# Learning Unsupervised Localized Activation Maps through Classification Model

# Abstract

Most deep learning models, although powerful, acts like a black box where it is hard to explain a model’s decision-making process. We review the Class Activation Mapping (CAM) technique that modifies the last layer of a classification model on some image dataset, so that in addition to the usual logits’ prediction, one can view a map that tells us which region from the image activates such prediction. When training an image classifier, researchers found out that the model can learn a certain map indicating the region of interest on the image without explicit supervision. Such activation map can be used to explain the reason of classification. Our report will be focus on building a simple baseline model on creating the activation maps of our datasets, and then improve the architecture hoping to generate more precise activation maps. We use the Flower-17 dataset that is available on Kaggle to experiment the CAM technique, which has not been done before (Sani, 2021). To speed up our experiment, we use the feature extractor of a pretrained VGG16 model and freeze its parameter during training, which allow efficient memory usage when storing our trained models.

***Keywords: Deep Learning, Image Classification, Convolutional Neural Network, Class Activation Mapping, Weakly Supervised Learning, Pseudo-mask.***

# Introduction and Background

We have seen competitive advancements on deep learning in the past few years, and it is only going faster. Among all of the deep learning categories, image classification remains the most researched and there are plenty of state-of-the-art models available. Some examples include Residual Neural Network (ResNet) that alleviates gradient vanishing issue in a very deep neural network (Kaiming He et al., 2016; Ross Wightman et al., 2021) and the YOLO series that keep improves the result of classification (Chien-Yao Wang et al., 2022).

With such improvements on classification accuracies, it strikes curiosity in our heart to wonder how exactly does a classification model make a decision that an image belong to one of the many classes? This process is kind of like mathematical theories, where people first assume something heuristically work and then take plenty of times to justify the rationale of its mechanics, in contrasts to common beliefs that mathematical theorems are first rigorously proved before widely used. However, current deep learning model, especially the ones involving convolutional neural network (CNN) layers are too complex to be able to clearly justified, thus we might take a step back and try to explain better than what we can for now. Class Activation Mapping (CAM) is one of such technique that improves the interpretability of a classifier (Bolei Zhou et al., 2015), as shown in Figure 1. The CAM technique clearly explain the decision of classifying the data as “Australian terrier” by showing that the last convolutional layer is activated by the “dog” part of the image, rather than the “human” part. In the discussion below we will use CAM to refer to the class activation mapping technique and the class activation map itself interchangeably.

