Comparison of CNN Architectures in Regards to Performance for Facial Recognition

Chris Ermel - 100934583
Seena Rowhani - 100945353

Contents

Background

Introduction

Parsing and performing classification on a set of faces represents a complex problem where the typical multi-layer perceptron (MLP) falls short in comparison to that of a convolutional neural network (CNN). When performing classification, MLP's fail to consider different features of the same class, and that can be located spatially in different regions of a given image vector. CNN's are able to extract a set of features by scanning the smaller subsets of a given input matrix, and determine significance of a relationship between several features. When given the task of identifying a face, most information has to do with segments of the face in relation to others. The standard neural network faces issues when considering how many parameters to train the network on, avoiding overfitting the weights, and avoiding local optimums. For example, consider that for the standard network, an image with dimensions 1000 by 1000 would result in over one billion parameters to train the network on. CNN's approaches the problem differently. By targeting spacial relationships, many of the parameters that a traditional network would store, the size of your parameter space is reduced drastically. Through use of pooling layers, we gain robustness to the exact spatial location of features. By subsampling the image, we allow the next layer to parse significance from groupings of these subsamples. It also allows us to condense information regarding position, and focus more on the features themselves.

Through experimentation of several convolutional network configurations, with regards to layering, pooling size, and kernel size, we will attempt to minimize loss and compare classification accuracies. Experimentation will be performed using the '*Labeled Faces in the Wild*' dataset contains images of various celebrities, artists, or famous figures. Several network configurations will be trained and evaluated with a working subset of the lfw-dataset, where number of output classes will be reduced to *106*, by filtering out images that have less than 14 representations. Each image, originally with dimensions 250 by 250, will also be scaled down to a more manageable 64x64, such that less memory will need to be allocated to process the images. The dataset will then be batch fed into a handful of Convolutional Neural Networks, each with different initialization parameters, and layer configurations. Using the tf.estimator.Estimator module, we can form a set of predictions, iterate over each prediction, and measure instances where the network's retrieved value matches the actual testing subset label.

Preprocessing

In an attempt to increase classification accuracy, preprocessing to the set of training vectors, can be performed in an attempt to increase the diversity for a given label's representation. Image manipulation are performed to increase variance of input; performing skew transforms for a small subset, as well as injecting noise to another given subset, to observe effect on overall convergence and loss. Combinations of the application, or lack of, preprocessing will be included as independent influences to overall accuracy measures, with regards to the different architectures of the CNN itself.

Architecture

As mentioned before, the network architecture subject to classification accuracy performance was a Convolutional Neural Network (CNN). We tested the architecture against the eigenfaces dataset, which contains pictures of various face, initially defined as 3-color channel split (RBG). These color channels are interpreted as features when fed into the first convolutional layer, where based on that layers configuration, we continue processing features through the network's architectural pipeline. Features are processed into many more features in each convolutional layer, which scans the input matrix based on the dimensions of the convolutional kernel. The output of each convolutional layer is pooled in order to subsample the feature data outputted by its component convolutional layer. This step is repeated several times, dependant on network configuration. Eventually, the final pooling layer is flattened into one or more fully connected layers (also called dense layers), in order to prepare the features for classification. Weight initialization for the layers is done using the Xavier method provided by tensorflow. The deep layers are then converted to logits and classified using softmax.

Results & Discussion

The purpose of this section is to communicate and analyze the results of the experimentation performed with the CNN architecture that was implemented. Various figures are presented in the results section, detailing classification accuracy as a factor of the network configuration used in the model. The discussion section summarizes these findings, and describes.

Results

Four different Convolutional Neural network configurations were used in order to determine the best one that fit the model. In all cases, we operated with a learning rate of 0.001 and trained the networks over roughly 158 epochs, while using the relu function for activation. The configurations and results of each are described below.

Single Convolutional Layer

An attempt at a convolutional neural network with a single convolutional layer was made to represent the given dataset. Network was configured to take input as 64x64 3 channel (RGB) tensor. Was then processed through the first convolutional layer using a 5x5 kernel, where over 128 features were extracted. Output is then inputted into 2x2 max pooling layer, where half of the volume of the inputted tensor will be downsampled - reducing the computational complexity and memory needed to process information. Pooling layer is then reduced into one dimension, and fed into a dense layer with 1024 units. Dropout was then utilized to prevent overfitting, and their corresponding activations were processed through softmax function post conversion to logits. Classification accuracy had managed to peak ~81% with loss of 1.23 after training. This configuration managed to obtain classification accuracy of 83% with a loss factor of 1.1. Figure 1 contains this network architecture.

