Animal Detection Using Deep Learning

Lei Lin
Department of Computer Science
Stanford University
ll1978ru@stanford.edu

Abstract

Using deep learning to study animals' demeanor and body language, we can find out if they are sick or not and provide necessary help in time. In order to achieve this goal, we need to start with animal species classification.

In this project I will train one of the deep learning models, VGG, to distinguish between images of cats and dogs. After using the Transfer Learning from VGG-16, the accuracy can increase from 80% to over 95%. Then two ways of visualizing the model outputs have been successfully demonstrated for more insight of the VGG model.

1. Introduction

Recently, animal detection for wildlife has been an area of great interest among biologists. Since there are many species, manually identifying them can be a daunting task. So, a deep learning algorithm that classifies animals based on their images can help monitor them more efficiently. A further possible application of this technology can be used to identify certain species' behaviors. Using deep learning to study animals' demeanors and body language, we may know if they are sick or not and provide necessary help and treatment in time. All these challenges necessitate an efficient algorithm for classification.

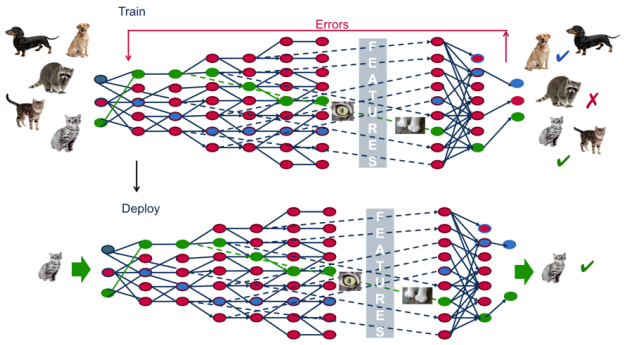

Figure 1. the demonstration of deep learning on animal classification [1].

In this project, Dogs vs Cats, the data set from Kaggle, will be applied for the classification model training. Although the problem sounds simple, it was only effectively addressed in the last few years using deep learning convolutional neural networks. While the dataset is effectively solved, it can be used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch.

This includes how to develop a robust test harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.

For this project, the simple CNN model, VGG-6, was first built from scratch. The training accuracy is above 80% and validation accuracyis around 86%. In order to further improve the accuracy, the transfer learning will be used in this project. The VGG-16 was applied for the tranfer learning and I got the training accuracy above 95%. And then training and validation accuracy and loss will be collected from all training models and used to evaluate the performance of the each classification model. To gain more insight of the VGG models, I will demonstrate the methods of visualizing the model outputs, which include visualizing intermediate layer activations and visualizing heatmaps of class acitvations.

2. Related work

For several years, animal detection in the wildlife has been an area of great interest among biologists. They often study the behaviour of the animals to predict their actions. Since there are a large number of different animals, manually identifying them can be a daunting task. So, an algorithm that can classify animals based on their images can help researchers monitor them more efficiently. Also, animal detection and classification can help prevent animal-vehicle accidents, trace animal facility, prevent theft, and ensure the security of animals in zoos.

The application of deep learning is rapidly growing in

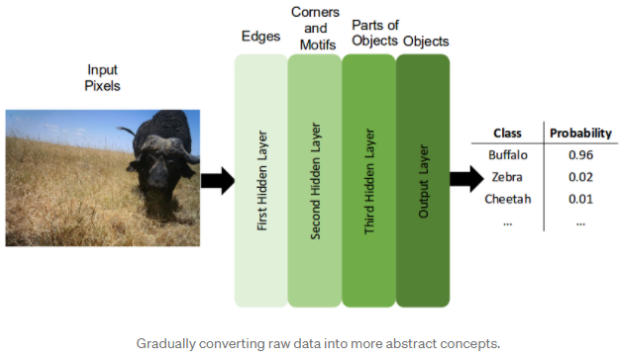

Figure 2. Description of some of the most popular Deep Neural Network architectures [7]

the field of computer vision and is helping in building powerful classification and identification models. We can leverage this power of deep learning to build models that can classify and differentiate between different species of animals as well.

