CMPE 452: Final Report

Image Recognition of Simpsons Characters Using Neural Networks

The Team

Name: Alex Bratton
Email: 12ab122@queensu.ca
Student #: 10085113

Name: Jake Fantin
Email: 13jf41@queensu.ca
Student #: 10139170

Name: Jean-Philippe McCluskey
Email: 13jpm11@gmail.com
Student #: 10139394

Name: Sam Ruscica
Email: 14stsr@queensu.ca
Student #: 10148585

Contributions

Alex Bratton	Performed deep learning using an LeNet, wrote the summary for this learning network, the data preprocessing for KNIME, and the Comparison of Tools in the report.
Jake Fantin	Built a Convolution Neural Net in Python using Keras on top of TensorFlow and described the process.
Jean-Philippe McCluskey	Performed character classification using Matlab and the Computer Vision System Toolbox / described dataset.
Sam Ruscica	Developed an AlexNet, deep learning prediction model, in KNIME. Described the design process and results of this model.

Problem Definition

To train an artificial neural network to recognize characters from the Simpsons in each image. Our goal was for our model to pick the correct category as often as possible. This task is called image classification.

Implementation Plan

To try to find the best ANN model for this problem, the group constructed different models across a series of platforms. Each member worked with technology they are familiar with to reduce learning time and were in constant communication with one another about their progress and breakthroughs. Each model used: LeNet, AlexNet, Convolutional Neural Net, and the Computer Vision System Toolbox in Matlab, were chosen for their usefulness in learning to categorize images. As each model is refined we became, by the end of the project, aware of the most successful option.

Data Set

For this project, the data set is from a Kaggle Site, created by Alexandre Attia. It is composed of screenshots of the characters from various episodes over multiple seasons. The dataset is composed of two sub datasets; training and testing. The training set includes a folder for each character with varying numbers of images. The testing set is a similarly structured set but of new images. The character is not necessarily centered in each image and sometimes they are with other characters. The character is always the largest part in the picture. There are 40 different characters and with any number of corresponding images ranging from zero to several thousand. Additionally, as each image was collected separately without a uniform sampling technique, and as a result, each image is of varied resolution and proportion. These are important factors that must be taken into account when performing data preprocessing. Using the same dataset allows us to objectively compare different modeled approaches with each other.

Related Works

"A Convolutional Neural Network with Dynamic Correlation Pooling"

LeNet Application, contributed by Alex

This work involved modifying an LeNet-5 network to use dynamic correlation pooling, which takes advantage of the correlation of adjacent pixels in an image to be improve feature recognition. The problem this paper seeks to address is a proposed method to improve the convergence rate and recognition accuracy of the LeNet-5 network using dynamic correlation pooling, as compared to max pooling, average pooling, stochastic pooling, and mixed pooling. Three datasets were used to test the theory, which were the MNIST dataset, the USPS dataset, and the CIFAR-10 dataset. The MNSIT dataset contains 0-9 handwritten Arabia numerals, including 60,000 training samples and 10,000 test samples. The USPS dataset is a postal recognition library, containing 7000 samples for training and 2000 samples for testing. The CIFAR-10 dataset contains labeled color images in 10 categories, with 50,000 images for training and 10,000 images for testing. The results showed that the network using the dynamic correlation pooling algorithm had higher accuracy and faster rate of accuracy, as well as a smaller error rate. While exemplifying that the LeNet algorithm is a reliable algorithm for image recognition, this paper is

particularly relevant to future work for our project. If we wished to increase the accuracy of the LeNet model, it may be beneficial to consider using the dynamic correlation pooling method.

AUTHORS: Chen, Junfeng; Hua, Zhoudong; Wang, Jungya; Cheng, Shi
PUBLICATION: IEEE Conference Publication, Hong Kong, China, 15-18 Dec 2017
DOI: 10. 1109/CISC.2017.00115

"Face recognition using ensemble support vector machine"

Matlab Application, contributed by Jean-Philippe

This paper aims to compare the use of multiple independently trained support vector machines aggregated to make a collective decision versus the single support vector machines methods. The task of the SVM's is to do facial recognition using fuzzy 2D maximum scatter difference (F-2DMSD) method of feature extraction. The authors used randomly selected training and testing samples from the AT&T, FERET face databases to show that the ensemble of SVM's outperforms the single SVM. The best average of the single SVM is 94.85% vs the best average of the ensemble of SVMs is 95.80%. The Matlab portion of the project uses an SVM therefore we could have used this method to slightly increase the accuracy of the facial recognition of the Simpson characters.

