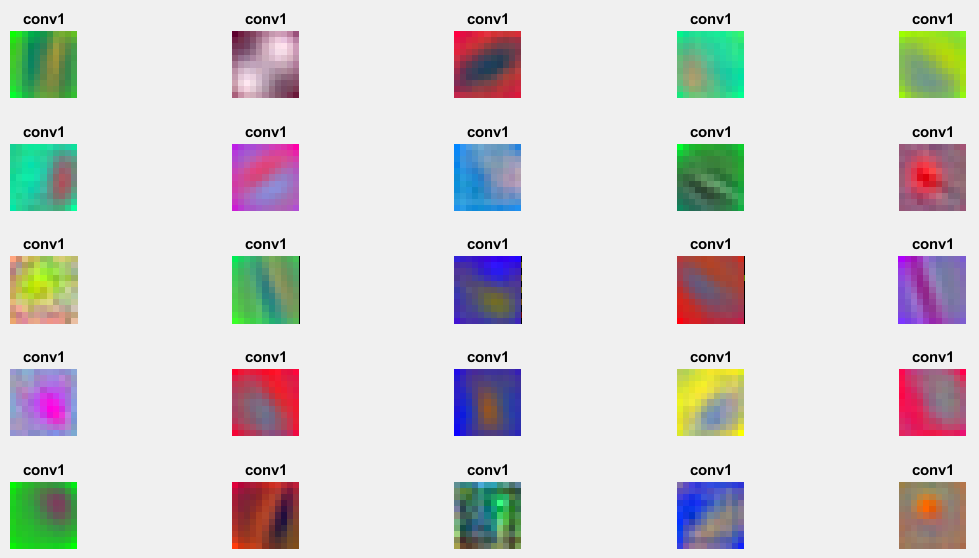


Designing and developing our own filters is interesting, we can observe the changes to our image that would take place. (For example, see the rabbit below)



However, allowing the network to obtain its filter values through training on the images would yield uniquely qualified filters for a general classification task. “The CNN learns the values of these filters on its own during the training process” (Class notes) In a CNN such as Alexnet which is trained on millions of images, its learning develops 96 features/properties/patterns/filters that are helpful for comprehending the nature of the image in that first convolution layer.

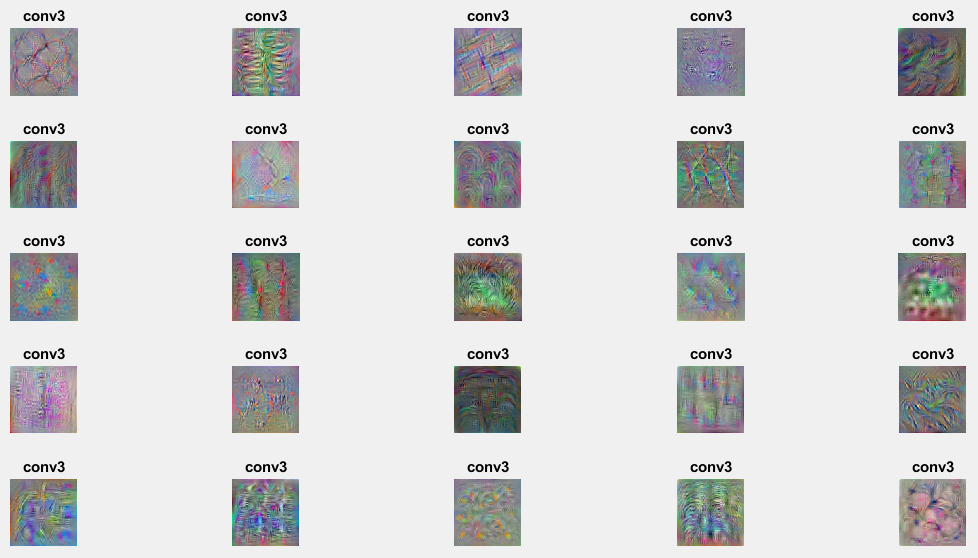
We used DeepDreamImage function in Matlab to “peak” inside the Alexnet CNN at various points in its training. The function visually shows the work of the convolution calculations that Matlab performed on the image matrix. We can see the “uniquely qualified” filters that the CNN crafted through the training process. Each of the images below is an 11x11x3 filters(matrix) with 363 carefully calculated values. Below are 25 features (of the 96) that Alexnet learned in the first convolution layer. Each is a filter that when applied to an image extracts those special unique aspects from an image.