Two Convolutional Layers

The next network configuration that we experimented on contained two convolutional and pooling layers. The first convolutional layer used a 5x5 kernel and generated 32 output features, which fed into 2x2 max pooling layer. The second convolutional layer again used a 5x5 kernel and generated 64 output features which then passed again into another 2x2 max pooling layer. The second pooling layer was then squashed and fed into a dense layer with 1024 hidden neurons. Dropout was then performed on these hidden neurons, and their activations were passed to a softmax layer after being converted to logits. This configuration managed to obtain classification accuracy of 83% with a loss factor of 1.1. The architecture of this network can be seen in Figure 2.

Three Convolutional Layers

The network containing the three convolutional layers had made use of extracting higher sets of activation maps, doubling that amount of perceived action maps per layer. Each convolutional layer was coupled with a 2x2 max pooling layer, reducing the size of the input tensor. Each convolutional layer had a 5x5 kernel, however the first layer extracted 64 features, the second convolutional layer output 128 features, and the final convolutional layer output 256 features which was then fed into a dense layer containing 1024 hidden neurons. Dropout was then performed on these hidden neurons, and their activations were passed to a softmax layer after being converted to logits. This configuration managed to obtain classification accuracy of 88% with a loss factor of 0.73. The architecture of this network can be seen in Figure 3.

Experimental Configuration

Due to the success of the three convolutional layer network, we attempted to classify the data set on a purely experimental configuration which we pulled out of thin air. The first and second convolutional layers both used a 5x5 kernel, with the first convolutional layer generating 32 features and passing these directly into the second convolutional layers which generated 32 features. The second convolutional layer than ran into a 2x2 max pooling layer after which dropout was executed. After this, another two consecutive convolutional layers were connected, each with a 3x3 kernel and each generating 64 features. This in turn was fed into a 2x2 max pooling layer, on which dropout is executed. Finally, this output is fed into a dense layer with 1024 hidden neurons, and their activations were passed to a softmax layer after being converted to logits. This configuration managed to obtain classification accuracy of 61% with a loss factor of 1.6. The architecture of this network can be seen in Figure 4.

Discussion

Of the four neural nets experimented on, clearly the three convolutional layer neural net represented by Figure 3 provided the optimal classification accuracy. We believe this to be a result of the fact that human faces contain many distinguishing features. As such, it is necessary to extract as many as those features as possible, possibly with many convolutional layers. However, we saw that there was not necessarily a direct increase in accuracy for each additional convolutional layer. Instead it was purely a factor of the number of features extracted, which multiple convolutional layers for an increase in.

It is worth noting that although four network configurations are shown here, there were many more models that were tested prior to these ones. Much trial and error was required prior to obtaining the network parameters that allowed for the high classification accuracies seen in this paper. In particular, tweaking of the batch size, learning rate, and number of training epochs was essential in order to allow the system to train to its maximum potential. Furthermore, we frequently ran into problems with overfitting the model to the data. In these scenarios, we would typically see a classification accuracy of roughly 50-55% during the test iterations. In these scenarios, we began training over less epochs as well as using dropout between layers.

Due to the number of layers in the convolutional nets, it was necessary that we trained the models using Tensorflow-GPU with an NVIDIA graphics card. This dramatically sped up the time required to train a model, however we still saw training times upwards of 30 minutes. As such, we sought ways in which to pre-process the input data so as to reduce the amount of time required to train a model. To this end, we experimented with different image input resolutions, eventually settling with 64x64.

Future improvements to this project could possibly be made through investigation of other methods of data preprocessing would include utilizing principal component analysis (PCA) on the input data to even further reduce the model training time. Once this has been done, it would be interested to add a fourth convolutional layer in order to double the number of processed features from the three convolutional layer network. In this sense, we hypothesize that we would attain better classification accuracy, not only because of the increased number of features, but also because we would be able to train over a larger number of epochs due the performance gained by using only the principle components of the data. Another potential suggestion would involve investigating other published facial recognition CNN architectures and attempting to implement and/or improve those.

