Visualization of encoder-decoder structure of Deep Convolutional Neural Network

Xiaohan Zhang
xzhan272@ucsc.edu
University of California, Santa Cruz
Department of Computer Science and Engineering

ABSTRACT

Deep Convolutional Neural Networks (DCNNs) achieve impressive results on image Classification, object detection and object tracking tasks. However, researchers still do not clearly understand what is the network learned during training phase. Recent works [8] and [7] shows the state-of-the-art result on visualization of DCNNs. However, DCNNs are still a "black-box" for most of beginners and other field researchers. In order to improve the interpretability and make the network easy to understand, I propose a manner to easily visualize the output of each layer. As part of the work, I also plan to conduct ablation study on how neural net work works and what it may learned during training.

Index Terms: Visualization—Computer Vision—Deep Learning—
;

1 MOTIVATION

Since the first time LeCun [4] Introduced the Convolutional Neural Network (CNN) to image classification problem, CNN shows its great performance in computer vision tasks such as image classification, object detection and object tracking. With the increase of computational power and complexity of tasks, Deep Convolutional Neural Networks (DCNNs) [3] have been proposed which achieved impressive result as error rate of 16.4% in ImageNet classification challenge. Nowadays, researchers are not satisfied regression problems. Generative models such as [1] shows that DCNNs has the ability to generate photo-realistic pictures.

Regardless of these significant progress, researchers still treat DCNNs as a "black-box" like architecture. Current researches merely focus on how and why the DCNNs made mistakes on certain problems. From a researcher's point of view, it is needed to show how this problem effect the DCNNs. Recent research [8] have already shows the feature vector reconstructed by deconvolution network. However, the most important concept in DCNNs are weight sharing and channel fusion which a single kernel can take the whole output from last layer as input. So a single kernel output can hardly decide the final output of the network. However, if a certain kernel made a "mistake", this "mistake" will spread to the next layer due to the nature of DCNNs. It is also hard for researchers to identify what this "mistake" is. To address this issue, I plan to design an interface which can show the output of difference layers simultaneously. The combination output of different layers on the original image which will help users have a more direct understanding on how one or more kernels effect the result. This method not only allow users to see the value of a single kernel but take the whole channel into account.

2 RELATED WORKS

Normally in shallow layer (in other words, the first one or two layers) of DCNNs, the kernel tends to learn gaussian-like or gabor-like filters which is interpretable and understandable by humans. However, with the layer go deeper, the outputs of kernels are compressed in width and length which make them uninterpretable. As I mention in Motivation, Zeiler and Fergus [8] shows a state-of-the-art result on visualization of DCNNs. Based on original AlexNet, they undo the operations of the previous layers of this layer which need to be visualized. That is they reverse the three main operations, convolution, MaxPooling and ReLU. To reverse the convolutional layers, they simply take the transpose of the convolutional kernel and use this convolutional kernel to convolve on the dialated original image. This operation is called deconvolution. For MaxPooling layers, they first record the location of the pooled value. In the reverse progress the pooled value is been placed in the recorded place and the remaining places are filled by zeros. The last operation is ReLU defined as $f(x)$ as below:

$$f(x) = max(0,x) \tag{1}$$

It is obvious that $f(x)$ keeps the positive value during the feed forward progress (inference stage). The reverse of $f(x)$ is also keeping the positive value and ignore the negative value. Zeiler and Fergus use these three reversed operations can visualize the kernel output in an original image size. They show different layers' outputs with different training epoch and find that with the deeper layers the kernels focus on the global features but not the local feature. The network potentially has the ability to separate the background and foreground to focus on an abstract features in certain space. They also conduct experiments on analyzing the feature output by certain kernel when abnormal input (such as cropping part of image, rotating, rescaling) is fed to the network. From their results, we can see with these abnormal images make some changes to the output. But we still do not have an direct understanding on how and why these features effect the output.

Another outstanding paper about the visualization of DCNNs presented by by Zhou [11] shows DCNNs has the ability to learn the position information with related to the label even the label do not contain the position information. And also, this ability can be used in transferring learning and other field. This paper proposed Class Activation Mapping (CAM) to visualize the focusing map of the entire DCNNs. Assume $f_k(x,y)$ is the value on kth feature map with position (x,y). CAM is calculated by Global Average Pooling (GAP) which define as below:

$$F_k = \sum_{x,y} f_k(x,y) \tag{2}$$

Now a single value F_k can represent a feature map k. To obtain the classification probability, We can use SoftMax output S_{class} to calculate P_{class} where S_{class} and P_{class} are defined below:

$$S_{class} = \sum_k w_k^{class} F_k \tag{3}$$

$$P_{class} = \frac{exp(S_{class})}{\sum_{class} exp(S_{class})} \tag{4}$$

Finally, combine (3) and (2) S_{class} can be written as:

$$S_{class} = \sum_{x,y}\sum_{k} w_k^{class} f_k(x,y) \tag{5}$$

Because, the S_{class} has a one-by-one relationship to P_{class}, they define the CAM $M_{class}(x,y)$ by taking the summation of all feature maps which is shown following:

$$M_{class}(x,y) = \sum_{k} w_k^{class} f_k(x,y) \tag{6}$$

To visualize $M_{class}(x,y)$, the author resize $M_{class}(x,y)$ to the original image size and overlay on it. The author conduct several experiment on different DCNNs to show their CAMs and prove that DCNN has outstanding ability in localization object. However, they did not conduct experiment on misclassification test. I think it is needed to show some failure cases of DCNNs which will help researchers to have a better understanding on DCNNs working.