Related works about animal classification were published in regard of wild animal monitoring [2–5]. The authors used convolutional neural networks to create models for animal species identification. Multiple versions of the snapshot dataset were used in order to probe the ability of the method to cope with different challenges that camera-trap images demand. The proposed methods can reach 94% accuracy. Similarly, convolutional neural networks were used to recognize 20 species common in North America [6] . This was the first attempt of the fully automatic computer vision based species recognition on the real camera trap images. The imagery data were captured with motion triggered camera trap and were segmented automatically using graph-cut algorithm. Another approach was taken for classifying different animals in automating species recognition [7]. The authors enforced ScSPM (Sparse coding Spatial Pyramid Matching) to extract and classify animal species over a 7 thousand image data set. After that multi-class pre-trained SVMs were applied to classify global features of animal species (Figure 2). All mentioned authors follow a similar pattern when using machine learning for image classification, but none of them concern household animals. Also, most of them lack the application of visualization technique. This technique gives insight into what types of features deep neural networks are learning at specific layers in the model and provides a debugging mechanism for improving a model.

3. Dataset and Features

Since this is my first deep learning project, I would like build up the deep learning model using the simple dataset, Dogs vs Cats, which is from Kaggle [8].

Figure 3. Dogs vs Cats from Kaggle

The dataset provides 25,000 labeled photos: 12,500 dogs and the same number of cats. Out of all the photos, 16% are for validation, 4% for testing, and the remainder for training. Kaggle also provides unlabeled photos for testing in another folder. The images (Figure 3) are of non-uniform size (averaged around 350×500) and varying image quality. Some of them are grayscale, and some files are corrupted, making them unreadable. We also can see that some photos are landscape format, some are portrait format, and some are square. Additionally, the perspective can differ from faces to full-body. In some images, part of the animal is obstructed from view, and others contain more than one of the same animals.

The photos will have to be reshaped prior to modeling so that all images have the same shape. This is often a small square image. There are many ways to achieve this, although the most common is a simple resize operation that will stretch and deform the aspect ratio of each image and force it into the new shape. We could load all photos and look at the distribution of the photo widths and heights, then design a new photo size that best reflects what we are most likely to see in practice. Smaller inputs mean a model that is faster to train, and typically this concern dominates the choice of image size. In this case, we will follow this approach and choose a fixed size of 150×150 pixels. As you may already know, data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network. (It is uncommon to feed raw pixels into a convnet.) In this case, we will preprocess our images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

4. From VGG-6 to VGG-16

4.1. Develop a simple CNN model

The baseline CNN model (Figure 4) is from the general architectural principles of the VGG [9] models. The main contribution of VGG is to show that classification/localisation accuracy can be improved by increasing the depth of CNN inspite of using small receptive fields in the layers, especially earlier layers.Neural networks prior to VGG used bigger receptive fields (7×7 and 11×11) as compared to 3×3 in VGG, but they were not as deep as VGG.

The architecture involves stacking convolutional layers with small 3×3 filters followed by a max pooling layer. Together, these layers form a block, and these blocks can be repeated where the number of filters in each block is increased with the depth of the network such as 32, 64, 128, 128 for the first four blocks of the model. Padding is used on the convolutional layers to ensure the height and width shapes of the output feature maps matches the inputs. Each layer uses the ReLU activation function. In the end it has 2 FC(fully connected layers) followed by a softmax for output.

The model has been fitted with RMSprop optimizer which is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.
Gradient descent with momentum:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$
$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db$$
$$W = W - \alpha \cdot v_{dw}$$
$$b = b - \alpha \cdot v_{db}$$

RMSprop optimizer:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$
$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$
$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$
$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

Sometimes the value of v_dw could be really close to 0. Then, the value of our weights could blow up. To prevent the gradients from blowing up, we can include a parameter epsilon in the denominator which is set to a small value.