A. Dey, S. Chowdhury and M. Ghosh, "Face recognition using ensemble support vector machine," 2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), Kolkata, 2017, pp. 45-50.
doi: 10.1109/ICRCICN.2017.8234479

"Neural Nets Vs. Lego Bricks"

Keras Application, contributed by Jake

In this article, the author has decided to try at his hand at the Lego secondary market by purchasing large quantities of unsorted Lego, and designing a system to sort them, as sorted quantities sell for much higher price. A conveyor belt with a camera and air nozzles was set up to take in images of the pieces and blow them into the correct bucket via air burst upon sorting. After attempting several different image classifying/identification methods, including openCV image processing and Bayesian classification, the final working solution he arrived at was using Keras on python, due to its high speed, ability to grow its own dataset over time, and reliably identify a high number of classes, even though many of the pieces can be similar. The dataset started as a small number of images the author manually classified, but grew every time the author ran the predictor and fixed the mislabeled outputs. This article helped direct the group to keras's ability to classify many closely related classes.

Author: Jacques Mattheij
Publication: IEEE 2017
Link: https://spectrum.ieee.org/geek-life/hands-on/how-i-built-an-ai-to-sort-2-tons-of-lego-pieces

"Object Recognition Using Deep Convolutional Features Transformed by a Recursive Network Structure"

AlexNet Application, contributed by Sam

This paper discusses how to recognise objects in faster and more compact ways. They wanted to know if it is possible to implement new techniques that would circumvent the limitations of an AlexNet. They had the AlexNet train on a ImageNet data set. This data set is a large-scale hierarchical image database that has fixed size RGB images. The technique they used was implementing a AlexNet with a RNN after it. This resulted in slightly higher performance than just the AlexNet, it had cheaper computation, and it allows for size-independent transferability and comparison of features between layers. They also used RGB clustering, grayscale clustering, and image normalization techniques to further improve their network results. In the end, they were able to have a network that was 89% accurate which was on average, 6% better than just using the AlexNet. This paper greatly helped with the refining and improving process of our project. It gave insight into how to improve a network while not increasing the computation costs greatly. Additionally, this paper provided reassurance that using a CNN was good for image recognition.

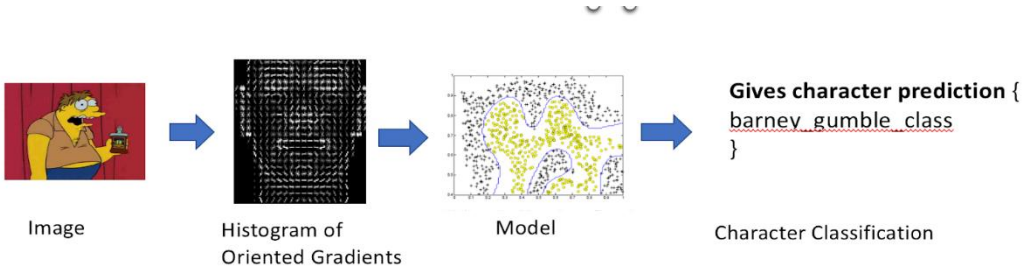
Authors: Hieu Minh Bui, Margaret Lech, Eva Cheng, Katrina Neville, Ian S. Burnett
Publication: IEEE, 2016
ISSN: 2169-3536
DOI: 10.1109/ACCESS.2016.2639543

Implementation Details

As mentioned in the *Implementation Plan* section, each team member took a different approach. JP implemented the Convolution System Toolbox using Matlab. Alex implemented the LeNet using KNIME. Sam implemented the AlexNet, also using KNIME. Jake implemented the Convolutional Neural Network using Python and Keras. In the following sections, the discussion will be divided into 3 sections, one for each tool used: Matlab; KNIME; and Keras.

