# COMP90086: Assignment 2 Report

Tanzid Sultan (ID# 1430660)

September 9, 2023

## 1. CNN Training

For this section, we have trained a convolutional neural network (following the network architecture described by the assignment specifications) to recognize scene images from 8 different categories: '*coast*', '*forest*', '*highway*', '*insidecity*', '*mountain*', '*opencountry*', '*street*' and '*tallbuilding*'. The input images are 32x32 pixels with RGB color channels. We rescaled the pixel values to be in the range $[0, 1]$ and chose a batch size of 80, then trained the network using the *Adam* optimizer. By monitoring the loss and accuracy for both the training and validation datasets throughout the training process, we were able to tune the learning rate to **0.05** such that the optimizer converges to a solution within 100 epochs. Figure 2 shows the loss and accuracy over epochs. We can see that the network quickly overfits to the training data, eventually reaching close to perfect training accuracy, however the validation accuracy plateus at 70%. This is due to the fact that our training dataset is quite small (with 180 images per class) and our network is over-parameterised, resulting in overfitting and poor generalization. Similar to the validation accuracy, the network is only able to achieve **69.5%** accuracy on the test dataset.

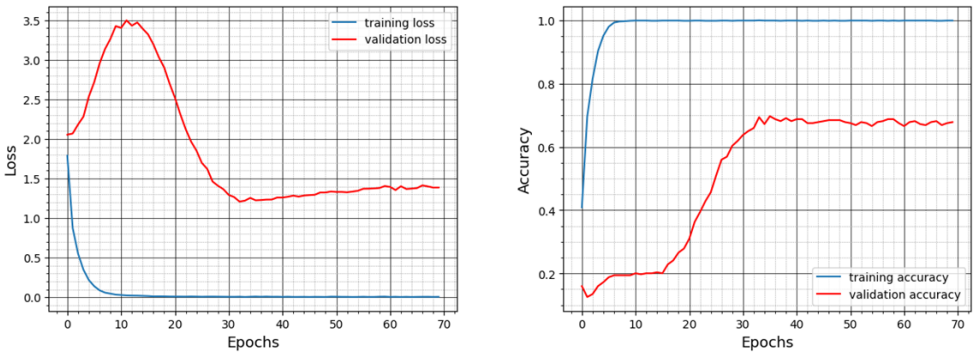


Figure 1: Loss and Accuracy over training epochs without image augmentation.

To allow our model to achieve better generalization capability and be less prone to overfitting, we chose to artificially increase the size of our training dataset using the following data

augmentation techniques [1]:

1. *Zooming:* New images are generated by randomly zooming the existing images by a factor in the range $[0.9, 1.1]$
2. *Translation:* Images are randomly shifted in direction, both horizontally and vertically, by a factor of [-0.3, 0.3]
3. *Horizontal flip:* Images are flipped horizontally
4. *Rotation:* Images are randomly rotated by angle [-10,10] degrees

The above image augmentation techniques were found to be most suitable for our dataset. In particular, including zoomed images in our training data makes the model more robust to learning scale-invariant features, similarly rotating the images allows the model to detect the same feature at different orientations. Also, since images of natural scenes have bilateral symmetry, flipping an image horizontally preserves it's semantic meaning. Finally, we also incorporated horizontal and vertical translation of the images, because this would make the model more robust to variations in positions of objects in the image (this can be specially important for enabling our model to correctly classify scenes containing mountains and buildings).

We re-trained our model with this augmented dataset and found that incorporating an exponentially-decaying learning rate helped to stabilize large fluctuations in the optimization steps (the fluctuations are most likely caused by the increased diversity in the training data batches due to augmentation) and also added a *dropout* layer to help prevent our model from over-fitting to the training data. We also stop training beyond 150 epochs as the validation acccuracy tends to plateue by then. The learning curves from Figure 2 show that the model does not overfit as easily to the augmented training dataset, a higher validation accuracy is achieved and the gap between the training and validation accuracy is also much smaller, which indicates an improvement in the model's generalization capability. Our trained network is now able to achieve a **78.4%** accuracy on the test dataset.