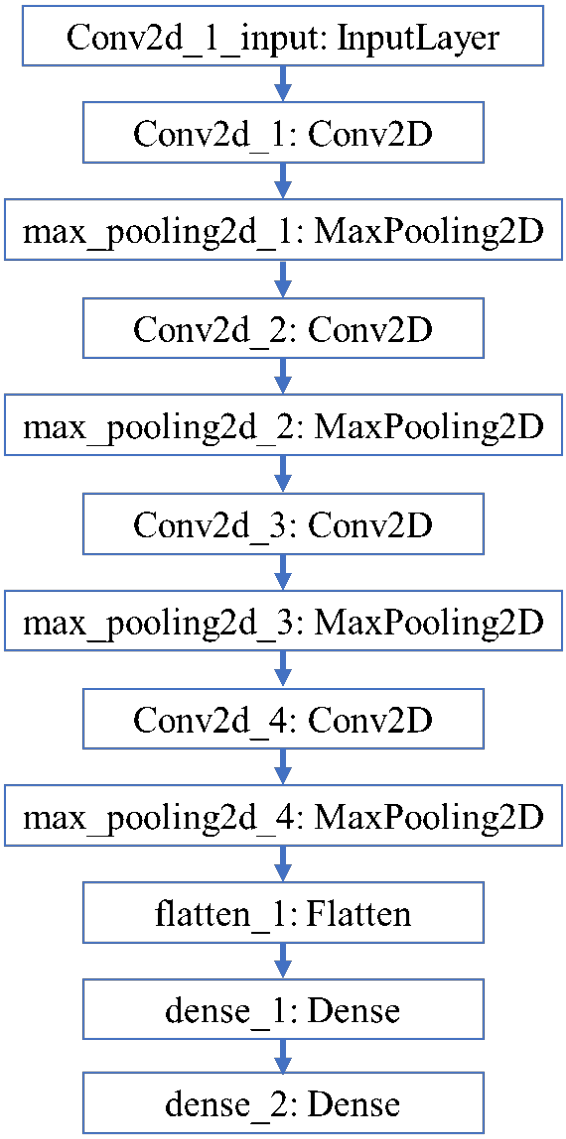

Figure 4. The structure of VGG-6

4.2. Transfer Learning from VGG-16

A more refined method would be to utilize a network which is pre-trained on a large dataset. This network would have already learned features that are useful for solving various problems such as Image Classification and Object Detection. This method would allow us to obtain better accuracy. Transfer learning involves using all or parts of a model trained on a related task. In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest. Transfer learning has the benefit of decreasing the training

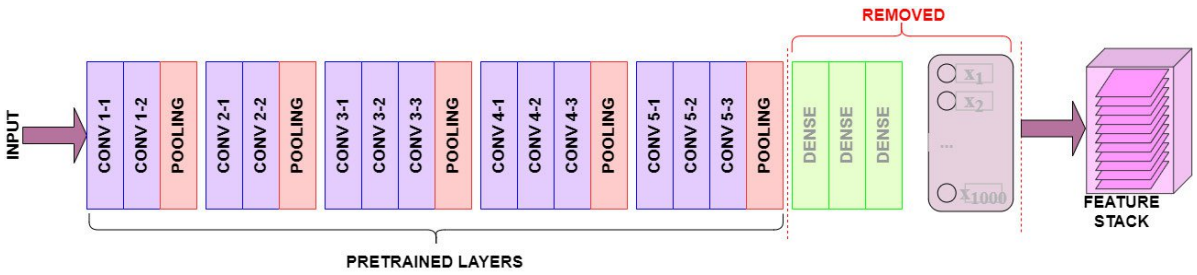

Figure 5. The above graphic shows the "top" portion of the model as removed. The model convolves the pre-trained output layers left behind as a 3D stack of feature maps.

