An Introduction to Capsule Networks and Implementation on CIFAR-10

By Joseph Conran

Introduction

From simple statistical models to large datasets used by Google, algorithms employed by machine learning serve as the foundation for a wide variety of data analysis. Machine learning, a field of computer science that allows computers to learn without being explicitly programmed, can be divided into two primary fields of study named supervised and unsupervised analysis. Supervised machine learning employs predictive learning methods where input data has a corresponding target whereas unsupervised trains the computer only on a set of inputs, and focused primarily on classification and exploratory analysis.

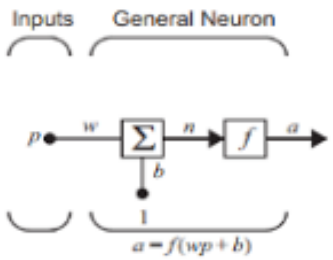
Serving as the final project in the second semester of machine learning at the George Washington University, this team focuses heavily on a type of supervised learning termed 'deep learning' to develop a breadth of understanding in both the fundamentals and modern machine learning techniques using the CIFAR-10 dataset. The learning outcomes of this analysis seek to improve our familiarity with image recognition algorithms and how to improve their accuracy, to better understand convolutional neural networks (CNNs), to develop a deeper understanding of Python and the Caffe and Keras frameworks, and finally to explore the limitations of CNNs and provide an introduction of capsule networks which seek to correct those limitations.

CIFAR-10 Dataset

The CIFAR-10[1] dataset is a subset of the Tiny Images dataset developed by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Considered 'big data' as it cannot be efficiently computed on a consumer PC, this dataset is made up of 60,000, 32x32 pixel, color images with 50,000 training and 10,000 test images. The dataset is equally distributed among 10 different classes of images including airplane, automobile, bird, cat, deer, dog frog, horse, ship and truck as seen in Figure 1. In addition to the MNIST handwritten digit dataset, the CIFAR-10 serves as a baseline resource for developing and measuring the efficacy of new network frameworks.

[1] https://www.cs.toronto.edu/~kriz/cifar.html

Deep Learning and Convolutional Neural Networks



At the core of a simple neural network is the neuron which includes an input, a weight, a bias, an activation function, and an output. To improve the performance of a network, the input data is propagated through the network with either random or set weights and biases to the final output (see figure 2 for an example of a simple neuron). The output data is then measured against the target data with the errors propagated in reverse throughout the network to update the weights and biases. These steps are completed over multiple iterations as the accuracy improves and the error decreases. Networks can be improved either by adding more nodes (or networks) in tandem, or by adding neurons sequentially. Moving beyond three or four sequential neurons begins to place the machine learning algorithm into what is known as 'deep learning.' The term also suggests a higher level of sophistication in the learning algorithm such as with convolutional neural networks.

Convolutional neural networks (CNNs) are a type of sequential deep learning networks that are used primarily for image analysis. While datasets like CIFAR-10 may have images small enough to be processed by simpler network, the weights and biases scale exponentially as the image size increases. For the CIFAR-10 dataset, each image results in 32 pixel height by 32 pixel width by a 3 channel RGB color depth. CNNs by design take a two-dimensional input such as an image. 32x32x3 results in 3072 weights and biases for each neuron. Moving slightly up to a still moderately-sized image in 2017 of 500x500x3 would result in 750,000 weights and biases.

Yann Lecun's LeNet architecture was the first fully-formed CNN in 1998[2] and still serves as the foundation for convolutional networks today. Starting with the 3-dimensional input data (32x32x3). Taking the input, the data is then convoluted to extract features from the input. This involves using a smaller matrix or filter and scanning the image in strides across the entire input image and computed using a dot product. The resulting matrix also known as an activation or feature map contains extracted features from the raw image. Moving on from the convolution step, the feature map is then passed through a Rectified Linear Unit (ReLU) operation in which all negative values are converted to 0 to add non-linearity to the network. After ReLU the data is then passed to a max pooling step in which the feature map is reduced in size by scanning areas of the map in pixel grids (such as a 2x2 filter) and selecting the largest value from each section and placing these into a smaller pooled matrix. The final Fully Connected layer is similar to that of the traditional neural networks where neurons are connected to all activations in the previous and then finally activated using the softmax function. The softmax combines the vector of scores into one final ranking between zero and one and assigns the image based off the highest score. At this point, the error is then backpropagated through the network gradients to update all the weights and parameter values. This process can be refined through multiple iterations, dropout functions, different numbers of nodes or layers, etc. An example of a network with two convolution/pooling layers can be seen in figure 3.

