COMP90086 - Computer Vision, Assignment 2

Student Name: Zixin Hao

Student ID: 1309180

Tutor Name: Jiayang Ao

Tutorial: Tutorial 1

Q1 - CNN training

1.1 CNN Architecture

CNN Architectur is shown in the follow Figure 1.

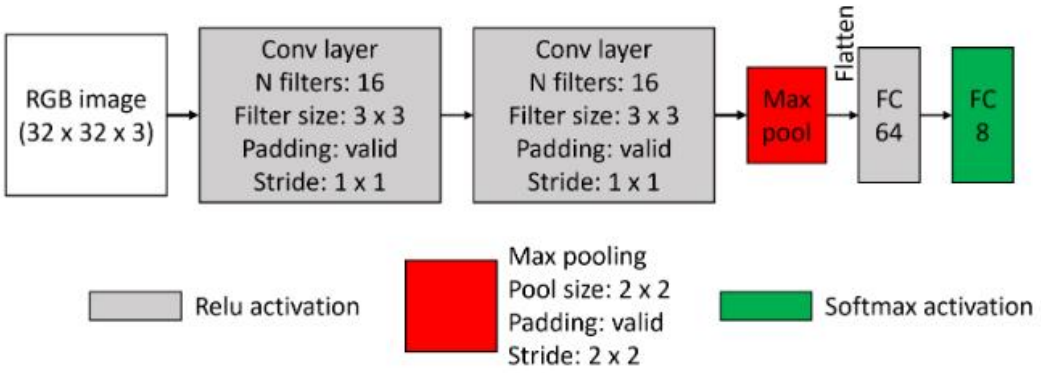


Figure 1: Network diagram

1.2 Implementation Details

Random seeds: np.random.seed(108); tf.random.set_seed(108); random.seed(108)

Data Preprocessing: The *ImageDataGenerator* function is employed to handle image scaling and data augmentation(can be set as parameters).

Activation Functions: ReLU (Rectified Linear Unit) is used as the activation function for all layers except the final layer, which employs the Softmax activation function. This choice is following the specified requirements.

Compilation and Training: The compile and fit methods are used for model compilation and training, respectively. The Adam optimizer and categorical cross-entropy loss function are specified during the compilation stage.

Evaluation and Visualization: The *evaluate*, *predict*, and *summary* methods are incorporated for model evaluation and performance visualization. Additionally, matplotlib.pyplot is used to plot the training and validation accuracy and loss over epochs.

1.3 Justification of Implementation Choice:

(A) Data Augmentation:

To prevent overfitting when the initial training set is too small, data augmentation can be employed. This also reduces the cost of collecting and labeling data. Below are the strategies being considered.

Slight Rotation: Applying a slight rotation to the image can increase the model's robustness. However, due to the low resolution of the images, large-angle rotations may make the content difficult to recognize. Therefore, a slight rotation within +/- 15 degrees is chosen.

Translation: Slight movement of the image in both horizontal and vertical directions is considered. Given the small size of the images, the distance of translation should not be too large.

Horizontal Flipping: For certain categories of scenes (such as natural environments, flora and fauna, food items, etc.), horizontal flipping is appropriate. Therefore, it is decided to use this technique.

Color Perturbation: Slight changes in the image's brightness, contrast, and saturation can help the model recognize scenes under different lighting conditions.

Cropping: Since the image size is already small, further cropping may not be suitable. Therefore, this technique is **not applied**.

Gaussian Noise: Due to the already low resolution, adding Gaussian noise would make the image even more difficult to recognize. Therefore, this technique is **not applied.**

(B) Number of Epochs:

I set a relatively large number of Epochs and use Early Stopping to automatically halt training when there's no improvement in performance on the validation set. The EarlyStopping callback monitors the loss on the validation set (val_loss). If there's no improvement in the validation loss for 10 consecutive epochs (patience parameter = 10), the training will automatically stop, and the model's weights will be restored to the best-performing iteration on the validation set. This way, the model can be trained as much as possible without overfitting.