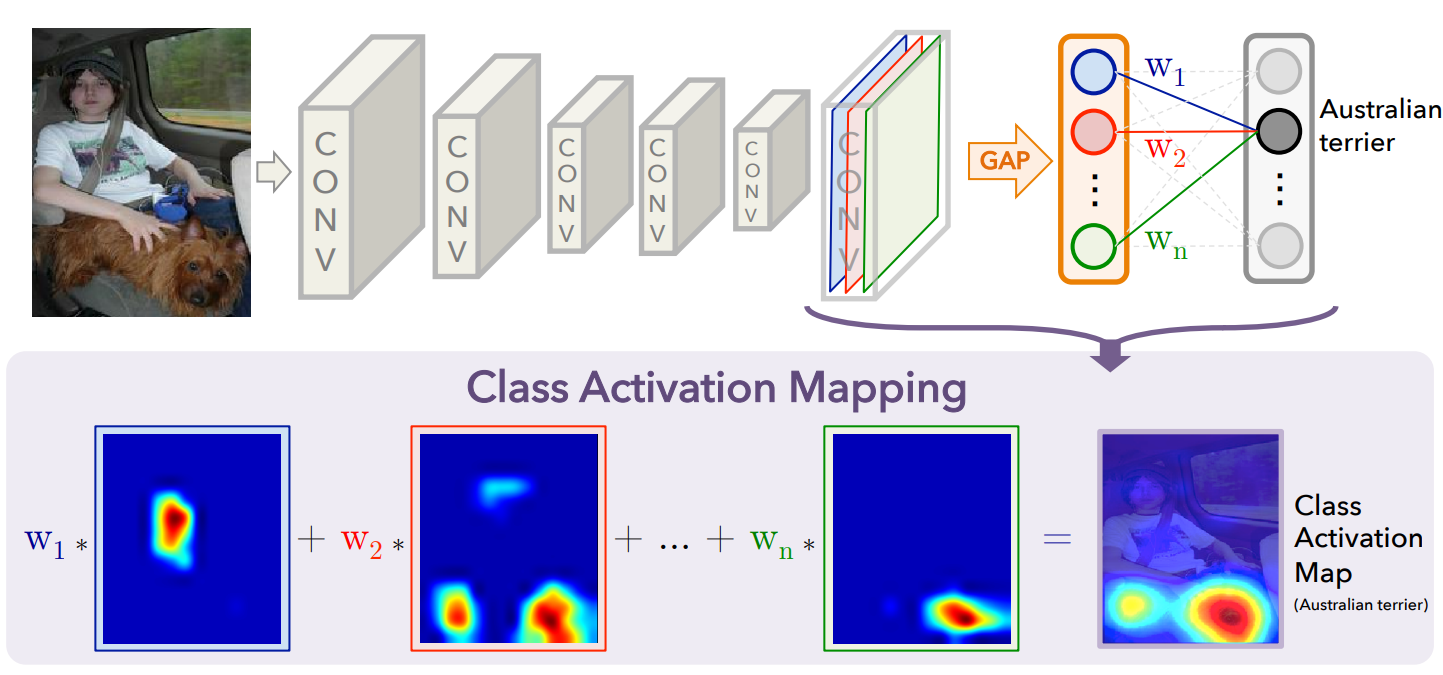


Figure 1: CAM visualization. (Image taken from Bolei Zhou et al., 2015)

The importance of CAM can sometimes help explain some ambiguity in a classifier, especially when the dataset is biased. For example, one classifier could be handling a task of classifying animals, and one of the classes is “polar bear”. It is highly possible that all of the “polar bear” images­ are taken in a snow-white scenery because it is unlikely for them to live in a more tropical environment, classifier could falsely rely on the white scenery for identifying “polar bear” class, which might lead to false classification when there is a white fox in a snow scenery. Therefore, we can analyze the CAM of some “white bear” images to know the model’s CAM is relying on the snow scenery or the bear itself, informing us potential issue with the model in advanced.

Our main objective is to investigate various implementations of CAM technique by using a pretrained VGG16 network and analyze its result critically. Moreover, different implementations will be carried out to generate fine grained CAM to better visualize the region of interest by the classifier. This project is mostly interesting in a particular way, that CAM generation can be done in a weakly supervised manner. This is because it only relies on the image level label where no mask is given. This provides another practical entry where instance segmentation is needed without ground truth masks at hand (Yang Liu et al., 2022; Zhaozheng Chen et al., 2022).

**Domain Knowledge**

Class Activation Mapping (CAM) can generate a mask similar to a heatmap indicating the region of interests. To use CAM, one does not need to create an entirely new model or heavily alter existing architecture. It is normally done by starting with a pretrained classification model then only modify the last linear layer of the model as in Figure 1. If the classifier is multiclass, one can generate CAM on a single class, where if it is a multilabel classifier, it can generate multiple CAMs overlaying on a single image, acting as pseudo-masks in a weakly-supervised manner (Peng-Tao Jiang et al., 2015) (Figure 2).