These filters came from Alexnet training on millions of images that were fed to it. We observe color variations, streaks in different orientations, and concentrated blobs. “These images mostly contain edges and colors, which indicates that the filters at layer 'conv1' are edge detectors and color filters. The edge detectors are at different angles, which allows the network to construct more complex features in the later layers.”

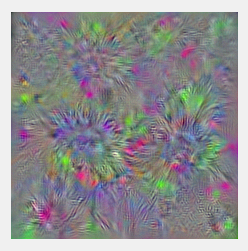
The Alexnet Diagram and layer descriptions above illustrate the mechanics of processing that happen next. After the first convolution our original image which was 227x227x3 is now split into two 55x55x48 “images” by the 96 11x11x3 filters applied to it. These split “images” now pass through relu, normalization, and pooling layers. In the pooling layer, the split “images” shrink to 27x27x48 each. These pooled “images” then goto convolution 2 which imposes 256 filters 5x5x48 filters on them. Each of these filters is crafted by the CNN through training and extract particular features. Each of these filters is populated with 5x5x48=1200 matrix/filter values. With 256 filters, that would be 256\*1200 = 307200 values are trained for the filters at this layer. The convolution layer 2 process splits each of the split “images”, now four, into 13x13x128 “images”. The split images are again relu, normalized and pooled, and then undergo convolution3.

The third convolution layer filters below (the first 25 out of 384) show more sophisticated visual developments.



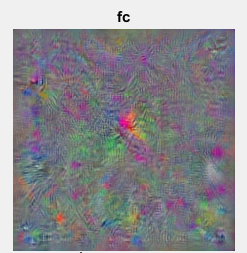
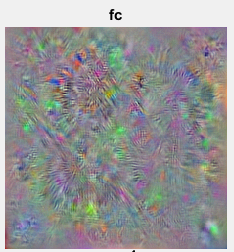
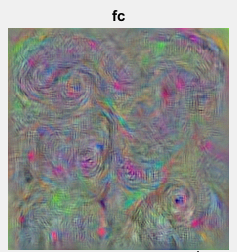
Alexnet has several more layers of convolutions, reLu, pooling, then it comes to fully connected layers to classify objects. Clearly, we see that with large pixel size images, like 227x227x3 (large for the CNN) the hundreds of filters training throughout the layers represents assigning hundreds of thousands, even millions of values to the filter/matrices. The learning task can be overwhelming and slow for the CNN and the processing time for models long. We will see this in the results we discuss later.

Alxnet is trained to classify 1000 objects, Coincidentally, one of them happens to be a daisy (One of the five types of flowers that we have in our classification problem. Therefore, in Alexnet, the fully connected layer (layer 23 in alexnet) has developed its filter to identify “daisy”, as seen below (using the deep dream net function in Matlab). Alexnet would be able to identify pictures of daisy’s with very high accuracy applying this filter.

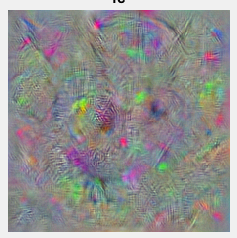
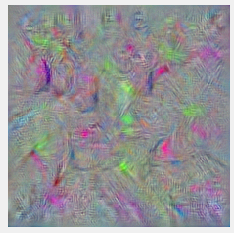


Modifying the last three layers of AlexnetMatlab code to classify our 5 types of flowers (instead of 1000 objects) and running a subset of our Kaggle flower images in this modified Alexnet model (using transfer learning, training on 50 images of each type: daisy, dandelion, roses, sunflowers, tulips) we obtain 74.6% accuracy in correctly classifying our 5 flower types. Using the Deep dream Image function, at the newly modified fully connected layer 23, for classifying our 5 flowers, we see below the corresponding five filters that would be used to classify each type of flower.

