Effect of CapsuleNet Discriminator on GANs

Abhijith C.
Sir. M. Visvesvaraya Institute of Technology
Bangalore
Karnataka, India
abhijith4m0505@gmail.com

Raghava G. Dhanya
Sir. M. Visvesvaraya Institute of Technology
Bangalore
Karnataka, India
raghava.dhanya@gmail.com

Shashank S.
Sir. M. Visvesvaraya Institute of Technology
Bangalore
Karnataka, India
mrknhash@gmail.com

ABSTRACT

Current advances in Generative Adversarial Networks allow us to obtain near realistic images but it is still quite distinguishable from actual photographic images. The technology is also not very amiable to changes in the orientation of images in Convolutional Neural Networks(CNN). Additionally, the amount of data required to train the network must be exhaustible, for example, in case different perspectives of a face are required the various perspectives must be explicitly present in the training data to achieve the result. Thus the network requires humongous amounts of data. In this project we propose a novel approach to accomplish the same results using CapsNet. CapsNet employs a dynamic routing algorithm which replaces the scalaroutput feature detectors of the CNN with vector-output capsules. A capsule is essentially a group of neurons describing a specific part of object or image. Active capsules at one level make predictions, via transformation matrices, for the instantiation parameters of higher-level capsules. In essence, the CapsNet is the reverse of the common Computer Graphics pipeline where we convert objects to their renders. The CapsNet works from the pixel level and works up towards the object. We propose that the amount of data required to train a comparable model is very small while it gives comparable, if not better, results

1. INTRODUCTION

A generative model is a mathematical or statistical model to generate all values of a phenomena. To train such a model, we first collect a large amount of data in some domain (e.g., think millions of images, sentences, or sounds, etc.) and then train a model to generate data like it. A generative algorithm models how data was generated to classify a data instance. It poses the question: according to my generation hypotheses, which category is most likely to generate this data instance? A discriminative algorithm does not care about how the data was generated, it just classifies a given data instance; that is, given the features of a data instance, they predict a label or category to which that data belong. Discriminative models

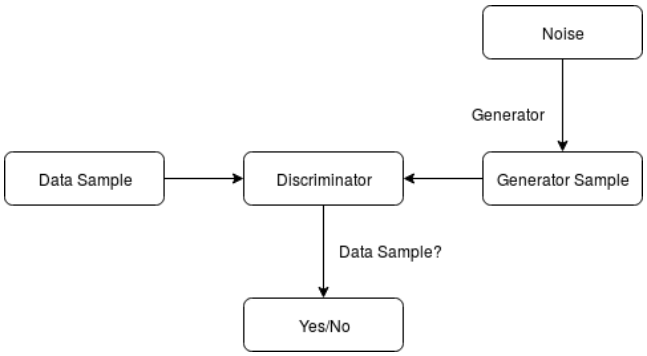

Figure 1: Vanilla Generative Adversarial Network

learn the boundary between classes while Generative models model the distribution of individual classes; that is, a generative model learns the joint probability distribution p(x, y) while a discriminative model learns the conditional probability distribution p(y | x) "probability of y given x". The trick is that the neural networks that we use as generating models have a significantly smaller number of parameters than the amount of data on which we train them, so the models are forced to effectively discover and internalize the essence of the data to generate it. There are multiple approaches to build a generative models.

1.1 Generative adversarial networks

One such model is the Generative adversarial networks (GANs), which are a class of generative algorithms used in unsupervised machine learning, implemented by a system of two neural networks competing in a zero-sum game framework. They were presented by Ian Goodfellow et al. [12]. This technique can generate photographs that seem at least superficially authentic to human observers, having many realistic features (though in tests people can tell real from generated in some cases). Generative Adversarial Networks, which we already discussed above, pose the training process as a game between two distinct networks: A neural network, called the generator, generates new instances of data, while the other, the discriminator, evaluates their authenticity; discriminator network tries to classify samples as either coming from the true distribution $p(x)$ or the model distribution $\hat{p}(x)$. Every time the discriminator notices a difference between the two distributions the generator adjusts its parameters slightly to make it go away, until at the end (in theory) the generator exactly reproduces the true data distribution

