Effectiveness of batch normalization, dropout and data augmentation in convolutional neural networks

Mihail Douhaniaris

August 20, 2018

Abstract

Convolutional neural networks have been widely successful deep learning models in several image classification and object recognition tasks. However, as images get larger and more complex, convolutional neural networks require more layers and parameters to capture the nonlinearities present in the data. Deeper networks are more prone to overfitting and a number of methods have been developed to improve the regularization of such networks. In particular, recent developments such as batch normalization, dropout and data augmentation have become de facto building blocks in many deep learning models used to tackle deep network overfitting. In this project, the effect of these regularization methods is studied through a convolutional neural network used to classify images in the CIFAR-10 dataset. The network architecture is incrementally deepened and regularized with the above methods, trained and then validated to observe the effectiveness of the regularization methods. The results show that all three methods improve the generalization performance of the constructed network significantly.

Keywords: batch normalization, dropout, dataset augmentation

1 Introduction

Image classification is a classical deep learning problem where a network is trained to assign a class to a given image from some set of predetermined image classes. A widely popular and succesful approach is to use convolutional neural networks with an input layer, several hidden layers consisting of convolution-pooling layer pairs and a densly connected output layer. As images become larger and more complex, convolutional neural networks typically require additional hidden layers in order to capture the increased number of nonlinearities present in the data, which can often lead to overfitting to the training set and bad regularization to newly observed images.

In this project, we study the regularization effects of batch normalization, dropout and data augmentation using convolutional neural networks trained with the CIFAR-10 dataset, which consists of color images from 10 different classes. We begin with a simple convolutional neural network architecture, originally used to classify simple MNIST digits, and incrementally make the network deeper as well as incorporate the above regularization methods. The implemented convolutional networks are trained and validated with the CIFAR-10 dataset, and the regularization effect of batch normalization, dropout and data augmentation is assessed.

2 Related work

Batch normalization is a recent method originally proposed by Ioffe and Szegedy (2015) used to tackle the problem of internal covariate shift in deep neural network training. The internal covariate shift problem refers to the change in the network layers' input distribution, caused when updating the network parameters during training. This causes difficulties in network training, resulting in slow convergence and bad generalization to new data. Batch normalization is a specific type of a normalization procedure applied to the network layer inputs for each training mini-batch. Batch normalization first normalizes the layer inputs to zero mean and unit variance, and then shifts and scales the normalized inputs so as to maintain the representation power of the network. This results in the inputs having a similar distribution between the training mini-batches, reducing the internal covariate shift and allowing the use of higher learning rates and providing a regularization effect (Ioffe and Szegedy, 2015).

Dropout is another method used to tackle the overfitting challenges associated with deep neural networks, originally proposed by Hinton et al. and inspired by biological systems (2012). In dropout, network units and their connections are layer-wise randomly chosen and left out during the neural network training phase according to a specified dropout probability (Srivastava et al., 2014). After the training phase, all the network units and connections are used by weighting them with the inverse of the dropout probability. In this sense, dropout can be seen as an ensamble method that shares parameters between the bagged models. Dropout is computationally cheap and results in significant reduction in overfitting and improved generalization performance (Srivastava et al., 2014).

3 Method

Overall, four network models are trained and validated and their complete architectures are shown in the appendix A. First, the network architecture used as the basis for the deeper and more complex architectures is a network originally designed by Tapani Raiko and used for MNIST image classification. The architecture is depicted in fig. 1. The architecture consists of:

- 1st convolution layer with ReLU activations
  - 5x5 kernel size
  - Same padding
  - 9 filters
- 1st max pooling layer
  - 2x2 pool size
- 2nd convolution layer with ReLU activations
  - 5x5 kernel size
  - Same padding
  - 16 filters
- 2nd max pooling layer
  - 2x2 pool size
- Dense output layer with softmax activations