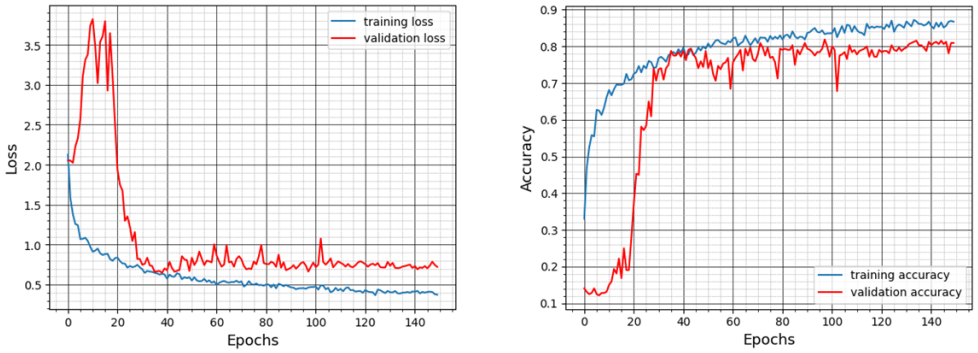


Figure 2: Loss and Accuracy over training epochs with image augmentation.

[1] this is done by setting appropriate parameters in the Tensorflow ImageDataGenerator instantiation

## 2. Error Analysis

In this section, we will analyse the performance of our model on the unseen test data. We will first look at the average prediction accuracy for each of the 8 categories and try to identify any particular classes which our model has more difficulty predicting.

| **Class** | **Accuracy** |
|---|---|
| coast | 0.725 |
| forest | 0.825 |
| highway | 0.825 |
| insidecity | 0.95 |
| mountain | 0.825 |
| opencountry | 0.5 |
| street | 0.775 |
| tallbuilding | 0.85 |
| **Overall** | 0.784 |

Table 1: Average accuracy per class

From Table 1, we can see that our model achieves above 80% accuracy for classes '*forest*', '*highway*', '*insidecity*', '*mountain*' and '*tallbuilding*'. For classes '*street*' and '*coast*', the accuracies are in the high and low 70% respectively. This indicates that the model is able to learn the right features which are required for correctly prediciting those classes and distinguishing between them. However, we can see that our model has some difficulty predicting the class

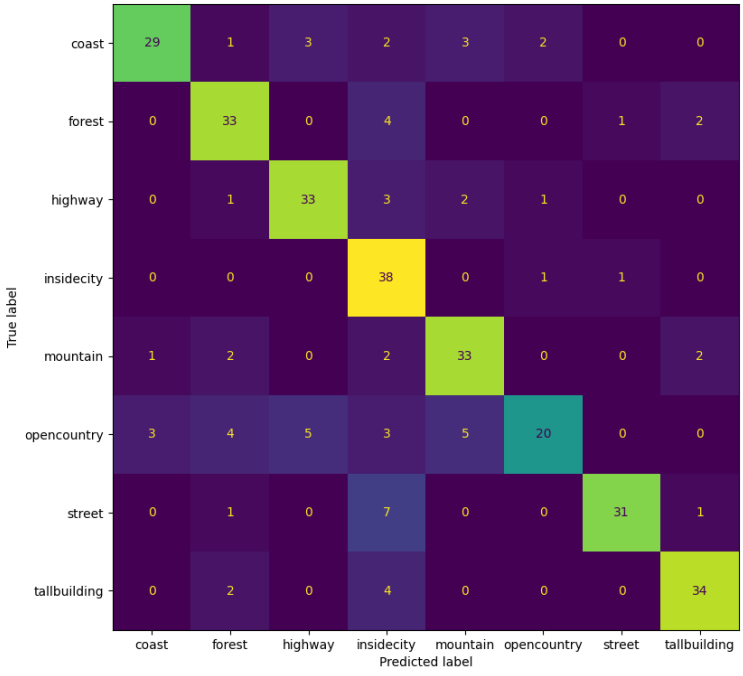


Figure 3: Confusion matrix for test dataset predicted labels.

'*opencountry*', which has an accuracy of only 50%. This could be due to the fact that the '*opencountry*' scenes typically contain features/characteristics such as open skies, large expanses of land and a sharp dividing horizon. Most of these features are also present across scenes from several other classes, such as '*coast*', '*highway*' and '*mountain*', and this could be the reason why the model get confused when it is trying to classify a scene belonging to the '*opencountry*' class. To confirm this hypothesis, we can also take a look at the confusion matrix summarizing the model's predictions, shown in Figure 3.

