CSCI 4155 GROUP 10

COMPARE AND ANALYZING CNN BASED MODELS' PERFORMANCE ON ANIMAL IMAGE CLASSIFICATION

Junqiao Qu
Yilong Su
Archer Zhou
Department of Computer Science
Dalhousie University

Apr.7 2022

We started with four members, but one of us dropped the course before the project, so only three of us participated in the project.

Contents

1 Abstract

With the development of the neural network model, there are a lot of variants of the CNN model, the key difference is most of them has enhanced structure compare to the simple CNN model. In this project, we are going to compare and analysis the certain structure's impact on the models' performance with image classification task. We will build AlexNet and VGG in our project and test their performance with our data to compare the performance. Our results shows "relu" activation function, dropout layer, deeper structure and use smaller kernel size to replace single larger kernel layer can achieve better performance than the baseline CNN. We learned these from AlexNet and VGG and validated the theories.

2 Introduction

CNN-based model has played a very important role in deep learning. In this project we are focusing on the CNN-based models' performance on image classification. We will implement AlexNet and VGG model in our experiment, we will test and analyze why they can improve the overall performance. We will also refine a simple CNN model based on AlexNet and VGG respectively to validate the impact of certain structure. We focus on four points in our project, the "relu" activation function, the dropout layer, the deeper structure and smaller kernel size. According to our experiment result, we noticed that AlexNet, VGG and all the CNN-based models performs better than our baseline CNN model. The refined CNN-based model shows better performance than our baseline by using "relu" activation function, adding "dropout" layer and adding more layers, using smaller kernel size. We conclude that all these four points we learned from AlexNet and VGG can improve the performance of CNN-based model.

3 Background

The fast growth of human population and the endless pursuit of economic development are making over-exploitation of natural resources, causing rapid, novel and substantial changes to Earth's ecosystems. The rapidly evolving computer vision can provide a powerful tool for biologist and animal photographers to classify different animals efficiently. In this project, we will explore the architecture different CNN based network, and use them to classify the animals in the data set. We will analyze the performance and try to make an explanation of relationship between performance and architecture.

4 Related work

4.1 CNN

Convolutional neural networks(CNN) are a class of feed-forward neural networks that include convolutional computation and have a deep structure and are one of the representative algorithms of deep learning. Convolutional neural networks have representational learning capability and can classify input information in a translation-invariant manner according to their hierarchical structure.

Research on convolutional neural networks began in the 1980s and 1990s, and LeNet-5 was the first convolutional neural network to appear [2]. After the twenty-first century, with the proposal of deep learning theory and the improvement of numerical computing equipment, convolutional neural networks have been developed rapidly and applied to computer vision, natural language processing, and other fields.

4.2 AlexNet

AlexNet was designed by Hinton, the winner of the 2012 ImageNet competition, and his student Alex Krizhevsky. [1] AlexNet contains several relatively new technical points.

It successfully uses ReLU as the activation function of CNN. It verifies its effectiveness over Sigmoid in deeper networks, successfully solving the gradient dispersion problem of Sigmoid when the network is deeper. Although the ReLU activation function has been proposed for a long time, it was not developed until the advent of AlexNet.

Dropout is used to randomly ignore some neurons during training to avoid over-fitting the model. It is mainly the last few fully connected layers in AlexNet that use Dropout.

It uses overlapping maximum pooling in CNNs. Previously, average pooling was commonly used in CNNs, but AlexNet uses maximum pooling to avoid the blurring effect of average pooling. Moreover, it is proposed in AlexNet to let the step length be smaller than the size of the pooling kernel so that there will be overlap and coverage between the outputs of pooling layers, which improves the feature richness.

4.3 VGG

The group of Visual Geometry Group from Oxford proposed VGG at ILSVRC 2014. [3] The main work was to show that increasing the depth of the network can affect the final performance of the network to some extent. One of the improvements of VGG16 over AlexNet

is using several consecutive 3x3 convolutional kernels instead of the larger convolutional kernels in AlexNet. For a given perceptual field, the use of stacked small convolutional kernels is preferable to large convolutional kernels. The multi-layer nonlinear layers can increase the network depth to ensure learning more complex patterns at a more negligible cost (fewer parameters).