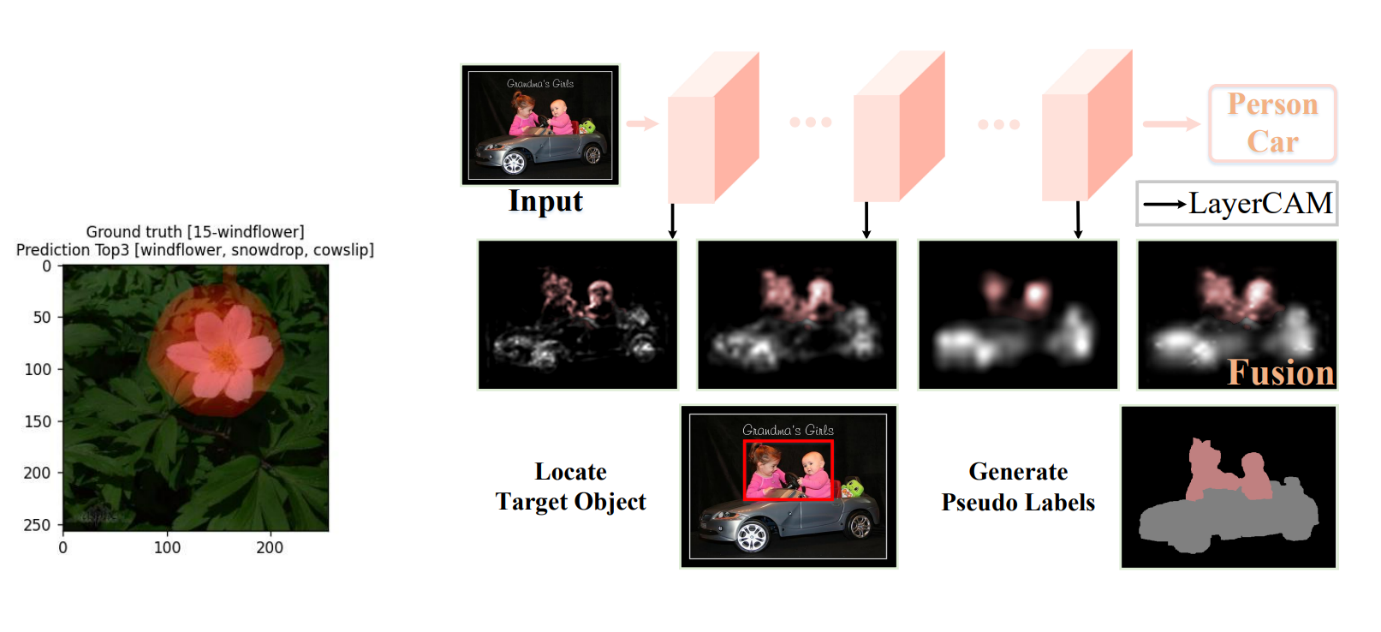
****

Figure 2 Multiclass and Multilabel CAM

The first implementation of CAM does not generate a detailed map, several improvements have been carried since then, either by using feature maps from shallow layers or rethink the weight coefficients.

# Justification and Review on Existing Models

A huge reason for selecting Class Activation Mapping (CAM) technique for this report is from the curiosity of understanding the rationale behind a working classifier. Modern state-of-the-art image classifiers that validate on the ImageNet Dataset (Jia Deng et al., 2009) are usually very heavy and huge, around 200 million parameters. While certain effort has been spending on keeping the model small (model distillation, pruning and quantization), understanding the decision-making process of a classifier is also equally important especially in the medical field.

The original CAM implementation (Bolei Zhou et al., 2015) simply modifies the last layer of a classification model to include a Global Average Pooling (GAP) layer before feeding the neurons into the last linear layer. Strictly speaking, a classification model can be viewed as a concatenation of a feature extractor and a direct classifier. Given an image of shape , let be its feature map of shape (we assume the feature maps are squares). Since we have discarded the original classifier from the model, we design a new end classifier as follows: A GAP layer that average the values from each map hence giving an output of shape , assuming this output is written as and there are classes in the dataset, we feed it through a linear layer to obtain an output of shape (Figure 3). This output can then be softmaxed and performed other operations normally for a classification model. Also note that GAP is a function for pure operation, it doesn’t have any trainable parameters.

