**Character Classification Project**

By: Cody Nicholson & Adam Gruszczynski

6/6/2017

**Abstract**:

For this project we were asked to choose a problem that could be solved using data mining techniques, and to use what we learned in our *CSC 367: Introduction to Data Science* class to solve it. It was our goal to create a program that could classify an image based on the character it looks the most similar to using both an unsupervised and supervised learning approach. We decided to use a deep convolutional neural network for our supervised learning approach and the k-means clustering algorithm for our unsupervised learning approach.

**Introduction**:

The ability to program a computer to classify objects found in images has been a very popular topic in the machine learning & data science communities as of lately. This is because, by teaching a program to understand the world around it, we are closer than we have ever have been to solving intelligence. In the words of Google’s AI Lead Developer Demis Hassabis: “creating a general artificial intelligence—something that, like a human, can learn to take on just about any task”. In order to create this “general artificial intelligence” we need to be able to train programs to recognize objects that it can “see” in images. Since videos are just large collections of images, by training a computer to recognize objects in images we can also easily make it work with videos as well. This is why we chose to create an image classifier for our project.  
 The applications of image classification could be detecting diseases given images of a person’s x-ray or MRI results. It is also being used heavily in self-driving car technology that will eventually make car accidents all but disappear. We also see image classification working in popular social media applications like facebook, snapchat, and instagram to name a few.

Solving intelligence will give us the power to solve just about any problem we put our minds to by using the decision-making assistance of computers. Hassabis envisions AI being able to do things: “as diverse as advancing medicine by formulating and testing scientific theories, and bounding around in agile robot bodies”. With this powerful statement made by a lead AI researcher, we can rest assured that this is no longer just a delusional future fiction. This revolution in technology is occurring in the world today, and we want to be a part of it.  
  
**Literature Review**:

The first paper we reviewed was “*A Survey on Image Classification Approaches and Techniques*” by Pooja Kamavisdar, Sonam Saluja, and Sonu Agrawal written in January of 2013. We used this paper early on in our project to review all of the different methods we could use to build our image classifier. The paper talks about using artificial neural networks, decision trees, support vector machines, and k-means among other methods. Since neural networks have been so popular lately, we decided we wanted to try one out for ourselves. We read all about the advantages and disadvantages of creating an artificial neural network. The advantages included: It is a non-parametric classifier, it is an universal approximator with arbitrary accuracy, it is capable of functions such as OR AND & NOT, it is a data-driven self-adaptive technique, it efficiently handles noisy inputs, and the consumption rate is high. The disadvantages include: It is hard to understand, it takes a long time to train, it is prone to overfitting, it is difficult to choose the type of network architecture.  
 In order to discover patterns we engineered a model (or pipeline) for all of our samples to go through that uses a convolutional neural network (ConvNet) inspired by Yann LeCun’s paper “*Gradient-Based Learning Applied to Document Recognition*”. ConvNets - as opposed to regular neural networks - are made specifically for image classification. The ConvNet parses through each sample image and for every patch/kernel it outputs a new image with different dimensions. This operation is called a Convolution. A patch or kernel is just a partition of the entire image. If your patch/kernel size was the entire image, it would output the same as a regular neural network. Instead of having stacks of matrix multipliers in a regular neural network, in a ConvNet we have stacks of convolutions. These stacks act as our feature maps, which you can see an example of by looking at **Appendix A**. In that image you will see the stack of convolutions in each row, with the top layer representing the input layer of the whole image. The idea is that these stacks will form a pyramid like the one in the image found in **Appendix B**. At the base of the pyramid is the input image, and at the peak is the output. This output is a representation where all the spatial information has been removed, and only parameters that map to content remain.

One of the most challenging parts about creating a model using deep learning is training it. There are dozens of different parameters to tune to make for the best training of the neural network. We explored different conventions for training our model in the research paper titled: “*On the importance of initialization and momentum in deep learning*”. They show that when stochastic gradient descent with momentum uses a well-designed random initialization and a slowly increasing schedule for the momentum parameter, it can train both deep neural networks and recurrent neural networks to levels of performance that were previously achievable only with Hessian-Free optimization. They found that both the initialization and the momentum are crucial since poorly initialized networks cannot be trained with momentum and well-initialized networks perform much worse when the momentum is absent or poorly tuned. We used the information in this paper to choose our batch size, our number of EPOCHs, our learning rate, and our cross-entropy for our back propagation.