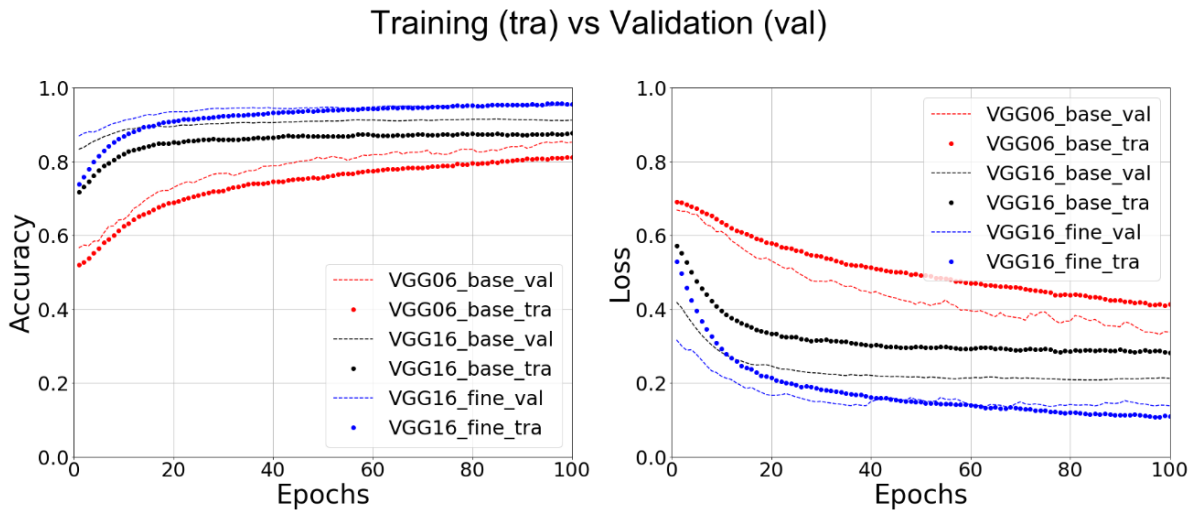

Figure 6. The result of training and validation from VGG model

time for a neural network model and can result in lower generalization error.

A useful model for transfer learning is one of the VGG models, such as VGG-16 [10] with 16 layers. In the 2014 ImageNet Classification Challenge, VGG16 achieved a 92.7% classification accuracy. But more importantly, it has been trained on millions of images. Its pre-trained architecture can detect generic visual features present in our dataset.

Now I can't use the entirety of the pre-trained model's architecture. The Fully-Connected layer generates 1,000 different output labels, whereas my target dataset has only two classes for prediction. Therefore, I use the feature extraction part of the model and but "cut off" the Fully-Connected layers (also called the "top" model) and then add a new classifier part that is tailored to the Dogs vs. Cats dataset (see Figure 5). Specifically, I held the weights of all the convolutional layers fixed during training, and only train new fully connected layers that will learn to interpret the features extracted from the model to make a binary classification.

4.3. Accuracy and Loss

The overfit is observed from my initial VGG-6 model. Therefore, I modify the baseline model through Dropoff and Data augmentation. The training accuracy is above 80% and validation accuracy is around 86% with the loss lower than 0.4 (red curves in Figure 6). We could keep tuning the network to improve training accuracy by lowering the Dropout rate a bit and to train more. But I guess we're not going to be able to reach 95% on this dataset.

The VGG-16 base model (black curves in Figure 6) with data augmentation can significantly improve the training performance. Reviewing the learning curves, we can see that the model fits the dataset quickly at the first 20 epochs. The training accuracy comes out to be around 90% and the validation accuracy being 91%.

To further improve the accuracy, we can fine-tune the weights of some layers in the feature detector part of the model.The goal of fine-tuning is to allow a portion of the pre-trained layers to retrain. In the previous approach, we used the pre-trained layers of VGG16 to extract features.

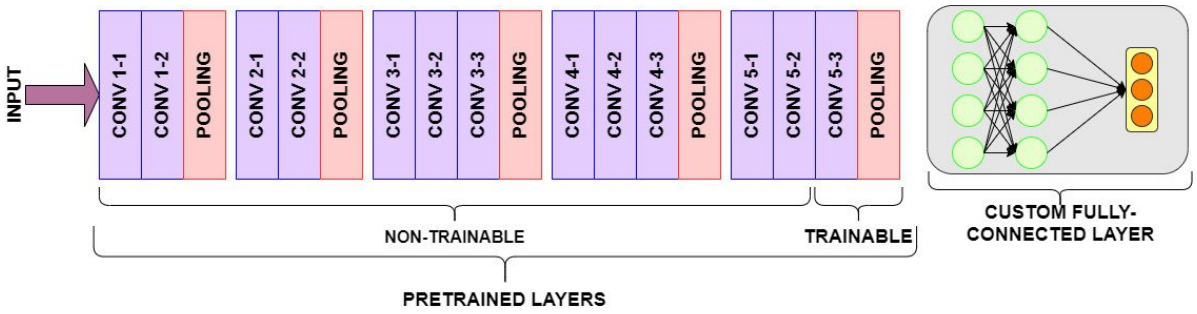

Figure 7. In the above graphic representation, the final convolutional and pooling layers are unfrozen to allow training. A Fully-Connected layer is defined for training and prediction.

We passed our image dataset through the convolutional layers and weights, outputting the transformed visual features. There was no actual training on these pre-trained layers. Fine-tuning a Pre-trained Model entails:

- Bootstrapping a new "top" portion of the model (i.e., Fully-Connected and Output layers)
- Freezing pre-trained convolutional layers
- Un-freezing the last few pre-trained layers training.