3 Proposed method

Inspired by [11] and [7], I propose a new manner to visualize the encoder-decoder struct of DCNNs. Encoder-decoder structure are widely used in computer vision research field such as [1] and [12]. However, it is hardly to visualize what is the network encoding during the training phase. Firstly, I plan to start with a pre-trained and fine-tuned ResNet-18 [2] network. He [2] shows this DCNN has outstanding performance in image classification task. The pre-trained ResNet-18 has achieve top-1 error rate 30.24% and top-5 error rate 10.92% on ImageNet challenge. The dataset used to train and test are tiny ImageNet which is a subset of ImageNet [3]. I will remove the last two layers of the ResNet-18 which are the fully connect layer and the average pooling layer. The output of the network now is a $(2,2,512)$ tensor. Due to the tiny ImageNet has 200 categories, firstly, expand the output to $(2,2,200)$ by two hundred 1×1 convolutional kernels. This is denote as the encoded feature map of the network which is the most compressed data in the network. The above is the encoder part of the network. Then the decoder part is composed of 3 convolution layer with batch normalization and usign Rectified Liner Unit as the activation function. The details of the network will be discussed in the later part.

3.1 Dataset

Due to lack of computational power, I originally would like to train my proposed network on whole ImageNet [3] which contains 1000 categories of object and 456567 images. Considering I only have one Nvidia Tesla K80 graphic card to train the network, it is impossible to train the whole ImageNet on it which takes about 2 to 3 days to finish training. So I use the compressed version of ImageNet, the tiny ImageNet. The tiny ImageNet is a subset of ImageNet which contains 200 classes. Each class has 500 images for training, 50 validation images and 50 testing images. For all images in the tiny ImageNet is resized to 64×64 pixels images which will greatly increasing the training time. Obviously, the small images will significantly effect the performance ResNet-18 pre-trained model and the whole network. But this is a trade-off between time and efficiency. For futher research, if there is enough computational power, it is necessary to get rid of this compressed version of ImageNet and use the original version of ImageNet dataset instead.

3.2 Encoder

As discussed in previous, the encoder part of the proposed model is a modified version of ResNet-18 proposed by [2]. The encoder would finally compressed the image into a $(2,2,200)$ tensor. This we usually called high dimensional feature of an image. The high dimensional feature of an image is hard to visualize. Each sub $(2,2)$ tensor in this high dimensional feature space has a large field of view corresponds to the original image.

$$C(s,t) = \sum_{m=0}^{Mr-1}\sum_{n=0}^{Mc-1} A(m,n)\cdot B(s-m,t-n) \tag{7}$$

Equation (7) shows how convolution works in 2-D dimension. A is the convolution kernel and B is the input image. The output size of C is depending on the stride. Current CNN normally set stride to 2 or higher value to compress input image. Two consecutive 3×3 convolution kernels with stride of 2 can compress a 7×7 iamge into 1×1 size. So this 1×1 point has a field of view 7×7 on original image. Due to this nature of CNN, just visualizing the compressed feature can hardly see the effectiveness of potential factors. However, the encoder-decoder structure is so useful and effective. It leads an idea that the high-dimensional feature is useful but needs an another way to visualize it.

3.3 Decoder

The most important part in the proposed model is the decoder. The decoder is responsible for classify the input image to a category. Although normaly, researchers do not normally use encoder-decoder structure for classification problem. But inspired by [11], weakly supervised learning can still benefit for visualization how encoder-decoder working.Based on the weakly-supervised training idea, I think it is reasonable to use decoder to do classification. The decoder is composed of two transposed convolutional layer just like other normal decoder structure. Basically, the decoder. Each transposed convolutional layer will enlarge the high dimensional feature by 2. For example, the $(2,2,200)$ tensor output by encoder will be enlarged to $(4,4,200)$. Finally the output of the decoder would be $(8,8,200)$. Until now all the things are similar to a normal encoder-decoder network. Nevertheless, normal convolutional layer has a nature to combining the high dimensional features. If I visualize the 8×8 features maps. It is still ambiguous just like the output of the encoder. To overcome this issue, I utilize the group convolution technique proposed by [10]. The idea of group convolution is decreasing the computation complexity. The normal convolutional layer has the same number of groups of convolution kernels as the number of output channel. Each convolution kernel in a group convolves on the input image and take the sum of all as the output to the corresponding output channel. Unlike the traditional way, group convolution do not have groups of convolutional kernel. Instead, the number of convolutional kernel is a multiplication of number of input channel. So the input is sliced into groups. And each group is convolved with its corresponding convolution kernel. In the proposed model the last convolutional layer is group convolution. This layer takes the output of last layer with size of $(4,4,200)$ tensor. And output size of $(8,8,200)$ with each 8×8 feature map corresponds to a class. I will discuss later in detail how to encourage the model to learn the feature map with corresponding class in the last group convolutional layer. The above is all about the decoder part. Additionally, I add two bacth normalization layer between three convolution layer and two drop out layer with drop out probability of 0.3 after the batch normalization layer.