Moreover, the training dataset consists of 1440 images, which is relatively small. Therefore, more Epochs may be needed to adequately train the model. Ultimately, I chose 60 Epochs, and the experiments showed that this is already sufficient. In most cases, the training stops before reaching this number.

(C) Batch Size:

A larger batch size may make the model more likely to converge to a local optimum rather than a global optimum. The convergence speed is fast. However, a smaller batch size can provide a regularization effect, helping to prevent overfitting. This may be due to the increased noise during the learning process. However, using a smaller batch size may also increase the training time.

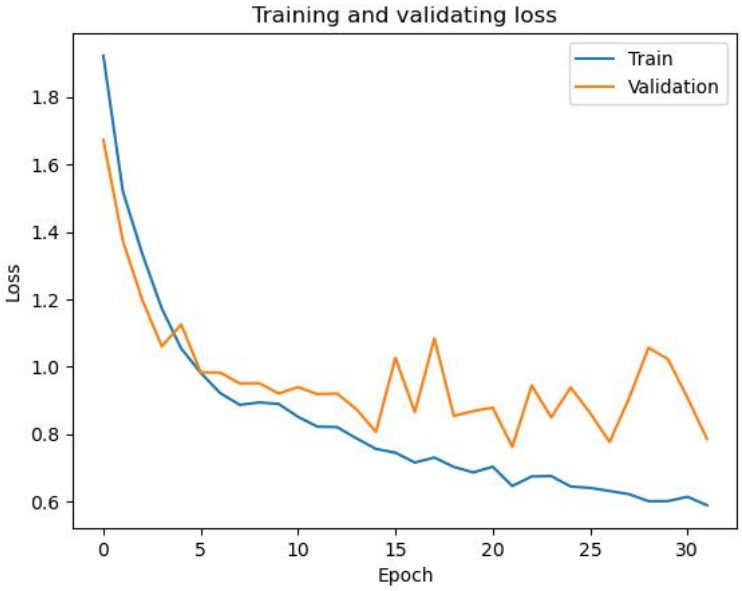
By balancing convergence speed, computational efficiency, and performance, we lean towards choosing a smaller batch size to achieve relatively better performance. Additionally, since our dataset is relatively small, using a smaller batch size is more appropriate. So I set 15 for the batch size.

1.4 Plots of the training and validation loss/accuracy

Based on the "Training and Validating Accuracy" figure, we can see that as the number of epochs increases, both the Training accuracy and Validating accuracy show a stair-step increase. In the early stages (around before epoch 10), the performance improves rapidly, while in the later stages (around after epoch 10), the rate of improvement slows down.

Towards the end, the validation accuracy even shows a slight decreasing trend.

The "Training and Validating Loss" figure shows that both "Training Loss" and "Validating Loss" exhibit a fluctuating decline. The rate of decline for "Validating Loss" is smaller compared to "Training Loss." Moreover, when the number of epochs exceeds 15, the rate of decline for "Validating Loss" starts to slow down.



1.5 Accuracy of the network on the test set

The accuracy of the final trained network on the test set is 0.8. Higher than benchmark. (The accuracy of 8-class random classification can be considered as 12.5%)

Q2 - Error analysis

2.1 Accuracy

Using the model implemented in the above question (with the same parameters set), the accuracy table shown below is tested and calculated using a model obtained from one of the runs. (Re-running may result in slight fluctuations in the data, and the results may not be exactly the same.)

Table 1. *Classification accuracy for different classes*

Class	coast	forest	highway	insidecity	mountain	opencountry	street	tallbuilding	Average
Accuracy	0.85	0.825	0.825	0.8	0.700	0.775	0.8	0.825	0.8

To get a more intuitive sense of the accuracy for each class, I created a confusion matrix. The larger the value, the darker the color. Only the diagonal line represents correct predictions, meaning the darker the diagonal line, the better the performance. Ideally, the values in other areas should be as close to zero as possible.

