Deep Learning - Homework 2 - Group 32

Alexandre Pires - 92414, Diogo Fouto - 93705, João Fonseca - 92497

January 2022

1 Question 1

1.1

In order to calculate the total amount of parameters a network has, we need to calculate the parameters for each layer and the size of the output of each layer, as that will be relevant for the linear layer.

The convolutional layer will have 8 kernels, which are independent and all have the same size and stride, so we can just calculate the number of parameters for one kernel and multiply it by 8. Having a 5x5x3 filter, there will be a total of $5 \times 5 \times 3 = 75$ parameters in it, plus a bias, for a total of 76 parameters. So for the 8 filters there will be $8 \times 76 = 608$ parameters.

In order to compute the resulting output size, one can use Eq. 1 (taken from [1]), where the input image is $N \times N \times D$, a filter is sized $F \times F \times D$, and there are K filters, and a stride of S, where the output will have an output size of $M \times M \times K$.

$$M = \frac{N - F}{S} + 1 \tag{1}$$

In our case, $M = \frac{28-5}{1} + 1 = 24$. The resulting stack of activation maps will be $24 \times 24 \times 8$, as we have 8 filters.

The next layer, the max-pooling layer, has no parameters, but knowing the output size will be relevant for the linear layer, we can yet again use Eq. 1 to calculate that the height and width of the output of this layer will be $M = \frac{24-4}{2} + 1 = 11$, and since there are eight filters, the output will have size $11 \times 11 \times 8$.

Finally, this output will go into a linear layer, so it will need to be turned into a vector. The $11 \times 11 \times 8$ tensor is transformed into a $11 \times 11 \times 8 = 968$ sized vector. Each of the 10 outputs will be calculated using the standard $z = w_i \cdot x_i + b$ formula, so there will be 968+1 parameters for each output, for a total of $969 \times 10 = 9690$ parameters for this layer.

The resulting total is $608 + 9690 = 10298$ parameters.

1.2

If instead of the convolutional and max-pooling layer we only had a feed-forward layer with 100 hidden units, we would first have to convert the image into a vector, concatenating each channel too. The input would be a vector with $28 \times 28 \times 3 = 2352$ values. Each hidden layer would have a weight associated with each value, plus a bias, for a total of 2353 parameters for each hidden unit, or 235300 for the entire layer. This would result in an 100 entry sized output.

The linear layer after that would yet again have to have a weight for each input, plus a bias, so 101 parameters, or, for the 10 output units, 1010 parameters total.

Therefore, the resulting network would have a total of $235300+1010 = 236310$ parameters, which is about 23 times more parameters than the CNN considered before, all the while loosing some of the proprieties CNNs have, like translational invariance (see Question 2.1).

1.3

1.3.1

$$\begin{aligned} P^h &= \text{softmax}(\frac{Q^h (K^h)^T}{\sqrt{d_k}}) = \\ &= \text{softmax}(\frac{X W_Q^H (X W_K^H)^T}{\sqrt{d_k}}) = \\ &= \text{softmax}(X \frac{W_Q^H (W_K^H)^T}{\sqrt{d_k}} X^T), \end{aligned}$$

so $A = \frac{1}{\sqrt{d_k}} W_Q^H (W_K^H)^T$, where A is $n \times n$.

Since both W_Q^H and W_K^H are $n \times d$, their rank can be, at most, $\min(n, d) = d$. So, $rank(A) \leq \min(rank(W_Q^H), rank(W_K^H)) = d$.

1.3.2

$$\begin{aligned}
P^1 &= \text{softmax}(XW_Q^1(W_K^1)^T X^T) \\
P^2 &= \text{softmax}(XW_Q^2(W_K^2)^T X^T) = \\
&= \text{softmax}(XW_Q^1 B(W_K^1 B^{-T})^T X^T) = \\
&= \text{softmax}(XW_Q^1 BB^{-1}(W_K^1)^T X^T)
\end{aligned}$$

Since B is invertible, we have that $BB^{-1} = I$, so $P^2 = \text{softmax}(XW_Q^1(W_K^1)^T X^T) = P^1$.