3.4 Classification

As well known, DCNNs are loss driven algorithms. For every model, loss function is responsible for encouraging the whole network to learn its target. Until now, the model can output a tensor of $(8,8,200)$. And ideally, each 8×8 feature map is corresponding to a target class. However, it is hard to design a loss function to compare the distance of a 8×8 feature map with a label. In order to compare the difference of predicted class and target class. We need to form $(8,8,200)$ into a $(1,200)$ vector. And comparing it with a one-hot-encoding $(1,200)$ vector which is the target class denote

as T. To achieve this goal, a global average pooling layer is added which takes the average of each 8×8 into a single number with the equation below.

$$S_k = \frac{1}{M \times N} \sum_{M,N} f_k(x,y) \tag{8}$$

M and N denote the number of columns and rows of feature map (here is 8×8). $f_k(x,y)$ are the value on M column and N row. S_k is the output of each feature map. In this model we have $S = \{S_1, S_2, S_3 .... S_{200}\}$. Finally, taking the SoftMax on S with equation (4) to convert S_k into probability P_K.Now, it is easy to compare S and the target class T. This operation can be easily achived with cross entropy loss with defined below.

$$L = -\sum_k T_k log(P_k) \tag{9}$$

Where T_K denote the corresponding class in target vector T. Cross entropy loss would encourage the corresponding class to have an output to 1. When doing backpropagation through the network, this will encourage the corresponding 8×8 feature map to activation more until the model converges.

Above all is about the model structure and loss function. Below Fig. 1 shows the proposed model architecture. Notice that the last two convolutional layer is group convolutional layer. The First orange box stands for a block of pre-trained ResNet-18 network.

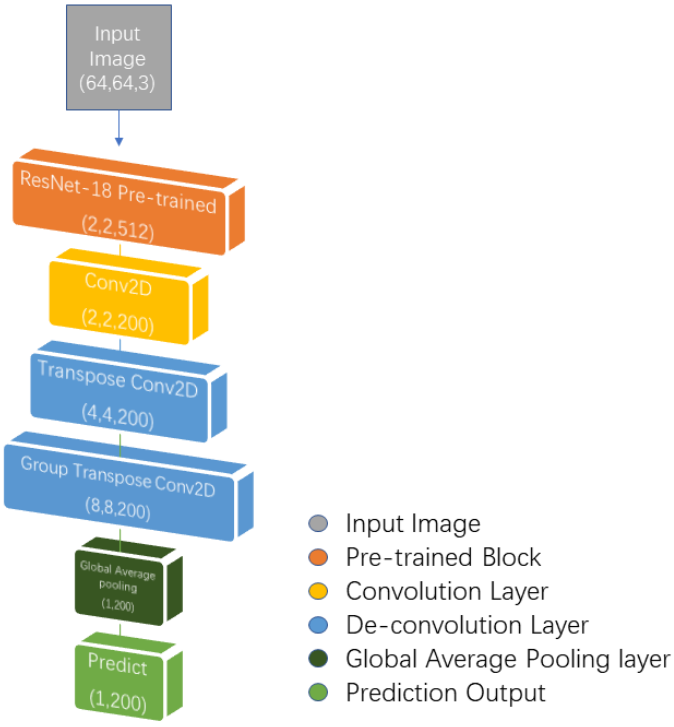

Figure 1: Model architecture

4 Experiment

In this section is mainly about the conducted experiments in this project. I will talk about the experiment setup and the experiment result in this section. Also I compare the result with other models. Our model is trained by PyTorch[1] opensource deep learning framework.

[1]https://github.com/pytorch/pytorch

4.1 Experiment Setup

As discussed in the previous section, our encoder part is a ResNet-18 [2] model without the last MaxPooling layer and the classification SoftMax layer. In detail, the ResNet-18 model is initialized with the pre-trained weights. And all the other layers (the rest of encoder, decoder and classification layers) are initialized with gaussian distribution random numbers. During all the experiment, the hyper parameters are learning rate set to 0.0001, batch size are 100 and number of epochs to train are 80. All the experiment has the same hyper parameters unless we specified. I follow the rules to split the data set by ImageNet official development kit. The learning rate has a cosine decay rate during the training. To compare our model performance, I find an same model on github [2]. I follow the exact the same way to train the network in their second experiment. The Table. 1 blow shows the accuracy of proposed model and baseline model. The proposed model is more accurate in top1 accuracy and worse a little in top5-accuracy which is an acceptable result. The Pre-trained ResNet-18 (224x224) shows the official version of the ResNet-18 performance on original ImageNet data set as a reference.