Matlab

In this method Matlab and the Computer Vision System Toolbox was used to recognize Simpsons characters using deep learning neural nets. The Computer Vision System Toolbox provides algorithms, functions, and apps for designing and simulating computer image recognition. It provides the ability to simply perform feature detection, extraction, and matching of the images that are in the database. The feature detection used was a histogram of oriented gradients that are fed into a support vector machine model to give an output classification.



KNIME

The AlexNet node, LeNet node, the deep network learner node, and the deep network predictor node are all part of the Deep Learning package in KNIME. KNIME can be downloaded from the KNIME website (https://www.knime.com/), and the workflows attached in the zip folder may be imported to recreate the experiments.

Keras

For this method, a high-level neural networks python API called Keras is used on top of the deep learning python library TensorFlow. Keras is built to be extremely efficient to produce results as fast as possible for convolutional and recurrent networks, which made it a desirable choice for this image categorization problem. Installing keras and tensorflow using pip gives the developer access to all the tools they need to build a model.

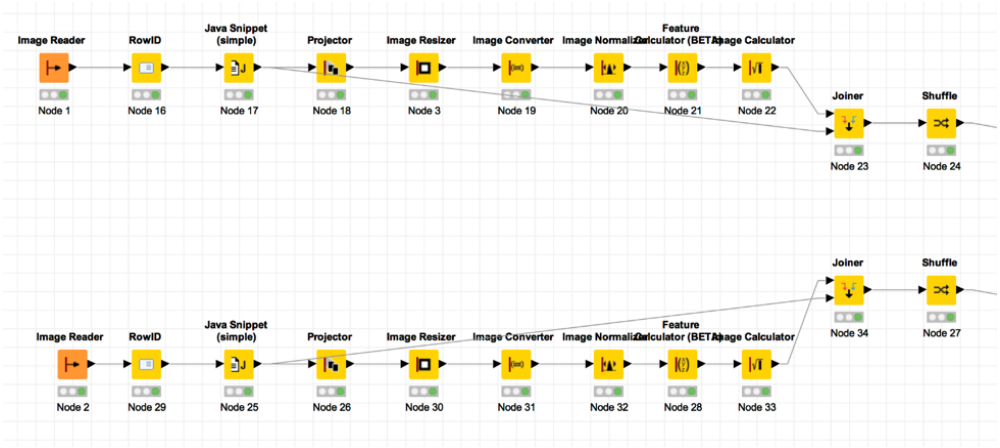
Data Preparation

Matlab

The dataset in matlab was easily imported into matlab in one function call due to the labeled and organized file structure of the dataset. To avoid looping through every single file in the directory testing that they are valid images formats and labelling them we used the Matlab function "imageDatastore('Z:\CMPE 452\simpsons_dataset')". This function automatically imports all photos and labels the images using the subfolder names as class names. This is ideal for the dataset structure that we have, each character having a folder containing all the images that they appear in. This function can also handle massive amounts of data, even if the all can't fit in memory at once. This was not a problem because there were only about 41,000 low quality images in the dataset, but would save time if more images were added to the dataset in the future. Then the dataset was divided it into 80 % training data and 20% testing data. Then the training data was transformed into a histogram of oriented gradients using the extractHOGFeatures function before being classified. This makes it easier to do feature extraction and get clear numbers rather then the image we started with.

KNIME

The dataset for both models implemented in KNIME was preprocessed using the following procedure:



The training images are read in using the ImageReader node to access the dataset, which consists of subfolders labeled with a character's name and containing up to 1000 images of that character. The RowID node extracts the full file path of the image, and the JavaSnippet node uses a short Java program to determine the name of the character associated with the image. The dataset now consists of two columns, one containing images and one containing the image's character's name. The images are not normalized and have different sizes, so the remaining preprocessing corrects these characteristics. The three colour channels in each image are projected onto one, as maximum projection, to reduce computation complexity. All the images are resized to 100x100 pixels, as they have to be the same size. Every pixel value is normalized to be between 0 and 1 for the neural network. The mean of the image is subtracted from its pixel values, to center the pixel values around zero. Finally, the order of the images is shuffled to prevent biasing the network by showing them one character at a time.

This workflow is shown twice in the above workflow because it is applied both to the testing dataset and the training dataset, which in this case come from different sources.