In brief, in VGG, three 3x3 convolutional kernels are used, and the primary purpose of this is to improve the depth of the network while guaranteeing to have the same perceptual field, which improves the neural network to some extent.

The advantage of VGGNet is that the structure of VGGNet is straightforward, using the exact size of convolutional kernel size (3x3) and maximum pooling size (2x2) for the whole network. And the combination of convolutional layers using several small filters (3x3) is better than one large filter (5x5 or 7x7) convolutional layer.

The authors also verify that the performance can be improved by continuously deepening the network structure.

5 Data set and Features

We have two data sets in our whole project, the first one is the data set from Kaggle and the other one is crawled by ourselves.

The Animal 10 is a data set contains 28000 medium quality animal images belonging to 10 categories: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, elephant. The images are all from google and manually checked. To simulate a real condition, some noise images were added.

The BING animal is a data set created by us containing images crawled from BING. The data set contains 10 animal species, butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, squirrel. Each species contains 229 images. 1-150 are original images after compression. And 151-224 are images with salt and pepper noise (with compression). And 225-229 are random images, which do not belong to this species. We use the keyword for search engine, for ground truth. That is, all images in a folder are considered to belong to this species.

We use this crawled data set as our evaluation data and the Kaggle one as our training set.

The crawled data set has the same format as Animal10 (Data set from Kaggle. And it has much better image quality and relevance than Animal10. If this turns out to be too easy, we can add noise manually. Adding noise is for the purpose that the network is less able to memorize training samples because they are changing all the time, resulting in smaller

network weights and a more robust network that has lower generalization error. According to this, we add the Salt and Pepper noise to some images in the data set. And randomly put some irrelevant images to each category. It is synthetic label noise.

We could also add more lower-quality images to the data set, which is closer to the real situation. It can make our network more robust and improve the generalization ability.

6 Methodology

6.1 Data Pre-processing

In addition to basic operations such as color and size adjustment of the data, this section focuses on data augmentation. As indicated earlier, we initially used a small data set obtained through crawlers for model building. In order to avoid the impact caused by the insufficient amount of training data, it is a good choice to use data augmentation. A technique to generate new training examples from existing training examples. Data augmentation is a low-cost and effective way to address data-constrained environments, thereby addressing the risk of over-fitting due to the use of small amounts of data. Not only that, but data augmentation also enables models to approximate real-world situations where data is encountered, such as images of the same object that can vary depending on different angles and different lighting conditions. As such, training with the support of such data set will make the model more robust. In this step, only one type of data augmentation are used, adding noise.

6.1.1 Noise Injection

Noise comes in many forms, but the effects and impacts caused are similar, so we will consider salt-and-pepper noise. The images with this type of noise will have a few dark pixels in bright areas and a few light pixels in dark areas. Pepper noise is also known as impulse noise. In reality, it can be caused by a variety of reasons, such as bad dots, analog-to-digital conversion errors, bit transfer errors, etc. Older movies often have black and white pixel specks in the picture where they shouldn't be, which is noise. Noise is a defect that is especially frustrating for machine learning models. It is one of the root causes of so-called adversarial attacks, in which small pixel changes that are imperceptible to humans can dramatically alter the ability of neural networks to make accurate predictions.

We therefore "strategically" add some noise to seek to increase the variability of certain images. This is used for training to mitigate adversarial attacks and to avoid over-fitting, but