and the discriminator is guessing at random, unable to find a difference. The generator takes noise as input and attempts to produce an image that belongs to the real distribution; that is, it tries to fool the discriminator to accept it as real image. Discriminator takes a generated image or a real image as input and attempts to correctly classify the image as real or fake (generated). To learn the distribution of the generator p_g over data $\boldsymbol{x}$, we define a prior on input noise variables $p_z(\boldsymbol{z})$, then represent a mapping to data space as $G(\boldsymbol{z};\theta_g)$, where G is a differentiable function represented by a neural network with parameters θ_g. We define a second neural network $D(\boldsymbol{x};\theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to the training examples and samples of G. We simultaneously train G to minimize $\log(1 - D(G(\boldsymbol{z})))$. This can be represented minimax game

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \quad (1)$$

1.2 Convolutional Neural Networks

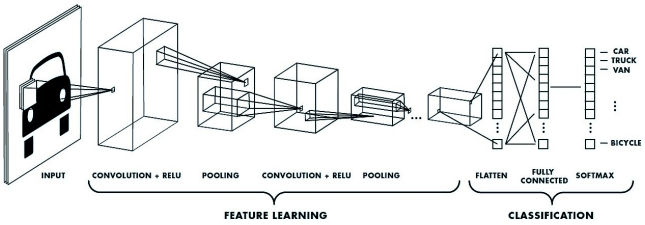

Figure 2: Convolutional Neural Network

Before we can jump to understanding Capsule Networks, we need to know about Convolutional Neural Networks(CNNs). Convolutional neural networks are very similar to ordinary neural networks, they consist of neurons that have learn-able weights and biases. Each neuron receives inputs, performs a scalar product and possibly follows it with a nonlinearity. The entire network expresses a single differentiable score function: raw image pixels at one end to class scores at the other end. And they still have a loss function on the last layer.

The major difference is that CNN explicitly assumes that the inputs are images, which allows us to encode certain properties in the architecture. These then make the forward functions more efficient to implement and significantly reduces the amount of parameters in the network.

Ordinary neural networks don't scale well to full images, for example, A colour image of dimensions of 150x150 (which is considered as low resolution by most people) has a shape (150,150,3), a fully connected neuron on first layer which receives this image would require 67500 weights. Unlike an ordinary neural network, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth as shown in figure 2. The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. CNN will reduce the full image into a single vector of class scores, arranged along the depth dimension.

CNNs use a "Pooling" layer to reduce the spatial size of the input for each convolutional layer. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, generally using the MAX operation, hence pooling layer is sometimes referred to as Max Pooling layer.

1.3 Capsule Networks

"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster." says Geoffrey Hinton, one of the founders of deep learning (Also known as Godfather of Deep Learning) and an inventor of numerous models and algorithms that are widely used today. CNNs perform exceptionally great when they are classifying images which are very close to the data set. If the images have rotation, tilt or any other different orientation then CNNs have poor performance. This problem is usually partially solved by adding different variations of the same image during training. But CNNs still require large amount of data to perform reasonably well. We use pooling after each layer to make it compute in reasonable time frames. But in essence, it also loses out the positional data.

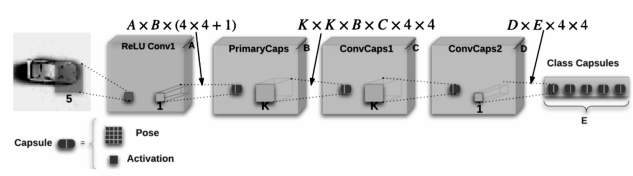

Figure 3: Capsule Networks

What we need is not invariance but equivariance. Invariance makes a CNN tolerant to small changes in the viewpoint. Equivariance makes a CNN understand the rotation or proportion change and adapt itself accordingly so that the spatial positioning inside an image is not lost. This leads us to Capsule Networks.

Capsule is a nested set of neural layers as shown in figure 3. Capsules are like cortical columns in human brains. Deep neural nets learn by back-propagation of errors over the entire network. In contrast real brains supposedly wire neurons by Hebbian principles: "units that fire together, wire together". Capsules mimic Hebbian learning in the way that: "A lower-level capsule prefers to send its output to higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsule". Capsules, combination of capsules encodes objects parts AND their relative positions, so an object instance can be accurately derived from the presence of the parts at the right locations, and not just their presence. Capsules produce equivariant features. Capsules predict the activity of higher-layer capsules to route information to right higher-layer capsules, this is called "Dynamic routing".