[2] http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

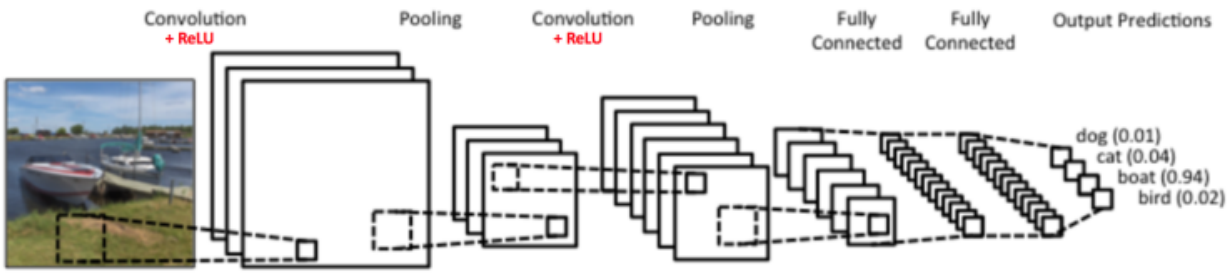
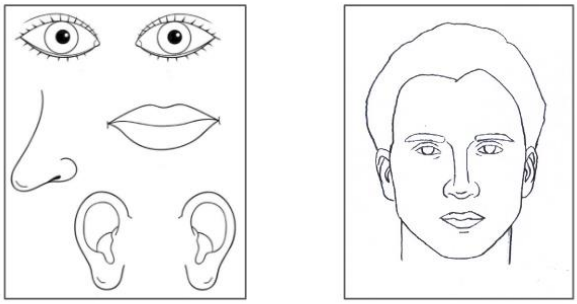


Figure 3 Sample CNN with 2 conv/pool layers[3]

The Limitations of Convolution Networks and Capsule Networks

Convolution networks excel at classifying images when they resemble the target image set such as with the MNIST handwritten number set, but classification suffers when the images have a different orientation than the target data. To solve this problem, methods such as data augmentation, which introduce altered versions of the input data (rotation, color scale, etc.) into the network help boost accuracy. Furthermore, the pooling layer's feature reduction inherently solves some of these issues but introduces others when trying to correctly classify an object of different scales or locations within the image. It can even classify abstract images of certain features as a true object, such as in figure 4. This is known as translational invariance.



In October 2017, Sabour, Frosst, and Hinton submitted a new paper introducing a new network method to avoid translational invariance and attempt to emulate human vision more accurately. Known as capsule networks, this new type of image processing essentially works as the inverse of computer graphics, taking a 2d image and modeling its 3d pose as seen below:

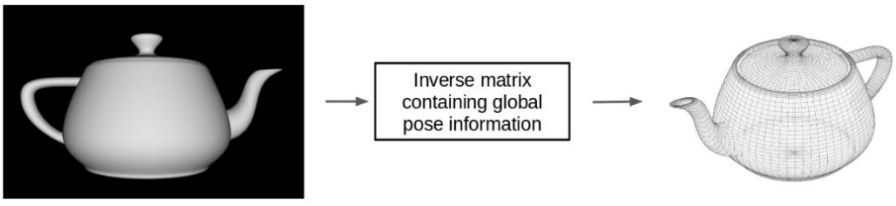


Figure 5 Concept of a capsule network

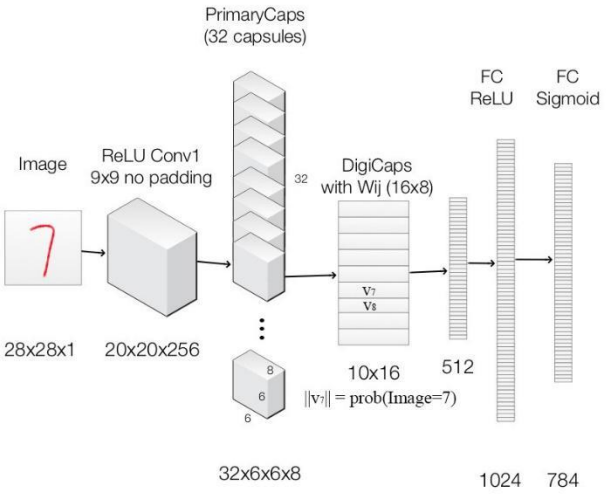
Using this method, allows the network to detect objects in different orientations, such a face head on or to the side. As a bonus to this method, the network can achieve the same accuracy as CNNs using less training data.

The paper titled "Dynamic Routing Between Capsules" presents a detailed explanation into the theory behind capsules. As defined in the paper, "a capsule is a group of neurons that

[3] https://www.clarifai.com/technology

not only capture the likelihood but also the parameters of the specific feature." In other words, the capsule is a nest of layers inside a layer. Using the MNIST handwritten numbers dataset as an example, this nested capsule of neurons not only can account for the features of each handwritten number, but also the rotation, size, and thickness.

Capsule network methodology relies on two key features: layer-based squashing and dynamic routing. Unlike convolutional networks where each individual neuron receives a non-linear function, capsules are transformed as a vector. The input is made up of a weight matrix, the previous layer's output (ReLU in the paper's CapsNet), and a coupling coefficient. Both the weights and the coupling coefficients are redefined throughout each iteration. Dynamic routing shares much in common with a convolution but includes an extra outer loop as the data moves forward in the network. The network is then updated in a method similar to backpropagation called reconstruction loss. Figure 6 offers a visualization of CapsNet on the MNIST dataset. [4]



Keras

Keras is itself a deep learning library written in Python that runs on top of more general data frameworks such as TensorFlow or Theano. Keras serves to create a user-friendly deep learning frameworks capable of easily running deep learning algorithms. Furthermore, as both Keras is heavily used within the data science community, there is significant documentation to build upon. From limited personal experience, it is noted that Keras files seem to be executed at a higher rate within the terminal as opposed to a development environment.

Methodology

Capsule Network

The CIFAR-10 capsule network was adapted from Xifeng Guo's 'CapsNet-Keras' on Github[5] used to assess the MNIST dataset as in the aforementioned article 'Dynamic Routing between Capsules." The network is visualized in figure 6. Notable changes to the code include changing the loaded Keras dataset to CIFAR-10, updating the pixel height, width, and channel (32x32x3), and changing the parameters of the capsule net to the parameters set in the article under the chapter, "Other Datasets" which include using 24x24 patches on the image along with 64 primary capsules. A batch size of 100 was used with 50 epochs, a learning rate of 0.001, a decay of 1.0, a reconstruction loss of 0.392, and 3 routings. The model uses the Adam function to optimized the function and measures the loss using mean square error. A complete copy of the code used can be found in the appendix with modifications highlighted.

[4] https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

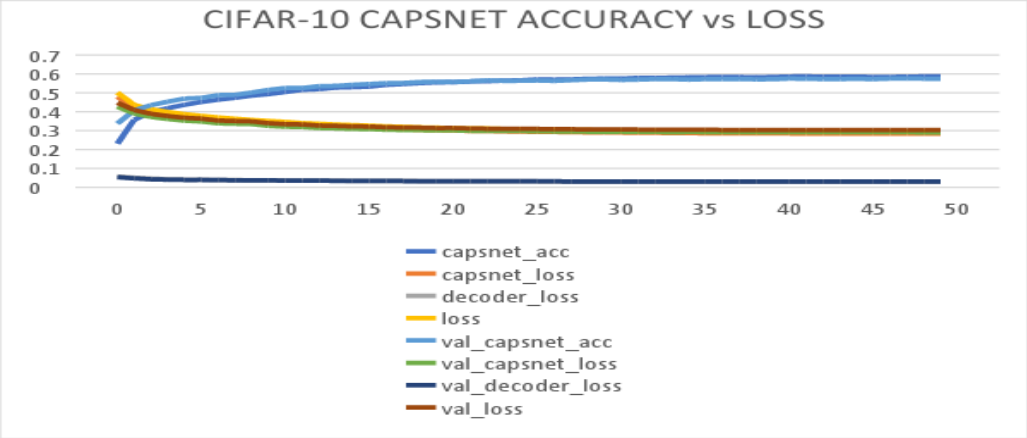
[5] https://github.com/XifengGuo/CapsNet-Keras/blob/master/capsulelayers.py