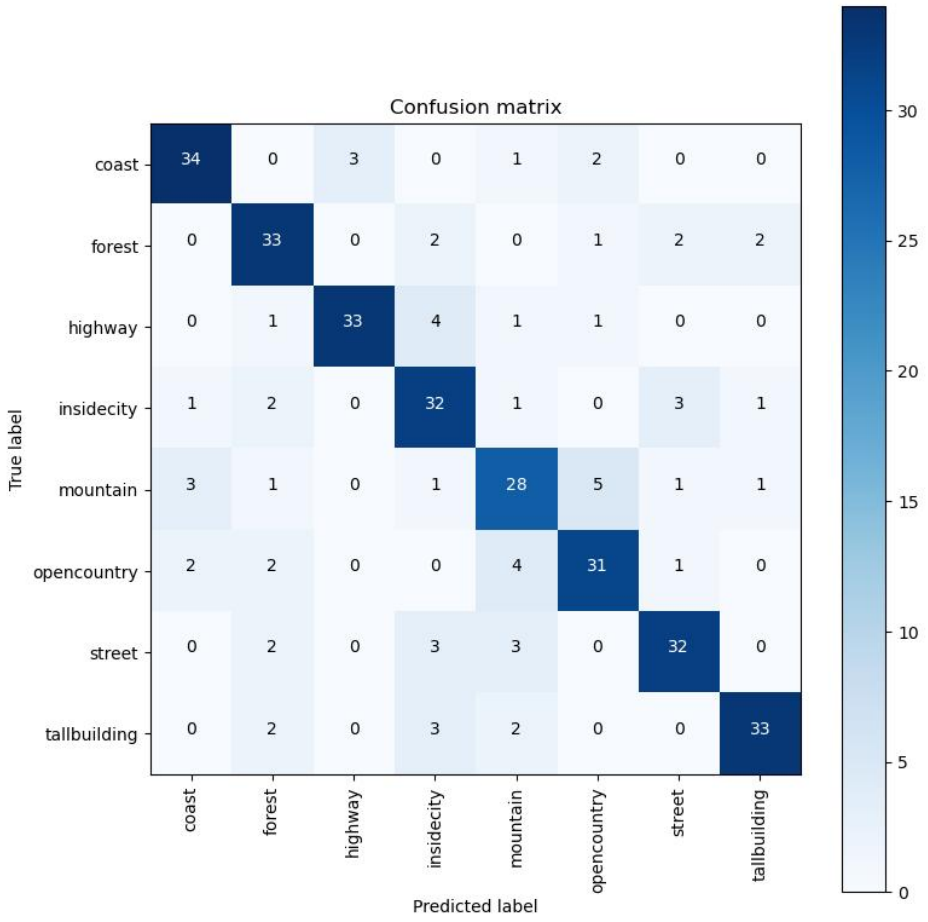


Figure 2. *Confusion matrix on test data for Q1 CNN model*

2.2 Performance of the CNN model

Strengths:

1. The performance of the model is consistent across classes, except for "mountain" and "opencountry", for which the accuracies are 0.8-0.85.
2. The average accuracy across all classes is 0.8, which is relatively high and suggests that the model is generally reliable.

Problems:

From the confusion matrix, it can be seen that the CNN model tends to confuse **'mountain'** and **'open country.'** **Five** 'mountain' images were predicted as 'open country,' and likewise, **four** 'open country' images were predicted as 'mountain'.

Based on accuracy, the model is relatively difficult to classify classes like **'mountain'** and **'open country'**. This could be due to various factors such as less training data for these classes, or these classes having features that are harder to distinguish.

Possible Reasean:

The possible reason could be the resolution of the image is too low; there is a high degree of similarity(features are similar) between the image categories.

E.g. The shapes of “mountain” and “opencountry” are similar in the Figure 3.