4.2 Training

Model	Accuracy@1	Accuracy@5
Baseline	52.80%	73.31%
Pre-trained ResNet-18 (224x224)	69.76%	89.08%
Proposed	53.78%	61.09%

Table 1: Comparison of proposed model and baseline models

During the training phase, I also record the training loss, training accuracy, validation loss and validation accuracy as a reference of correctness of my training procedure. The result as below.

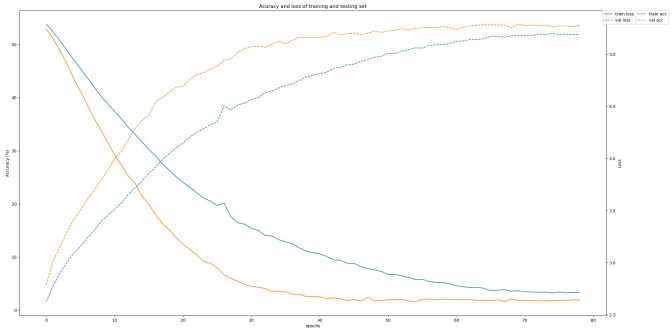

Figure 2: Model architecture

Fig.2 shows the training and validation accuracy and loss progress. The model converges at about 70 epochs which demonstrate that the proposed model is learning the feature and perform at a stable stage. In addition the doted line shows the accuracy and the solid line shows the trend of loss.

My training environment is on GCP (Google cloud platform)[3] with ubuntu 18.04, PyTorch 1.0.0 and a single Nvidia Tesla K80 GPU. The training time totally cost about 12 hours. I totally trained the network from scratch three times. Each time the model shows a very close performance and I take the best one which is shown in Table.1. All the rest experiment is performed on this model unless we specified.

[2]https://github.com/tjmoon0104/pytorch-tiny-imagenet
[3]https://cloud.google.com/

Above all is the details about the experiment. We will look in deep of how this encoder-decoder CNN performs in the next section.

5 ANALYSIS

In this section, I utilize different ways to compare and visualize the perform of proposed encoder-decoder CNN model. This section has mainly three parts which show how is the proposed network learning. Secondly, I compare the different aspect to show the effects to the output. And finally, I did the error analysis.

5.1 How CNNs learn?

It has been a long time topic to reveal how CNNs learn the useful feature to do classification or clustering. [11], [9], [6] and [5] shows a wide variety of way to explain how CNNs learn and how make them lear better and fast.

In section 3, I propose a novel way to visualize the $f_k x, y$ which called activation map. Remind that activation map is generated by the last group convolution layer. The reason why deploy the group convolution layer is inspired by the nature of group convolution can split the fusion features. So that the final stage output can get rid of a fully-connected layer to do the classification which results that the group convolution layer's output is sliced into category with no feature fusion to impact each other. By comparing the equation (6), the proposed model is get rid of the last layer w_k^{class}. To better visualize the activation map, I normalize the activation map by the equation below.

$$f_{k_{norm}}(x,y) = f_k(x,y)/max(f_k(x,y)) \tag{10}$$

The $max()$ function will return the max pixel value in the raw activation map. Here, I didn't process the negative value because the last ReLU layer would not output negative values. To show how CNNs learn the feature, I first compare the output of a same image in different training stage. The examples are shown below.

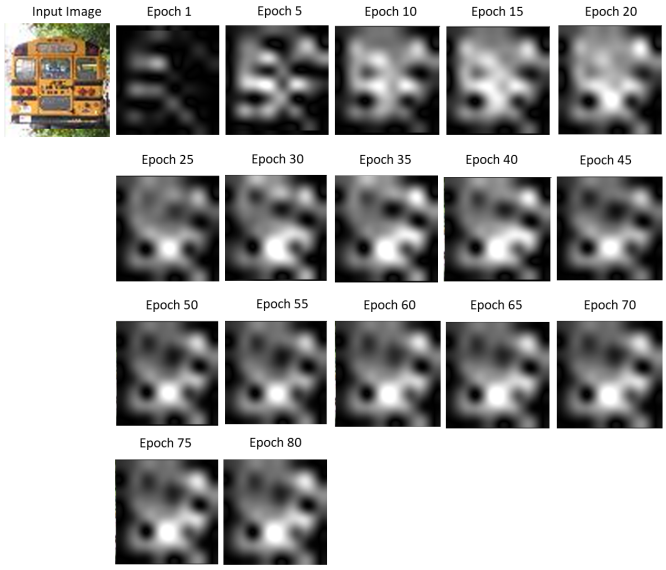

Figure 3: Different activation map output from different epoch with input image school bus

The fig.3, fig.4 and fig.5 shows the model's prediction of three images in epoch 1, 5, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75 and 80. As we can see, the activation map forms a unique pattern during the training for each input image. Actually, the model will generate similar patterns for each class. We will reveal that later.