Keras

For data preprocessing and organization, the Keras API has a function that imports and resizes images from folders using the folders name as the class. As the dataset at hand is already prepared in that format, the only issue was in creating a validation set for use in the model. As a result, a separate program was created to syphon out 30% of each class folders images into a separate directory for this use. It was at this stage that the small, sometimes non-existent number of training images available for certain characters was brought into question, with some being removed entirely. For preprocessing, each image was scaled to 64 by 64 pixels as the size of each image must be the same for the Keras convolution layers, and some new images were generated by blurring, rotating, and scaling the images provided to produce a more robust model. As data preparation is all done in one step, the code that performs this operation is included below rather than a flow diagram.

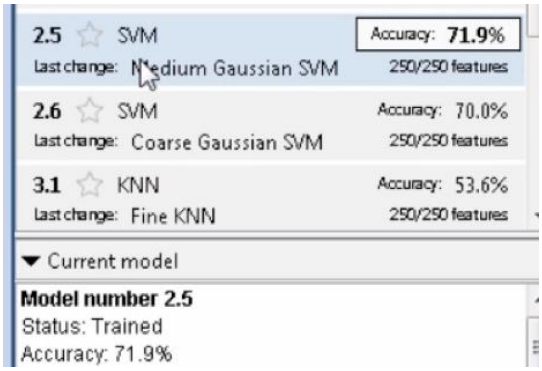
```
train_datagen = ImageDataGenerator(rescale = 1./255,
featurewise_center=False,
samplewise_center=False,
featurewise_std_normalization=False,
samplewise_std_normalization=False,
rotation_range=0,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
vertical_flip=False)
```

Solution Approach

Matlab

Once the images have been converted into readable data it is entered into a training model. The built-in Matlab app called classification learner was used to find the best classifier for the data. The app runs multiple training models and compares their validation errors side-by-side to help us decide which algorithm to use. A few different support vector machines and nearest neighbor classifiers were tried to

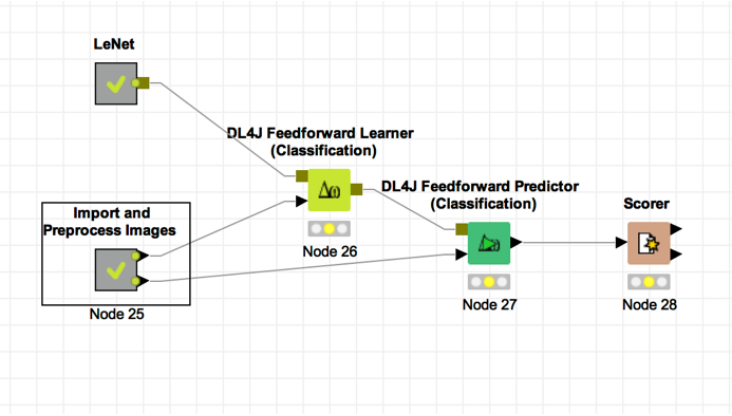
find the best classifier with highest accuracy. The Medium Gaussian SVM was the best classifier for the simpson character images. The trained classifier was able to predict the correct characters images in the test set at a rate of 71.9%.



KNIME

LeNet5 Convolutional Neural Network

Image processing was performed with a LeNet5 convolutional network, using the LeNet image processing node in KNIME. The LeNet5 model was selected because it is known to accurately recognize features in images. The KNIME workflow used is shown below.