For “mountain” and “opencountry”, both types of images often feature expansive views and open spaces; Both may contain similar natural elements, such as sky, clouds, and vegetation, so a large areas are the same color.

Figure 3. *Examples of mis-classified pictures*

Q3 - Kernel engineering

The figures 4 and 5 show the performance change of the new model as we increase kernel size.

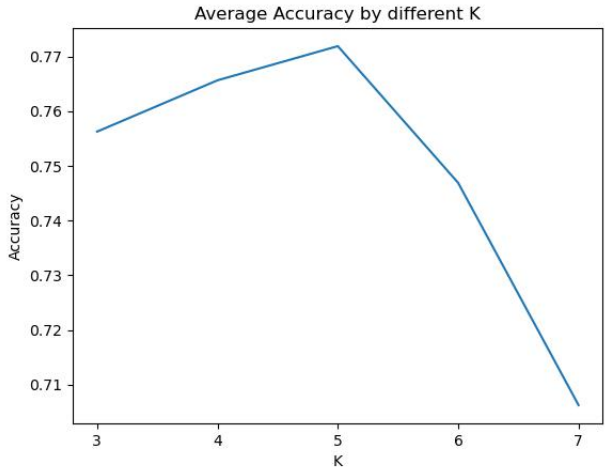


Figure 4. *Average accuracy on test data set for the model in Q3 by different K.*

