Scene Classification and Grad-CAM Visualization

Olivia Qu, Shiyue Wang, Wenlu Dong, Yuqi Wang

December 2022

https://github.com/oliviaqu/COM576.

Contents

1 Introduction

1.1 Motivation

Nowadays, Microsoft AI For Earth, has created the most detailed United states forest map using satellite imagery and AI, which would essentially be a game changer in reducing deforestation, pests and wildfires. Based on this, We wish to use a deep learning model which will be train based on Convolutional Neural Networks (CNNs) and Residual Blocks to detect the type of scenery in images. This project could be practically used for detecting the type of scenery from the satellite images. In addition, this project will cover the use of a technique known as Grad-Cam to observe and explain how AI models think. In detail,How can we understand what our convolutional neural network (black box) sees and understands when making a decision?

1.2 Dataset

We find a dataset which contains 14034 training and 3000 test images with highly accurate, detailed and consistent annotations from a wide range of natural scenes from all around world. The dataset have 6 categories: street, sea, buildings, forest, mountain, glacier.

2 Data Visualization

Firstly, We'd like to take a look of our dataset. We use Python to choose 5 images from every single categories. After that, we want to make sure that if we have a balanced dataset. So as for the pie chart, we can see that the 6 categories have similar amount of the images which is good for our future analysis.

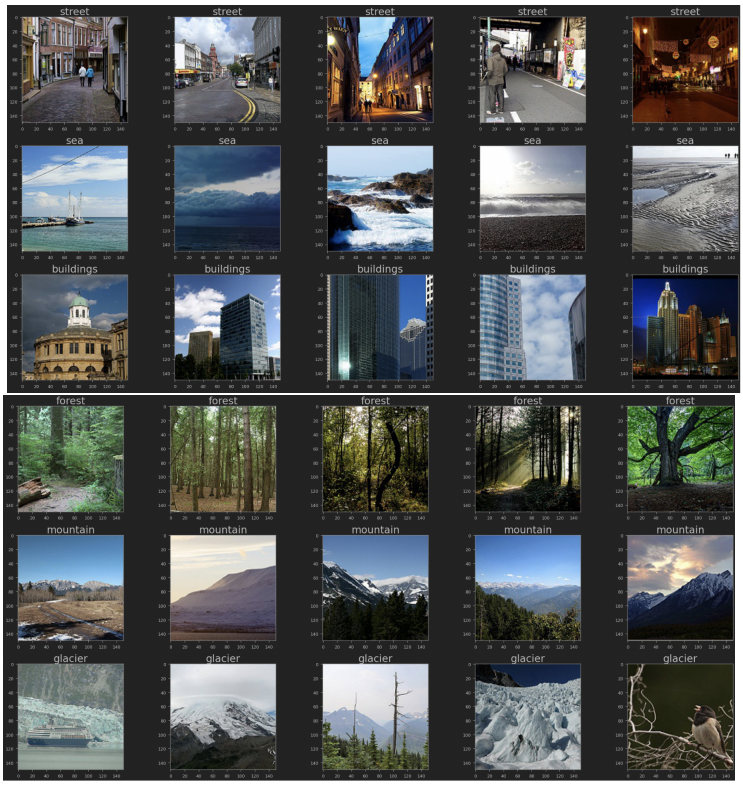

Figure 1: Classified Images

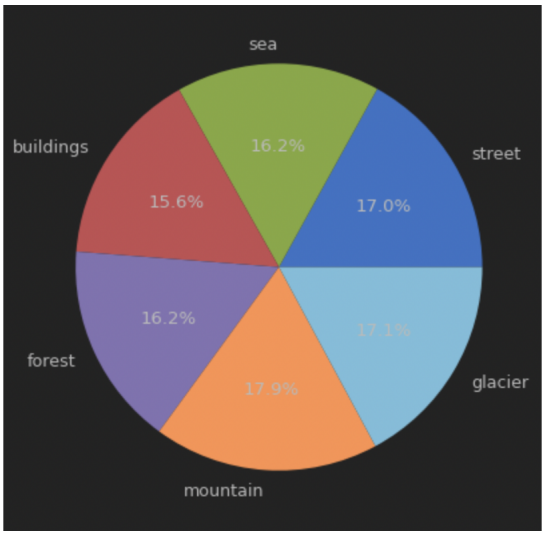

Figure 2: Pie chart of different categories

3 Deep Learning

3.1 Convolutional Neutral Networks

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other[1].