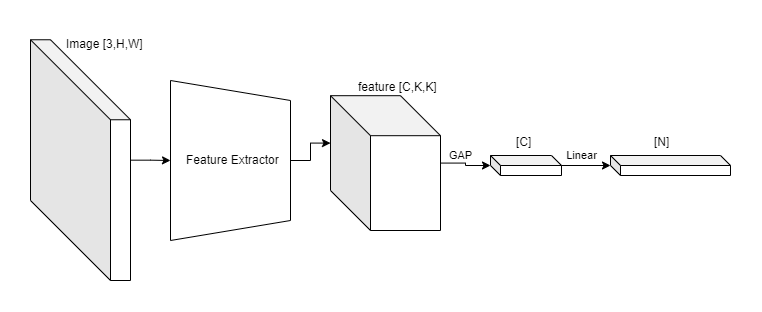


Figure 3 CAM architecture

At first glance, this is just a standard image classification model, but after the training is completed, it can be used to generate CAM. The last linear layer has weight and bias parameters that are of shape and . By skipping the GAP layer, one can pass the feature map of an image directly to the last linear layer to obtain a collection of CAMs with shape . Then, resizing the CAMs to constitutes as a rough activation map for each of classes.

The role of Global Average Pooling layer in CAM is subtle: It merely average the neuron activations from the feature map. While requiring no trainable parameters, it suits perfectly well with the objective. Since the model’s last linear layer is trained on the GAP output from the feature map, it reacts well to the original feature map as well. This is why CAM can be generated by skipping GAP layer, letting us observe the regions activating the decision (Figure 4).