2. RELATED WORK

GANs were first introduced by Ian Goodfellow et al. [12] in Neural Information Processing Systems 2014. The paper

proposes a completely new framework for estimating generative models via an adversarial process. In this process two models are simultaneously trained. According to [12] the network has a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. This original work by Ian Goodfellow uses fully connected neural networks in the generator and the discriminator.

2.1 DCGAN

Since GANs were introduced, there has been tremendous advancements in Deep Learning. A convolutional neural network (CNN, or ConvNet) [**?**] is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. The convolution layer parameters consist of a set of learn-able filters, also called as kernels, which have a small receptive field, but they extend through the full depth of the input volume. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

A breakthrough development that occurred in Adversarial Networks was the introduction of "Deep Convolutional Generative Adversarial Networks" by Alec Radford *et al* [**?**]. DCGAN uses CNNs as generator and discriminator as shown in 4. He applied a list of empirically validated tricks as the substitution of pooling and fully connected layers with convolutional layers.

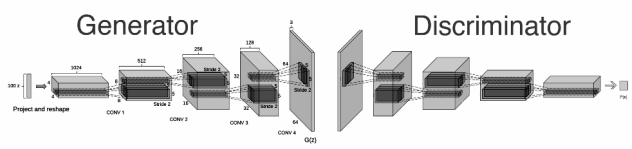

Figure 4: Deep Convolutional Generative Adversarial Network

Today, most GANs are loosely based on the former shown DCGAN [**?**] architecture. Many papers have focused on improving the setup to enhance stability and performance. Many key insights was given by Salimans *et al*. [**?**], like Usage of convolution with stride instead of pooling, Usage of Virtual Batch Normalization, Usage of Minibatch Discrimination in DD, Replacement of Stochastic Gradient Descent with Adam Optimizer [6], Usage of one-sided label smoothing.

2.2 InfoGAN

The power of the features encoded in the latent variables was further explored by Chen at al. [**?**]. They propose an algorithm which is completely unsupervised, unlike previous approaches which involved supervision, and learns interpretable and disentangled representations on challenging datasets. Their approach only adds a negligible computation cost on top of GAN and is easy to train.

2.3 ACGAN

Augustus Odena *et al* [**?**] came up a improved training of generative adversarial networks and variant of GAN with employing label conditioning that results in image samples exhibiting global coherence. ACGAN uses an auxiliary classifier to control the minimax game between generator and discriminator. In their work they demonstrate that adding more structure to the GAN latent space along with a specialized cost function results in higher quality samples

2.4 WGAN

Another huge development came with the introduction of Wasserstein GANs by Martin Arjovsky [**?**] . He introduced a new algorithm named WGAN, an alternative to traditional GAN training. In this new model, he showed that the stability of learning can be improved, remove problems like mode collapse, and provide good learning curves useful for debugging and hyperparameter searches.

This recently proposed Wasserstein GAN (WGAN) [**?**] makes progress toward stable training of GANs, but sometimes can still generate only low-quality images or fail to converge. Ishaan Gulrajani with Martin Arjovsky proposed an alternative in [**?**] to fix the issues the previous GAN faced. This proposed method performs better than standard WGAN and enables stable training of a wide variety of GAN architectures with almost no hyperparameter tuning, including 101-layer ResNets [**?**] and language models over discrete data.

2.5 Other GANs

Work by Mehdi Mirza *et al*. [**?**] introduced the conditional version of GAN which can be constructed by simply feeding the data, y, we wish to condition on to both the generator and discriminator. The CGAN results were comparable with some other networks, but were outperformed by several other approaches – including non-conditional adversarial nets.

Sebastian Nowozin *et al*. [**?**] discussed the benefits of various choices of divergence functions on training complexity and the quality of the obtained generative models. They show that any f-divergence can be used for training generative neural samplers.

Ming-Yu *et al*. [**?**] proposed coupled generative adversarial network (CoGAN) for learning a joint distribution of multi-domain images. The existing approaches requires tuples of corresponding images in different domains in the training data set. CoGAN can learn a joint distribution without any tuple of corresponding images.