2 Question 2

2.1

In order to be invariant to a certain transformation, a model has to be able to correctly classify instances that have been affected by such transformations. Examples of possible transformations include crops, rotations, translations, scales and hue shifts.

In the case of convolutional layers, they are only invariant to translation. This is because the same filters are applied in all parts of the image (depending on the stride), so the same information will eventually be captured by the convolution, only in a different place. One can follow intuition to understand why, in the case of CNNs, there is no invariance to:

- Rotations: filters are applied in a square shape and don't follow rotations;
- Scaling: filters will stop capturing certain features, and instead capture only parts of such features (e.g.: what was a circle can turn into just a semicircle by increasing the size, if captured by a square of constant size);
- Hue Shifts: the filter will be applied, but the color channel information will shift, leading to different values when the filter is applied, which could lead to different classifications.

2.2

The CNN model had the best performance – read: highest accuracy – with a learning rate of 0.01. Below are that model's charts for Validation Accuracy and Training Loss, figures 1 and 2 respectively, as functions of the epoch number.

The results present a maximum accuracy of around 0.918 for the validation set, and a minimum loss of approximately 0.35, with both results improving consistently throughout the epochs, which means no over-fitting occurred.

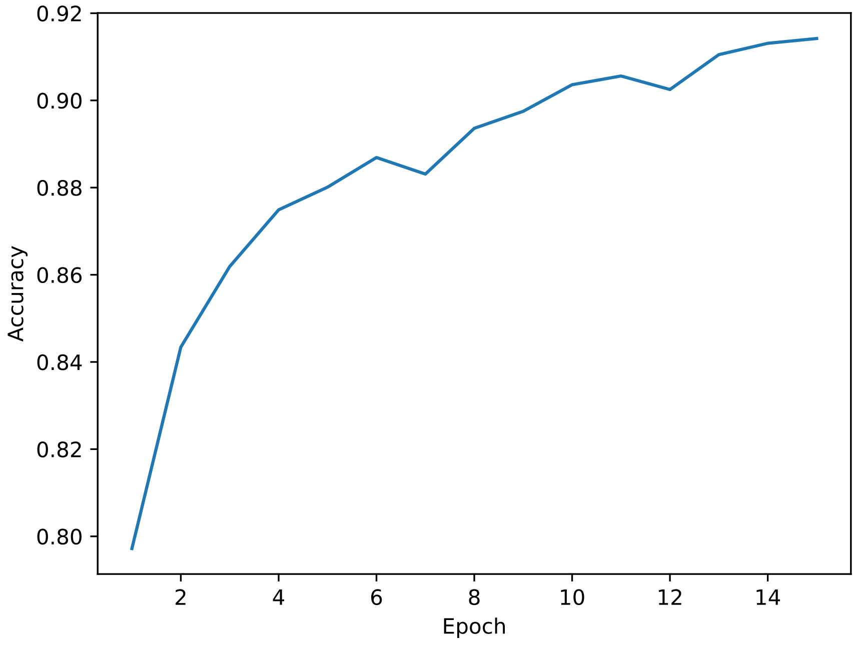