Daisy Dandelion Roses

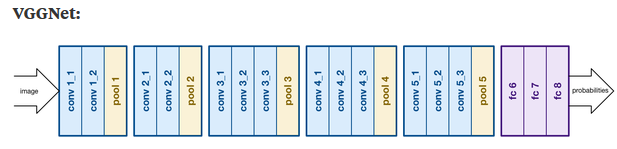
  

Sunflowers Tulips

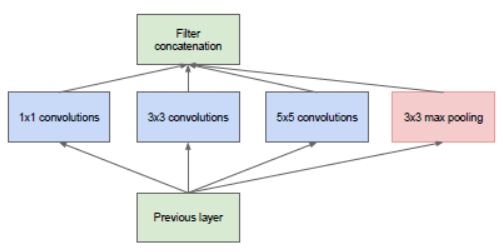
 

These filters are not as sharp or clear as the image that they are trying to classify. The “daisy” filter trained by Alexnet is superior to the “daisy” trained through transfer learning. Training on more images should improve these flower filters. However, it may also be that the quality/diversity of the Kaggle images may make it more difficult to make a clear filter. Images that Alexnet may have been trained on may have been more uniform or cleaner. Yet, still with Roses, for example, we can make out definite swirls that are consistent with roses, and we also see spikey bursts that are consistent with dandelions. Even with these “poor” classification filters, transfer learning through alexnet obtains good accuracy (74.6%).

The good results achieved are in a great part due to the architectural design of Alexnet as well as its design parameters. A CNN’s architecture, as well as the determined hyperparameters are key to potentially improve the accuracy in classification. Alexnet and many other famous architectures, including VGGNet (below) are styled to run many layers in series. And like all architectures, they learn, build and improve upon one another.



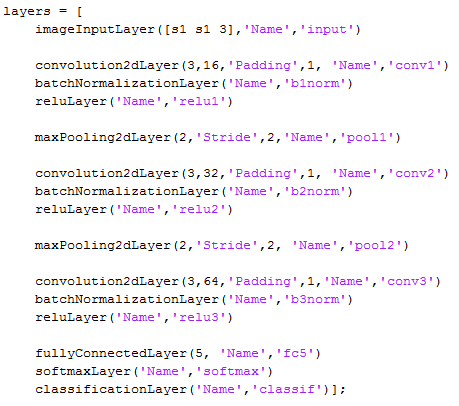
Developers of VGGnet have determined that 3x3 filters work best, as well as strategic placing of max-poolings, and finally doubling the number of filters after each max pooling works well. “GoogleNet” has quite a sophisticated architecture: it uses parallel convolutions, in combinations, called inception modules, which use 1x1 feature convolutions that condense the number of channels.



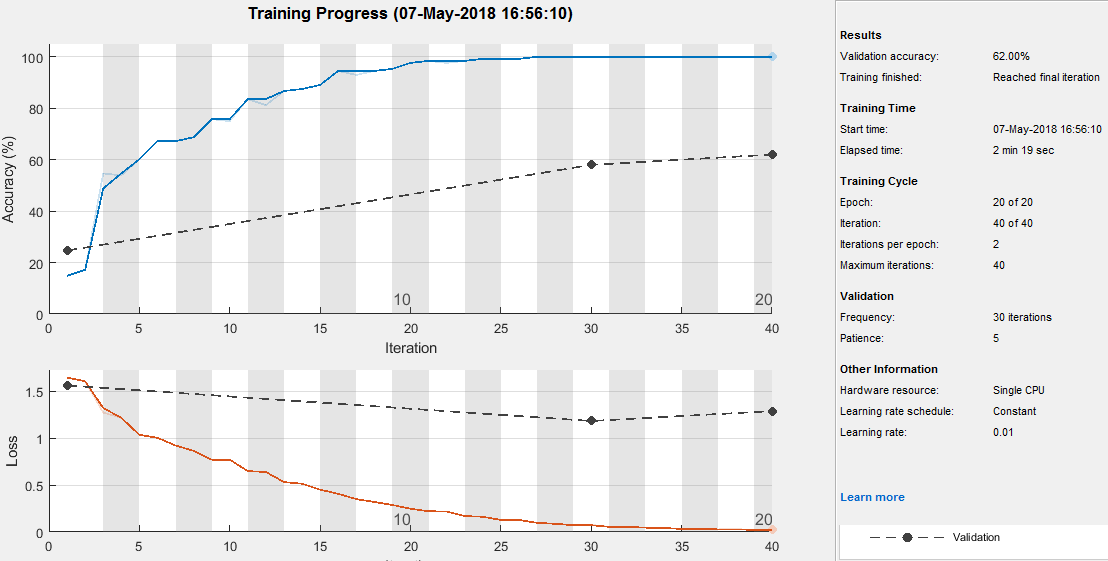
“The module basically acts as multiple convolution filters, that are applied to the same input, with some pooling. The results are then concatenated. This allows the model to take advantage of multi-level feature extraction . For instance, it extracts general (5x5) and local (1x1) features at the same time.”