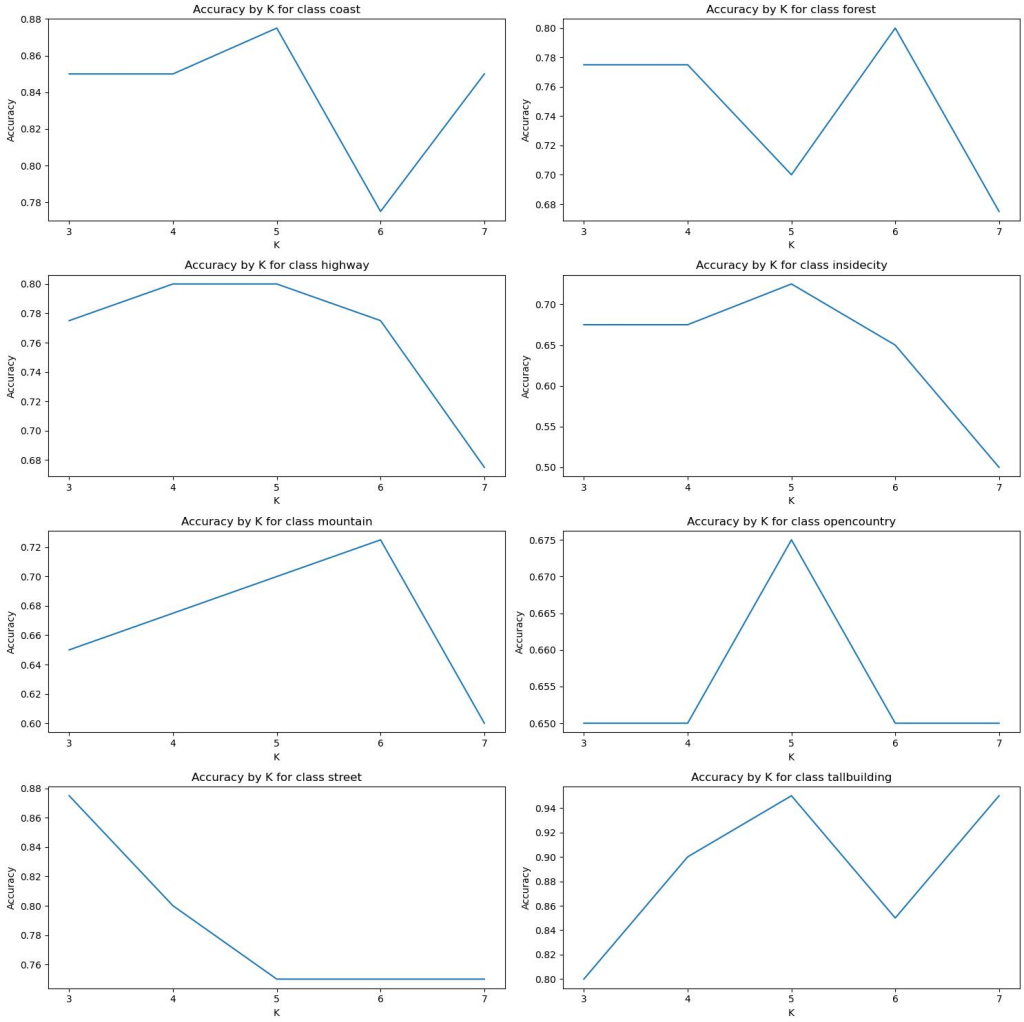
