Introduction to Deep Learning
Assignment A2

Chloé Tap, Evan Meltz, Giulia Rivetti (Group 36)

November 2023

Contributions

Chloé Tap: Part 1 text-to-text code and report, Part 2 adding LSTM layers code and report
Evan Meltz: Part 2 text-to-text and image-to-text code and report
Giulia Rivetti: Part 1 image-to-text, text-to-image and adding LSTM layers code and report

Introduction

Recurrent Neural Networks (RNN) are powerful networks that can "predict" the future: for example in autonomous driving systems, they can anticipate car trajectories and help avoid accidents; they also have useful applications in natural language processing, such as speech-to-text conversion and language translation. For this assignment, we focused on training decoder-encoder RNN networks to learn how to perform simple arithmetic operations, e.g. addition, subtraction and multiplication. The goal was to develop various RNN models to compute these operations in both text and image formats. In particular, we have built three models: a text-to-text model, which after having received as input a text query containing two integers and an operand between them (+, - or x), it outputs a sequence of integers that match the actual arithmetic result of the given operation; an image-to-text model doing the same operations as the previous model, but on a given sequence of images containing individual digits and an operand; a text-to-image model which receives a text query and outputs a sequence of images corresponding to the result of the operation (only used for addition/subtraction data in this case). In this report, we outline the results for each of these models when applied to sequences of addition/subtraction (Section 1) and multiplication (Section 2) operations of MNIST digits and analyze the performance for various model architectures.

1 Addition and Subtraction

In this section, we test the various models for the addition and subtraction of MNIST digits in the range of [0,99] and compare the performance of each model for different size splittings of training and test data. Hence the expected output range is [-198,198]. For all cases where the text strings are used, the "hot encoded" vectors are used as input and extracted from the model as output. These encoded vectors indicate the probability that a given digit is represented in this position. In the case of inputs, a "1" is indicated in the position associated with the given digit or operator (+/-) used. Spaces are also included in case the sequence does not make up the maximum string length for both inputs and outputs.

1.1 Text to text model

In this section, we have built a text-to-text RNN model for the addition and subtraction operation dataset. The inputs for this model are "hot-encoded" vectors representing sequences of pairs of MNIST digits which are either added or subtracted from one another. The aim of this model is to train itself to understand how to compute the operation, decode the result and output the encoded representation for the result of such an operation. For visualisation purposes, these results are decoded and transformed into float objects. We compare these predicted results with our expected results to determine the accuracy of the model. Our model structure is shown in Table 1. We trained our model over 100 epochs for multiple splits (%) of the training and test data: 50-50, 25-75 and 10-90. A full summary of the accuracies for each of these splits is listed in Table 3.

Layer type	Parameters
LSTM	u=256
RepeatVector	rf=3
LSTM	u=256
TimeDistributed	l=Dense
Dense	u=13

Table 1: Architecture for a text-to-text RNN model. Here u indicates the number of units of a layer, rf is the repetitions factor and l indicates the type of layer.

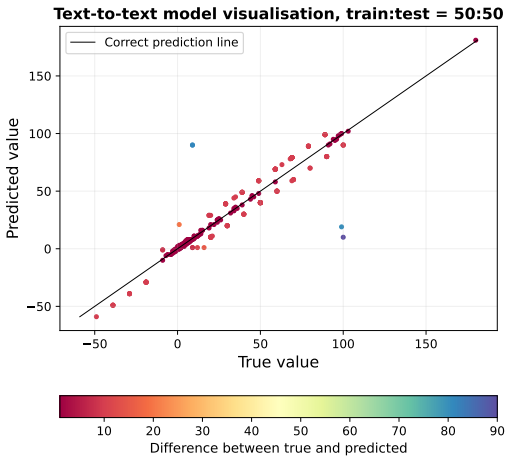
Some trends observed in all cases for incorrect predictions are described as follows:

- For expected outputs in the range [90,99], the predictions can sometimes appear in the range [900,999]. In particular, 99 is often predicted as either 900 or 990. We neglected such predictions from our visualisation.
- The model can sometimes "add" or "drop" a particular digit from the true result. For example, 9 may be predicted as 90.
- One digit in the prediction might be changed relative to the true result (e.g. 10 is predicted as 80). Given that some MNIST digits may be misinterpreted as others with a similar shape (as found in Assignment A0), it is likely that this may occur here also.
- Digits may be reversed in placement. So for example, expected output 78 is predicted as 87.
- In many cases, the predictions seem to be quite random and are not easily explained. An example of this is true digit -12 predicted as -73. Most of these are only small differences (within 20 of true digits).