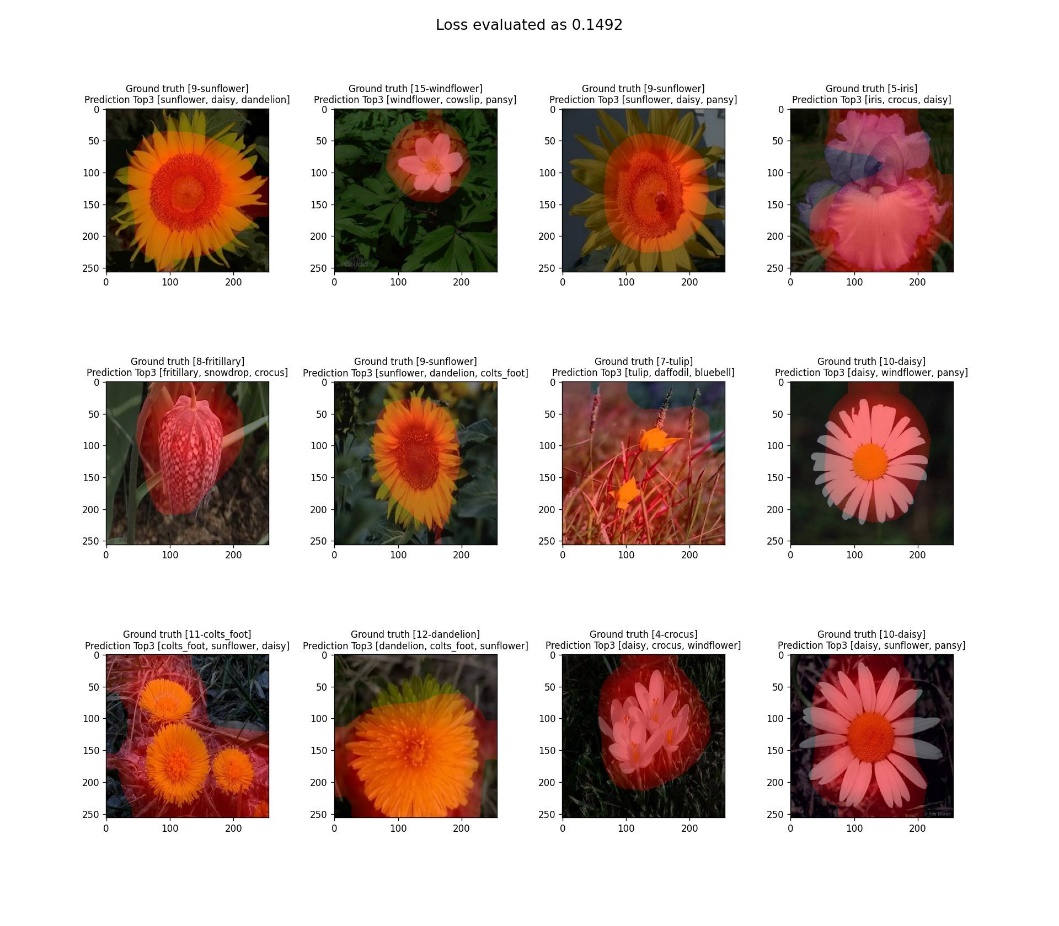


Figure 4 Original CAM Implementation

However, the approach of CAM generates blurry mappings because the feature maps has very small dimensions, the last linear layer’s weight is heuristically used in the process, and the generated CAM might include false positive regions (for example, the 7th image from (Figure 4)). To remedy the situation, Zhaozheng et al. (2022) created ReCAM that aims to reactivate class activation mapping in order to reduce false positive. Designed originally for multilabel model, the model is trained to first generate a set of CAMs for the multi-hot classes on an image. Then, normalized CAMs are used to mask over the original feature maps before generating another set of CAMs, which is called ReCAMs (Figure 5). The authors argue that this approach can make the CAM focus on a more prominent features of the image since the importance was highlighted by the first set of CAMs.

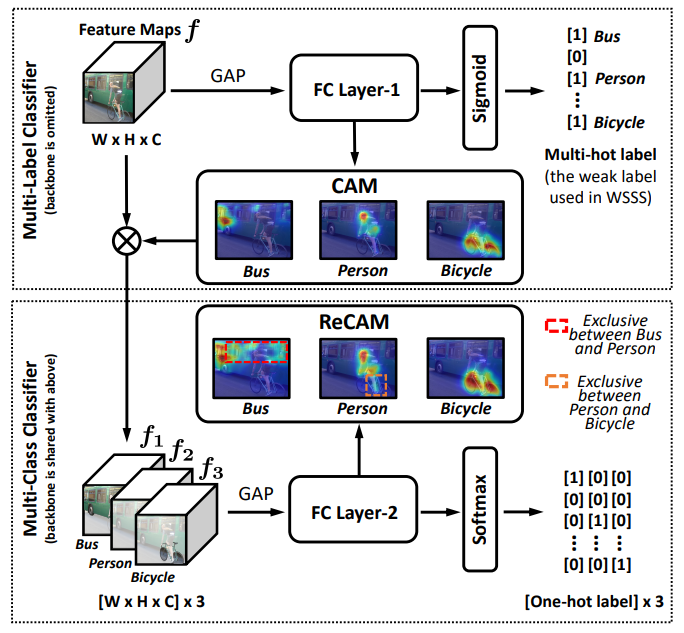


Figure 5 ReCAM architecture using a shared feature extractor, taken from (Zhaozheng et al., 2022). The masking effect can be seen from the processed feature maps on the lower-left corner.

However, ReCAM still uses the same choice of weight parameters from the last linear layer, it is heuristic and assumes the weight suitable for the output of global average pooling layer is also suitable for the features itself. There are more rigorous approaches providing finer details on the generation of CAMs.