In addition to the code, a chart tracking accuracy and loss, visualizations including real and reconstructed images, and manipulations of reconstructed data from each class are included along with Tensorboard output of accuracy vs loss, and a depiction of the network can also be found in the appendix.

Analysis

Capsule Network

Using Ubuntu 14 with a Tesla K80 video card, each epoch of the CapsuleNet took about 101s to complete. The following graph displays the improvement in accuracy and loss over epochs with a convergence in the final network at 0.587 accuracy with a 0.283 loss.



Xifeng Guo's implementation of the CapsNet architecture achieves a 99.66% after 50 epochs on the MNIST handwriting dataset. Using the same architecture along with the reported parameters in the paper "Dynamic Routing between Capsules," the same model only achieved an accuracy of 0.59 while the authors report an accuracy of 0.89 in the paper. Other users on Xifeng's Github issues page report similar issues with accuracies ranging between 0.54 and 0.72.[6] While analyzing the performance of the code is beyond this paper, this disparity between open-source users and the publishers of the paper is significant and requires further analysis. Furthermore, this introductory capsule network place well below the top performing CIFAR-10 networks that all employ convolutional networks.[7]

Conclusion

Completing this analysis, it is clear that while capsule networks attempt to move beyond the limitations inherent to convolutional neural networks, refinement is needed before they contend equally in predictive modeling. Discovering the concepts and math behind capsule networks also served to be an edifying experience, as well as being introduced to Github and Reddit collaboration with authors of the paper and other cutting-edge data scientists. To be able to implement the CIFAR-10 dataset onto high-end code and obtain comparable results that other users on the internet struggled to complete served as an achievement itself.

[6] https://github.com/XifengGuo/CapsNet-Keras/issues/10

[7] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

Joseph Conran, conran@adnumerant.com

References

Dr. Amir Jafari, Machine Learning 2, The George Washington University

http://cs231n.github.io/convolutional-networks/

https://keras.io/

https://arxiv.org/abs/1710.09829

https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

https://hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc

https://www.youtube.com/watch?v=VKoLGnq15RM

https://github.com/llSourcell/capsule_networks/blob/master/Capsule%20Networks%20What%20Comes%20after%20Convolutional%20Networks%3F.ipynb