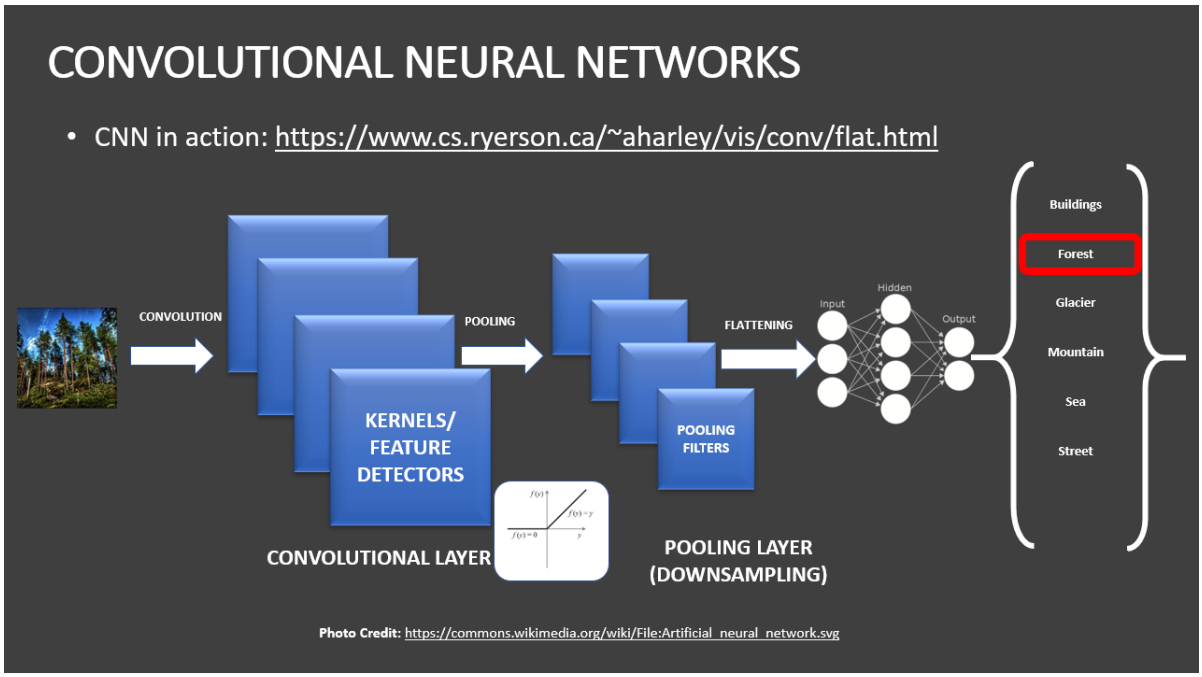

Figure 3: Convolutional Neutral Networks

We start with a picture and then add numerous layers of feature extraction. So we have kernels, or feature detectors, in this case. And what these kernels do is they take the picture, scan it, and extract characteristics from it. And I'll have many feature detectors in here. That is the convolution layer that we apply after that. This will be the first layer. Following that, we often do what is known as pooling or downsampling. And the goal here is to take all of these feature maps and then decrease their size via pooling or downsampling. Then we flatten all of these feature maps, transform them to a 1D array format, and send them to a dense, fully connected artificial neural network to conduct the classification.

Convolutional Neural networks (CNN) are very useful when it comes to visual input analysis. But when it comes to training these networks, there are certain problems that are present. As we keep on increasing the number of layers of networks, it becomes more and more difficult to train them. That's called the vanishing gradient problem encountered while training artificial neural networks involving gradient-based learning and backpropagation. We already know that in backpropagation, gradients are used to update the weights in a network. However, occasionally the gradient becomes vanishingly tiny, thus preventing the weights from changing values. This causes the network to cease training since the same values are transmitted over and over again, resulting in no productive work being done. That's why we will overcome this issue using what we call ResNet, or residual neural networks.

3.2 Residual Neutral Networks

3.2.1 Improvement

This design introduces the notion of Residual Blocks to overcome the issue of the vanishing/exploding gradient. In this network, we use a method known as skip connections. The skip connection links layer activations to subsequent levels by skipping some layers in between. This results in the formation of a residual block. These leftover blocks are stacked together to form ResNets.

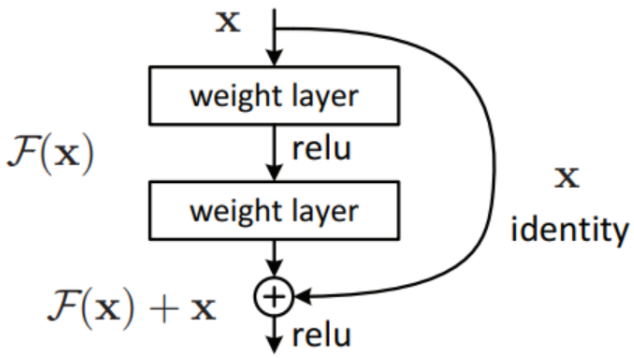

Figure 4: skip connection

From the above figure a basic residual function can be summarized as follows: If x is the input and F(x) is the output from the layer, then the output of the residual block can be given as Y = F(x)+x