The frozen pre-trained layers will convolve visual features as usual. The non-frozen (i.e., the 'trainable') pre-trained layers will be trained on our custom dataset and update according to the Fully-Connected layer's predictions (Figure 7). In this project, we unfreeze from the layer 'block5_conv1' along with our Dense layers, resulting in more than 95% validation accuracy (blue curves in Figure 6). Applying fine-tuning allows us to utilize pre-trained networks to recognize classes they were not originally trained on, reducing the loss to 0.1.

5. Understanding the Convolution Network with Visualizations

Despite the wide availability of large databases and pre-trained CNN models, sometimes it becomes quite difficult to understand what and how exactly your large model is learning, especially for people without the required background of Machine Learning. Although, reading on basic Statistics and Probability helps to overcome some hurdles, but when it comes to debugging large convolution architectures like Inception models, many people take the fall. The goal of most people is to just use the pre-trained model to some image classification or any other related problem to get the final results. They are least bothered about the internal workings of the network, which can actually tell them a lot about how and what their network is learning and also debug its failures.

Visualizing the output of the model is a great way to see how its progressing. While training deep networks, most people are only concerned with the training error(accuracy) and validation error(accuracy). Judging these two factors does give us an idea of how our network is performing at each epoch. When it comes to deep CNN networks like VGG-16 there is so much more that we can visualize, thus allowing us to learn about network architecture. [11] In this project, I will demonstrate two ways of visualization the model outputs (intermediate as well as final layers), which can help us gain more insight into working of the model.

5.1. Visualizing Intermediate Layer Activations

For understanding how the deep CNN model is able to classify the input image, we need to understand how the model sees the input image through studying the output of its intermediate layers. By doing so, we are able to learn more about the workings of these layers. On the following part, I will pick an image of a dog and to try to see what will be the visualized outputs from some of the intermediate convolution of the trained VGG-6 model.

If we take a look at the different images from Convolution layers filters, it is pretty clear to see how different filters in different layers are highlighting or activating different parts of the image (see Figure 8).

- The first layer acts as a collection of various edge detectors. At that stage, the activations are still retaining most of the information present in the initial picture, although there are several filters that are not activated and are left blank.
- As we go higher-up, the activations become increasingly abstract and less visually interpretable. They start encoding higher-level concepts such as "dog ear" or "dog eye". Higher-up presentations carry increasingly less information about the visual contents of the image, but adding more information related to the class of the image to the model.