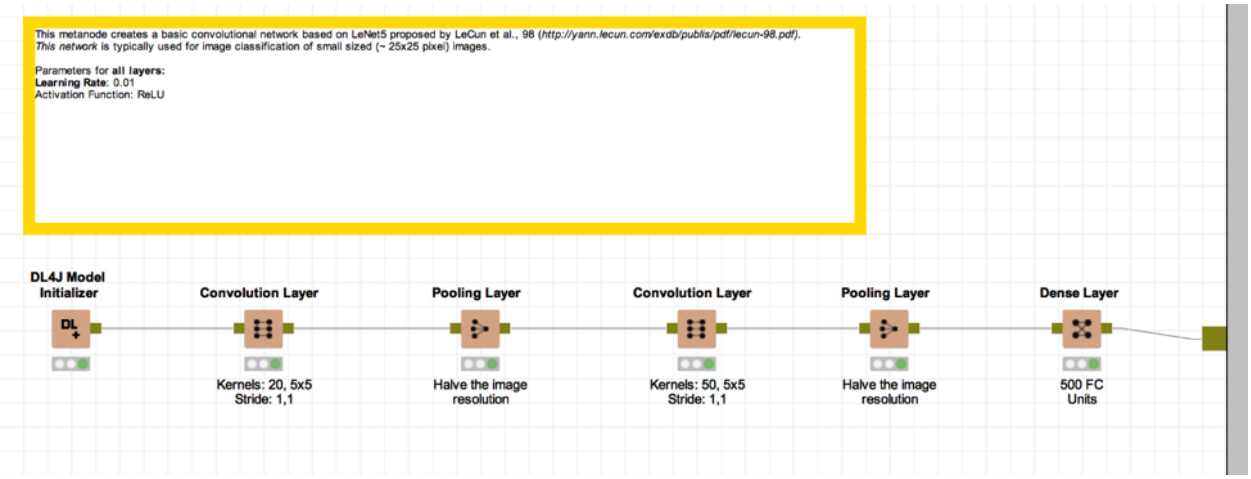
First, the images are imported and preprocessed. The first (top) output of the preprocessing metanode is the training set of images and the second (bottom) output is the testing set. The training set is fed into a feedforward learner that uses an LeNet to learn the character's faces. The feedforward predictor then takes the model from the learner and the test dataset to attempt to predict the image's character. The scorer determines the accuracy of the model.

Once the images had been preprocessed, they were passed through an LeNet neural network in KNIME. The LeNet is a basic convolutional neural network, with a learning rate of 0.1 and a rectifier activation function. This network works by using the following procedure:

1. A model initializer node, which creates an empty architecture for the neural network.
2. A convolution layer node, which convolves the images using 20 kernels of size 5, 5.
3. A pooling layer node, which performs maximum pooling on patches of the images with kernel size 2, 2.
4. Another convolution layer node, which convolved the images using 50 kernels of size 5, 5.

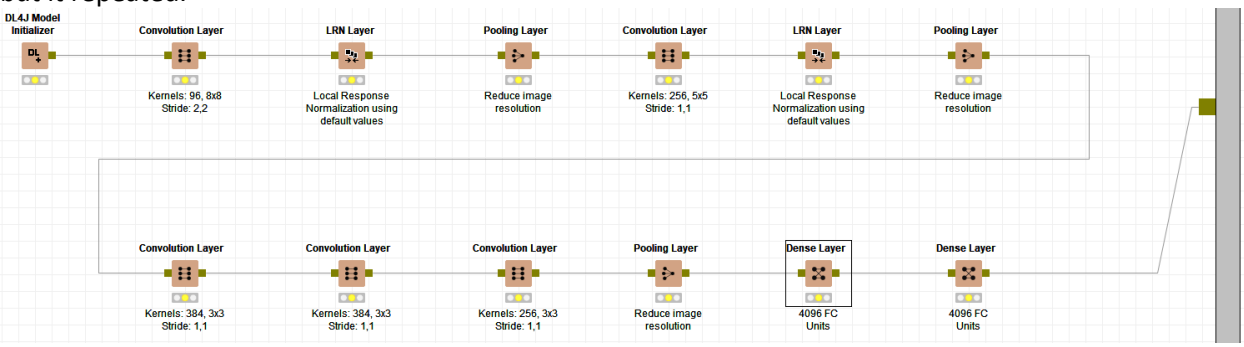
5. Another pooling layer node, also performing maximum pooling with kernel size 2, 2.
6. A dense layer node, which adds a fully connected layer to the deep learning model.

A screenshot of the workflow for the LeNet node is below.



AlexNet

The AlexNet works very similarly to the LeNet. It was designed by the SuperVision group with the purpose of creating a GPU friendly network so it could run faster by running in parallel. The network consists of the same nodes but they are stacked on top of each other. By performing the convolution layer and the pooling layer multiple times in a row, the network trains to a higher dimension than a simple LeNet. Below is a screenshot of the workflow for the AlexNet node showing how it contains the same nodes as LeNet but it repeated.