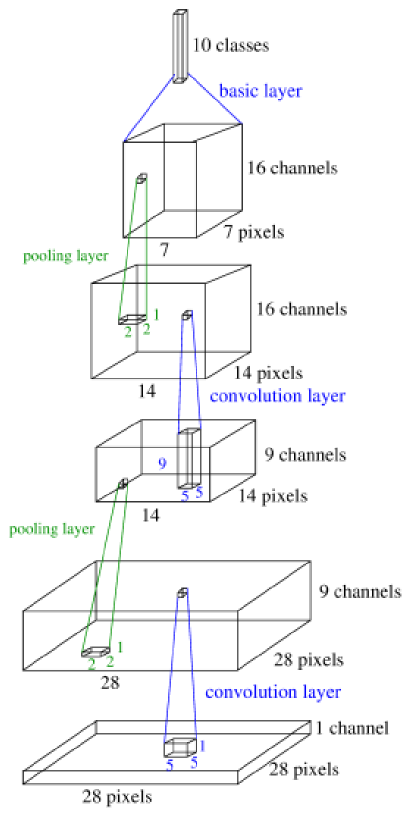

Figure 1: The network architecture of the base model. Image by Tapani Raiko, taken from Kivinen, 2017.

The second architecture is an enhanced version of the base architecture built by adding a third convolution-max pooling layer pair with 16, 32 and

64 filters in the 1st, 2nd and 3rd convolutional layers, respectively. We then add an additional convolutional layer after each convolutional layer, in order to increase the nonlinear representation power of the network. Finally, we add an intermediate ReLU activated dense layer with 128 units before the output layer.

In the third architecture, we add batch normalization layers before the second and third convolution-max pooling layer groups, as well as before the two dense layers in the enhanced network architecture.

In the fourth network architecture, we add dropout layers after each of the convolution-max pooling layer groups as well as after the intermediate dense layer. The dropout probabilities for the four dropout layers are 0.2, 0.3, 0.4 and 0.5, selected such that the deeper the dropout layer, the closer the dropout probability is to 0.5, as advised by Srivastava et al. (2015).

4 Data

The dataset used in this project is the CIFAR-10 dataset, consisting of 60000 32x32 RGB images from 10 classes, with 6000 images per class (Krizhevsky, 2009). The training set contains 50000 images and the test set contains 10000 images and before feeding them to the models, the image channels (RGB) are scaled to values between 0 and 1 by dividing them by 255. The image classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. An example image from the training set is shown in fig. 2

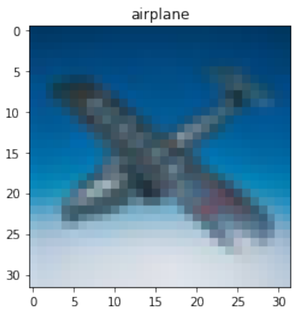

Figure 2: An image from the training set of the CIFAR-10 dataset. The image class is airplane.

5 Experiments

The network architectures described in section 3 were implemented in Python using the deep learning library Keras with the TensorFlow backend (Chollet, 2015; Abadi et al., 2015).

The networks were trained with the RMSProp optimizer with a learning rate of 0.001 and a decay rate of 1e-6. The loss function used in training the networks is the categorical crossentropy loss and the training was carried out with batch size 64 for 128 epochs. To monitor the training convergence, we record the loss function value for each epoch. The performance of the trained networks and the effectiveness of the tested regularization methods are assessed by the training and validation accuracies and the generalization gap.

As the final experiment, we perform data augmentation and retrain and revalidate the fourth network. The data augmentation is performed for each batch by randomly rotating the images up to 20 degrees, shifting horizontally and vertically the images by up to 0.15 times their width and height, respectively, and horizontally flipping the images.

6 Results

Base model

The training loss as well as the training and validation accuracies for the base model are shown in fig. 3. The final training set loss was 1.06 and the training and validation accuracies were 0.6488 and 0.5805, respectively.