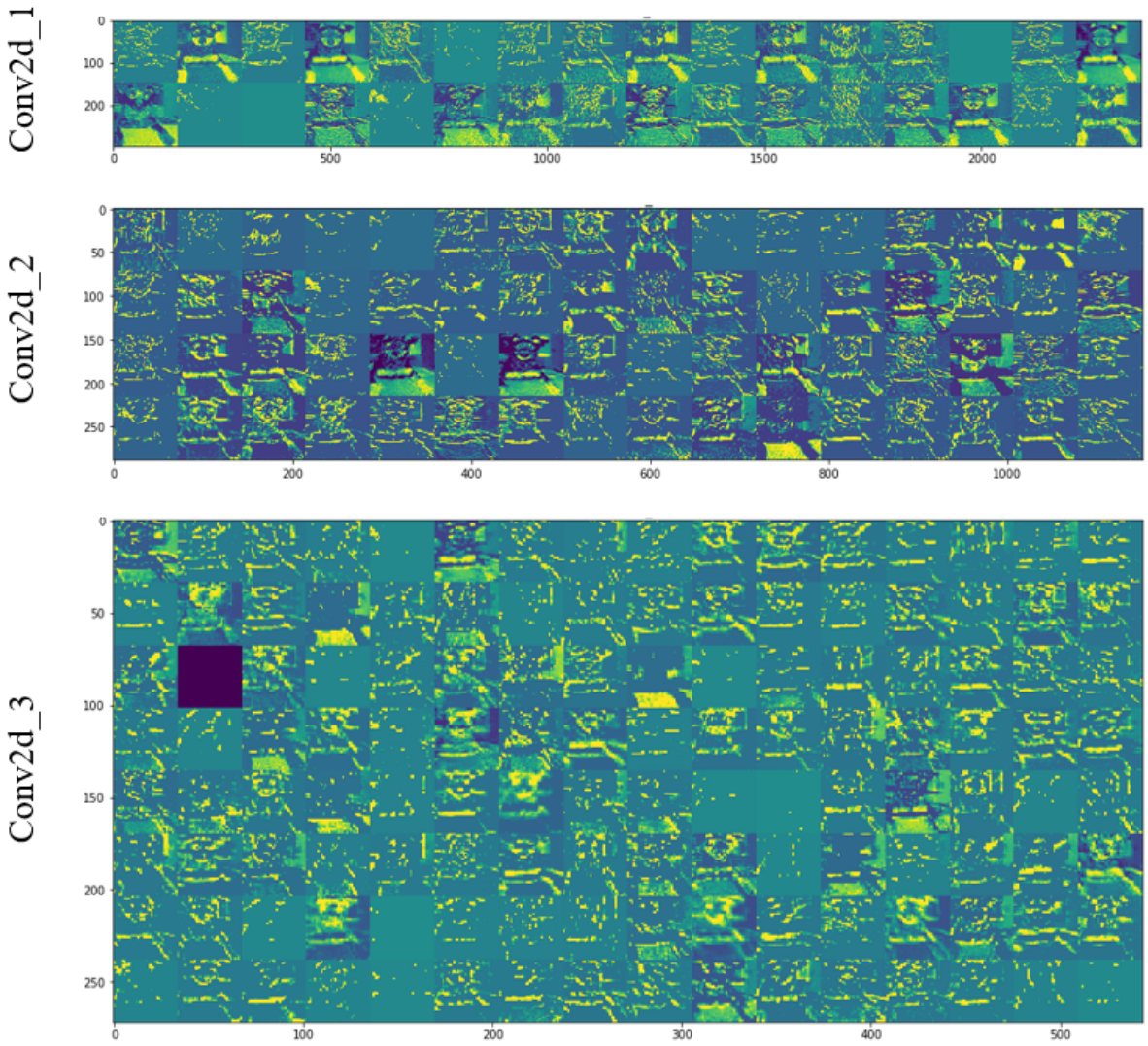

Figure 8. The filters from conv2d_1, conv2d_2 and conv2d_3 by running the trained VGG like model on one of the test images.

- The sparsity of the activations increases with the depth of the layer: in the first layer, all filters are activated by the input image, but in the following layers more and more filters become blank, meaning that the pattern encoded by the filter is not found in the input image.

So this is it! We have visualized how a convolutional neural network finds patterns in some basic figures and how it carries the information from one layer to another one.

5.2. Visualizing Heatmaps of Class Activations

While predicting the class labels for images, sometimes the model will predict the wrong label for the class, i.e. the probability of the right label will not be maximum. In such cases, it will be helpful if we could visualize the parts of the image the convnet looking at to deduce the class labels. The general category of such techniques is called Class Activation Map (CAM) visualization. CAM can produce heatmaps of class activations over input images [12]. A class activation heatmap is a 2D grid of scores associated with a particular output class. It is computed for every location of an input image, indicating how important each location is with respect to that output class.

The Global Average Pooling layer (GAP) is preferred over the Global MaxPooling Layer (GMP) which is used in VGG-16 model because GAP layers help to identify the complete extent of the object as compared to GMP layer which only identifies the discriminative part. This is because in GAP we take an average across all the activation that helps to find all the discriminative regions while the GMP layer only considers the most discriminative one.

The spatial average of the feature map of each unit in the

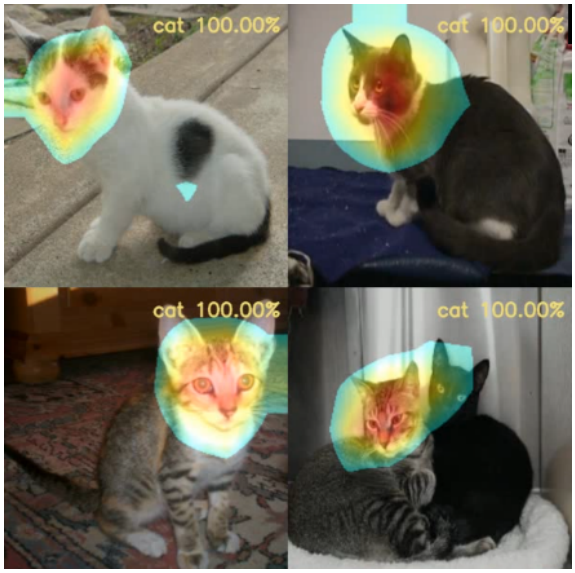

Figure 9. The demo of CAM visualization.

last convolutional layer is produced by the GAP layer which is then weighted summed to generate the final output. Similarly, we produce the weighted sum of the last convolutional layer to obtain our class activation map.