Conclusion

In conclusion, through the experimentation with several configurations of CNNs, we were able to find that there was a positive correlation between the number of features extracted by the number of convolutional layers, and the classification accuracy of the system. Of the four network configurations presented in this paper, the three layer convolutional network achieved the best classification accuracy with 88%.

Figures

Figure 1

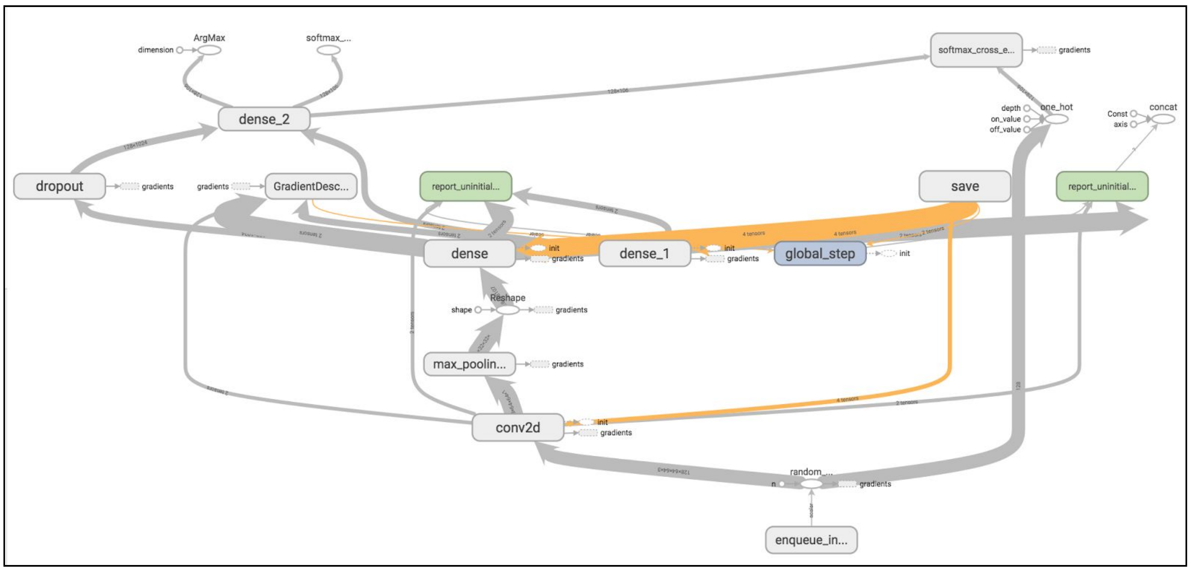


Figure 2

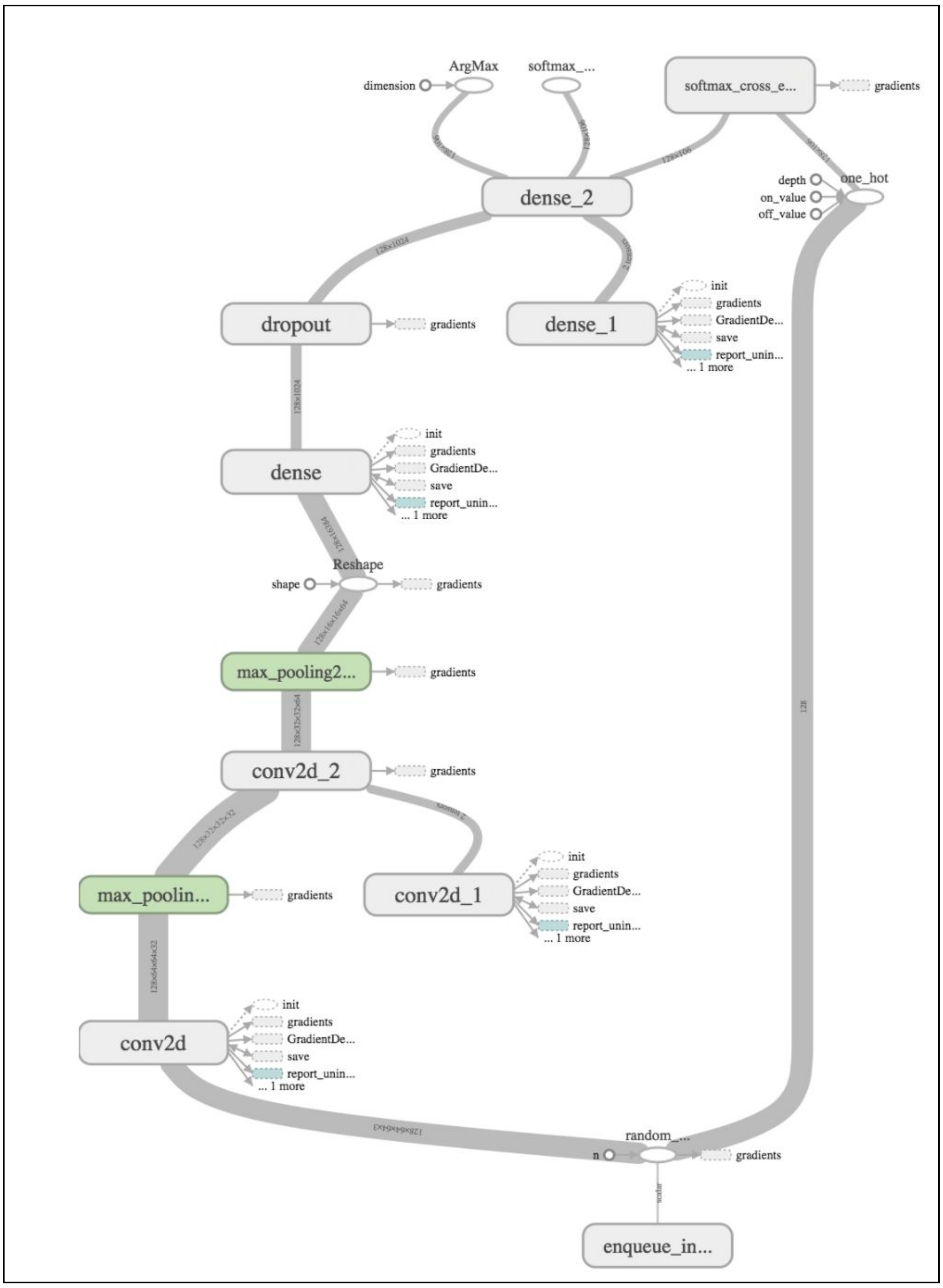

ArgMax
softmax_...
dimension
softmax_cross_e...
gradients
128×106
one_hot
depth
on_value
off_value
dense_2
128×1024
2 tensors
dropout
gradients
dense_1
init
gradients
GradientDe...
save
report_unin...
... 1 more
128×1024
dense
init
gradients
GradientDe...
save
report_unin...
... 1 more
128×16384
Reshape
shape
gradients
128×16×16×64
max_pooling2...
gradients
128×32×32×64
conv2d_2
gradients
128
128×32×32×32
2 tensors
max_poolin...
gradients
conv2d_1
init
gradients
GradientDe...
save
report_unin...
... 1 more
128×64×64×32
conv2d
init
gradients
GradientDe...
save
report_unin...
... 1 more
128×64×64×3
random_...
n
gradients
scalar
enqueue_in...