Recent innovation of gradient-based CAM models, such as Grad-CAM and Grad-CAM++ achieved far better visual results compared to the previous discussed approaches (Ramprasaath et al., 2017; Aditya et al., 2018). In the framework of Grad-CAM, classification from an image classifier can generate gradients from the end to the start of the model. On the feature extractor level, the gradient is inspected and normalized (usually by a tangent hyperbolic function on nonnegative values) to obtain maps ranging from 0 to 1, thus constructed CAM in this way. The design is based off the believe that model will make a decision based on the more influential neurons, hence assign higher relevance to the neurons with higher gradients in magnitude. The Grad-CAM++ framework increases this sensitivity by including second order gradients during the calculation, but it inevitably makes the computation more expensive.

LayerCAM is yet another gradient-based CAM model which utilizes low-level and high-level feature maps to generate realistic CAM (Peng-Tao et al., 2015). It is based on a pretrained VGG16 model, consisting of 5 main convolutional blocks, each generate a feature regarding different level of details. Using the 3rd, 4th and 5th features from the VGG16 model, the authors generate a CAM in a hybrid manner that outperforms the previous two models (Figure 6). Although it is only being done on VGG16, its core implementation can be done on other standard architectures as well.

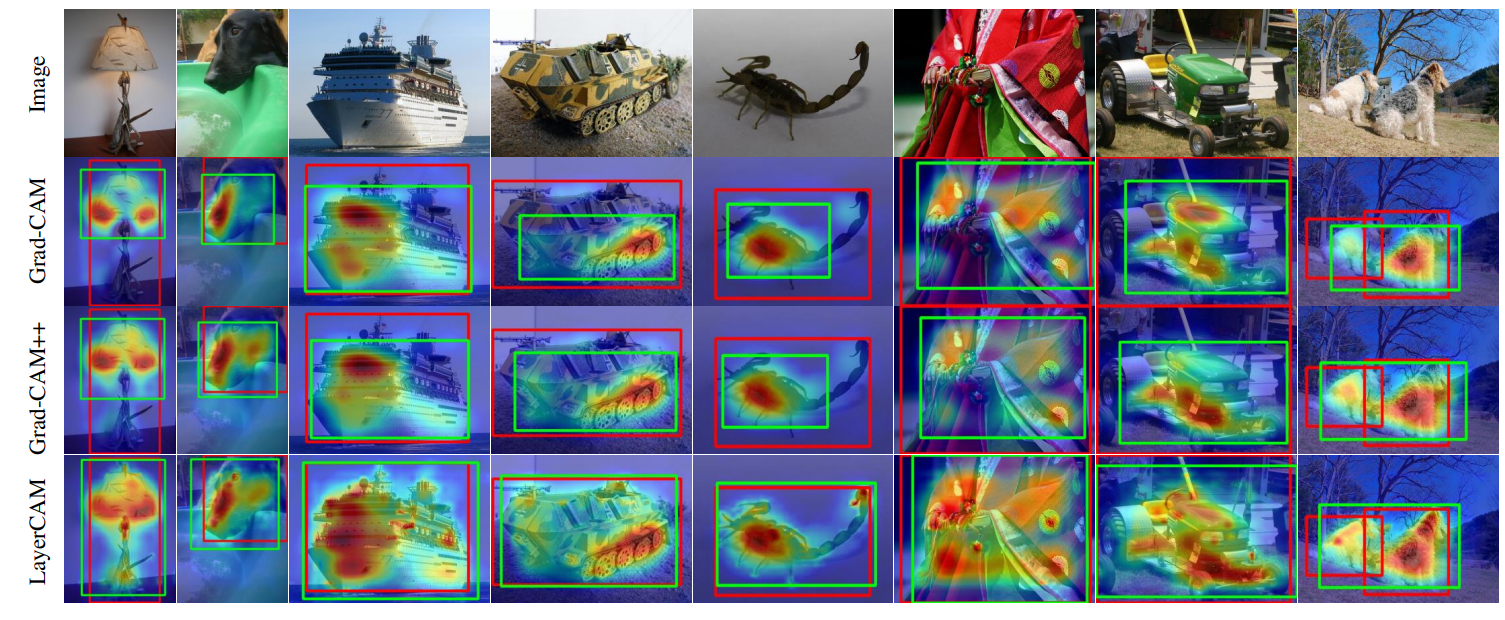


Figure 6 LayerCAM compared to Grad-CAM and Grad-CAM++, taken from (Peng-Tao et al., 2015).