In Figure 9, we can see how this technique works. Start from top two plots. At the early training phase, the model seems to randomly pick up interest locations, plotting heatmap points by chance. As the training moves forward, the model slowly converges on the cat face and paws with the accuracy of 99.99%. At the end, the heatmap finally captures the entirety of the cat face with the output of 100% accuracy. So basically, what the heatmap is trying to tell us is the important locations in the image for that particular layer to classify it as the target class, which is cat in this case. I have uploaded one small video onto YouTube*, which demonstrates how the CAM works from the beginning to the end.

Class Activation Maps or CAMs are a great way to see what the CNN models are seeing while they are training and offers insights to the developer as well as the stakeholder which can be very crucial to the model's usability and lifecycle. By the help of the CAMs we can see which region in the image the model was looking at and can be used to examine the reasons for bias in the model, lack of generalizing power or many other reasons which when fixed can make the model more robust and deployable in the real world.

6. Conclusion and Future Work

In this report, I discussed how to train a CNN model to classify Dogs vs. Cats images. I trained the models from scratch at first and then using Transfer Learning, I got over 95% accuracy. Meanwhile, two ways of the model outputs visualization have been demonstrated, which can help us gain more insight into how the model works.

One of the original tasks for this project is applying the CNN model to detect whether animals or pets are sick or not? In real life, it may be difficult to know if subtle changes in the animals indicate a health problem. Using the CNN model, I may be able to keep track of animals' behavior in order to prevent extreme illnesses. The discharge from eyes or nose may indicate a possible upper respiratory infection; or skin irritation or hair loss may be a sign of allergies, external parasites, or another skin condition. Because of the time limitations, I did not collect enough dataset for this kind of study. For the future works, I will keep collecting datasets for a month or a year in order to find the correlation between illnesses and animal behavior. Through the deep learning model, I will be able to predict the probability of a symptom for every animal picture or video. Future work might also include robotic systems that monitor the state of the animals, adjust food distribution depending on image readings or alert when the animals from any kind of illness.

Moreover, different types of models can be employed to see, which one fits the needs the most. The project shouldn't be limited to VGG models.

GitHub and YouTube Links*

Github code: Project Code
CAM YouTube demo link: Heat map Demo

Acknowledgements

I would like to express special thanks to the CS231n instructors as well as the teaching assistants who led me into the brand new field: Convolutional Neural Networks for Visual Recognition. With their help, I gained the capability to try, work and finish my first deep learning project.

References

[1] Carol McDonald. Demystifying ai, machine learning and deep learning. 2020. https://mapr.com/blog/demystifying-ai-ml-dl/. 1

[2] Alexander Gomez, Augusto Salazar, and Francisco Vargas. Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks. 2016. https://https://arxiv.org/pdf/1603.06169.pdf/. 2

[3] Margarita Favorskay and AndreyPakhirka. Animal species recognition in the wildlife based on muzzle and shape feature using joint cnn. *Procedia Computer Science*, 159:933–942, 2019. 2

[4] Ahmed Zaitoon. Deep learning for animal identification. 2018. https://github.com/A7med01/Deep-learning-for-Animal-Identification/. 2

[5] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S. Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, 115(25):5716–5725, 2018. 2

[6] Guobin Chen, Tony X. Han, Zhihai He, Roland Kays, and Tavis Forrester. Deep convolutional neural network based species recognition for wild animal monitoring. *2014 IEEE International Conference on Image Processing*, pages 858–862, 2014. 2

[7] Xiaoyuan Yu, Jiangping Wang, Roland Kays, Patrick A Jansen, Tianjiang Wang, and Thomas Huang. Automated identification of animal species in camera trap images. *EURASIP Journal of Image and Video Processing*, 2013(52), 2013. 2

[8] Dogs vs. cats. 2014. https://www.kaggle.com/c/dogs-vs-cats/data. 2

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015. https://arxiv.org/pdf/1409.1556. 3

[10] Abhay Parashar. Vgg 16 architecture, implementation and practical use. 2020. https://medium.com/pythoneers/vgg-16-architecture-implementation-and-practical-use-e0fef1d14557. 4

[11] Ankit Paliwal. Understanding your convolution network with visualizations. 2018. https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b. 5

[12] Divyanshu Mishra. Demystifying convolutional neural networks using class activation maps. 2019. https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-class-activation-maps-fe94eda4cef1. 6