2.6 Capsule Neural Network

A big breakthrough in the field of Deep Learning came with the introduction of CapsNets or Capsule Networks [**?**] by the Godfather of Deep Learning, Geoffrey Hinton *et al*. CNNs perform exceptionally great when they are classifying images which are very close to the data set. If the images have rotation, tilt or any other different orientation then CNNs have poor performance. A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an

object part. They use the length of the activity vector to represent the probability that the entity exists and its orientation to represent the instantiation parameters. Active capsules at one level make predictions, via transformation matrices, for the instantiation parameters of higher-level capsules. When multiple predictions agree, a higher level capsule becomes active. They show that a discrimininatively trained, multilayer capsule system achieves state-of-the-art performance on MNIST and is considerably better than a convolutional net at recognizing highly overlapping digits. To achieve these results they use an iterative routing-by-agreement mechanism: A lower-level capsule prefers to send its output to higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsule.

3. PROPOSED ARCHITECTURE

The generator will use noise as input to generate faces. We will use random data as this noise. This ensures the data is unique and across the spectrum while retaining a normal distribution.

The CapsNet making up the discriminator consists of a small convolutional network to convert low level data in the form of pixels into an artifact called "pose". These poses can be anything, like nose, ear, eye, etc. These poses are then passed on as input to the later lower layers consisting of components called Capsules. A capsule is analogous to the human brain containing different modules to handle different tasks. The brain has a mechanism to route the information among the modules, to reach the best modules that can handle the information.

A capsule is a nested set of neural layers. Each capsule is able to handle one particular pose and communicate its calculation to other capsules which can use that calculation. This calculation is in the form of a probability prediction of the current pose that takes place in its logistic unit. This working is fundamentally different from convolutional networks, which utilizes Max Pooling. Max pooling selects the most active input node from the next layer to pass on the information. CapsNet on the other hand selects the next capsule based on which capsule would be capable of handling that information. This is called Dynamic routing. This results in invariance of information to the position and orientation of features in an object while ignoring the invariance in very low level features as, at the pixel level, this does not matter.

3.1 Generator

We use a deep convolutional generator model, which is similar to what DCGAN uses as generator. It starts with a latent vector or noise of shape 100 which connects to a densely connected neural layer. We then use Reshape layer which reshapes it to an (8, 8, 128) matrix and which is then sent to the batch normalization layer. We later perform up-sampling by using DeConv (De-Convolutional layer: transposed convolutional layer) layer. DeConv layer is internally implemented by an up-sampling layer and a convolutional layer. We perform DeConv two more times to get the shape of the image as expected, which should be (64, 64, 3), which stands for a 64x64 RGB image.

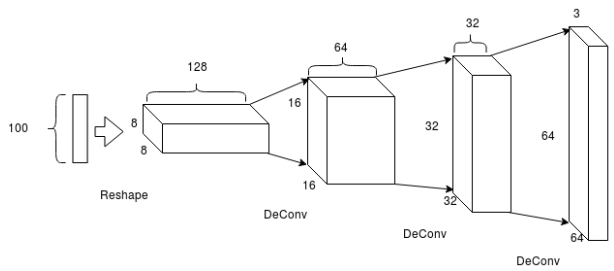

Figure 5: Generator architecture

3.2 Discriminator

The discriminator in the original models is replaced by our modification of CapsNet. We use a binary classifier CapsNet to distinguish between real and fake images.

The CapsNet has 2 parts: encoder and decoder. The first 3 layers are encoder, and the second 3 are decoder:

Layer 1. Convolutional layer Layer 2. PrimaryCaps layer Layer 3. DigitCaps layer Layer 4. Fully connected #1 Layer 5. Fully connected #2 Layer 6. Fully connected #3

In our implementation, we do not use the decoder layers as we do not need the reconstruction aspects of the network for classification. Hence only the encoder layers are used.

The encoding part of the network takes as input a digital image of 64x64 and learns to encode it into a vector of 16 dimensions of instantiation parameters, this is where the capsules do their job. The output of the network during prediction is a 10-dimensional vector of the lengths of the DigitCaps' outputs.

3.2.1 Layer 1: Convolutional Layer

Input: 64x64 image (three color channel).
Output: 56x56x256 tensor.
Number of parameters: 62464.