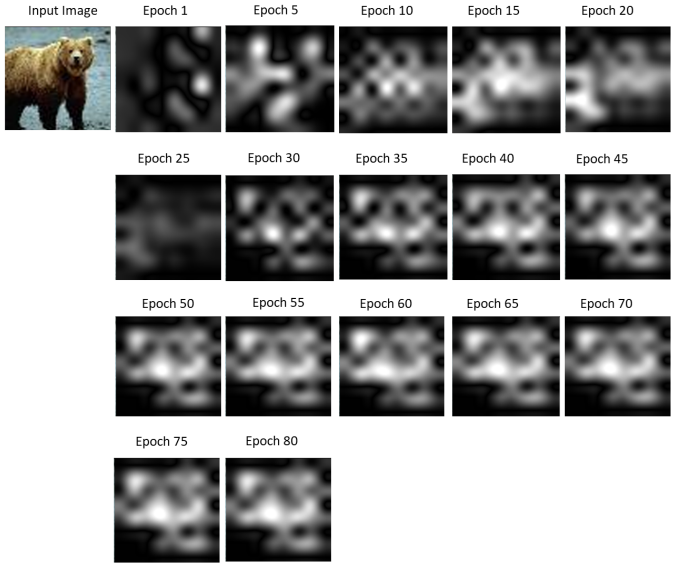

Figure 4: Different activation map output from different epoch with input image brown bear

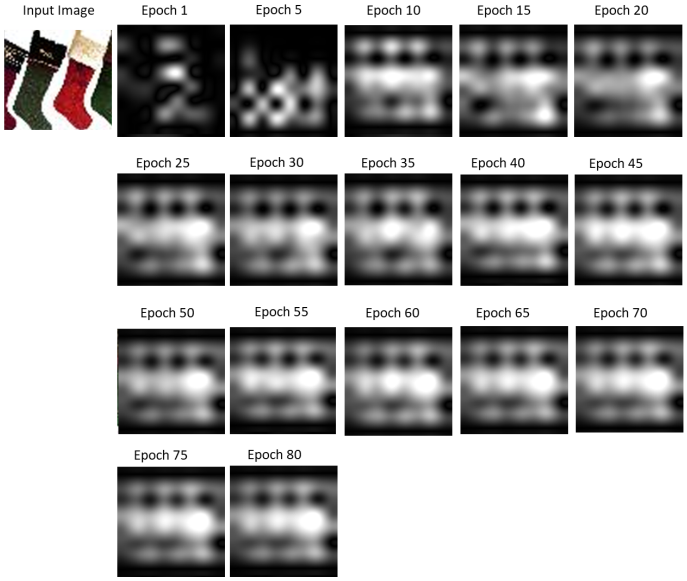

Figure 5: Different activation map output from different epoch with input image christmas socks

From the figures, the unique pattern is changing rapidly during the initial stage of training and becomes stable in the later epochs. This trend is the same as the loss and accuracy. And another interesting result is that the model learns gradually for each class. However, some class learns fast and some slow. Fig. 3, the school bus's unique pattern forms at epoch 5. Fig.5, the christmas sock's pattern forms around epoch 10. The brown bear's unique pattern forms at epoch 30. Actually, the first 6 epochs misclassified the brown bear as golden retriever or bison. And we can see there is a big change during epoch 20 to epoch 30 in Fig.4 which this big change I guess impact a lot in form the unique pattern of brown bear.

From my observations, it is easy to see that different class may learn differently. I guess there is a latent variable to control how to learn different class. Ideally, when researchers train model, we

assume every class learns the same. But from the result, this assumption may not correct. Because brown bear's pattern and school bus' pattern forms at different time, these two classes' accuracy may different. And I guess the proposed model will perform better in predicting school bus.

5.2 Pattern analysis

After understanding the form of each categories' unique pattern, I am also interested in analyzing how different pattern effect intra-class. I hope that The model will generate the same-like pattern when predict a same class. And also, I am still interesting in how the confidence score P_k related to the activation map.