Keras

Keras works with its own unique data structure called a 'model', which holds the layer structure of the neural net the developer wants to build. The type of Keras model used was a sequential type, meaning that the stacking of layers is linear, as this is easier to understand and implement than an arbitrary graph of layers. Through a simple series of .add() methods the model is created using a series of two-dimensional convolution layers, Max pooling layers, Flattening layers and Dense neural net connecting layers. The convolution layers perform the convolution operation on the image to produce features maps. The Max pooling layers condense the image based on the feature maps produced from the convolution layers in order to reduce the number of nodes needed for later layers. The Flattening layer converts the pooled 2D matrix into a vector that feeds into our fully-connected Dense hidden layer which connects to the outer layer of some 42 classifying nodes each using a sigmoid activation function. The model is then

compiled with a function called .compile() that includes the training set and validation set To run the model, it's as easy as specifying the number of epochs and validation steps desired.

After a few first attempts at running the model with example epochs and steps-per-epoch from online tutorials on using the API with decent success, more research was done into the construction of keras convolutional networks. When this model was chosen to go further than the others, the group colluded further into its development. Three more convolutional layers were added to further isolate the key forms of each class, to a total of four. Following every two of these layers is a max pooling layer and a dropout function to reduce overfitting. These additions greatly increased the efficiency of the model.

Results

Matlab

The Matlab model was more accurate than the KNIME models, but much less then the final Keras model. It took 6+ hours to train, and resulted in an accuracy of 71.9% using the medium gaussian SVM classifier.

KNIME

The LeNet5 model was the least accurate of all the models attempted. It took 6+ hours to train, and resulted in an accuracy of 64%. The AlexNet was more successful reaching 81% accuracy. The network took over 10 hours to train.

Initial Keras

The initial model was not spectacular in its performance, only reaching 68% after running for half an hour and then seemingly breaking.

Final Keras

The final keras model, after experimentation and adding even more overfitting reduction techniques, achieved 95% in 2.5 hours of training.

	KNIME LeNet	KNIME AlexNet	Matlab	Keras (initial)	Keras (final)
Accuracy	64%	81%	72%	68%	95%
Runtime	6 hours	10+ hours	6 hours	0.5 hours	2.5 hours

Discussion of Results

Comparison of Tools

There were several pros and cons to each tool. These will be summarized below. The best tool used was Keras, for reasons specified below.

Matlab

The advantages of the Matlab approach include the well-built tools that come included with Matlab, the large library of built-in function classifiers, and the built-in preprocessing methods for images. The main disadvantage of having access to built-in tools and methods is that they are inflexible and hard to

customize, especially in pursuit of improving the algorithms. The training time was also long for the Matlab implementation, exceeding 6 hours.

KNIME

Like Matlab, KNIME provides pre-made models that are easy to implement and understand. The workflows are also easy to troubleshoot, and KNIME provides much in online support. The disadvantages of using KNIME is that the logic behind the algorithms in inaccessible, rendering the models difficult to customize or improve. The training time was quite long as well, with the longest training time exceeding 10 hours. This was the training time of the AlexNet model, which was the slowest to train.

Keras

There were notable disadvantages to the Keras model. The learning curve for mastering all the functions was steep and there is no built-in way to separate data into validation, testing, and training sets. In spite of these disadvantages, the Keras model was named the best tool used in this project for several reasons. The Keras model was easily modified with many options, enabling great flexibility. It also included built-in preprocessing methods for images, simplifying the process. The Keras model also had the fastest training time of 2.5 hours.

Discussion of Models

Support vector machine (Matlab)

The support vector machine model is a supervised learning models with associated learning algorithms that analyze data used for classification. It is non-probabilistic binary classifier that can efficiently perform a non-linear classification of many classes by mapping their inputs into high-dimensional feature spaces. In this project we used the built in Matlab app to run and determine the most accurate algorithm determined to be the medium gaussian SVM. This model could have been improved by using by using multiple SVM's and aggregated them to improve the models overall accuracy as seen in the research paper "Face recognition using ensemble support vector machine".