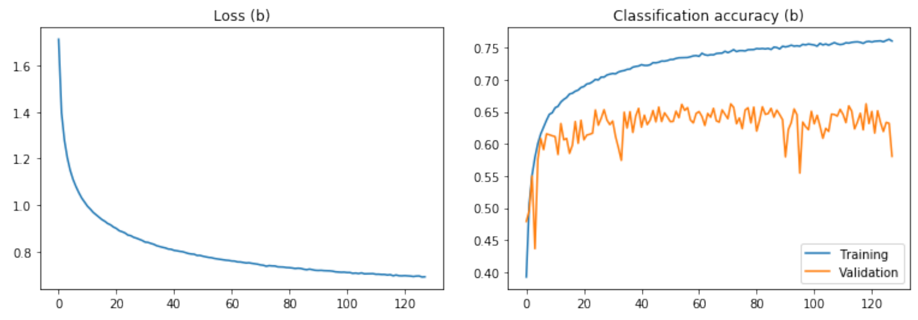

Figure 3: The training loss (left) as well as the training and validation accuracies (right) for the base model.

```
50000/50000 [==============================] - 13s 262us/step
Final train set loss: 1.06
Final train set classification accuracy: 0.6488
------------------------------------------------------------------------
```

```
10000/10000 [==============================] - 3s 261us/step
Final test set loss: 1.4592
Final test set classification accuracy: 0.5805
----------------------------------------------------------------------
```

Enhanced model

The training loss as well as the training and validation accuracies for the enhanced model are shown in fig. 4. The final training set loss was 0.3826 and the training and validation accuracies were 0.9152 and 0.6891, respectively.

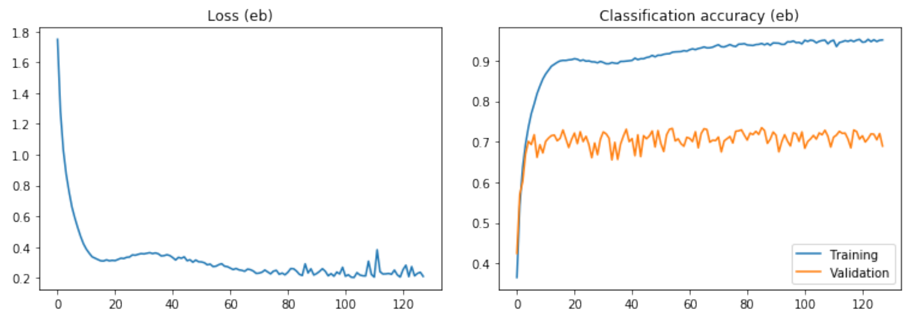

Figure 4: The training loss (left) as well as the training and validation accuracies (right) for the enhanced model.

```
50000/50000 [==============================] - 52s 1ms/step
Final train set loss: 0.3826
Final train set classification accuracy: 0.9152
----------------------------------------------------------------------
10000/10000 [==============================] - 10s 1ms/step
Final test set loss: 2.6152
Final test set classification accuracy: 0.6891
----------------------------------------------------------------------
```

Enhanced model with batch normalization

The training loss as well as the training and validation accuracies for the enhanced model with batch normalization layers are shown in fig. 5. The final training set loss was 0.0029 and the training and validation accuracies were 0.9991 and 0.775, respectively.

```
50000/50000 [==============================] - 59s 1ms/step
Final train set loss: 0.0029
Final train set classification accuracy: 0.9991
----------------------------------------------------------------------
```

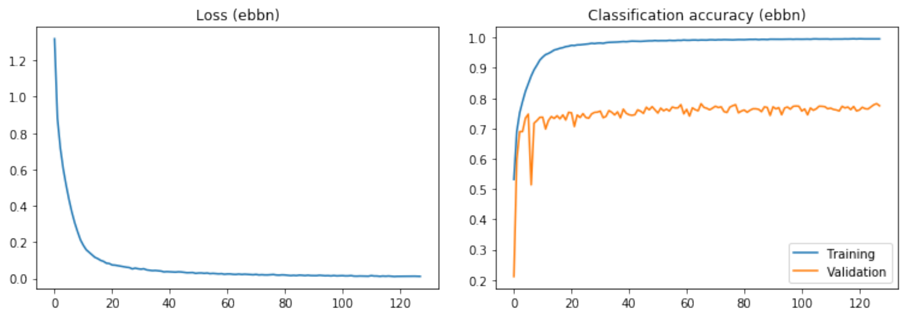

Figure 5: The training loss (left) as well as the training and validation accuracies (right) for the enhanced model with batch normalization.

```
10000/10000 [==============================] - 12s 1ms/step
Final test set loss: 1.791
Final test set classification accuracy: 0.775
----------------------------------------------------------------------
```

Enhanced based model with batch normalization and dropout

The training loss as well as the training and validation accuracies for the enhanced model with batch normalization and dropout layers are shown in fig. 6. The final training set loss was 0.2201 and the training and validation accuracies were 0.9391 and 0.8242, respectively.