Indeed, from the confusion matrix, we can see that a large portion of the 20 misclassified '*opencountry*' test images are classified as 'either *highway*', '*mountain*' or '*forest*'. Like the '*opencountry*' scenes, scenes from'*highway*' and '*mountain*' classes also typically contain a sharp horizon and expansive blue sky. Similarly, some '*opencountry*' scenes contain trees and large expanses of grassy land, which also happen to be a common characteristic of '*forest*' scenes. We can also confirm these observations by directly comparing some of the mis-classified test images to actual images belonging to that class from the training set, as shown in Figure 4 (The top row of the figure shows some of the mis-classified images and the bottom row shows similar looking training images from that incorrect class). For instance, in the top left, we have an '*opencountry*' test image which has been misclassified as '*forest*'. Underneath this image, we have shown an actual '*forest*' image from the training set and we can immediately note the similarities. (I personally would have a hard time classifying this test image as '*opencountry*' because of the striking similarities with some of forest images from the training set, so it's not surprising that the model also gets confused!)

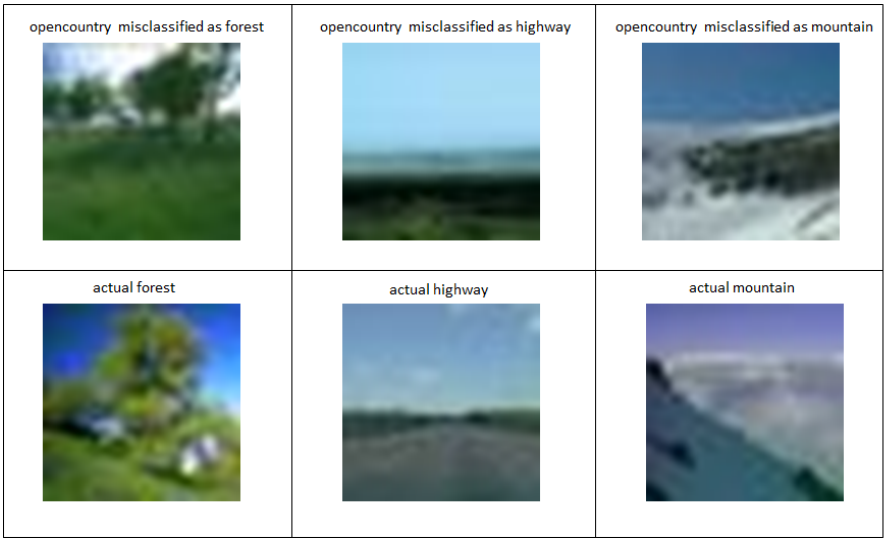


Figure 4: *Top Row:* mis-classified *opencountry* images. *Bottom Row:* training images from the incorrect predicted class. Note the striking similarity between the images in the top row and corresponding images in bottom row.

The confusion matrix also reveals that an overwhelming majority of mis-classified test im-

ages from the classes '*street*' (7 out of 9) and '*tallbuilding*' (4 out of 6) are incorrectly classified as '*insidecity*'. In this case, it is also obvious why the models gets confused. Most of the training images for '*insidecity*' scenes also contain building and streets, so there is inevitably a significant overlap in features with '*street*' and '*tallbuilding*' scenes. So, in general, we can conclude that images containing important features that are present across multiple categories will be harder for our model to classify.

## 3. Kernel Engineering

In this final section, we modify our convolutional network architecture by removing the second convolutional layer so that our model now has only one convolution layer, keeping all other parameters fixed. Then we create multiple copies of this model with varying kernel sizes and train each of these models on the augmented training dataset in order to investigate the effect of the kernel size on classification performance.

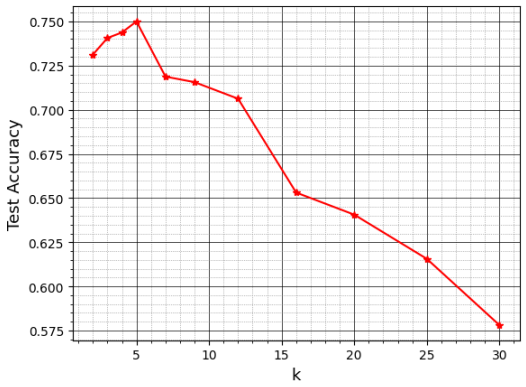


Figure 5: Test set accuracy vs kernel size.