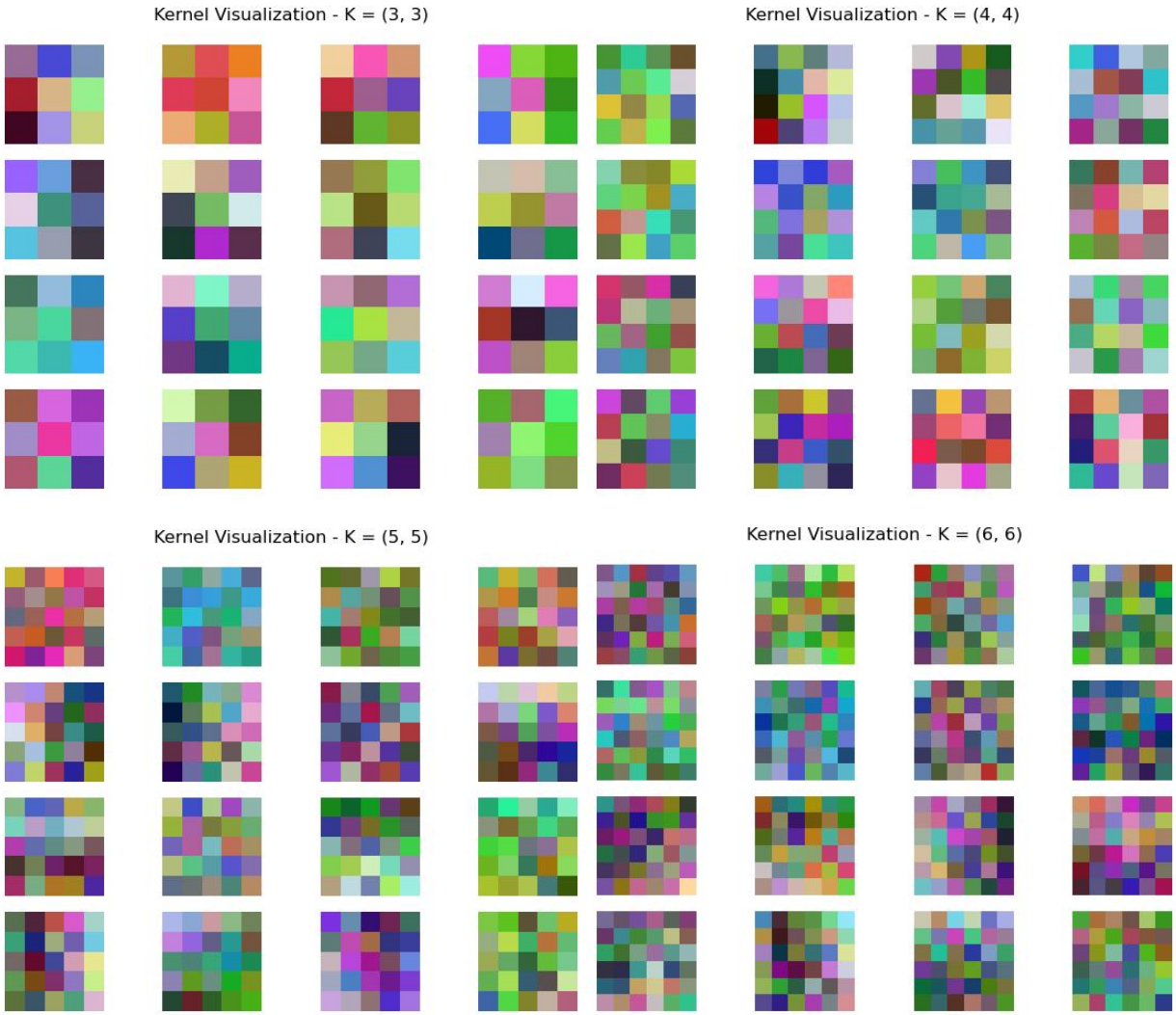