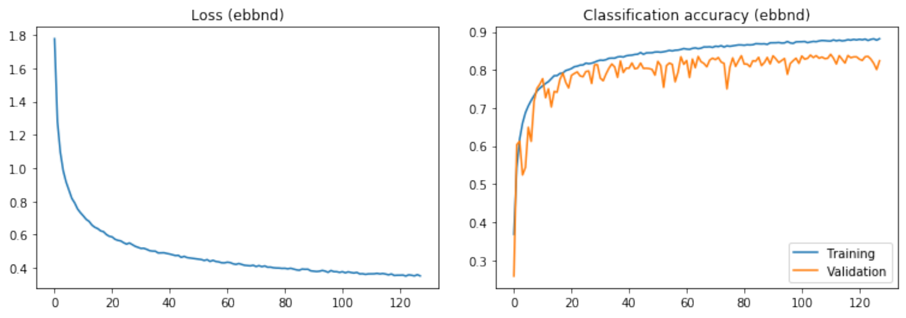

Figure 6: The training loss (left) as well as the training and validation accuracies (right) for the enhanced model with batch normalization and dropout.

```
50000/50000 [==============================] - 59s 1ms/step
Final train set loss: 0.2201
Final train set classification accuracy: 0.9391
----------------------------------------------------------------------
```

```
10000/10000 [==============================] - 11s 1ms/step
Final test set loss: 0.5709
Final test set classification accuracy: 0.8242
----------------------------------------------------------------------
```

Enhanced based model with batch normalization and dropout trained with augmented data

The training loss as well as the training and validation accuracies for the enhanced model with batch normalization and dropout layers are shown in fig. 7. The final training set loss was 0.4052 and the training and validation accuracies were 0.8646 and 0.8376, respectively.

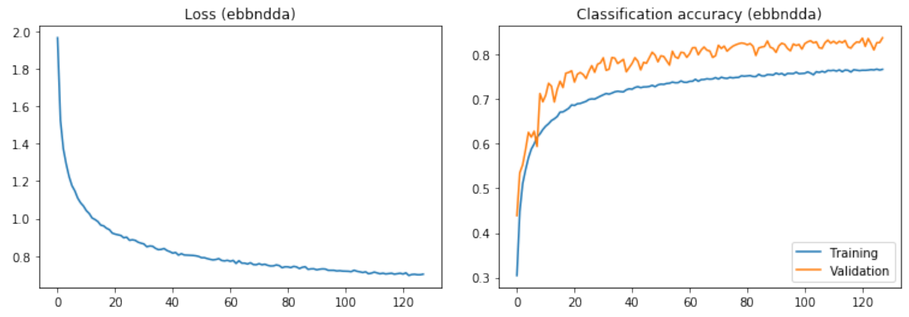

Figure 7: The training loss (left) as well as the training and validation accuracies (right) for the enhanced model with batch normalization and dropout trained with augmented data.

```
50000/50000 [==============================] - 57s 1ms/step
Final train set loss: 0.4052
Final train set classification accuracy: 0.8646
----------------------------------------------------------------------
10000/10000 [==============================] - 10s 1ms/step
Final test set loss: 0.4933
Final test set classification accuracy: 0.8376
----------------------------------------------------------------------
```

7 Discussion

The base model performed poorly, which was expected as the network is originally used to classify much simpler MNIST digits. The network architecture is too simple and its number of parameters is too low for the network to capture the nonlinearities present in the CIFAR-10 data. This is indicated by the high training loss of 1.06 as well as the low training accuracy of 0.6488. The network also generalizes badly, as the generalization gap is quite high with the validation error at 0.5805.

The enhanced model is clear improvement to the base model. The training loss is much lower now at 0.3826, although some training instability can be observed from the training loss graph in fig. 4. The deeper network architecture and the higher number of parameters result in the network being able to capture the nonlinearities of the data much better. The enhanced network achieved a higher training accuracy, 0.9152, although the validation accuracy is only 0.6891, meaning that there is severe overfitting in the model.