However, even a simply designed CNN still has reasonable classifying power, as we will now demonstrate in building a simple CNN of our own. Below is a CNN that we built in Matlab. It is composed of 3 convolution layers, each followed by batch normalization, relu, and pooling layers. Finally, there is a fully connected, softmax, and classification layers.

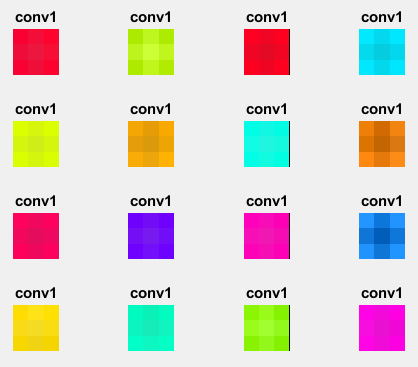
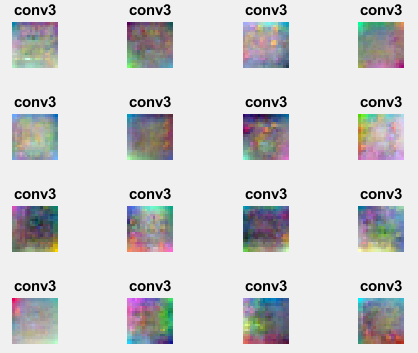




Running our CNN with 3X3 size filters, and with doubling the number of filters at each convolution layer (16,32,64) With pixel size only 30x30x3, and on 100 images. We obtain 62% accuracy (below) which is not bad for a simple CNN and shows the power of convolutions to classify images.



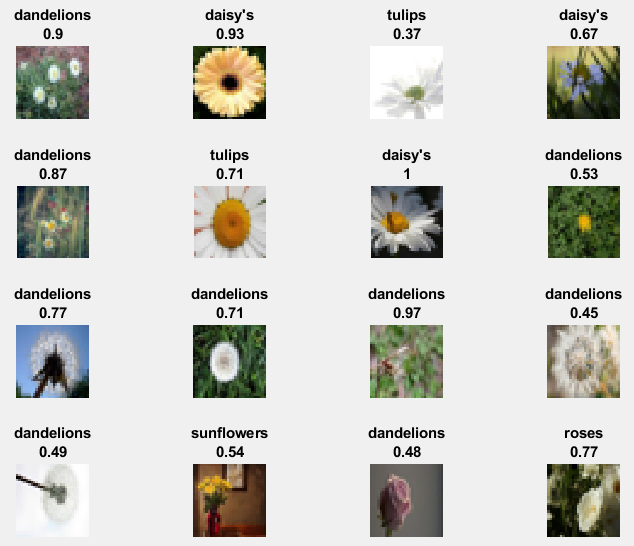
Like we did with Alexnet, we can deep dive into our model, and “peak” into what it is happening inside the CNN. We can compare the deep dream images that we obtain in our model with those we obtained in Alexnet earlier. The 16 3x3 filters learned for Convolution 1 are shown below. The images are quite simple. Next to this are 16 out of the 64 3x3 filters for the conv 3 layer. These appear to have some learned complexity.

Finally, the fully connected filters for each class are seen below: Daisy, Dandelion, Roses, Sunflowers, Tulips. Hardly distinguishable, yet the model obtains 62% classification accuracy on 800 images. Compared to Alexnet, where we could distinguish the features of roses and dandelions, these deep dream images do not provide the same clarity. The Kaggle images, however, I believe, are much more diverse. For exmple, the “daisy” object in the original Alexnet, deep dream is likely so sharply defined due to being trained on clean daisy images.