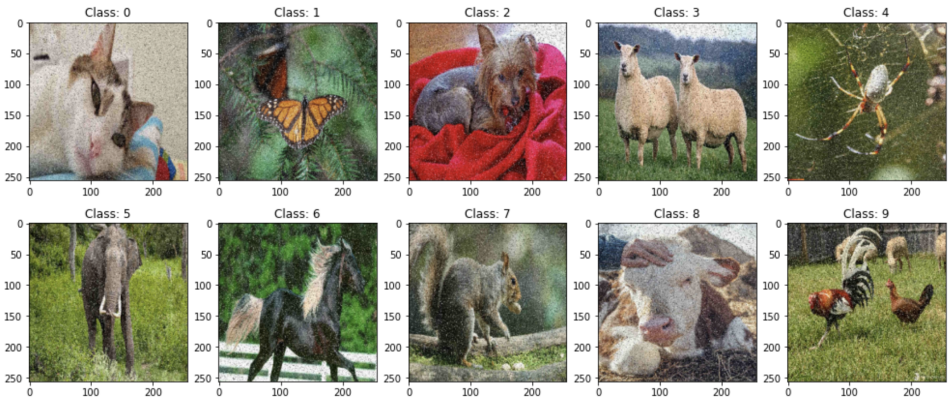

Figure 1: Examples with salt-and-pepper noise

we also do not seek to introduce noise into our validation and test sets. As shown in Figure 1, these are some images after adding noise.

6.2 Models Building

We choose to conduct our experiment section on google colaboratory with python. We will use keras library to build our CNN-based models. We will also build refined CNN model based on our baseline to better compare and analysis the performance between different CNN-based models.

For the baseline CNN model, we build a simple CNN model with four convolutional layer and "tanh" as our activation function. This will be defined as the baseline of our experiment, we will compare its performance with AlexNet, VGG and our refined CNN-based models.

For the AlexNet, we will build AlexNet model according to its paper. [1]We will compare its performance with our baseline and analyze the result. The point we are focusing on is whether "relu" activation function and dropout layer will improve the performance.

For the VGG, we will also build the model according to its paper [3] Since VGG has deeper structure and smaller kernel size, we will compare and analyzing their impact on the performance.

The detail structure and parameter please refer to the appendix section.10

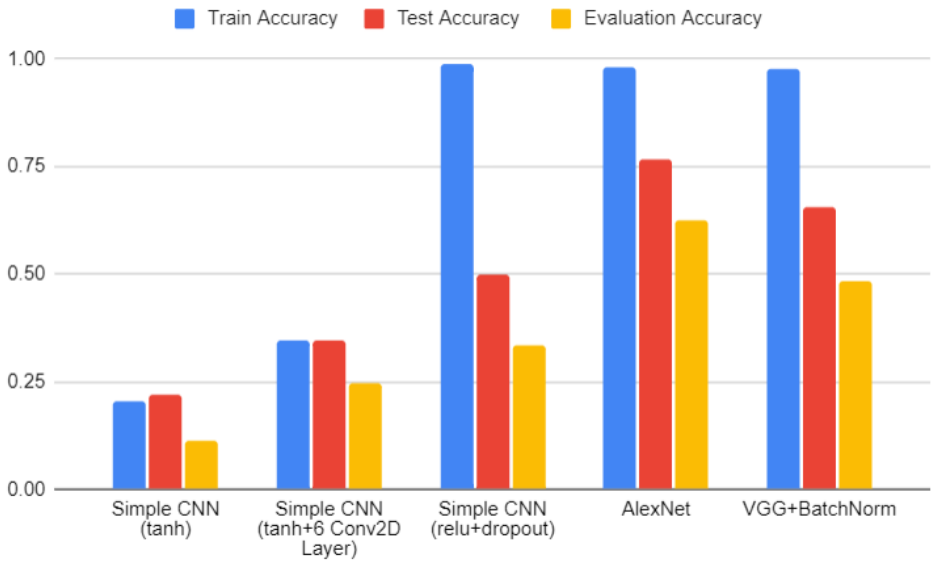

Figure 2: Evaluation result

7 Result and Discussion

7.1 Experiment result

According to the figure 2, we can noticed that AlexNet, VGG and the refined CNN-based models all have better performance than our baseline CNN model. And AlexNet has the best performance in all the models. Our refined CNN model has a decent improvement compare to the baseline, but not as good as the AlexNet and VGG model.

According to the figure 3, we found that batch normalization is very important to the VGG model, the performance drops a lot without batch normalization.