Figure 1: CNN Model - Validation set accuracy with respect to the epoch

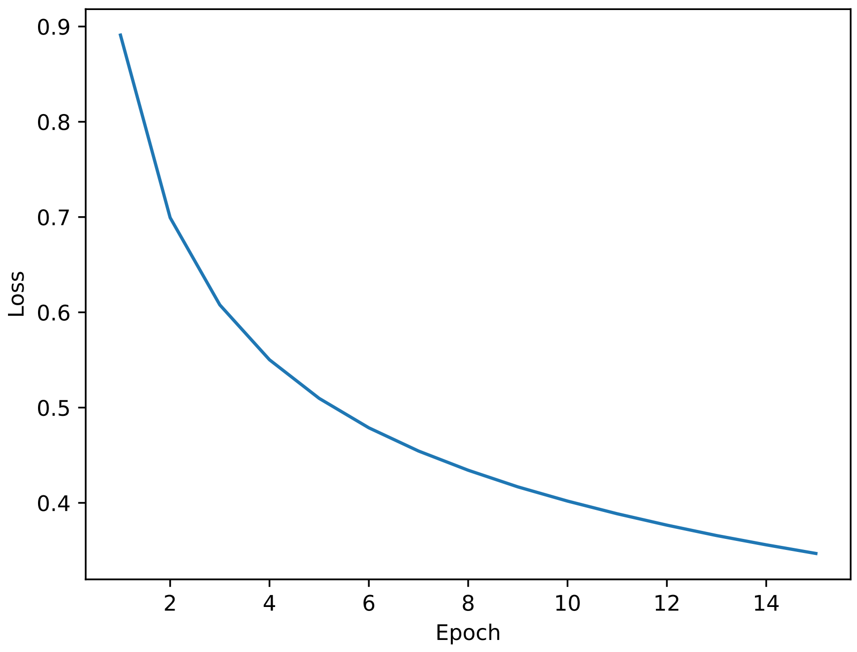

Figure 2: CNN Model - Training Loss with respect to the epoch

2.3

The filters for the first and second layers are presented in figures 3 and 4 respectively.

While the resolution is small, it is sometimes possible to see certain filters that are sensible to general shapes. Examples are the 2nd row 3rd column (2,3) filter capturing a circular shape, and the (3, 7) filter capturing a vertical line in the middle of the figure.

These are most prominent in the second layer, as expected, since deeper filters should capture higher level features (see [2]), which are more recognizable. Initial layers will only capture vague concepts like contrasts between certain sides, and deeper layers should start capturing shapes that are present in the objects being recognized (specific curves or patterns), and finally the last layers should instead be able to recognize defining features that are, preferably, only present in one of the different classes, as to be able to recognize it and differentiate it from other classes. This, of course, is not to be expected from filters with such a low resolution, and such a shallow depth.

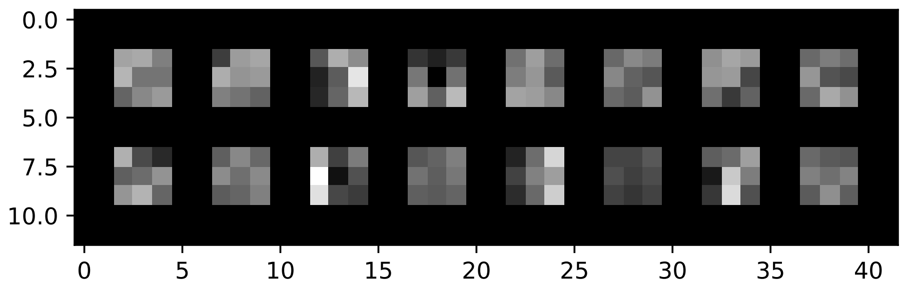

Figure 3: CNN Model - First Convolutional Layer

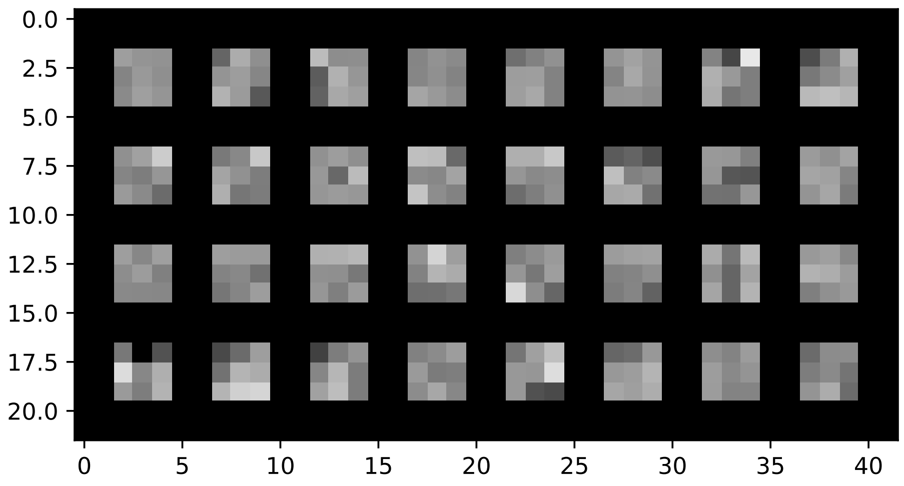

Figure 4: CNN Model - Second Convolutional Layer

3 Question 3

3.1

For the initial model for this task, we used an Encoder-Decoder architecture, where the encoder is a pre-trained ResNet, and the decoder is a simple auto-regressive LSTM with dropout and a fully connected linear layer for the output.