Figure 5. *Accuracy on test data set for the model in Q3 by different K, for each class.*

As K increases, the performance of the model rises, reaching a peak when K = 5, and subsequently declines. When K is very small, the convolutional kernel may not be able to capture sufficient features, leading to underfitting of the model. As K increases, the model is able to capture more complex and global features, which enhances its accuracy. However, as K becomes too large, there's a risk of losing too much local information, which could compromise the model's performance.

The accuracy at K=5 is also the closest to the accuracy of the model in Question 1. This is because the network in Question 1 (which stacks two convolutional layers with kernel size of 3*3) and the network at K=5 in Question 3 (which uses a single convolutional layer with a kernel size of 5*5) have the same receptive field. Both consider a 5*5 area in the input. However, the network in Question 1 has greater non-linear expressive power. This is because using two times kernels allows the insertion of a non-linear activation function (such as ReLU) between the two convolution operations, while using a single times kernel permits only one application of a non-linear function.



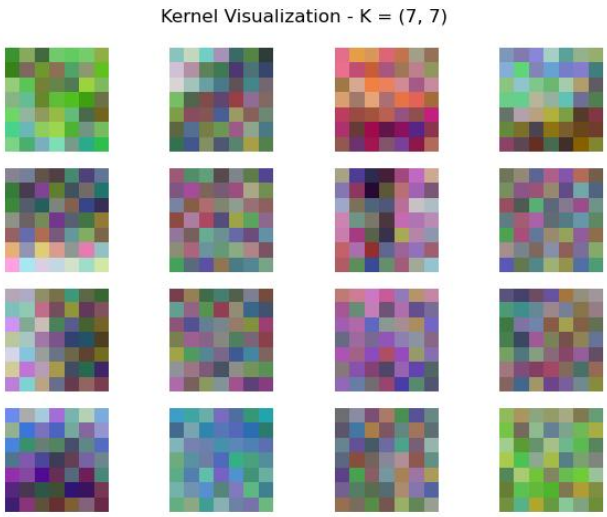


Figure 6. *Kernel Visualization for different K*

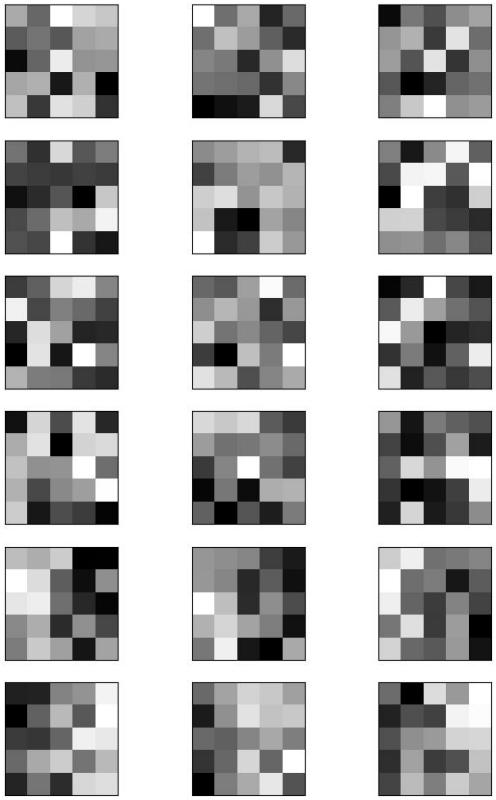


Figure 7. *Kernel Visualization for first 6 filters from the only convolutional layer - Kernel Size = (5,5). One row for each filter and one column for each channel. From left to right, they are Red, Green, Blue channel.*

In Figure 7, white indicates that the convolutional kernel is looking for a specific pattern or feature at that particular position and channel. Black shows it's looking for the opposite. We can see where each convolutional kernel is capturing features.

To better understand what the kernel learned from an image, I visualized the feature map, Figure 8.

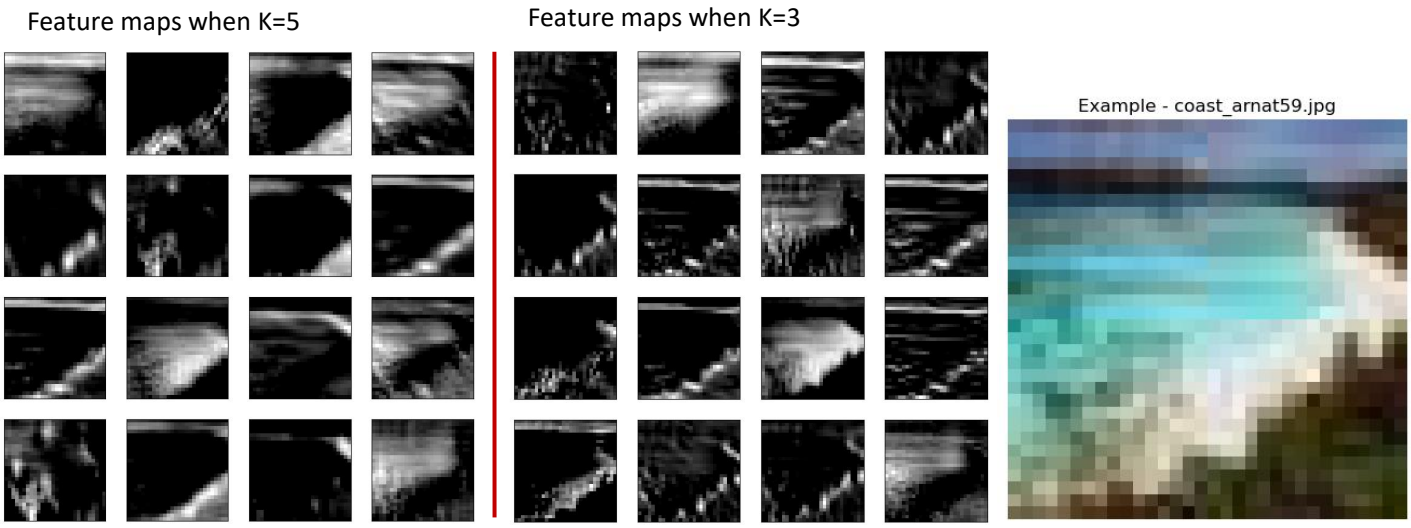


Figure 8. *Feature maps for the example image(coast_armat59.jpg) when K = 3, 5*

We can see the kernel (k=5) can capture more features from different areas of the image. 3x3 kernels are more sensitive to minor local changes due to their smaller receptive field, leading to feature maps with more high-frequency details. In contrast, 5x5 kernels, with their larger receptive field, tend to produce smoother, lower-frequency feature maps better suited for capturing broader patterns and structures. Larger kernels are more apt at detecting wider edges and bigger structures, while smaller kernels tend to highlight finer textures and details.