7.2 Discussion

According to the two comparison, baseline and AlexNet, baseline and refined CNN based on AlexNet, we can see the model's performance improves. We address this to the "relu" activation function and dropout layer.

ReLU allows neuron to express a strong opinion, it is less sensitive to random initialization and the gradient doesn't saturate.

According to the two comparison, baseline and VGG, baseline and refined CNN based on VGG, we can see the model's performance improves. We address this to the deeper structure and replace large kernel size layer with smaller kernel size layer.

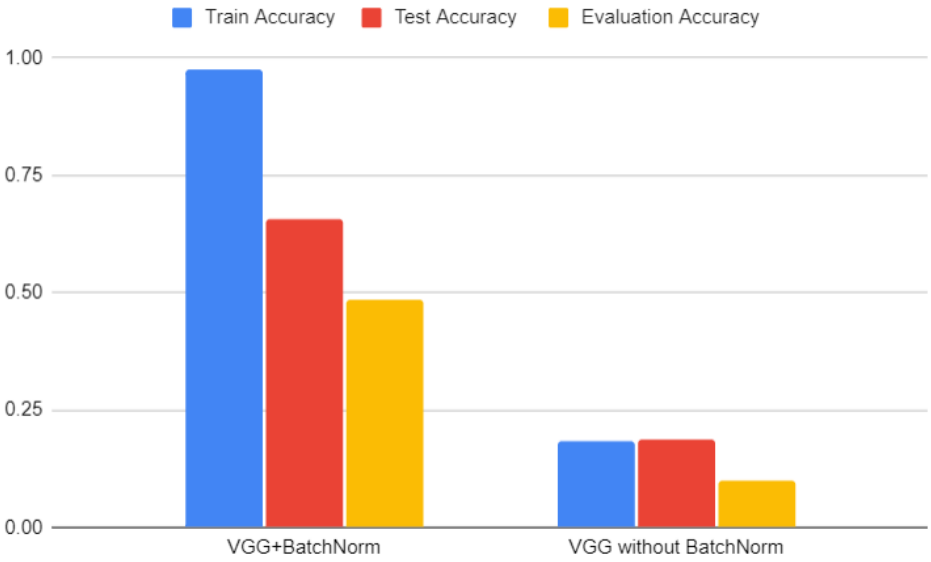

Figure 3: The impact of batch normalization

The deeper the structure, the better the model can fit the features. The deeper the structure, also the easier for each layer to do the job. For these reasons, a deeper structure can improve the performance. According to the research in the VGG paper [3], a smaller kernel size can improve the performance, we also validate this in our experiment. Most of the useful features in an image are usually local and it makes sense to take few local pixels at a time to apply convolutions.

Another important thing we have found during our experiment is the importance of batch normalization to VGG model. When we are building and testing the VGG model, we tried to build it without the batch normalization and the result turn to be terrible. The batch normalization plays an important role in the VGG model, since this is not our main goal of the project, we will leave it to future work.

8 Conclusion

According to our experiment result, we can get following conclusion. Using "relu" activation function and dropout layer would improve the model's performance. Using a model with deeper structure and replace one large kernel size layer with multiple smaller kernel size layers could improve the model's performance.

9 Future Work

We will further compare the models' performance with pure noisy data and pure clean data. In our experiment, we combine noisy and clean data to get a more generalize result. We can experiment with the baseline model and our refined model to see how performance varies between pure noisy data, pure clean data and combined data.

We can also experiment with some other CNN based data to see the structure's impact on the model's performance, we can use model such as ResNet.