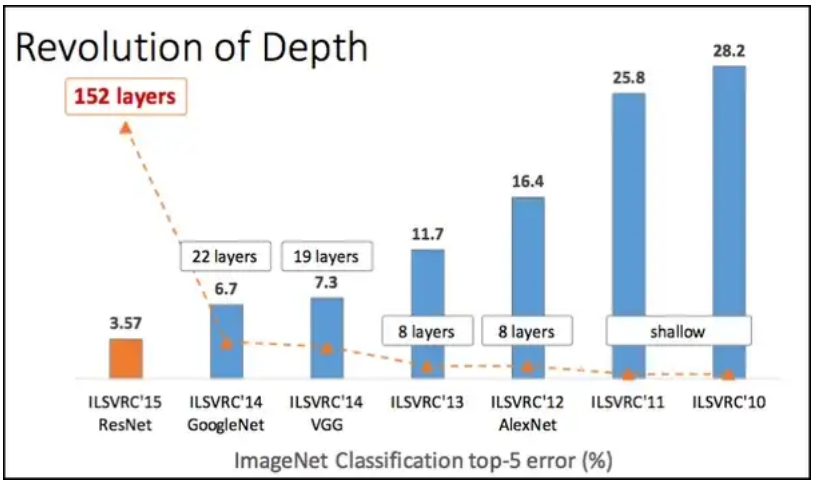

Figure 5: Revolution of depth

Here we have a summary, that shows the performance of various networks on the ImageNet dataset[2]. The ResNet error is 0.0357. And this error is actually pretty incredible. ResNet's actual performance is quite superior compared to all the other networks.

3.2.2 Res-Block

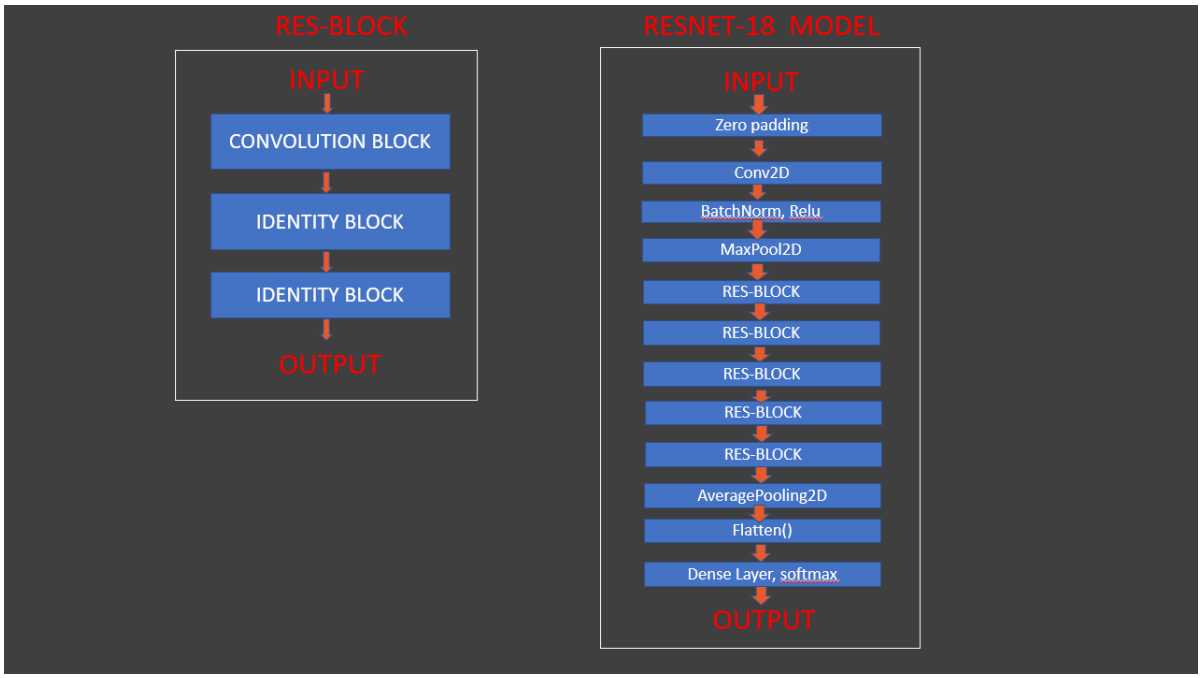

Figure 6: Res-Block

So this is the model from a very high level. We are going to have our input. We have Conv2D, Batch Normalization, Max pooling 2D and then we're going to have a series of rest blocks. And all of these rest blocks are represented here. The rest block is represented here as a convolution, followed by two identities.