Below are this model's charts for training loss and its BLEU-4 score on the test set as functions of the epoch, on figures 5 and 6, respectively. The final BLEU-4 score on the test set was 0.50. We can see that the loss lowered consistently, but that BLEU-4 score was less stable. This may be due to the nature of the scoring system itself, since it resorts to overlaps of 4-grams and not to the evaluation of the semantic meaning of the captions generated.

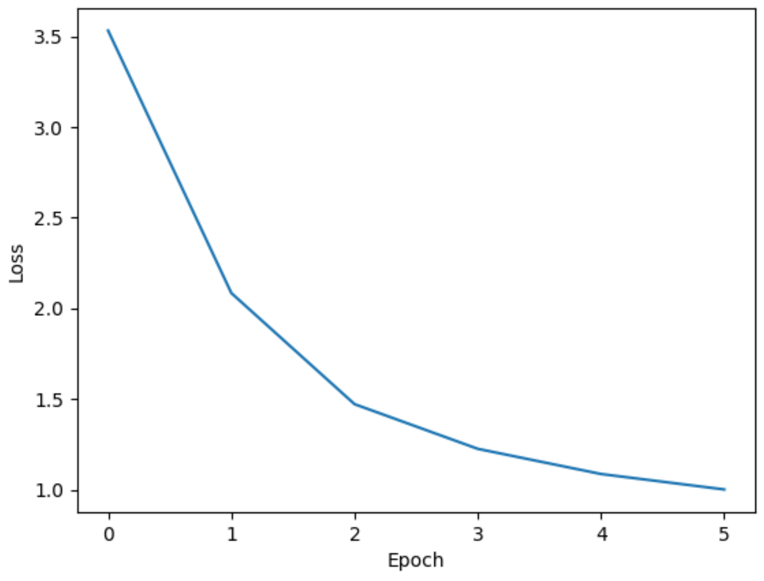

Figure 5: Loss of the Encoder-Decoder model without Attention

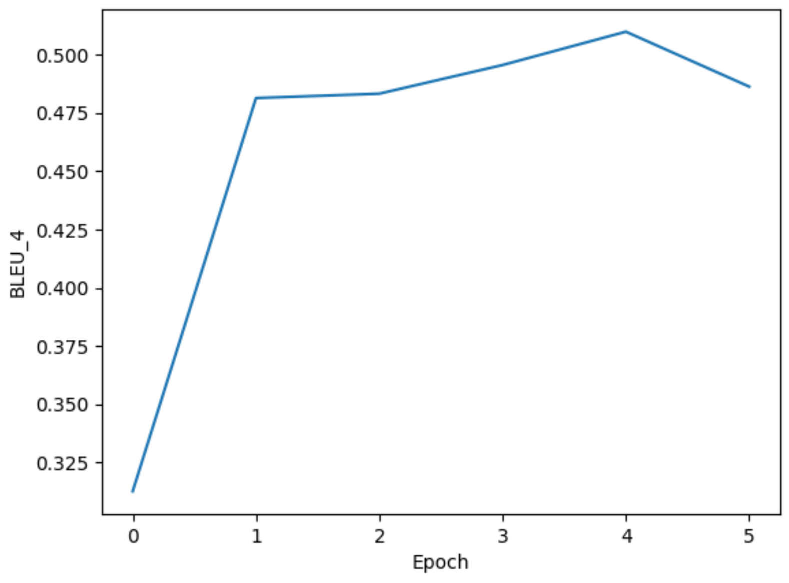

Figure 6: BLEU-4 score of the Encoder-Decoder model without Attention

3.2

In this model we used the same Encoder-Decoder architecture as before but with (Bahdanau) attention on the decoder.

Below are this model's charts for training loss and its BLEU-4 score on the test set as functions of the epoch, on figures 7 and 8, respectively. The final BLEU-4 score on the test set was 0.51. Notice the stable decrease of the loss and the slight instability of the BLEU-4 score, which were also present in the previous model.