The enhanced model with the batch normalization layers is again an improvement on the previous model. The training is a lot faster than in the previous case, as explained by Ioffe and Szegedy (2015). The training is also much more stable and the network archieves a very low training loss of 0.0029 and a high training accuracy. Although there is also an improvement in the validation accuracy at 0.775, the generalization gap is still very high, meaning that there is again a lot of overfitting present in the model. In this case, the batch normalization achieves a slight performance gain and a lot faster training, however, it is not very effective as a regularizer.

The enhanced model with the batch normalization and dropout layers is once more an improvement to the previous model. The training is again very stable and it has somewhat slowed down compared to the previous model without the dropout layers. The slower training is expected, as each mini-batch trains a different network each time, a "thinned" version of the full network. The added dropout layers are able to eliminate the overfitting issues present in the previous model: the training loss is now higher at 0.2201, the training accuracy is lower at 0.9391 and the validation accuracy is higher at 0.8242, leading to a much improved generalization gap.

Finally, the enhanced model with batch normalization and dropout trained with the augmented dataset achieves the best validation performance and best generalization. The network overfits much less now with a training loss of 0.4052 and a training accuracy of 0.8646. The validation accuracy is higher than in the previous model at 0.8376. From the accuracy graph shown in fig. 7, we observe that the validation accuracy is higher than the training accuracy during training. This is because of the data augmentation, as the dataset is randomly permuted at each epoch, effectively giving the model new data every time.

8 Conclusions

The final model still falls behind the current state-of-the-art model in CIFAR-10 validation accuracy by Graham (2015), which uses a more advanced pooling scheme, fractional max-pooling. Although the performance of the final model could be improved with more careful parameter selection and additional experimentation, the main objective of this project was to explore to explore the effectiveness of the widely popular regularization methods of batch normalization, dropout and data augmentation. We have showed that these methods are very effective regularizers and can significantly boost the classification performance of the relatively simple networks used in this project.

References

[1] Chollet F. and others. *Keras.* GitHub, 2015

[2] Graham B. *Fractional Max-Pooling.* arXiv preprint, arXiv, 2015

[3] Hinton G. E., Srivastava N., Krizhevsky A., Sutskever I. and Salakhutdinov R. R. *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv preprint, arXiv, 2012

[4] Ioffe S. and Szegedy C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* ICML'15 Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 448 – 456, 2015

[5] Kivinen J. *CS-E4890 Deep Learning - Lecture 4: Convolutional Neural Networks I (Lecture slides).* Department of Computer Science, Aalto University, 2017

[6] Krizhevsky A. *Learning Multiple Layers of Features from Tiny Images.*

[7] Srivastava N., Hinton, G., Krizhevsky A., Sutskever I. and Salakhutdinov R. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, vol. 15, pp. 1929 – 1958, 2014

[8] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A., Dean J., Devin M., et al. *Tensorflow: Large-scale machine learning on heterogeneous systems.*, 2015

Appendix A: Network architectures

Base model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 9)         684
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 9)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 16)        3616
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 14,550
Trainable params: 14,550
Non-trainable params: 0
_________________________________________________________________
```

Enhanced model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 16)        1216
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 16)        6416
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 32)        12832
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 32)        25632
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 64)          102464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 332,314
Trainable params: 332,314
Non-trainable params: 0
_________________________________________________________________
```

Enhanced model with batch normalization

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 32, 32, 16)        1216
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 32, 32, 16)        6416
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 16, 16, 32)        12832
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 32)        25632
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 64)          51264
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 8, 8, 64)          102464
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dense_4 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 337,114
Trainable params: 334,714
Non-trainable params: 2,400
_________________________________________________________________
```

Enhanced model with batch normalization and dropout

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 32, 32, 16)        1216
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 32, 32, 16)        6416
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 16, 16, 32)        12832
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 16, 16, 32)        25632
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 64)          51264
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 8, 8, 64)          102464
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
batch_normalization_7 (Batch (None, 1024)              4096
_________________________________________________________________
dense_6 (Dense)              (None, 128)               131200
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
```

```
=================================================================
Total params: 337,114
Trainable params: 334,714
Non-trainable params: 2,400
_________________________________________________________________
```