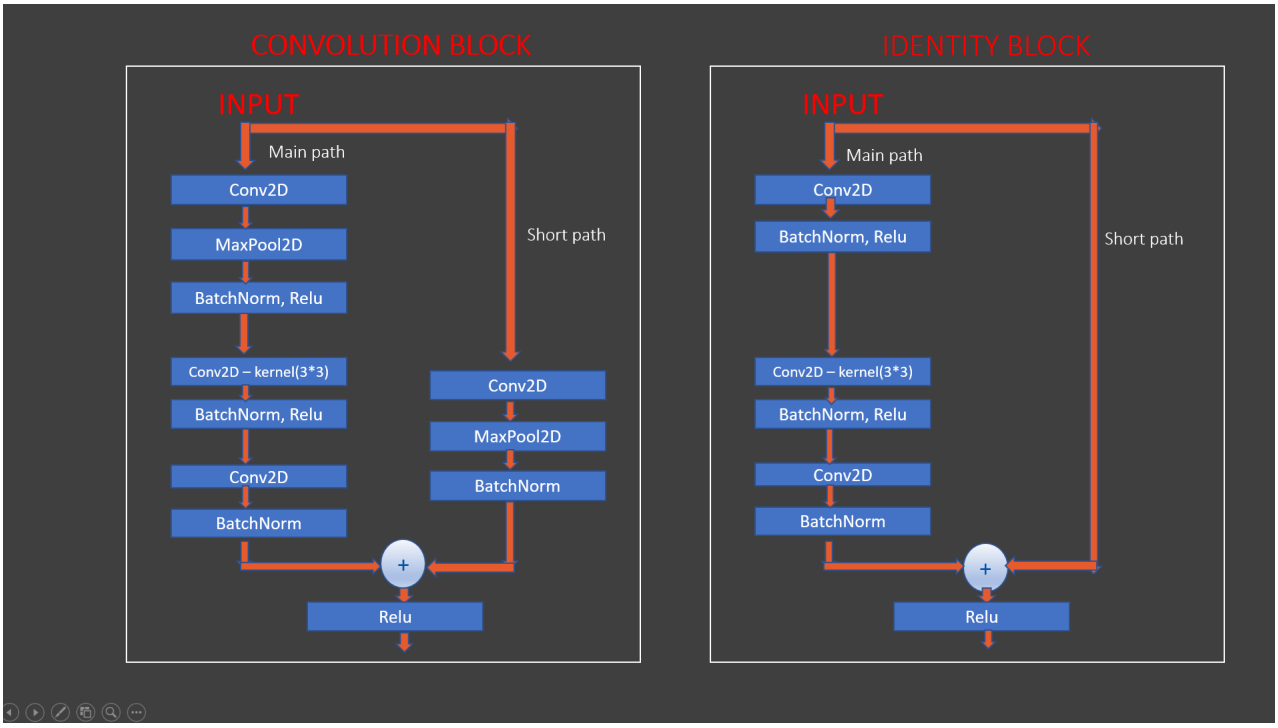

Figure 7: Convolution Block and Identity Block

So within the convolution block, I'm gonna take the input and apply it to Conv2d, which as we discussed before, is followed by Max pooling, batch normalization, Ralu function. And they have another batch normalization, and then Conv2D, and the Batch normalization. That will be my main path.

And for short path, they're gonna have to be max pulling 2D and batch normalization. And then we sum them up and apply them to renew the activation function at the end. So that will be every single convolution. We called it the convolution block.

So identity block from its name, we're going to take the input, and feed it along to a series of convolutions and batch normalization. And because it's called identity block, we're going to have only one identity. The short path will be fed. So the input will feed it as is. And will add it to whatever comes from this path and then we are going to sum them up and then apply the Relu activation function at the end.

3.3 Compile the Model

After construct our main network, we compile the model and train it. Here we use Adam optimizer here. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data[3]. The results of the Adam optimizer are generally better than every other optimization algorithms, have faster computation time, and require fewer parameters for tuning. Because of all that, Adam is recommended as the default optimizer for most of the applications[4]. There are 6 categories in our data, so, the loss type is "categorical cross-entropy".

Of course, we do not want to over-fitting the model, so, we set an early-stopping, it set patience to 15, which means if the error does not improve for 15 epochs, we are going to perform early stopping or just exit the training. We saved the best model with lower validation loss.

Then we fit the generator method on our model and feed it along the train generator. Next, we will show some outputs of the model.

3.4 Assess the Model

We firstly evaluate the accuracy of our model. The model is pretty good with 86.59 % accuracy, which means most pictures can be predicted precisely. Then we are going to load the images and their predictions. Here, we opened the image and resized the images to 256 by 256. Then we convert the image to array and normalize the image and reshaping it into 4D array. Here are some pictures of the prediction.