There was a very slight improvement in performance with this model, in relation to the no attention model. The lack of improvement is probably due to the small dataset we are using (about 600 image-caption pairs), and due to the lack of diversity of the training captions (many of them have similar word choices and structure). It should also be noted that aerial images don't have clear-cut objects, making the encoding task more difficult, which also has a negative impact on the auto-regressive decoding.

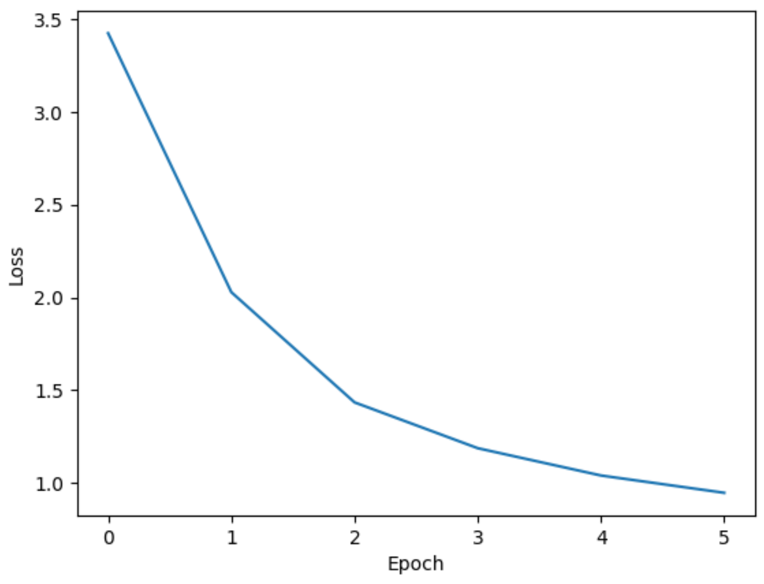

Figure 7: Loss of the Encoder-Decoder model with Attention

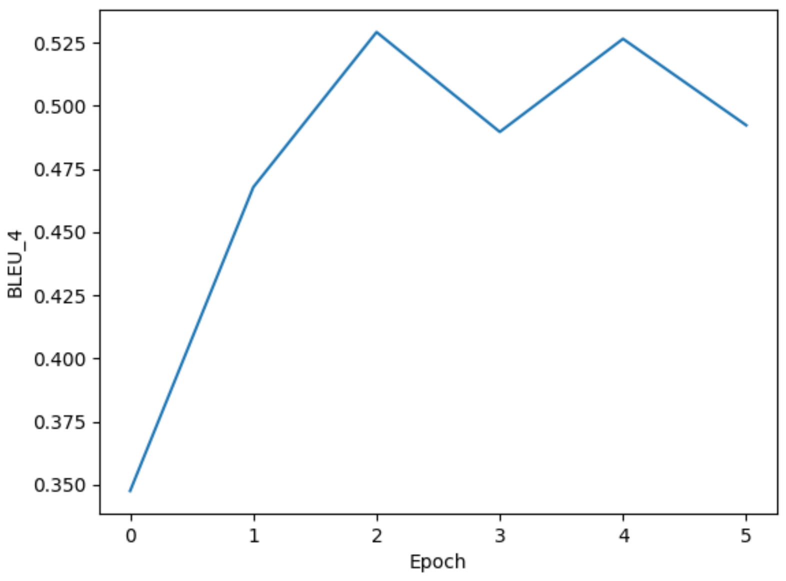

Figure 8: BLEU-4 score of the Encoder-Decoder model with Attention