Figure 3

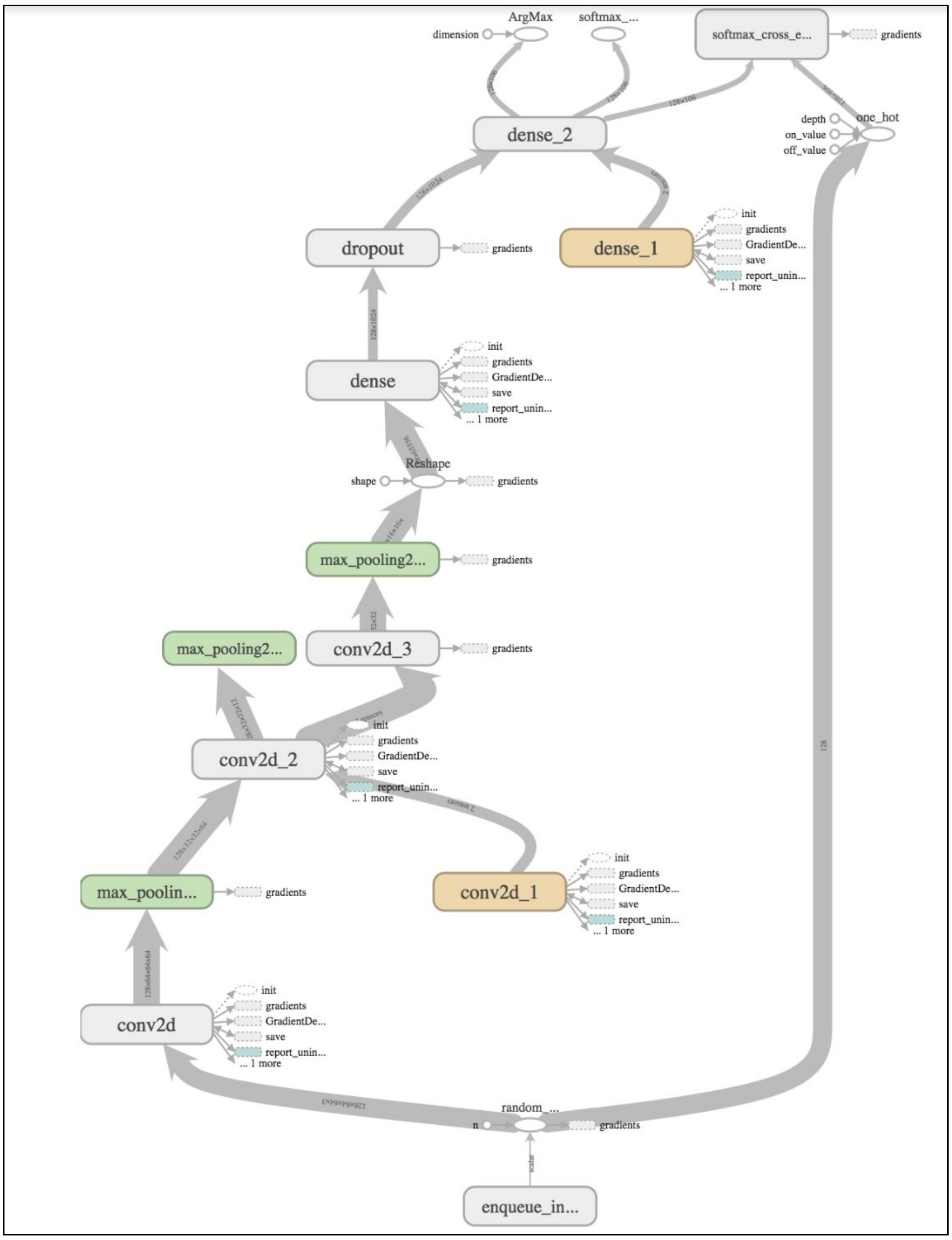