We noticed that batch normalization is very important to VGG model, we can further test the impact of batch normalization on different models.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

10 Appendix

Data set

The detail of crawled data set information please refer to the P1 file in the teams chat.
The detail of Animal10 data set please refer to the following link.
Link: `https://www.kaggle.com/alessiocorrado99/animals10`

Model Structure

Baseline-CNN

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_17 (Conv2D)          (None, 222, 222, 32)      896

 conv2d_18 (Conv2D)          (None, 220, 220, 32)      9248

 batch_normalization_14 (Bat  (None, 220, 220, 32)     128
 chNormalization)

 conv2d_19 (Conv2D)          (None, 110, 110, 32)      25632

 conv2d_20 (Conv2D)          (None, 55, 55, 64)        51264

 flatten_2 (Flatten)         (None, 193600)            0

 dense_5 (Dense)             (None, 128)               24780928

 dense_6 (Dense)             (None, 10)                1290

=================================================================
Total params: 24,869,386
Trainable params: 24,869,322
```

```
Non-trainable params: 64
_________________________________________________________________
```

AlexNet

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_9 (Conv2D)           (None, 54, 54, 96)        34944

 batch_normalization_6 (Batc  (None, 54, 54, 96)       384
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_10 (Conv2D)          (None, 26, 26, 256)       614656

 batch_normalization_7 (Batc  (None, 26, 26, 256)      1024
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_11 (Conv2D)          (None, 12, 12, 384)       885120

 batch_normalization_8 (Batc  (None, 12, 12, 384)      1536
 hNormalization)

 conv2d_12 (Conv2D)          (None, 12, 12, 384)       1327488

 batch_normalization_9 (Batc  (None, 12, 12, 384)      1536
 hNormalization)
```

```
 conv2d_13 (Conv2D)          (None, 12, 12, 256)       884992

 batch_normalization_10 (Bat  (None, 12, 12, 256)      1024
 chNormalization)

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten_2 (Flatten)         (None, 6400)              0

 dense_5 (Dense)             (None, 4096)              26218496

 dropout_4 (Dropout)         (None, 4096)              0

 dense_6 (Dense)             (None, 4096)              16781312

 dropout_5 (Dropout)         (None, 4096)              0

 dense_7 (Dense)             (None, 10)                40970

=================================================================
Total params: 46,793,482
Trainable params: 46,790,730
Non-trainable params: 2,752
_________________________________________________________________
```

Refined-CNN based on AlexNet

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 222, 222, 32)      896

 conv2d_1 (Conv2D)           (None, 220, 220, 32)      9248
```

```
 batch_normalization (BatchN  (None, 220, 220, 32)     128
 ormalization)

 conv2d_2 (Conv2D)           (None, 110, 110, 32)      25632

 dropout (Dropout)           (None, 110, 110, 32)      0

 conv2d_3 (Conv2D)           (None, 55, 55, 64)        51264

 flatten (Flatten)           (None, 193600)            0

 dropout_1 (Dropout)         (None, 193600)            0

 dense (Dense)               (None, 128)               24780928

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 24,869,386
Trainable params: 24,869,322
Non-trainable params: 64
_________________________________________________________________
```

VGG

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 batch_normalization (BatchN  (None, 224, 224, 64)     256
 ormalization)

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928
```

```
batch_normalization_1 (Batc  (None, 224, 224, 64)     256
hNormalization)

max_pooling2d (MaxPooling2D  (None, 112, 112, 64)     0
)

conv2d_2 (Conv2D)            (None, 112, 112, 128)    73856

batch_normalization_2 (Batc  (None, 112, 112, 128)    512
hNormalization)

conv2d_3 (Conv2D)            (None, 112, 112, 128)    147584

batch_normalization_3 (Batc  (None, 112, 112, 128)    512
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 56, 56, 128)      0
2D)

conv2d_4 (Conv2D)            (None, 56, 56, 256)      295168

batch_normalization_4 (Batc  (None, 56, 56, 256)      1024
hNormalization)

conv2d_5 (Conv2D)            (None, 56, 56, 256)      590080

batch_normalization_5 (Batc  (None, 56, 56, 256)      1024
hNormalization)

conv2d_6 (Conv2D)            (None, 56, 56, 256)      590080

batch_normalization_6 (Batc  (None, 56, 56, 256)      1024
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 28, 28, 256)      0
```

```
2D)

conv2d_7 (Conv2D)           (None, 28, 28, 512)       1180160

batch_normalization_7 (Batc  (None, 28, 28, 512)      2048
hNormalization)

conv2d_8 (Conv2D)           (None, 28, 28, 512)       2359808

batch_normalization_8 (Batc  (None, 28, 28, 512)      2048
hNormalization)

conv2d_9 (Conv2D)           (None, 28, 28, 512)       2359808

batch_normalization_9 (Batc  (None, 28, 28, 512)      2048
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 14, 14, 512)      0
2D)

conv2d_10 (Conv2D)          (None, 14, 14, 512)       2359808

batch_normalization_10 (Bat  (None, 14, 14, 512)      2048
chNormalization)

conv2d_11 (Conv2D)          (None, 14, 14, 512)       2359808

batch_normalization_11 (Bat  (None, 14, 14, 512)      2048
chNormalization)

conv2d_12 (Conv2D)          (None, 14, 14, 512)       2359808

batch_normalization_12 (Bat  (None, 14, 14, 512)      2048
chNormalization)
```

```
 max_pooling2d_4 (MaxPooling  (None, 7, 7, 512)        0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 4096)              102764544

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 10)                40970

=================================================================
Total params: 134,318,410
Trainable params: 134,309,962
Non-trainable params: 8,448
_________________________________________________________________
```