For our second model we chose to use k-Means to cluster the MNIST images into groups based on the digit they most closely resemble. We got our idea by reading the research paper “*Image Classification through integrated K- Means Algorithm*”. The paper explains that clustering analysis is a valuable and useful tool for image classification and object diagnosis. We learned that the K-Means algorithm splits the given image into different clusters of pixels in the feature space, each of them defined by its center. To do this our model determines the centroid coordinate. The centroid coordinate is simply the center of the digit in the image relative to its size. You can see an example of the centroid of a triangle image by looking at **Appendix C**.

After getting both our unsupervised and supervised learning models built and working we started to search for research papers that could help us to make our models better. The first paper we looked at was “*ImageNet Classification with Deep Convolutional Neural Networks*”. This paper was about building a ConvNet that could scale to be used to classify almost *any* image since it was trained using 1.2 million high-resolution images. ImageNet had a record-breaking error rate of only 15.3% during its best test run. We learned about techniques like dropout that is used to prevent overfitting, and ReLU’s that are used to help the model learn faster by making the learning non-linear. You can see the difference in training speed by looking at the image in **Appendix D**. We were able to use these techniques in our project to get an error rate of only 1% for our model.

The last paper we looked at was called “*Generative Adversarial Nets*” by Ian J. Goodfellow. We chose to look at this research paper because generative adversarial networks (GANs) are some of the most cutting-edge machine learning techniques currently available. The paper defines a GAN as two deep learning networks, one generator and one discriminator, that work together to train a single model to classify images. Basically, the generator network creates random “noise” images to try and trick the discriminator network into classifying the “noise” images as being a part of a class. You can find a helpful image portraying this technique at **Appendix E**. However, the discriminator is also simultaneously receiving valid images that it *should* classify as being part of a certain class. By having to differentiate between that fake images and the real once, these deep neural networks can collaboratively classify images better than any single neural network could on its own. We were unable to get this model to work in time for our presentation, but this is a great example of the future work we would like to do on this project.

**Methodology**:

We chose to stick very closely to the five steps of the data mining process for our project methodology. Thus, we started with domain understanding. We wanted to build a model that could classify images. Given the our limited time and experience, we knew it would be smart to start with an easy to use image dataset.

The dataset we chose was the MNIST dataset of hand-drawn images from TensorFlow, which is Google’s machine learning library. Since we chose to program our project in python, the data collection was as simple as writing this line of code: from tensorflow.examples.tutorials.mnist import input\_data. This loaded all 70,000 images in the dataset into our program to be used for testing and training our model. You can see a small subset of our 70,000 samples at **Appendix F**.

Next, we moved on to preprocessing our dataset. First, we split our dataset into three groups: a training set with 55,000 samples, a validation set with 5,000 samples, and a testing set with 10,000 samples. It is important to note that each sample at this point in the process has the dimensions 28x28x1. The model that we later designed only accepts images that have the dimensions 32x32x1, so for our next preprocessing step we added 1 pixels of padding to each side (4 pixels total) of every sample in all three divisions of the dataset to give each sample the correct dimensions. For our next preprocessing step we shuffled all three of our divisions of the original dataset. We do this because when we train our dataset over 10 iterations later, we don’t want our model to depend on the order of the data.

We then moved on to data reduction. We used a technique called one-hot encoding to encode the ten different classes we use to classify our images (0-9). In order to determine which class a sample belongs to, we used convolution operations to create feature maps. We create a convolution by dividing our image into smaller partitions called patches/kernels. Since these patches are only subsets of the entire image, they naturally have much smaller dimensions. After collecting all of these convolutions from the original sample into a list, we can then use this list as a feature map and compare to the feature maps we found when we trained our model on the training set of sample images. The process of finding these feature maps is a type of feature selection, so it does fall under the umbrella of data reduction. You can find a visualization of the feature maps created by our convolution operations at **Appendix G**.

As seen in **Appendix B**, we take the wide and short original image and after every convolution we change it to be narrow and tall. The narrow and tall output image is a feature map that contains all the features of the sample image that we use for classification. After we compare the feature map of our sample with the feature maps found in our training set, we can find the class that the sample is most probable to be. this is how we do the pattern discovery in our project. We trained our model over 10 EPOCHs and used that model to classify samples in our validation set to receive a final accuracy of 98.8%. Training over 10 EPOCHs simply means that we iterated 10 times over the entire set of 55,000 samples getting more accurate over most iterations. We made sure not to train it over too many EPOCHs so that we would not overfit.

Once our model is trained we can begin the pattern evaluation and interpretations step of the data mining process using our testing sets. To evaluate, we simply use the model that we created on the testing dataset that we defined in our data preprocessing step. Our model was able to classify 98.9% of our testing samples correctly.