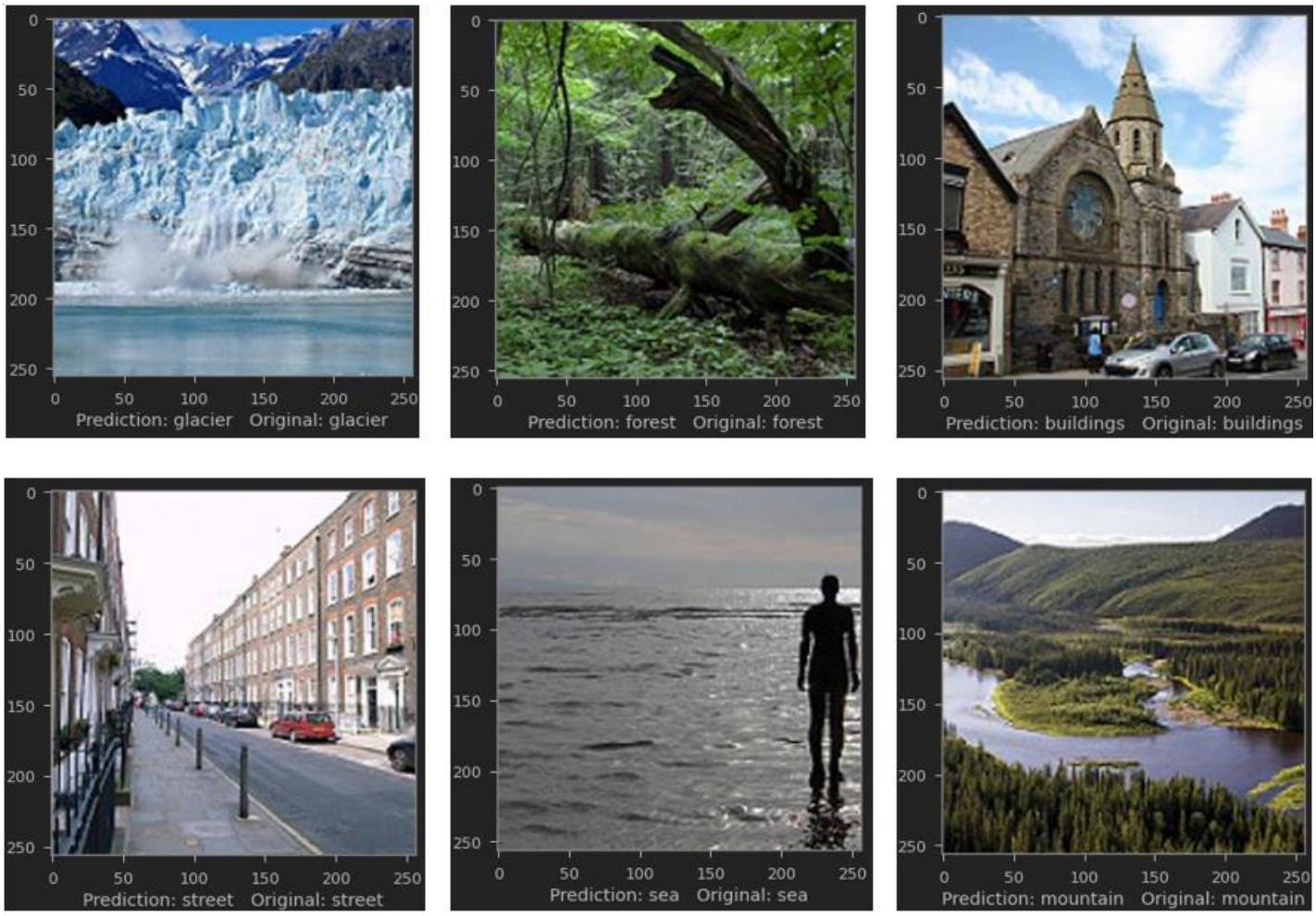

Figure 8: Correct predictions

There are some examples that the model predict the picture totally wrong(shown as following).

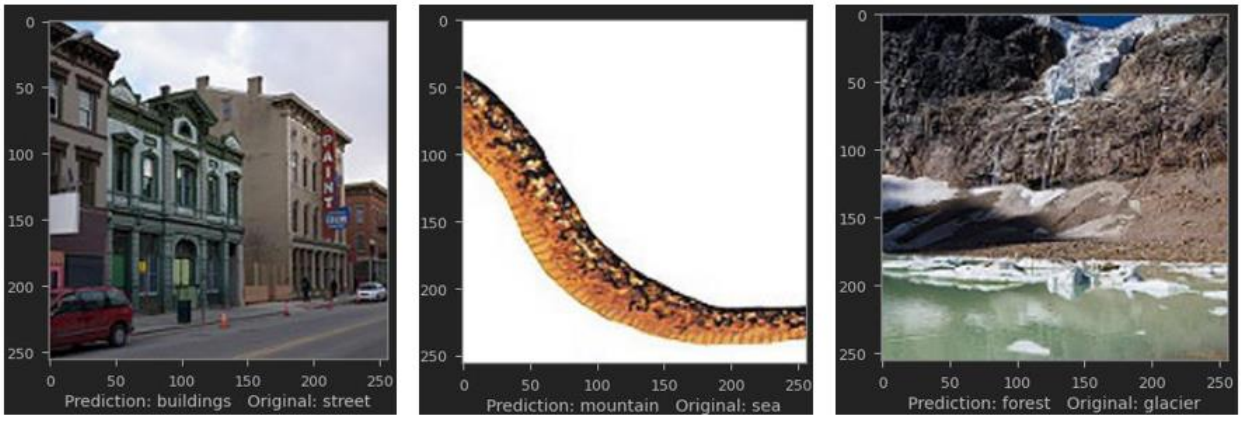

Figure 9: Wrong predictions

But for most pictures, the model can predict it correctly. There are some examples that do not have correct output but still make sense. Here are some examples.