In our report, we will implement the original CAM model, together with the ReCAM and LayerCAM variant. Finally, we compare their results visually and eventually conclude the best model out of the three on the Flower-17 dataset.

# Implementation

We implemented CAM, ReCAM and LayerCAM using Python with PyTorch as our deep learning framework. The feature extractor of a pretrained VGG16 are used in this implementation just like the LayerCAM implementation (Peng-Tao et al., 2015), we merely change the last classifier to include a global average pooling layer. CAM and LayerCAM model use the same underlying architecture, the only difference of them is the methods of generating CAMs. On the other hand, ReCAM uses a slightly different architecture change as ReCAM is generated based on CAM. Therefore, we only implement models for CAM and ReCAM, while writing CAM generations as needed in the class methods. During training, we freeze the weights from the pretrained VGG16 feature extractor, while only train the last and customized part of the model.

We select the Flower-17 dataset from Kaggle (Sani, 2021), it contains only 1360 images, within which has 80 images for each class. To make the image classification more robust, we perform offline augmentation on the dataset by a composition of random rotate from -20 to 20 degrees, random shift of 20% horizontally and vertically, and a 50% probability of horizontal flips. This increases our dataset to 13,600 images. We use the first out of three train and validation splits as given from the Kaggle website.

During training on the Flower-17 dataset, we use cross entropy loss function from PyTorch to track the model performance and save the model if there are improvements from the last best validation loss. We can afford to save a model for each epoch since only the last layer’s weights need to be saved (around 37KB). Moreover, we train the model for 40 epochs using learning rate ranging from 0.00001 to 0.0002 with Adam optimizer. As a standard evaluation of image classification model, we calculate and report the Top1, Top3 and Top5 accuracies on the Flower-17 validation dataset and use it as a guide to choose the best model for CAM generation. This is because we notice a heavy overfitting starting from epoch 20 and it only becomes more serious until epoch 40, hence we figure the best model choosing strategy is to base on the top-k accuracies on validation set.

When generating CAM (or ReCAM) from a model, we often uses the feature maps or the gradients of it, but its values are real values or nonnegative (if ReLU is applied). To make its value suitable for visualization, we tested different normalization methods. Minmax method is to take the minimum and maximum values of the map, then use it to scale the map into [0,1]. Sigmoid is another method that simply maps any number (negative or positive) to [0,1]. We also have a hybrid ReLU method that first apply ReLU to the map, then divides it by the maximum value so that final values are clipped into [0,1]. Out of the three methods, it is clear that ReLU might lead to loss of information. The three methods are generally only applied to CAM and ReCAM model, while LayerCAM model uses another new method since it inherently uses gradient-based method to generate CAM. With gradient at hand, it can be negative or non-negative. We use the standard approach as in GradCAM and LayerCAM (Peng-Tao et al., 2015; Ramprasaath et al., 2017) that first apply ReLU to the values of gradients so that only positive gradients are kept, divides it by its maximum value, double it, then apply tangent hyperbolic function to obtains a normalized values as in range of [0,1]. The formula is as below (Peng-Tao et al., 2015):

The use of tangent hyperbolic function is to magnify small but positive activation, makes the effect more salient on our eyes.

We experiment with hyperparameter and model tuning. We create 4 CAM models respectively by training for 40 epochs with learning rate 0.00001, 0.00005, 0.0001, and 0.0002. Since the normalization method for creating CAMs is only applied during inference, we do not need to training CAM models separately for different normalization methods. However, ReCAM requires creating CAM during training, hence we trained 12 ReCAM models respectively similarly as above, but including the three normalization methods (Minmax, Sigmoid and ReLU hybrid) into the grid. For LayerCAM, we just reuse the 4 CAM models since the gradient-based method can also be applied during inference stage. We report our results onto the next section.