3.3

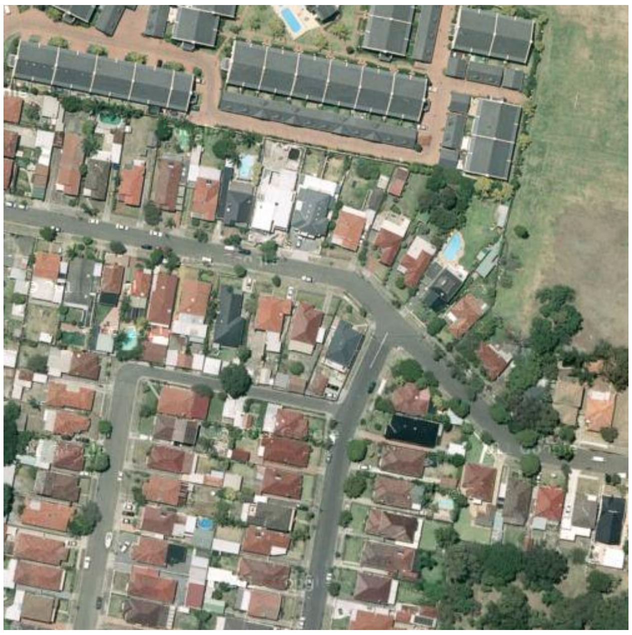

Figure 9: "219.tif" - a residential area with houses arranged neatly while many plants on the roadside

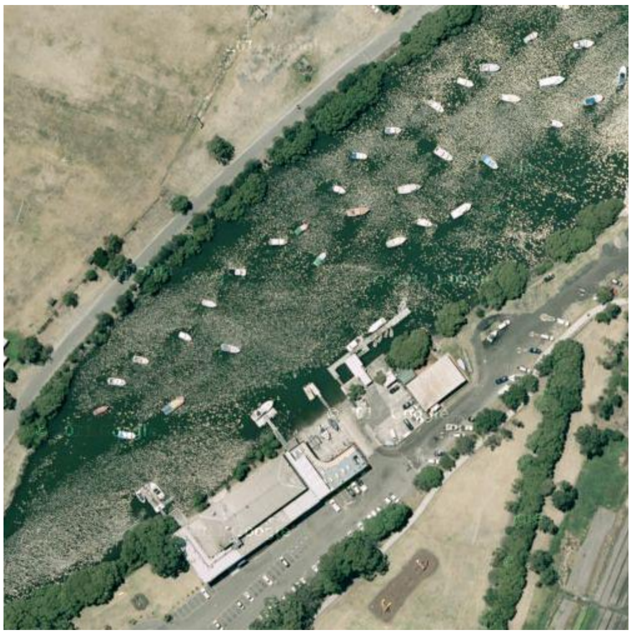

Figure 10: "357.tif" - a meadow with some green bushes and white bunkers on it

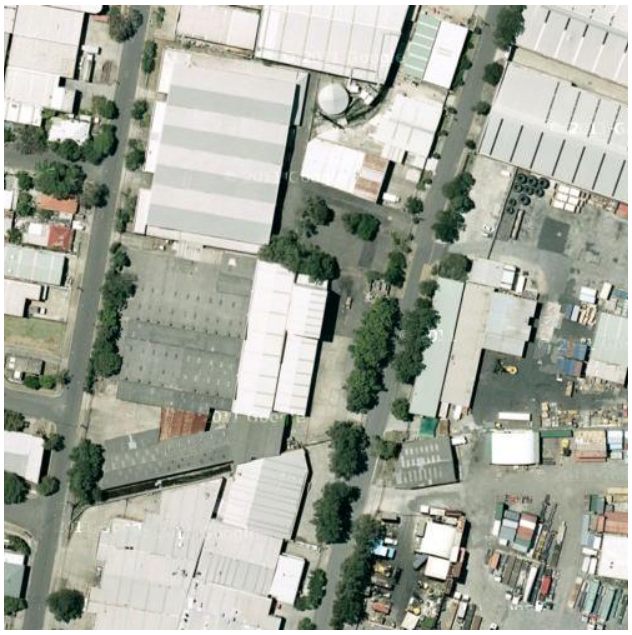

Figure 11: ”540.tif” - an industrial area with many white buildings and some roads go through this area

References

[1] A. Martins, F. Melo, and M. Figueiredo, “Lecture 8: Convolutional neural networks,” *Deep Learning Course, IST*, p. 14, Fall 2021.

[2] A. Martins, F. Melo, and M. Figueiredo, “Lecture 7: Representation learning,” *Deep Learning Course, IST*, p. 9, Fall 2021.