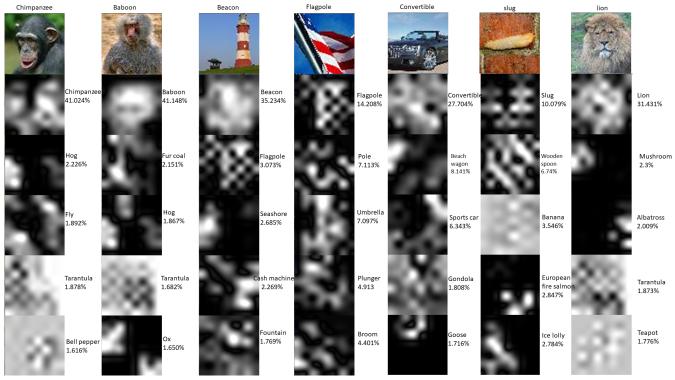

Figure 6: Randomly sampled correct prediction from validation dataset

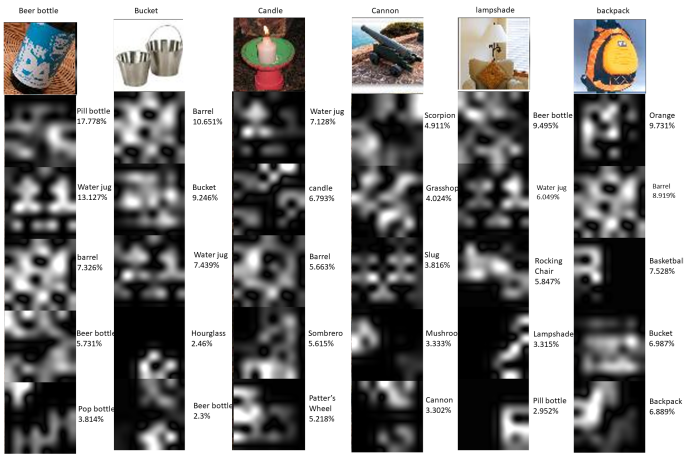

Figure 7: Randomly sampled wrongly prediction from validation dataset

Above shows some randomly choosing image from validation set. Fig.6 shows the correct prediction examples and Fig.7 shows the wrongly predict examples. The label on top is the ground truth and the label on right of each activation map is the prediction and the P_k value which is the confidence score. Notice the $Flagpole$, Hog from Fig.6 and $Water\,jug$, $barrel$, and $beerbottle$ from Fig.7. We can see these activation maps nearly has the same pattern even if they are not the top1 prediction with high confidence score. Interestingly, the three $Water\,jug$ prediction shows that with the increasing of the P_k the activation map is more sharp. Otherwise, the activation map is dark and blur. In Fig.7, the confidence score has less difference than Fig.6. This property may help researcher that we can check this score to identify if potential wrong output occurs. To afflict this idea, I examines on other pictures which also shows the same property. Below are two examples, *schoolbus* and *lion* with each has two different input but show similar activation map pattern. In addition, these two example are both predict correctly (top1 accuracy).

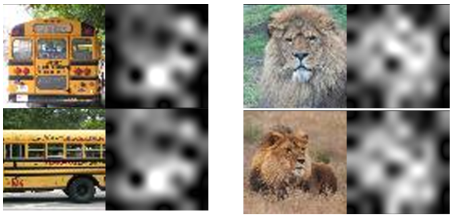

Figure 8: Two examples of same class has same activation map left is input and right is activation map

From the above example, it is easy to see when predict correctly, the model would generate the same activation map pattern. In conclusion, proposed network can learn an unique and identifiable activation map which can be used to object classification. More importantly, this activation map pattern can be used to find the confidence of the output which may help to identify the potential wrong output.

5.3 Noisy analysis

To learn how CNNs learn during training, it is not enough to see the examples from input to output. In this section, I use some conventional computer vision techniques to change the input images' color space, shape or element. But keep the images' main object which is still can be identified by human eyes in some cases.

5.3.1 Gray image

The first experiment I conduct is eliminate all the color information from the input image and keep the shape information. To do that I convert the image into gray scale before feed into the model. Below is some examples generated from this method.

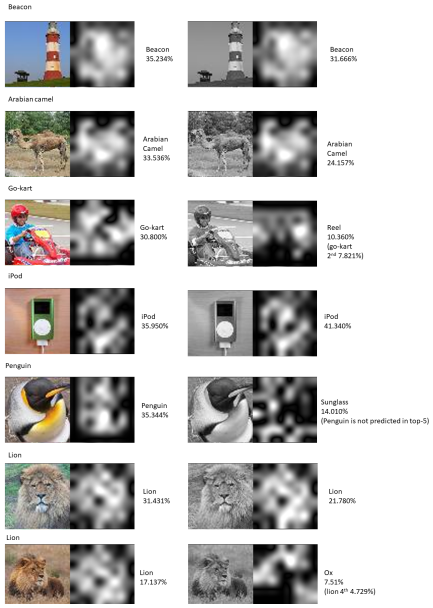

Figure 9: Gray image as input compared with original image

Fig.9 shows some random examples of gray scale image compared with original image. As I expect, I originally think the model

will perform very bad due to loss of color information. However, the output astonishing me. I test on 100 random sampled image and the accuracy is still 45.52%. From the observation, the activation map for predicted image is still unique for that class which is identifiable. But noticing on the wrong answer, the confidence score of wrong answer can hardly achieve 20% which just like the comment stated in section 5.1. The wrong answer always have low confidence score and most scores are less different. In details, the confidence score of iPod(4th row) is even increasing. The reason, I guess is that in the training dataset most iPod is white or silver. Due to the same color, that may be one of the reason the confidence score is increasing.