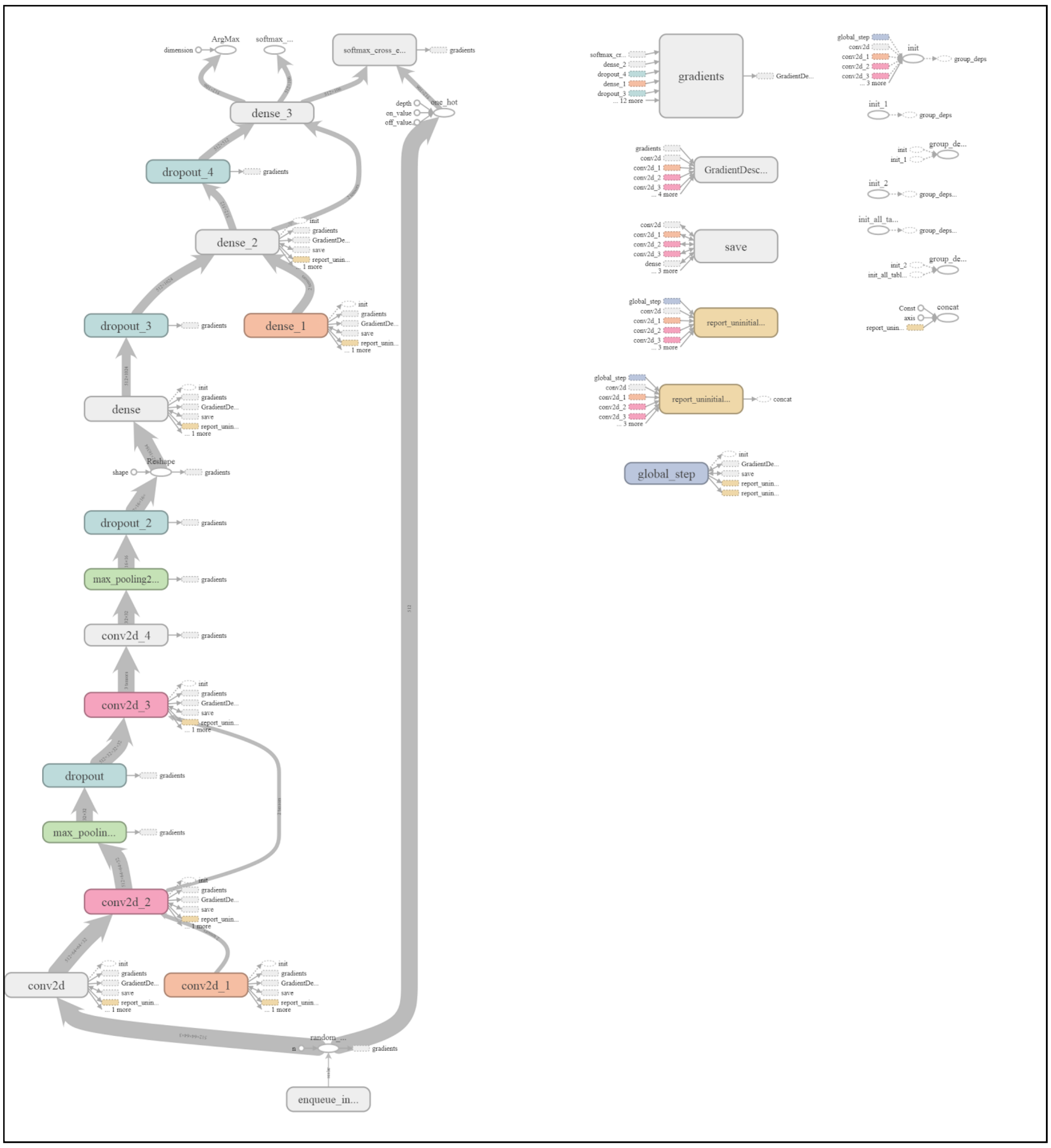