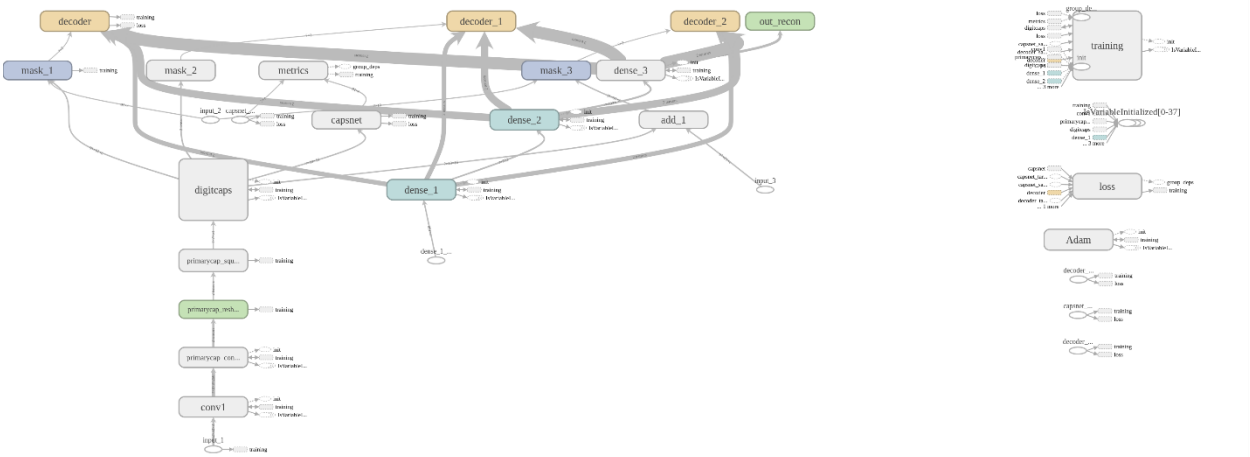
Appendix

CapsuleNet Output

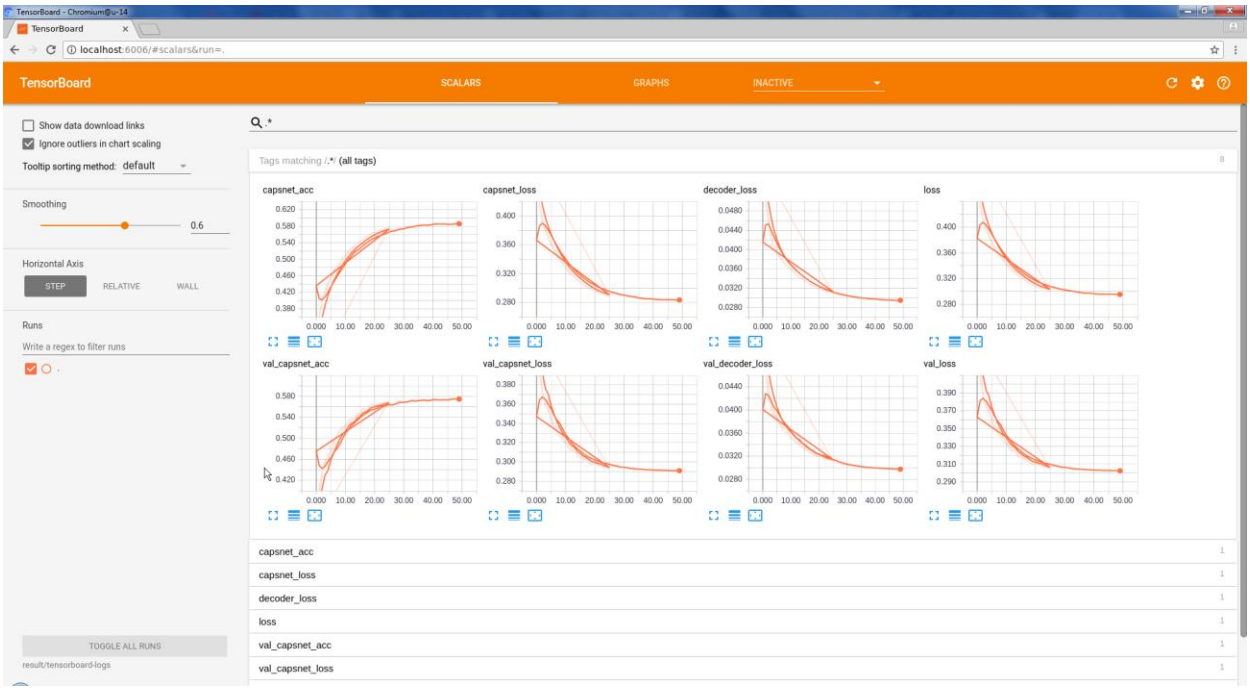
epoch	capsnet_acc	capsnet_loss	decoder_loss	loss	val_capsnet_acc	val_capsnet_loss	val_decoder_loss	val_loss
0	0.22998	0.479232975	0.056972057	0.50156602	0.337	0.426546517	0.05305731	0.447344982
1	0.356680001	0.416110002	0.050495083	0.435904074	0.406799999	0.390408774	0.046922247	0.408797295
2	0.398139999	0.39498929	0.045645121	0.412882177	0.434599998	0.372088193	0.0421378	0.388606212
3	0.418779998	0.382133231	0.04208005	0.398628611	0.452899998	0.361298949	0.040040062	0.376989653
4	0.436699998	0.37098254	0.040509727	0.386862353	0.469499995	0.352466656	0.038892544	0.367712534
5	0.451679998	0.362465234	0.039448917	0.377929209	0.472999997	0.347971664	0.039019409	0.363267271
6	0.463719998	0.354426452	0.038516307	0.369524844	0.487699997	0.338325421	0.037713648	0.353109168
7	0.473559997	0.348029762	0.037781726	0.362840199	0.489099997	0.336233167	0.037036656	0.350751536
8	0.485099997	0.341811023	0.036860923	0.356260504	0.501499997	0.334408753	0.036702454	0.348791113
9	0.493199996	0.3367502	0.036229858	0.350952304	0.515499995	0.324906432	0.036358382	0.339158918
10	0.504399997	0.332108673	0.035804625	0.346144085	0.525499997	0.320315408	0.035087896	0.334069864
11	0.515819996	0.325991696	0.035187645	0.339785252	0.5252	0.318852397	0.034881774	0.332526053
12	0.519179996	0.322641911	0.034861385	0.336307573	0.535299998	0.313256235	0.034279415	0.326693763
13	0.527839996	0.318452698	0.034382264	0.331930545	0.536499997	0.311682279	0.034063039	0.325034991
14	0.529939998	0.315956552	0.033932829	0.329258221	0.5424	0.308855574	0.03363698	0.32204127
15	0.532719996	0.312987469	0.033502659	0.326120511	0.546699996	0.308253523	0.03318484	0.32126198
16	0.542159998	0.30984638	0.03308765	0.322316738	0.551199993	0.304441136	0.03290957	0.317341687
17	0.546399997	0.307160313	0.032811964	0.320022602	0.551999997	0.303311519	0.032517326	0.316058311
18	0.551339998	0.304192696	0.032484074	0.316926453	0.556999998	0.302494057	0.032404637	0.315196673
19	0.555139997	0.302800595	0.032227618	0.31543382	0.558899997	0.300709256	0.032004997	0.313255215
20	0.556159997	0.300879746	0.031903412	0.313385883	0.556999998	0.301149922	0.031985618	0.313688284
21	0.560439998	0.297922887	0.031660715	0.310333886	0.560599997	0.29866612	0.031863209	0.311156497
22	0.561259997	0.298273499	0.031451548	0.310602506	0.565699995	0.297732582	0.031355944	0.310024112
23	0.564599998	0.295939519	0.031216513	0.308176392	0.563499998	0.297405534	0.031399784	0.309714248