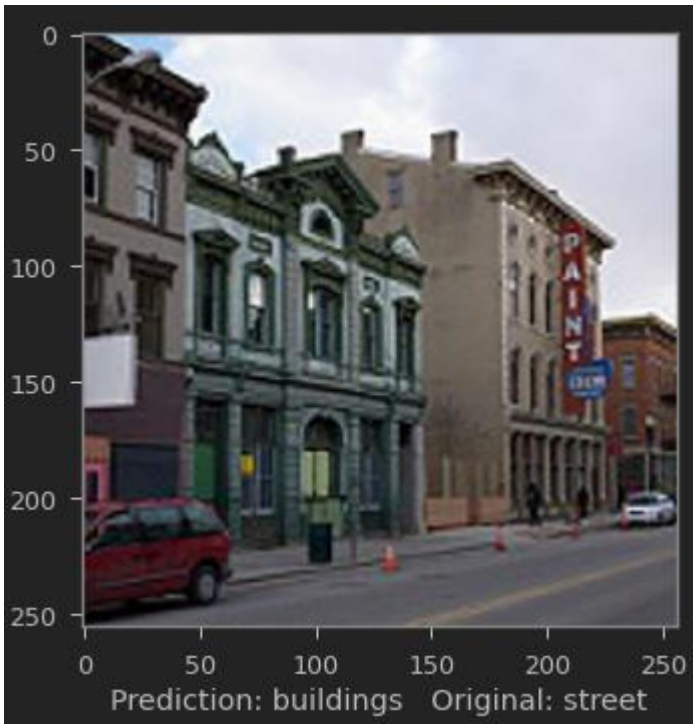

Figure 10: A wrong prediction with explanation

The above picture shows that the original tag is street, but the prediction output is building. If we look at the picture, we can figure out why this happened. You can see there is a street but there are also many buildings in it. So, if we write tag of these pictures, different people may attach different tags on it. So, for our model, it's fine.

We also check the confusion matrix. 0-5 stands for different categories.

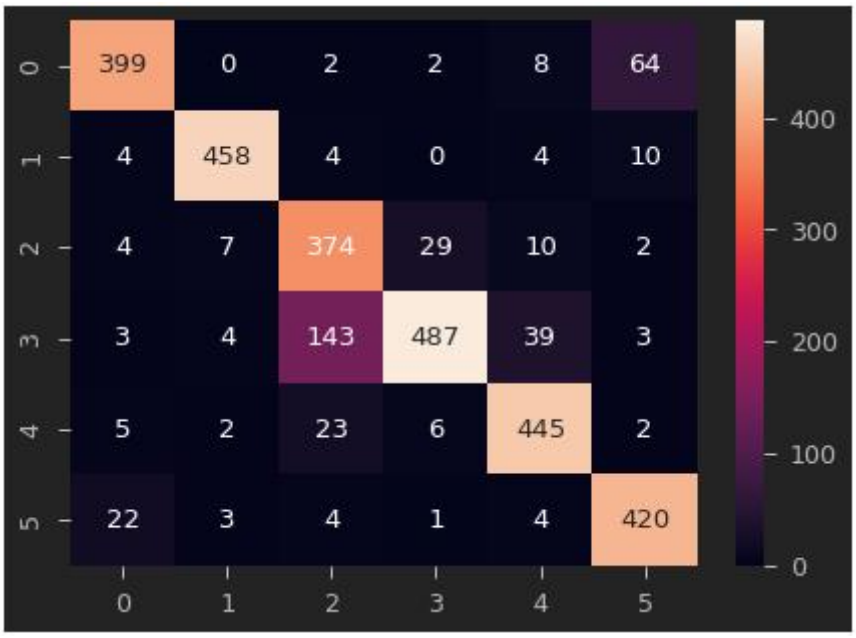

Figure 11: Confusion Matrix

	precision	recall	f1-score	support
buildings	0.91	0.84	0.87	475
forest	0.97	0.95	0.96	480
glacier	0.68	0.88	0.77	426
mountain	0.93	0.72	0.81	679
sea	0.87	0.92	0.90	483
street	0.84	0.93	0.88	454
accuracy			0.86	2997
macro avg	0.87	0.87	0.86	2997
weighted avg	0.87	0.86	0.86	2997

Figure 12: Precision for each category

The above table shows our conclusion derived before is proved.

3.5 Grad-CAM Visualization

3.5.1 Definition

As we said in the motivation part, How can we understand what our CNN (black box) sees and understands when making a decision? We introduce the grad-cam visualization[5]:

Gradient-Weighted Class Activation Mapping (Grad-CAM) helps visualize the regions of the input that contributed towards making prediction by the model.

It does so by using the class-specific gradient information flowing into the final convolution layers of CNN to localize the important regions in the image that resulted in predicting that particular class.

3.5.2 Steps

1) To visualize the activation maps, we pass the image through the model to make the prediction.

2) Use argmax to find the index corresponding to the maximum value in the prediction, which gives the predicted class.

3) Then, calculate the gradient that is used to arrive at that value with respect to feature map activations A^k of the convolution layer. We use using tensorflow.GradientTape() to get the value of gradients.

4) We multiply the gradients obtained with the filter values in the last convolutional layer, since we want to enhance the filter values that contributed towards making this prediction and lower the filter value that did not contribute towards the result.

5) Then, we perform weighted combination of activation maps and follow it by a ReLU to obtain the heatmap.

$$L^c_{Grad-CAM} = ReLU(\sum_k a^c_k A^k)$$

6) Finally, we super-impose the feature heatmap on the original image to see the activation locations in the image.