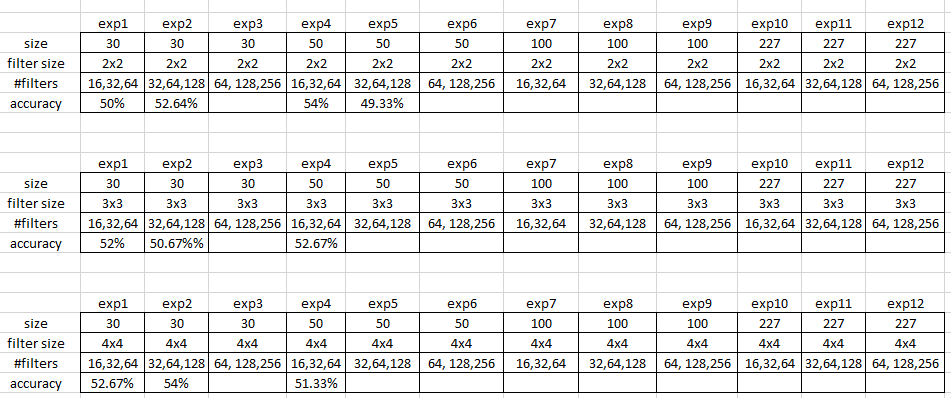


With 62% accuracy we can observe a sample of some of the successes and failures of our CNN in properly classifying our images. We also can see a sampling of the diversity of image types in Kaggle. Along with the image is the %confidence in the prediction. Some of these %confidence values may seem very high given the “very poor” classification filters we have just seen above. The reason for this confidence is because it was trained and validated on over 20 epochs. When trained on a few epochs most of the %confidence was below 0.50. This is appropriate because like a human vision test by an optometrist, when asked to read the small letters on a line far away one might be thinking, ‘there is a chance it is an “F”, but there is also a chance it is a “P” and there may be chances it may be other letters too like “D” or B” ‘, but whatever you say is your best attempt to classify something that you cannot really see. And we need to remember this, that the machine, our CNN, really does not see.

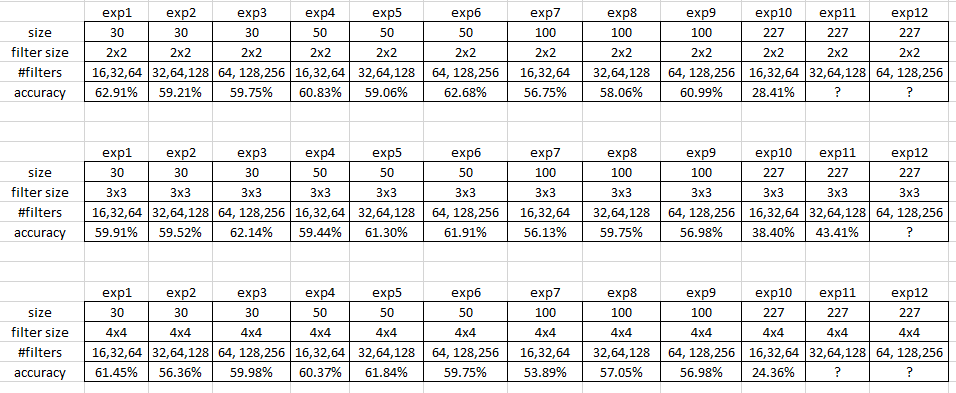


We have shown a detailed example of how our CNN performed in a single experiment. Now we will run several experiments on our model exploring the effects of image pixel dimensions, size of filters, and the number of filters. Although not expecting to obtain vast accuracy improvements to compete with the sophisticated architecture of Alexnet, VGGnet, Googlenet, or others, this exploration will also give us insights into the robustness of our CNN under various specifications. We try to sample the VGGNet recommendations discussed earlier, exploring filter sizes and number, whether doubling the number of filters after each convolution (ex 16,32,64) is helpful. We also consider pixel size of images. We run these experiments on a smaller dataset (100 images per class type), as well as on a larger data set, (about 800 images of each class type).

100 images per class (70 training, 30 validation)



About 800 images per class (560 training, 240 validation)



With 100 images per class we see the accuracy is about 50% for the conducted experiments. With about 800 images per class, the accuracy is about 60%. There is no significant differences among the results with the variables that we have changed: filter sizes, # filters, and #filters. Except in the 800 images case when the pixels are 227x227, the accuracy drops from near 60% to below 40%. This may be because the CNN has so much to learn that there becomes a lot of noise. The advantages of more possible data (more pixels) is overshadowed by so much more noise in the data to comprehend and understand. More images per class (800 vs 100) makes a significant improvement in accuracy. Images with size 30X30 pixels were classified with 60% accuracy. This reveals that with nonhomogeneous data (like we have with kaggle) a more general perspective that captures general shapes, colors, is more robust than a detailed image of each fold, hue, orientation, size, of the flower.