Refined-CNN based on VGG

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_6 (Conv2D)           (None, 222, 222, 32)      896

 conv2d_7 (Conv2D)           (None, 220, 220, 32)      9248

 batch_normalization_1 (Batc  (None, 220, 220, 32)     128
 hNormalization)

 conv2d_8 (Conv2D)           (None, 110, 110, 32)      9248

 batch_normalization_2 (Batc  (None, 110, 110, 32)     128
 hNormalization)
```

```
 conv2d_9 (Conv2D)           (None, 55, 55, 32)        9248

 conv2d_10 (Conv2D)          (None, 28, 28, 32)        9248

 batch_normalization_3 (Batc  (None, 28, 28, 32)       128
 hNormalization)

 conv2d_11 (Conv2D)          (None, 14, 14, 64)        51264

 flatten_1 (Flatten)         (None, 12544)             0

 dense_2 (Dense)             (None, 128)               1605760

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,696,586
Trainable params: 1,696,394
Non-trainable params: 192
_________________________________________________________________
```

VGG without batch normalization

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 112, 112, 64)     0
 )

 conv2d_2 (Conv2D)           (None, 112, 112, 128)     73856

 conv2d_3 (Conv2D)           (None, 112, 112, 128)     147584
```

```
max_pooling2d_1 (MaxPooling  (None, 56, 56, 128)      0
2D)

conv2d_4 (Conv2D)           (None, 56, 56, 256)       295168

conv2d_5 (Conv2D)           (None, 56, 56, 256)       590080

conv2d_6 (Conv2D)           (None, 56, 56, 256)       590080

max_pooling2d_2 (MaxPooling  (None, 28, 28, 256)      0
2D)

conv2d_7 (Conv2D)           (None, 28, 28, 512)       1180160

conv2d_8 (Conv2D)           (None, 28, 28, 512)       2359808

conv2d_9 (Conv2D)           (None, 28, 28, 512)       2359808

max_pooling2d_3 (MaxPooling  (None, 14, 14, 512)      0
2D)

conv2d_10 (Conv2D)          (None, 14, 14, 512)       2359808

conv2d_11 (Conv2D)          (None, 14, 14, 512)       2359808

conv2d_12 (Conv2D)          (None, 14, 14, 512)       2359808

max_pooling2d_4 (MaxPooling  (None, 7, 7, 512)        0
2D)

flatten (Flatten)           (None, 25088)             0

dense (Dense)               (None, 4096)              102764544
```

```
 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 10)                40970

=================================================================
Total params: 134,301,514
Trainable params: 134,301,514
Non-trainable params: 0
_________________________________________________________________
```

Model parameters

Baseline CNN and refined CNN

```
epochs=15,batch_size=64,verbose=1
```

AlexNet

```
epochs=100,batch_size=64,verbose=1
```

VGG

```
epochs=30,batch_size=64,verbose=1
```