The work of the convolutional layer consists of detecting the basic functions in the 2D image. In the CapsNet system, the convolutional layer has 256 kernels of size 9x9x1 and stride of 1, followed by the activation function, ReLU. To calculate the number of parameters, we must also remember that each kernel in a convolutional layer has 1 bias term. Therefore, this layer has (9 x 9 x 3 x 256 + 256 =) 62464 trainable parameters in total.

3.2.2 Layer 2: PrimaryCaps Layer

Input: 56x56x256 tensor.
Output: 24x24x8x32 tensor.
Number of parameters: 5308672.

This layer has 32 primary capsules whose job is to take basic features detected by the convolutional layer and produce combinations of the features. The layer has 32 "primary capsules" that are very similar to convolutional layer in their nature. Each capsule applies eight 9x9x256 convolutional kernels to the 56x56x256 input volume and therefore produces 24x24x8 output tensor. Since there are 32 such capsules, the output volume has shape of 24x24x8x32. Doing calculation similar to the one in the previous layer, we get (9 x 9 x 256 x 256 + 256 =) 5308672 trainable parameters in this layer.

3.2.3 Layer 3: DigitCaps Layer

Input: 24x24x8x32 tensor.
Flattened to: 147456
Output: 1x1 matrix.
Number of parameters: 1497600.

We have a 3 hidden-layer densely connected neural network which takes the flattened input to give a binary classification output of 1x1. Each hidden layer consists of 160 neurons. From the flattened input we get the input of size 147456 which is connected densely to the first hidden layer. Thus the weights in the first layer turn out to be (147456 x 160 + 160 =) 23593120 in number. The first hidden layer is connected to the second hidden layer densely, similarly the second and the third. Therefore there are (160 x 160 + 160 =) 25760 trainable parameters for the second and third hidden layers.

The last last hidden layer is connected to a output layer with one neuron, which essentially gives the binary classification result. Its parameters are (160 x 1 + 1 =) 161 in number.

4. IMPLEMENTATIONS

The first step is to implement the state-of-the-art in image regeneration to guage the improvements. We use DCGAN to start of with. The results of the training and testing will be recorded to compare it with the results of our CapsNet-based approach later. We will be using CapsNet as the underlying technology to implement our GAN (CapsGAN). The goal is to replace the CNN inside DCGAN with CapsNet and compare the results. The GAN internally consists of two components - a generator and a discriminator - which we build out of CapsNet. The discriminator is initially trained separately to distinguish real and fake data, and later they work together to improve upon their performance by acting as adversaries.

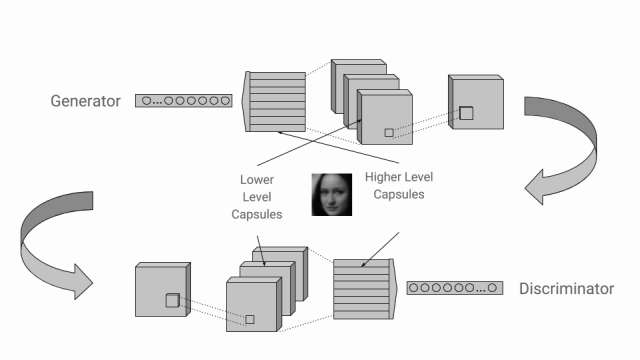

Figure 6: Proposed architecture

During the course of our research we were forced to conclude that building a CapsNet based generator was not feasible. A fundamental aspect of a GAN is back-propagation in the discriminator which is not possible to replicate in the generator. Hence we implemented just the discriminator in CapsNet.

We concentrated on four networks: DCGAN, WGAN, ACGAN and InfoGAN. Our training laboratory was Colloboratory - the cloud machine learning research platform. We trained each network individually for 20,000 epochs each. For our preliminary training we used the MNIST dataset. The MNIST dataset is a large dataset of handwritten digits commonly used for image processing training tasks. Each of the networks had it's discriminator augmented with the CapsNet code. The networks with the CapsNet discriminator were then individually trained on the same dataset, for 20,000 epochs each. Overall, it took us a few days to train all the networks and gather all the data.

4.1 Network Architectures

The following are the network architectures of the generator and the discriminator.