5.3.2 Gaussian blur

Gray scale image keeps the shape information of the original image. How about we inverse this process which we keep the color information and eliminate the shape information. To achieve this goal I apply gaussian filter to the input image. The equation of gaussian filter I use is below.

$$G_{(x,y)} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{11}$$

Where σ is the standard deviation of the gaussian distribution. To analyze more carefully, I apply two types of gaussian filter, kernel size 3×3 and kernel size 5×5. Below is some example images comparing the original images and gaussian blur images.

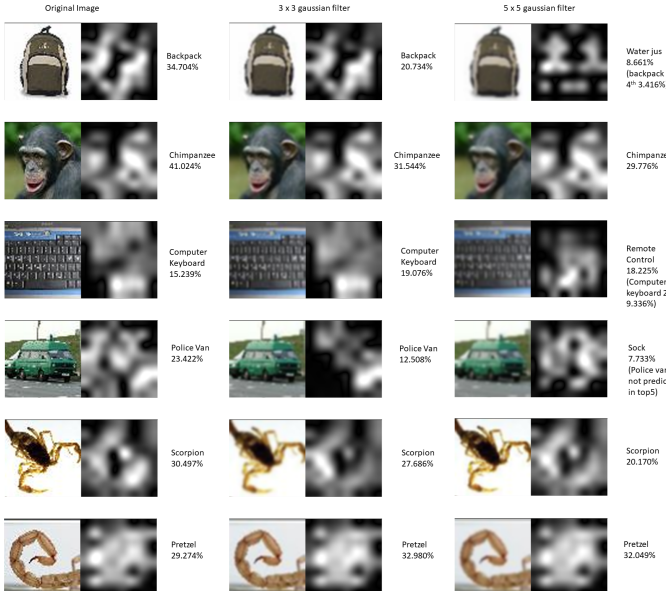

Figure 10: Gray image as input compared with original image

Compared with the gray scale image, 3×3 gaussian filter do not impact a lot. However, 5×5 gaussian filter make the result worse which is reasonable in this experiment. Interestingly, the last row of the scorpion is misclassified as *Pretzel*. With the gaussian filter, the confidence score is even increasing. This shows a weakness of the CNNs. But we see other examples are still correct with 3×3 gaussian filter. That is a good news for the robustness of the proposed model. Notice that the *Policevan* example when apply 3×3 gaussian filter, the left lower corner of the pattern is lost compared with the original image. But the other part keeps the same. That shows the usefulness of the activation map pattern.

5.3.3 color channel

In previous section, I discuss the effect of color and shape to the model. Nevertheless, I still interesting in digging more deeply in this field. So I design two more experiments to show deep effect of color and noise. First, I would like to show the effect of different color effect the result but keep the shape the same. Compared with gray scale image, this method bring more challenges to the model and I expect the model will perform worse than gray scale image.

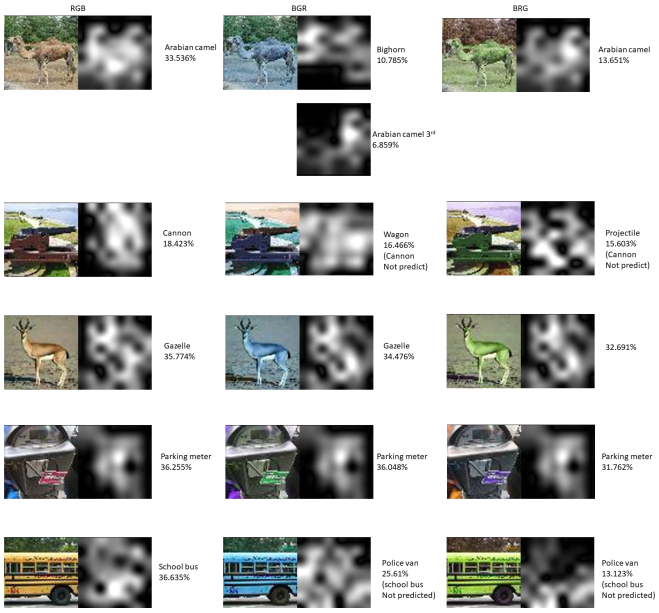

Figure 11: shuffle the color channel compared with the original image

Fig.11 shows the comparison of original image and the shuffled channel image. The original images have channel order $[R,G,B]$ before feeding into the model. To shuffle the channel, I choose two different channel order, $[B,G,R]$ and $[B,R,G]$. The results are interesting. In the example. Gazelle with shuffled channel order has merely no change. However, with almost the same color, camel are impact by this color channel ordering. Same as camel, school bus are identified as police van with high confidence. After checking the training data, camel data are captured with a wide variety of angle and some of them only has part of the bogy. However, gazelle always have full body with different angle. That's why may be some class are easy to learn and some other class are hard to learn (section 4). But why school bus are identified as police van? I believe that the first reason they are both vans. The second reason, in the training data the police van are some times blue and green (police van skin in europe or some where). From the parking meter, it is easy to explain why it has no effect for the channel ordering. The parking meter is almost black. Changing the channel ordering has nearly not effect for the color.