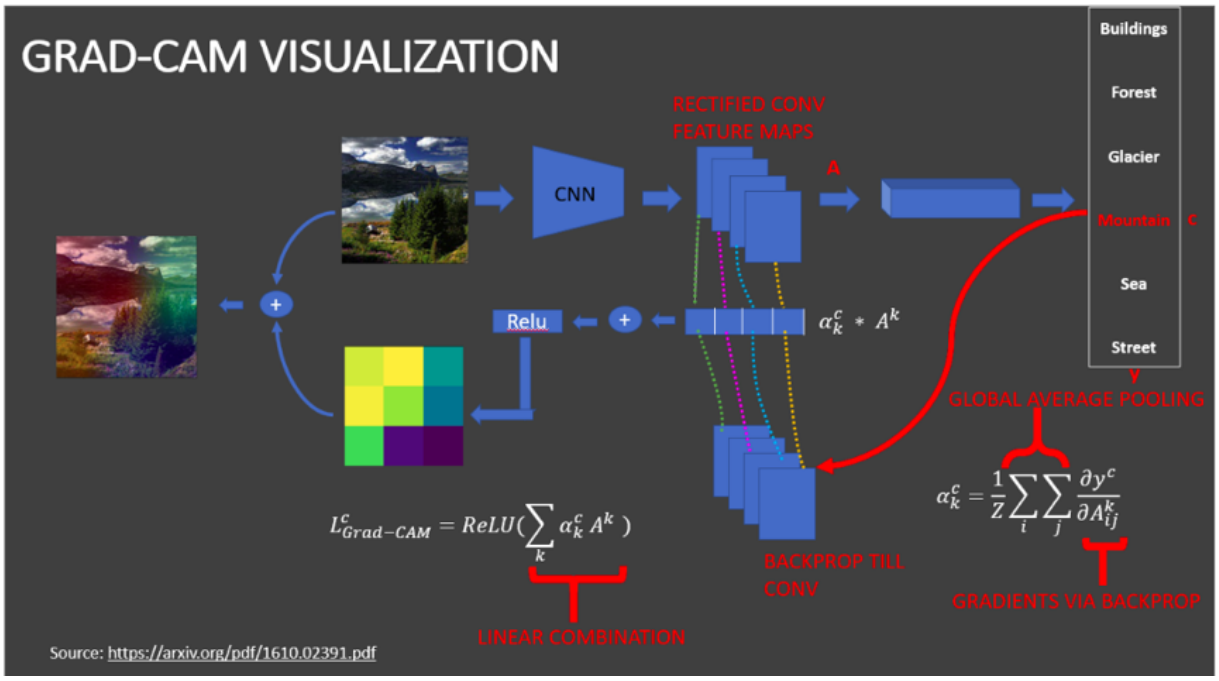

Figure 13: Grad-CAM Visualization

In short, It has two main parts, the first step is fit the image pass through our CNN to generate activation maps and use argmax to find the index corresponding to the maximum value in the prediction. The second part is derive gradient which is used to arrive at the predicted value, and multiplied by the activation map from the last layer.(enhance the filter values that contributed towards making the prediction and lower the filter values that did not contribute towards to result).Then sum up and apply the Relu function to obtain the heatmap. And when we get the heapmap,like the left figure shows, we super impose the heapmap on top of the original image to tell which part of image particularly contribute to the prediction.

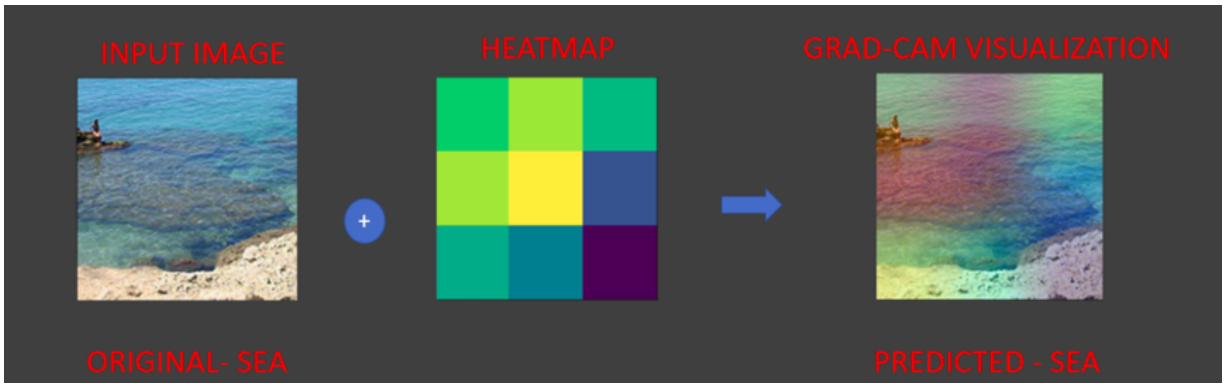

Figure 14: Grad-CAM Application

4 Result of Grad-CAM Visualization

4.1 Analysis of the Results

Based on the heatmap scaled on the top of the original images, the difference change in color shows the regions which model focus on to make the prediction. We random selected three examples. please look at the plots below, from the third images on the first and third rows, the lower right areas are been focused on, and give the sea and building prediction respectively which are the right classification. While for the third plot on the second row, we can see that the importance is also given the lower right area and the prediction is given as the mountain which was wrong because the right classification should be glacier.