To better evaluate our model, we created another model using k-Means to see how it would compare. The k-Means model will take in the same MNIST dataset as input and cluster each sample based on what that sample looks like. Our k-Means model first finds the center of the given sample that we denote as the centroid. Once we have found the centroid we can then calculate the distance from that point to our nearest object point - where an object point is the nearest point that is not white. Once we collect all of the distances between object points and our centroid, we can use this to plot each image on a graph based on these collected distances. The distances to centroids collected from our “four” images will all be fairly similar to each other, while being significantly different from the distances collected from our “one” images. We tried many different “k” values for our model as you can see in **Appendix H**, but we really only needed 10 clusters since we only had 10 classes. After 10 the algorithm is overfitting which is why the accuracy becomes so high as the number of clusters increases. Using this clustering method we were able to get an accuracy of about 66% on our entire MNIST dataset (We did not divide it since we don’t need to train because it is unsupervised learning). After experimenting with the k-Means model, we decided that the unsupervised approach was a better method for classifying images in the MNIST dataset.

Finally, we got to the knowledge discovery step. Our character classification model can correctly classify 98.9% of samples correctly. This could be used to read automate machines to read labels off of mail in a post office, or serial numbers off of items in a store, among many other tasks. By making this model we can make these processes more efficient so that people can spend time doing more complicated and important tasks. The k-Means model will need some more work before it is ready to try and handle any of these kinds of tasks since it only has an accuracy of 66%.

For details on the programming that went into this project please view our project website at: <https://codynicholson.github.io/Character_Classifier_Project/>.

**Results & Discussion**:

Of the two resulting models we created, the deep neural network is the only one that we would say has practical use outside of this project since it correctly classifies 98.9% of the validation samples. We might be able to improve this deep learning model even more by making it into a Generative Adversarial Neural Network. Since the k-Means model is only correct 66% of the time, it won’t be useful in many applications. We look forward to improving it in the future. This project was a great first step for us in learning how machine learning and artificial intelligence comes together with data science to create useful applications.

**Conclusions & Future Work**:

To conclude, we were successful in creating a model using deep learning that practical uses outside of the classroom. It took a lot of research, but this was a great introduction into some of the latest technology in the machine learning, artificial intelligence, and data science communities.

There is an unlimited amount of potential for future work because our deep learning model can easily be tweaked to work for facial detection, classifying traffic signs, classifying famous pieces of art, among other uses. To take an even more ambitious step we could use this model as a starting point to build a Generative Adversarial Network (GAN) that would be even more accurate and adaptive to change than the one we currently have. Although we didn’t have time to engineer a model, we did read all about GANs in “*Generative Adversarial Nets*” by Ian Goodfellow. The paper defines a GAN as two deep learning networks, one generator and one discriminator, that work together to train a single model to classify images. Basically, the generator network creates random “noise” images to try and trick the discriminator network into classifying the “noise” images as being a part of a class. You can find a helpful image portraying this technique at **Appendix E**. Using these two neural networks the GAN can train itself better than any single neural network ever could since it is constantly generating new random data to classify.

We could also improve our k-Means model by adding an support vector machine (SVM) to classify the images after the k-Means clustering. Basically, the k-Means algorithm will produce a graph with clusters corresponding to different classes. The SVM is a supervised learning technique that can be used to draw a line called the *decision surface* that can divide the clusters into classes.

**References**:

Pooja Kamavisdar et al. “*A Survey on Image Classification Approaches and Techniques*” January, 2013

Yann LeCun et al. “*Gradient-Based Learning Applied to Document Recognition*” November 1998