LeNet Convolutional Neural Network

This was the least accurate model of all that were attempted. It was easy to use and understand, and the network was pre-made which saved time, but it was not easy to customize and had a long runtime, exceeding 6 hours. This model could have been improved by using dynamic correlation pooling, as noted in *A Convolutional Neural Network with Dynamic Correlation Pooling.*

AlexNet Convolutional Neural Network

This model was initially the most accurate model. It was very simplistic to implement and had lots of online resources. This network was not what we chose to pursue because of the runtimes for each test. It would have resulted in unnecessary wasting of time by trying to improve the network through trial and error.

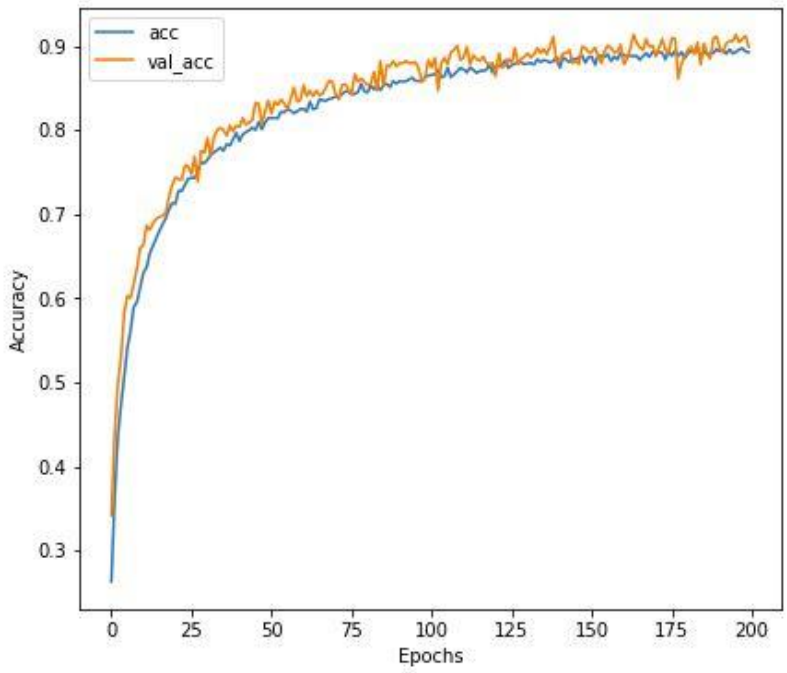
Sequential Convolution Network

The initial sequential convolutional network built with Keras only took 30 minutes to train, but it was built without a good understanding of the model itself. Several of the initial executions was ended by the user after only one epoch out of 200 as it was pausing for an indeterminate amount of time, and thought to be in error. Upon reflection, it most likely would have resulted in a much a higher accuracy if allowed to finish, however the training would have lasted for 100 hours.

The final model built with Keras was still a sequential convolutional network, but much more efficient. With greater understanding of how Keras operated, the options available to the API became a blessing and less of an obstacle. The model was moved to a more powerful computer, cutting down on processing time with a number of combinations tested in a smaller time frame. Upon experimentation an optimal balance of epochs and steps was reached, that only needed 2.5 hours of training to reach 95% accuracy.

Notes for Improvement

The dataset used in this project is still being added to regularly from people manually going through episodes of the Simpsons and saving frames. This has lead there to be many classes that only have a few images due to the small role of the character and some characters that have over 1000 images. All models could have been improved by reducing the number output classes to the characters that have at least 100 images worth of training data. This would not only reduce the possible number of classes but reduce the size and computational complexity if the neural net. Another improvement would be having the network train using more Epochs. This however has diminishing returns and would have resulted in even longer training times. This can be seen in the graph below.



https://medium.com/alex-attia-blog/the-simpsons-character-recognition-using-keras-d8e1796eae36

Future Applications

Uses for Facial recognition algorithms like the models described above are already being used in the technology we use today. Here are 3 examples of this:

1. Social Media: Social media sites like facebook are recommending people to tag in photos.
2. Photo Organization: Services like google photos are able to let users sort through their photos by people even tho they have never been formally labeled as being that person.

3. Security Passwords: Screen unlock like on the new IPhones use face recognition to replace password logins to unlock and pay for things on the phone.