From this experiment, it is obviously that the CNN model is very sensitive to the color information. And we can conclude that, CNN is a data driven model. With more data, the noise can be canceled during the training in a single image. So more data and data augmentation is needed in training the CNN model.

5.3.4 Salt & Pepper noise

The last experiment I did is using a conventional and general way to add noise to the input images. The previous experiment are dedicate in fix one element and see the effects of the other element. In this experiment, I utilize the Salt & Pepper noise (S&P noise) which is a general noise would effect both shape and color. To generate the S&P noise, the simplest idea is change random choose channel pixel value to 0 for pepper or 255 for salt. The amount of pepper M_{noise} is controlled by λ showing in the equation below.

$$M_{noise} = \lambda \times (H \times W \times C) \tag{12}$$

Where H, W and C denote the input image height, width and channel. We pick two λ in this experiment 0.005 and 0.0005. Here I show some example images.

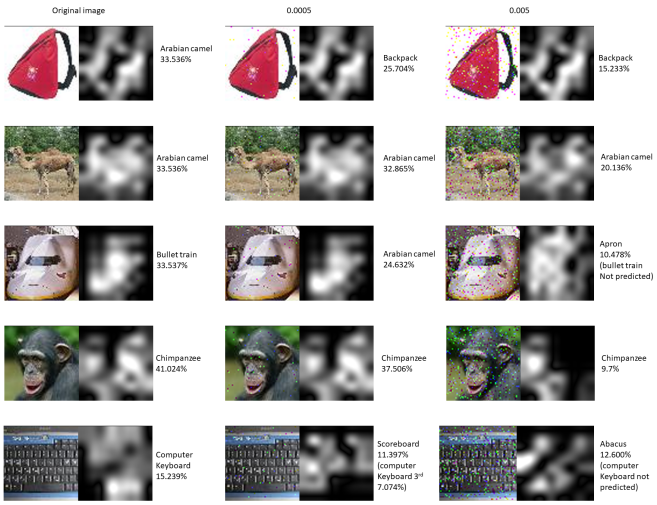

Figure 12: Salt & Pepper noise image compared with the original image

From the above examples, the S&P noise show different action of the previous experiment. Nearly all examples above are effected by this kind of noise. And intuitively, all the confidence score are decreased linearly. S&P noise not only changes the shape but also changes the color which is hard for the model to capture the changes. I also send these images to 10 people to identify what is this. Unlike the above examples, this examples all can be correctly identified by human eyes (the previous experiment are not). This shows a new idea that the CNNs identify the object different than humans. Maybe CNN is not as string as we think. It is also capture the subtle information but not a global information. So that this S&P noise can effect the results. Nevertheless, this result shows a new way to do the data augmentation which is to add the S&P noise when training the model.

6 Conclusion

In this CSE 261 project, I propose a new model architecture which is suitable for visualizing a certain kind of CNN models, encoder-decoder model. And proposing a new activation map technique to visualize the CNN model via group convolution. I conduct a bunch of experiments to show the potential factors to effect the CNN model and use the comparison visualize technique to compare and find the most informative factor. We explore the progress of how CNN learns during training stage and find that for each class the learning time is different and that's why the model has some bias for some classes. I conduct a bunch of experiment to show the noise effect the models. And the result is interesting. The CNN model utilize both shape and color information to predict the result. But for different objects, the discriminating information seems different. That's probably what CNN learns during training. The final experiment shows the effectiveness of S&P noise effect the model in for linearly decreasing the confidence score may be can be utilize in the future of training CNN models.

All in all, in this project I combine the computer vision and data visualization and try to explain how CNN learns and its strength and weakness in a data visualization way. This new perspective encourage the future study of computer vision and I hope more researchers in data visualization field can be interesting in deep learning and computer vision. All the code can be found on my github repo [4]

[4] https://github.com/zxh009123/visualization-DCNNs-CSE261-W19-UCSC

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. pp. 2278–2324, 1998.

[5] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 91–99. MIT Press, Cambridge, MA, USA, 2015.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[7] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding Neural Networks Through Deep Visualization. *arXiv e-prints*, p. arXiv:1506.06579, June 2015.

[8] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *arXiv e-prints*, p. arXiv:1311.2901, Nov. 2013.

[9] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds., *Computer Vision – ECCV 2014*, pp. 818–833. Springer International Publishing, Cham, 2014.

[10] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang. Interleaved group convolutions for deep neural networks. *CoRR*, abs/1707.02725, 2017.

[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[12] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.