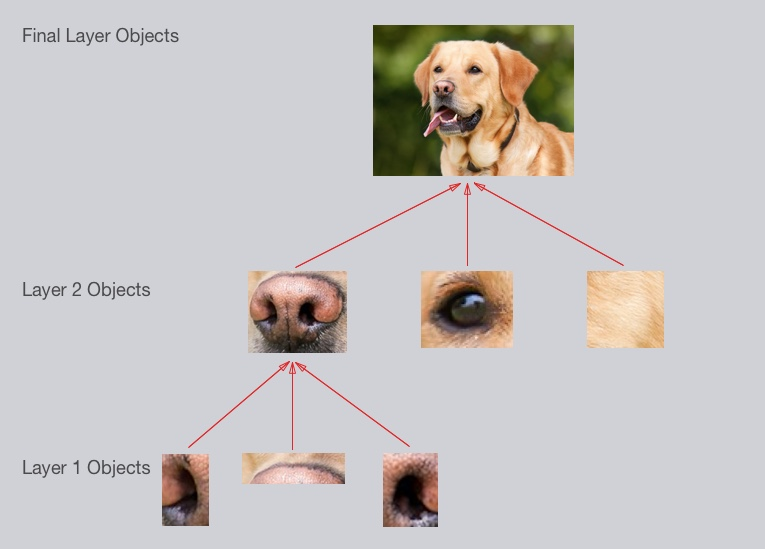
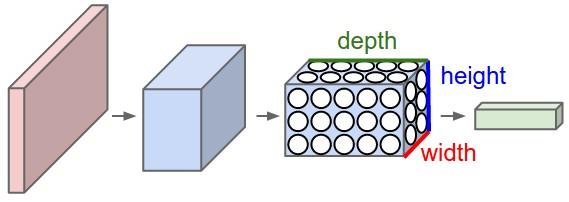
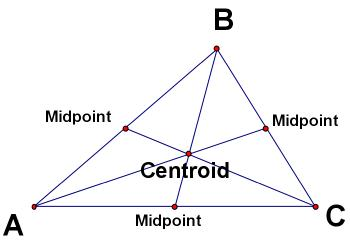
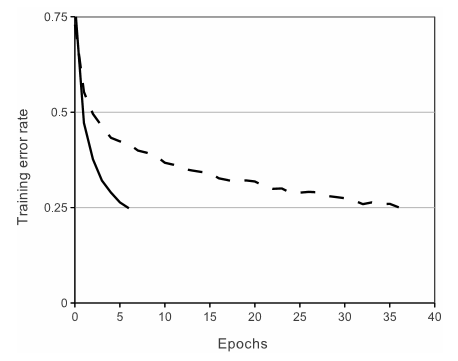
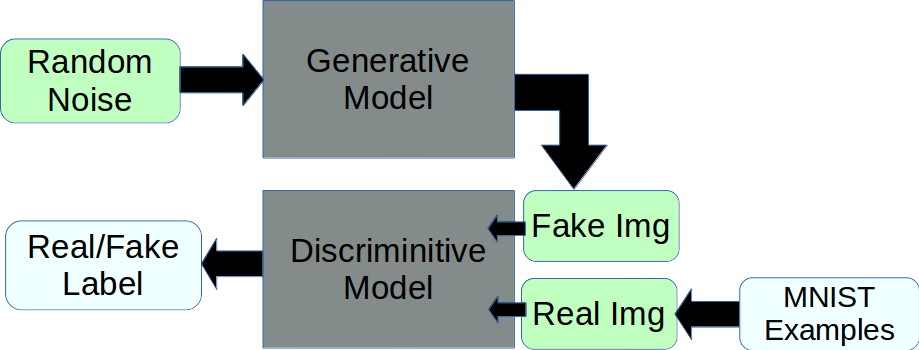
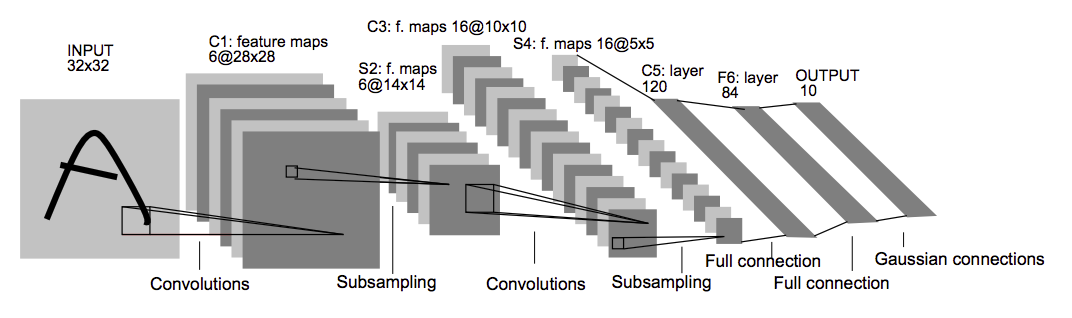
Ilya Sutskever et al. “*On the importance of initialization and momentum in deep learning*” 2013

Balasubramanian Subbiah et al. “*Image Classification through integrated K- Means Algorithm*” March, 2012

Alex Krizhevsky et al. “*ImageNet Classification with Deep Convolutional Neural Networks*” 2012

Ian J. Goodfellow et al. “*Generative Adversarial Nets*” June, 2014

**Appendix**:

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 