24	0.566119999	0.294775273	0.031084149	0.306960259	0.5638	0.297737493	0.031323598	0.310016343
25	0.571359997	0.293167008	0.030916404	0.305286238	0.565499998	0.296498485	0.031274406	0.308758052
26	0.570059998	0.292670364	0.030731679	0.304717181	0.563199998	0.295560389	0.031018685	0.307719713
27	0.571119998	0.291757152	0.030627593	0.303763168	0.566799998	0.295207503	0.030788321	0.307276524
28	0.574599998	0.290417182	0.030508294	0.302376432	0.570899998	0.294009265	0.030635638	0.306018434
29	0.575359999	0.289573323	0.030393276	0.301487487	0.570799997	0.293783495	0.030496725	0.30573821
30	0.575059999	0.289320368	0.030279766	0.301190086	0.568200001	0.294131013	0.030427126	0.306058446
31	0.579399998	0.288472725	0.030203042	0.300812317	0.569999999	0.293436593	0.030420493	0.305361426
32	0.579019998	0.287982109	0.030085098	0.299725467	0.572799994	0.292594567	0.030324129	0.304481624
33	0.580159997	0.287076384	0.03000281	0.298837484	0.572899996	0.292304375	0.03021625	0.304149144
34	0.581799998	0.286594941	0.029924222	0.298325235	0.569999998	0.29252059	0.030142355	0.304336393
35	0.583019999	0.285510444	0.029877806	0.297222544	0.572399999	0.292442519	0.030126463	0.30425209
36	0.583699999	0.285007489	0.029876433	0.29671905	0.573199995	0.291859365	0.030042259	0.303635981
37	0.583479999	0.28466116	0.029776705	0.296333628	0.573199996	0.291723667	0.029995743	0.303481998
38	0.582039998	0.285349915	0.029685353	0.296986572	0.571499998	0.291635079	0.029970364	0.303383462
39	0.583659998	0.284724391	0.029675128	0.296357041	0.571999997	0.291814538	0.029933427	0.303548441
40	0.586279999	0.284411695	0.029652775	0.296035583	0.575499996	0.291290556	0.029908005	0.303014495
41	0.586539997	0.283571736	0.029606209	0.29517737	0.574199992	0.291236495	0.029889216	0.302953067
42	0.585859998	0.283805008	0.029549607	0.295388454	0.572999996	0.291343928	0.029873785	0.30305445
43	0.584719998	0.28359195	0.029538753	0.295171141	0.572999996	0.291085296	0.029832979	0.302779824
44	0.585859999	0.284233721	0.029495955	0.295796134	0.574099993	0.291017354	0.029826029	0.302709158
45	0.583899998	0.284057544	0.029516647	0.295628069	0.572699995	0.291116797	0.029787215	0.302793385
46	0.584019999	0.284082423	0.029475371	0.295636767	0.575299996	0.29094262	0.029759758	0.302608446
47	0.585959999	0.283423004	0.029438003	0.294962701	0.576099994	0.290966515	0.029739135	0.302624256
48	0.586499999	0.283351582	0.029445608	0.29489426	0.574899996	0.290748798	0.029724419	0.30240077
49	0.586739999	0.283447905	0.029401408	0.294973256	0.574699996	0.290841718	0.029708947	0.302487624