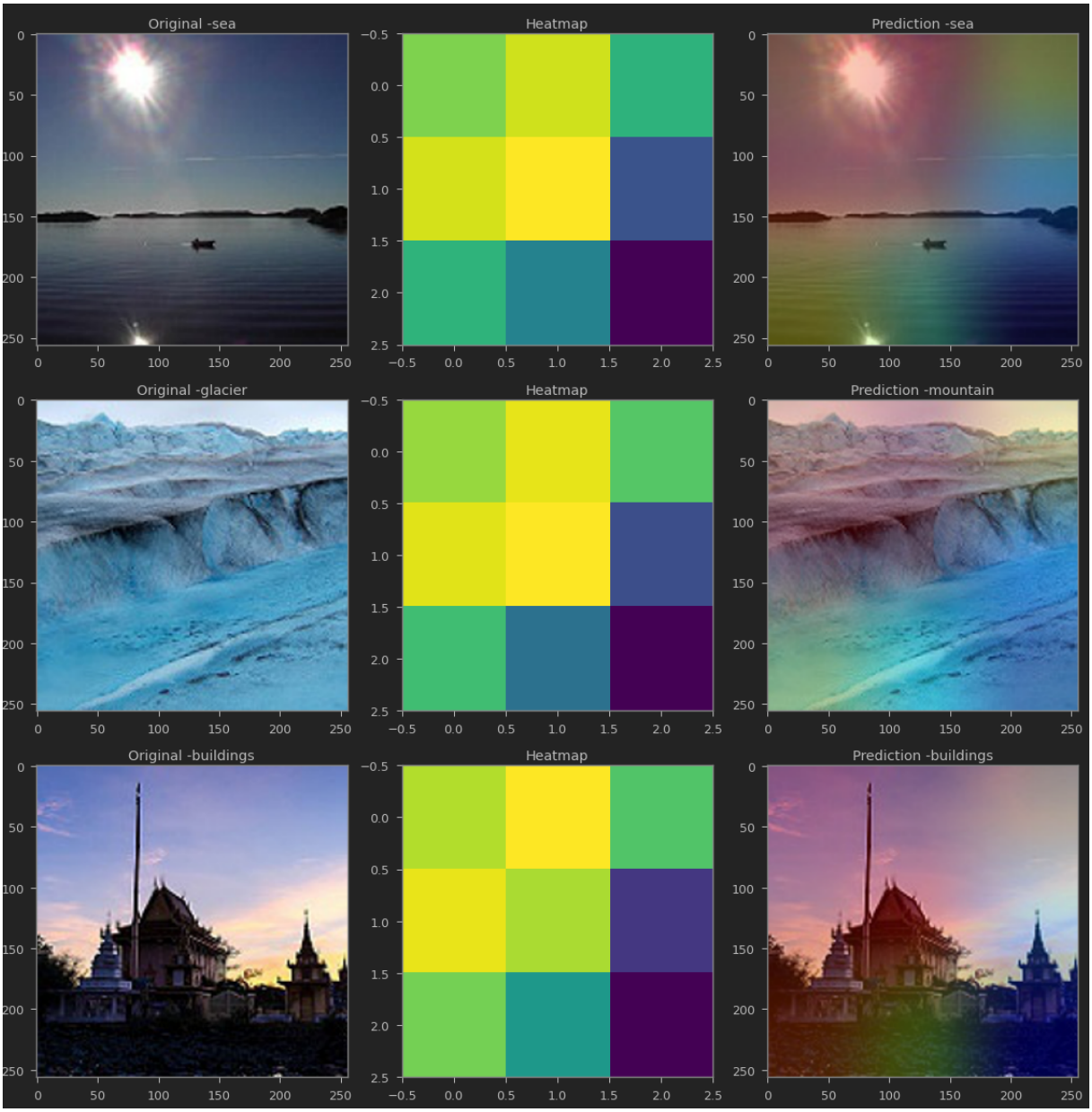

Figure 15: Three examples of the results of Grad-CAM Visualization

4.2 Improvement

How can we make the Grad-CAM heatmap more precise?

In the case, we only use the last layer to derive gradients and get the heatmap. We can incorporate earlier layers, to get an averaged together Grad-CAM heatmaps from all model layers. Because the earlier layers are not useless, they just focus on different part. For example, the earlier layers could detect contours and borders of images and depthwise layers de-emphasize concepts in the image.

5 Conclusion

1) The model performs pretty good with 86.59% accuracy. The confusion matrix also shows this result. Among 6 categories, 'forest' got the highest precision(97%) and glacier got the lowest precision(68%). The glacier category could be further improve by tuning the model or more experimentation in the image augmentation.

2) Gad-CAM can really show how the machine thinks by showing which regions of the image the model focus on. We can use the Grad-CAM heatmap to give us clues into why the model had trouble making the correct classification.

References

[1] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed 12/13/2022.

[2] Priya Dwivedi. *Understanding and Coding a ResNet in Keras*. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. Accessed 12/13/2022.

[3] Jason Brownlee. *machinelearningmastery*. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. Accessed 12/13/2022.

[4] Ayush Gupta. *analyticsvidhya*. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learningoptimizers/. Accessed 12/13/2022.

[5] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7. URL: https://doi.org/10.1007%2Fs11263-019-01228-7.