Future Work

* Continue to explore pixel size studies and perform sensitivity, also data set size sensitivity
* Experiment with other hyperparameters: learning rate, softmax function, use something other than relu for nonlinearity, etc
* Experiment with network design and architecture. 3d convolution filters. Inception modules.
* Use cluster analysis in combination with autoencoders and/or CNN’s to see how these it could classify
* Study the image data set and Identify impacts of outliers (images where the flower is not cleanly presented) on deep dream images

Summary

Autoencoders which work to reconstruct the inputs after being squeezed to a lower dimension (or expanded to a higher dimension) work at reconstructing the original image. Here we have seen that the larger the pixel size of the original image the better accuracy that we obtain. However, the larger the pixel size the significantly longer the processing time of the model to run. With this limitation as well as tuning some parameters such as number of images and the size of the hidden layers we saw that autoencoders achieve nearly 50% accuracy. With autoencoders we were also able to render reconstructed images which provided a very intuitive feel as to what the autoencoder was working at and how successful it was in capturing and redrawing individual images. However, we were unable to see how the autoencoder handled the diversity of images within each class. Extreme close-up images, obscured or hidden flower images, as well as various flower sizes, colors, shapes, orientations all took its toll on the ability of the autoencoder to correctly classify images. More images in a class only modestly helped the autoencoder.

CNN’s which work to understand the features of a class of images through combining various filters, performing pooling, etcwas able to render, (amidst the diversity of a particular class of images), the visual patterns that the class images mostly shared. Through the deep dream image visualization function in matlab we could see what the CNN worked hard at learning. The more images that the CNN was trained on the better the accuracy. Unfortunately, these deep dream images were not very intuitive, and even at the fully connected layer, where we would hope that a distinguishable pattern/image would emerge, we mostly obtained noisy colors and grainy resolution that only made clear sense to the CNN. However, unlike autoencoders which had no tools to grasp within-class features, these crude images provided the CNN the ability to classify with over 60% accuracy. With Alexnet, these deep dream images were clear enough to see intuitive pictures of each class of flower. Using Alexnet’s network architecture by transfer learning we obtained 75% classification accuracy. Network Architecture powerfully impacts models. Googlenet’s sophisticated architecture improved accuracy even more. Interestingly, the larger the pixel size of the images, the poorer the CNN seemed to perform. (opposite from autoencoders) This is because a CNN’s strength is in extracting features from a class, therefore, the simpler the presented image the better the training process will be able to extract these features to serve up to the classifier. However, if presented with rich detailed images, the CNN will struggle to properly identify the key features amidst the overwhelming variation within the class, burdened with hundreds of thousands/millions of filter values to assign, it collapses under the weight of noise.

**References**

Bengio, Y., LeCun, Y., Bottou , L., & Haffner, P. (1998). *Gradient-Based Learning Applied to Document Recognition*. Retrieved from <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>

Deshpande, A. (2016 , August 24). *The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3).* Retrieved from <https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

Ding, Z. (2007). A new Improved K-mean Algorithm with Penalized Term, K-means Algorithm is popular method in cluster analysis.

Faber, V. (194). Clustering and the Continuous k-Means Algorithm. *Los Alamos Science*, 138-144.

Fergus, R., & Zeiler, M. (n.d.). *Visualizing and Understanding.* Retrieved from <https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

Harrison, R. (2005). Improved K-Means clustering algorithm.

Ordonez, C. (2006). Integrating K-Means clustering with a relational DBMS.

Seldin, Y., Starik, S., &Werman, M. (n.d.). Unsupervised Clustering of Images Using their Joint Segmentation. 1-24.

Simonyan, K., & Zisserman, A. (n.d.). *Very Deep Convolutional Networks for Large Scale Image Recognition.* Retrieved from <https://arxiv.org/pdf/1409.1556v6.pdf>

Szegedy, C., Liu, W., Jia, Y., Sermanet , P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). *Going Deeper with Convolutions.* Retrieved from <https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf>

Tayal , M., &Raghuwanshi, M. (2011). Review on Various Clustering Methods for the Image Data‖. *Journal of Emerging Trends in Computing and Information Sciences*, 34-38.

Tsai , C.-W. (2007). A Time Efficient Pattern Reduction Algorithm for K-means Based Clustering.

Xiong, Z. (2006). The study of Parallel K-Means Algorithm .