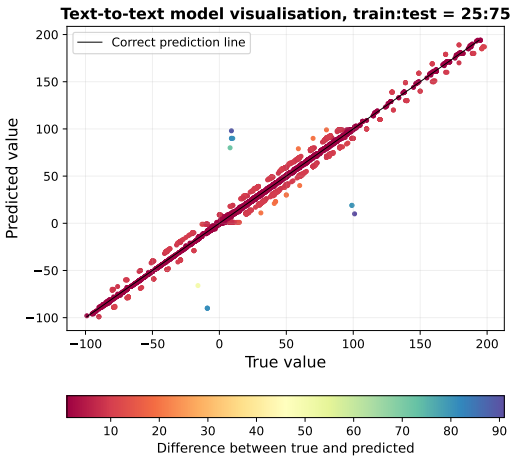
In addition there are some cases in which there are empty predictions and the decoded label consists of either all spaces or a negative sign (-) and two spaces. In other words, there are no digits present in the label, only spaces and sign arguments. These were removed from the dataset before visualisation. The visualisations of each of the train-test splits for this model are illustrated in Figure 1. For these visualisations, we plotted only those queries for which the prediction does not match the expected (true) value to avoid a larger clustering of points. Instead we plot a line following $y = x$ indicating where correct predictions should lie.

For the 50% train - 50% test split, we found that only 3.5% of output queries were incorrectly determined. However the accuracy determined by the model finds a lower error rate than this, on the order of $\sim 1\%$; we must note that the model accuracy does not fully illustrate the fraction of predicted results that we correctly evaluated, since it may decide that determining 2 out of 3 characters correctly for an output query is accurate enough. As illustrated in Figure 1(a), most of the incorrect predictions lie very close to the correct prediction line. Only for some small cases does the model either under- or over-estimate the expected output (shown as blue points, differences of 80-90). Notice also that any cases of outputs below -50 are generally predicted accurately well, similar for those above 100 with a few minor exceptions.

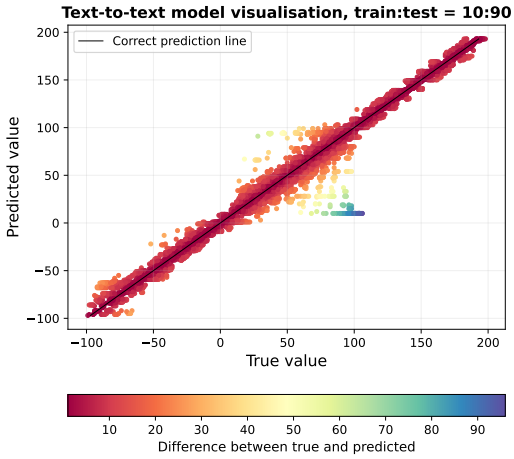
For the 25% train - 75% test split,, we begin to see some additional deviations relative to the previous split. As the training size is reduced, the model has less information to learn from and must begin to make some generalizations based on few data samples. Since the data is randomly shuffled, it may be possible that the model must learn without much information for some output ranges. Hence we find that there is an additional factor of randomness associated with the predicted results. We find that 22.2% of output queries are incorrectly predicted, an increase from the last split as one would expect. We again find under- and over-estimations scattered across the plot shown in Figure 1(b). As mentioned, we begin to see a larger scattering about the correct prediction line as an addition of randomness in predictions occurs. The randomness observed is likely associated with the model attempting to generalize from a small training set to a larger test set.



(a) 50% train - 50% test



(b) 25% train - 75% test



(c) 10% train - 90% test

Figure 1: Visualisation of test data predictions against expected outputs for text-to-text RNN model. The colourbar indicates the difference between the true and predicted values

For the 10% train - 90% test split,, we notice a dramatic increase in the percentage of incorrect test predictions, up to a staggering 82.6% of output queries. Once again we find that the model accuracy of $\sim$ 65% (see Table 3) does not reflect this large error since many predictions may only be missing one character from the true output query but the model deems this result as good enough. This shows how the model tries to make generalizations based on only 10% of the data and makes greater errors on the full extent of the test data. Although we find a large density of test predictions close to the correct prediction line as seen in Figure 1(c), there is also a wide spread of predictions with large differences from 40 to 90 as a result of this random behaviour. We also found a larger number of outliers (not visible in the figure) in the range [900,999] which we described previously.

Thus overall we find that increasing the test size significantly reduces the accuracy of the model as it attempts to generalize the addition/subtraction computation based on a small sample of training data. Such low train sizes (such as 10%) should be avoided for RNN models as the generalization capabilities are very weak.