ArgMax
softmax_...
softmax_cross_e...
gradients
dimension
dense_2
one_hot
depth
on_value
off_value
dropout
gradients
dense_1
init
gradients
GradientDe...
save
report_unin...
... 1 more
dense
init
gradients
GradientDe...
save
report_unin...
... 1 more
Reshape
shape
gradients
max_pooling2...
gradients
max_pooling2...
conv2d_3
gradients
conv2d_2
init
gradients
GradientDe...
save
report_unin...
... 1 more
max_poolin...
gradients
conv2d_1
init
gradients
GradientDe...
save
report_unin...
... 1 more
conv2d
init
gradients
GradientDe...
save
report_unin...
... 1 more
random_...
n
gradients
enqueue_in...
128x1024
128x1024
128x106
128

Figure 4


ArgMax
dimension
softmax_...
softmax_cross_e...
gradients
dense_3
depth
on_value
off_value
one_hot
dropout_4
dense_2
dropout_3
dense_1
dense
Reshape
shape
dropout_2
max_pooling2...
conv2d_4
conv2d_3
dropout
max_poolin...
conv2d_2
conv2d
conv2d_1
init
GradientDe...
save
report_unin...
... 1 more
random_...
enqueue_in...
softmax_cr...
dropout_4
dense_1
dropout_3
... 12 more
GradientDesc...
conv2d_1
conv2d_2
conv2d_3
... 4 more
... 3 more
global_step
report_uninitial...
concat
group_deps
init_1
group_de...
init_2
init_all_ta...
init_all_tabl...
Const
axis