# Discussion

## Classification Metrics

Below shows the training result on the Flower-17 dataset (Table 1). We report on validation dataset, the best top-1 accuracy and the other corresponding top-k accuracies and cross entropy loss based on that epoch. To avoid taking results from overfitted models, we discard any models that have training loss less than half of validation loss. Again, we should emphasize that CAM only uses normalizing method during inference, so there is only one set of CAM models. On the other hand, ReCAM may use different normalizing methods during training process, hence model weights will be affected by the chosen normalizing method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model/LR | 1e-5 | 5e-5 | 1e-4 | 2e-4 |
| CAM | (epoch 40)  Top1: 81.47%  Top3: 96.18%  Top5: 99.12%  Loss: 1.003636 | (epoch 37)  Top1: 92.65%  Top3: 98.53%  **Top5: 99.71%**  Loss: 0.288743 | (epoch 22)  **Top1: 92.94%**  **Top3: 99.12%**  **Top5: 99.71%**  **Loss: 0.271448** | (epoch 11)  Top1: 92.06%  Top3: 98.82%  Top5: 99.41%  Loss: 0.282351 |
| ReCAM (minmax) | (epoch 40)  Top1: 77.06%  Top3: 91.18%  Top5: 96.18%  Loss: 1.367373 | (epoch 39)  Top1: 87.65%  Top3: 96.76%  Top5: 99.12%  Loss: 0.457615 | (epoch 25)  Top1: 89.41%  Top3: 97.35%  **Top5: 99.41%**  **Loss: 0.349892** | (epoch 12)  **Top1: 90.29%**  **Top3: 98.53%**  Top5: 99.12%  Loss: 0.370260 |
| ReCAM (sigmoid) | (epoch 40)  Top1: 79.12%  Top3: 92.94%  Top5: 97.65%  Loss: 1.052183 | (epoch 40)  **Top1: 90.59%**  **Top3: 99.12%**  **Top5: 100%**  **Loss: 0.289081** | (epoch 18)  Top1: 88.53%  Top3: 97.94%  Top5: 99.12%  Loss: 0.373524 | (epoch 9)  Top1: 88.24%  Top3: 97.35%  Top5: 99.12%  Loss: 0.366691 |
| ReCAM (relu hybrid) | (epoch 40)  Top1: 75.88%  Top3: 89.12%  Top5: 94.41%  Loss: 1.457482 | (epoch 39)  Top1: 88.82%  Top3: 97.65%  **Top5: 99.12%**  Loss: 0.381621 | (epoch 24)  **Top1: 90.00%**  **Top3: 98.24%**  Top5: 98.82%  Loss: 0.365495 | (epoch 14)  Top1: 89.71%  Top3: 97.94%  Top5: 98.82%  **Loss: 0.341185** |

Table 1 Training results

By evaluating the models solely as classifiers, we notice a learning rate of 1e-5 trains too slowly, while a learning rate of 2e-4 seems to lead to mildly unstable result. The best of our models roughly lies on those which uses a learning rate of 5e-5 or 1e-4. After training these models, we notice a consistent trend that the vanilla CAM models always outperform the ReCAM models, regardless of normalizing methods, in which we will discuss the reason soon. Nonetheless, out of the three normalizing methods for training ReCAM models, the sigmoid method seems to perform slightly better than the other two, we believe it is because sigmoid maps values to the range [0,1] consistently unlike minmax method which its range might heavily affected by the raw minimum and maximum values. Moreover, ReLU hybrid method perform slightly worse than sigmoid mostly because it discards some useful information (negative raw values) hence hinder the classification power.

## CAM generations

We compare the generated CAMs from vanilla CAM, ReCAM and LayerCAM models.

# Conclusion

# References