Figure 5 shows the accuracy on the test dataset for varying kernel sizes. We can see that smaller kernel sizes, i.e. $k = 2, 3, 4, 5$, result in the largest test accuracies. As kernel size increases beyond 5, the test accuracy falls steadily. To understand why this is the case, we need to consider the types of features that are being learned by the kernels for different sizes. When the kernel size is small, the CNN is able to extract local patterns from images, specifically lines and edges. On the other hand, larger kernels are unable to capture these local patterns and can only extract large-scale high level features. For **low-resolution images**, these fine-grained details carry most of the information content. For example, consider the high-resolution image and it's down-sampled low-resolution counterpart, shown in Figure 6. In the high resolution image, we can easily recognise the main content of the image being the person sitting on the ground. We can even see the fine details, such as the pattern on her skirt and the texture of her hair. Now, in the low resolution image, much of these finer details are lost and we see mostly uniform regions separated by edges. However, these edges are precisely what still allows us to

recognise the main contents of the image, i.e. the person sitting on the ground.

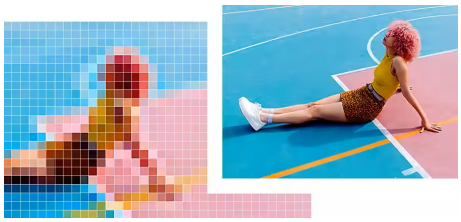

Figure 6: High resolution image with it's down-sampled low resolution version (*Image Credit: https://www.adobe.com/creativecloud/photography/discover/increase-resolution.html*)

Since our model has been trained on 32 x 32 pixel **low-resolution images**, we can therefore expect the local patters such as edges to be crucial for recognizing the scene category. As the filter size becomes larger, it becomes increasingly difficult for the CNN to extract these local patters, which hinders the network's ability to correctly classify unseen test images (which supports the trend shown in Figure 5). Figure 7 shows some of the filters learned by the CNN at four different filter sizes (k = 2,5,12,25) along with the feature activation map for the test image 7(e). For the small filter size in 7(a), we can see the filter are able to capture local patterns. For example, the filter on the top row of 7(a) are horizontal edge detectors in the first two channels and a vertical edge detector in the third channel, and the feature activation map, i.e. the filtered image, is brightest at the locations of these edges. Conversely, from 7(d), we can see that the feature activation maps for the larger filter size are very sparse and local pattern such as edges are not being captured by the filters. Thus the larger filter size leads to lower classification performance on the test dataset.

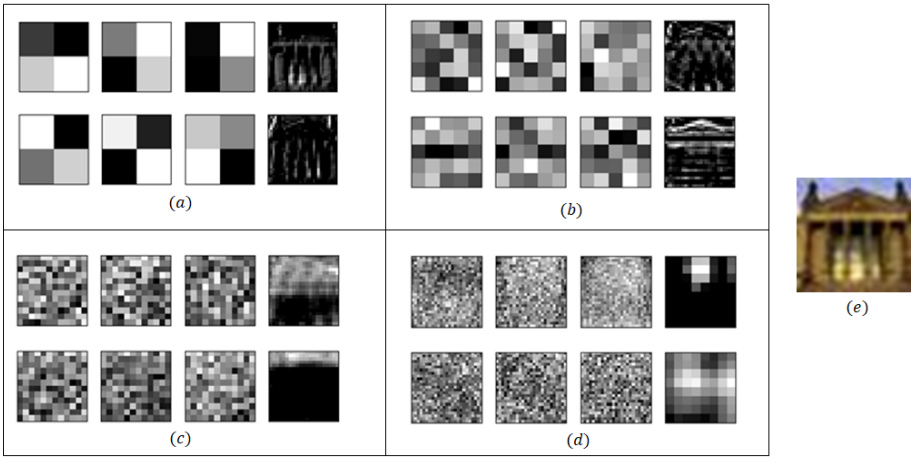


Figure 7: (a) k = 2, (b) k = 5, (c) k = 12, (d) k = 25 (e) test image. Each panel (a) - (d) shows two different filters learned by the convolutional layer on each row along with the feature activation map computed for a test image (three channels followed by feature activation map)