4.1.1 Generator

Layer (**type**)	Output Shape	Param #
dense_5 (Dense)	(None, 8192)	827392
reshape_1 (Reshape)	(None, 8, 8, 128)	0
batch_normalization_3 (Batch	(None, 8, 8, 128)	512
up_sampling2d_1 (UpSampling2	(None, 16, 16, 128)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	147584
activation_1 (Activation)	(None, 16, 16, 128)	0
batch_normalization_4 (Batch	(None, 16, 16, 128)	512
up_sampling2d_2 (UpSampling2	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	73792
activation_2 (Activation)	(None, 32, 32, 64)	0
batch_normalization_5 (Batch	(None, 32, 32, 64)	256
up_sampling2d_3 (UpSampling2	(None, 64, 64, 64)	0
conv2d_3 (Conv2D)	(None, 64, 64, 32)	18464
activation_3 (Activation)	(None, 64, 64, 32)	0
batch_normalization_6 (Batch	(None, 64, 64, 32)	128
conv2d_4 (Conv2D)	(None, 64, 64, 3)	867
activation_4 (Activation)	(None, 64, 64, 3)	0

Total params: 1,069,507
Trainable params: 1,068,803
Non-trainable params: 704

4.1.2 Discriminator

Layer (**type**) *Connected to*	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 3)	0
conv1 (Conv2D) input_1[0][0]	(None, 56, 56, 256)	62464
leaky_re_lu_1 (LeakyReLU) conv1[0][0]	(None, 56, 56, 256)	0
batch_normalization_1 (BatchNor leaky_re_lu_1[0][0]	(None, 56, 56, 256)	1024
primarycap_conv2 (Conv2D) batch_normalization_1[0][0]	(None, 24, 24, 256)	5308672

```
primarycap_reshape (Reshape)    (None, 18432, 8)     0           leaky_re_lu_4 (LeakyReLU)       (None, 160)          0
primarycap_conv2[0][0]                                           multiply_3[0][0]
primarycap_squash (Lambda)      (None, 18432, 8)     0           dense_4 (Dense)                 (None, 1)            161
primarycap_reshape[0][0]                                         leaky_re_lu_4[0][0]
batch_normalization_2 (BatchNor (None, 18432, 8)     32          Total params: 29,042,753
primarycap_squash[0][0]                                          Trainable params: 29,042,225
flatten_1 (Flatten)             (None, 147456)       0           Non-trainable params: 528
batch_normalization_2[0][0]
uhat_digitcaps (Dense)          (None, 160)          23593120
flatten_1[0][0]
softmax_digitcaps1 (Activation) (None, 160)          0
uhat_digitcaps[0][0]
dense_1 (Dense)                 (None, 160)          25760
softmax_digitcaps1[0][0]
multiply_1 (Multiply)           (None, 160)          0
uhat_digitcaps[0][0]
leaky_re_lu_2 (LeakyReLU)       (None, 160)          0
multiply_1[0][0]
softmax_digitcaps2 (Activation) (None, 160)          0
leaky_re_lu_2[0][0]
dense_2 (Dense)                 (None, 160)          25760
softmax_digitcaps2[0][0]
multiply_2 (Multiply)           (None, 160)          0
uhat_digitcaps[0][0]
leaky_re_lu_3 (LeakyReLU)       (None, 160)          0
multiply_2[0][0]
softmax_digitcaps3 (Activation) (None, 160)          0
leaky_re_lu_3[0][0]
dense_3 (Dense)                 (None, 160)          25760
softmax_digitcaps3[0][0]
multiply_3 (Multiply)           (None, 160)          0
uhat_digitcaps[0][0]
dense_3[0][0]
```

5. CONCLUSIONS

6. ACKNOWLEDGMENTS

6.1 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references).

APPENDIX

You can use an appendix for optional proofs or details of your evaluation which are not absolutely necessary to the core understanding of your paper.

A. FINAL THOUGHTS ON GOOD LAYOUT

Please use readable font sizes in the figures and graphs. Avoid tempering with the correct border values, and the spacing (and format) of both text and captions of the PVLDB format (e.g. captions are bold).

At the end, please check for an overall pleasant layout, e.g. by ensuring a readable and logical positioning of any floating figures and tables. Please also check for any line overflows, which are only allowed in extraordinary circumstances (such as wide formulas or URLs where a line wrap would be counterintuitive).

Use the `balance` package together with a `\balance` command at the end of your document to ensure that the last page has balanced (i.e. same length) columns.