Joseph Conran, conran@adnumerant.com

Tensorboard Network Visalized

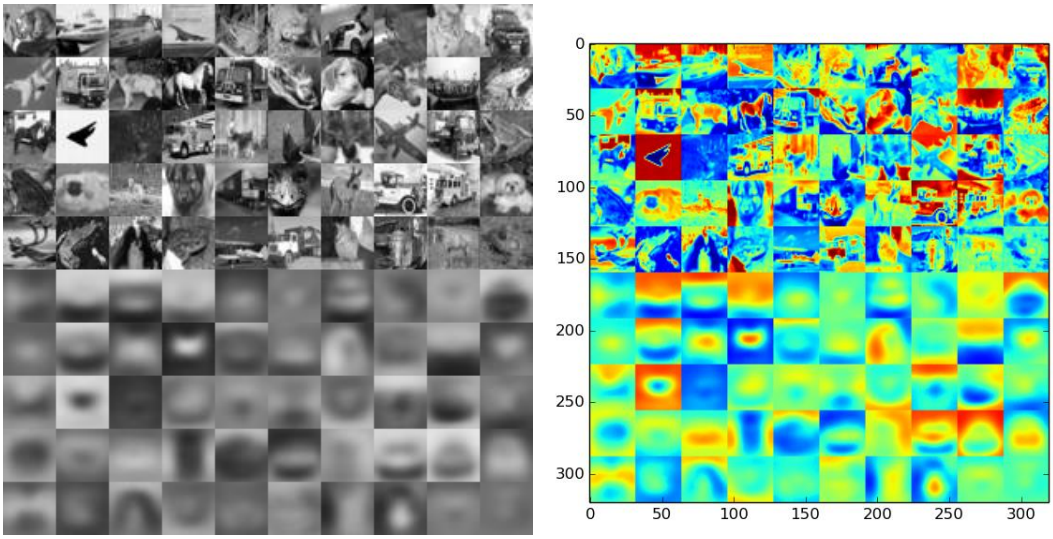


Tensorboard Accuracy and Loss Rates Visualized



Joseph Conran, conran@adnumerant.com

Real vs Reconstructed Images from CIFAR-10 using CapsNet



Joseph Conran, conran@adnumerant.com