1.2 Image to text model

In this section, we have built an image-to-text RNN model, which takes as input a sequence of MNIST images representing a query of either an addition or a subtraction and outputs the answer to the given operation in a text format. The architecture of this model is shown in Table 2. First an encoder is used to encode the image sequence received as input into a vector of size 256. Then, this encoded representation is repeated for a number of times equal to the maximum length of the answer, which in our case is 3. After a decoding step, the model applies a dense layer to each temporal slice of the output, allowing us to obtain in text form the answer to the given operation.

Layer type	**Parameters**
LSTM	u=256
RepeatVector	rf=3
LSTM	u=256
TimeDistributed	l=Dense
Dense	u=13

Table 2: Architecture of image-to-text RNN model.

We have trained the model for 100 epochs on the test set for different test sizes (50%, 75%, 90%). Table 3 shows the accuracy obtained on the test set for different splits with both the text-to-text and the image-to-text models. As expected, for both models, the accuracy decreases as we increment the size for the test set; indeed in this way the model has less data to learn from and train on, resulting therefore in lower performance. By comparing the two models, we can see that the text-to-text RNN reaches much higher accuracy with respect to the image-to-text model. This occurs because computing the result in text form of an operation which is also inputted in text form is a much simpler task than computing the result when the operation is given as an image. In the image-to-text case, the network would require much more training to achieve a better performance.

Split (train:test)	**Text-to-text accuracy**	**Image-to-text accuracy**
50:50	0.9875	0.4638
25:75	0.9185	0.4110
10:90	0.6491	0.3592

Table 3: Accuracy obtained with the text-to-text and image-to-text models for different dataset splits.

Lastly, we attribute the relatively low accuracies for the image-to-text model to the fact that we applied regular LSTM cells (with flattened images as vectors) to the model as opposed to the alternative of using recurrect convolutional layers ConvLSTM2D. While we thought using the convolutional layers would increase the accuracies, due to the time constraints and computational difficulties we encountered in implementing this, we decided against it.

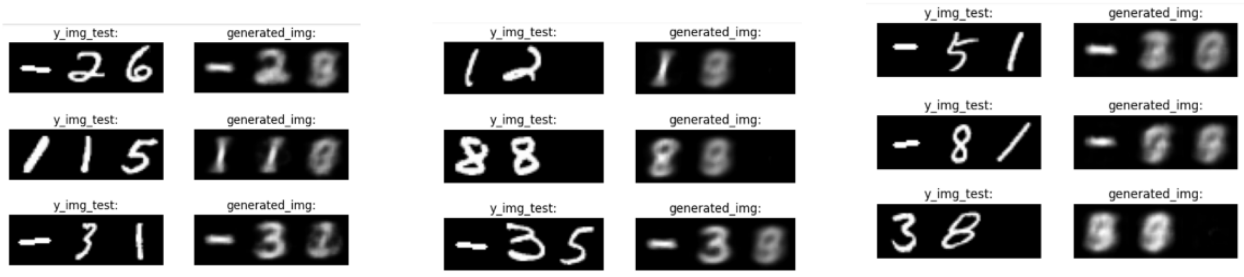
1.3 Text to image model

For this part of the task, we have built a text-to-image RNN model which takes as input an operation (addition or subtraction) in a text form, and outputs the result as an image. The architecture that we have employed is similar to the one for the previous model, just with small changes to the input dimensions to adapt it for input images (see Table 4). In this case, before feeding the 28x28 input images to the model , we have reshaped them. We have encoded the inputs using an RNN and produced an output of dimension 255, then we have repeated the encoded representation for the maximum answer length and employed a decoder. Finally, to obtain the result of the operation in text form, we have applied a dense layer to each temporal slice of the output.

Layer type	**Parameters**
LSTM	u=256, input_shape=(5, 784)
RepeatVector	rf=3
LSTM	u=256
TimeDistributed	l=Dense
Dense	u=13

Table 4: Text-to-image RNN model architecture

In this case, the accuracy does not turn out to be a strong indicator for the performance of our model, since the output is represented by images and it is not in text form. Therefore, to better understand how our model performed, we have visualized the generated images for the three different splits of the test and train sets. By looking at Figure 2, showing the generated images, we can observe that, as expected, the model is more accurate for the 50% train - 50% test split, since this is the case where the train size is larger, meaning that the model has more data to learn from. We can notice that in all the three cases the last digit of the result is the one which has been more often predicted incorrectly. This can be attributed to the fact that since the model we have employed is rather simple, it has some difficulties in retaining long-term dependencies. Being at the beginning of the sequence, the information of the first two digits is more easily preserved, but as the sequence progresses, the model struggles to capture dependencies that span a long context, resulting in the last digit to be more difficult to correctly predict and generate.



(a) 50% test - 50% train (b) 75% test - 25% train (c) 90% test - 10% train

Figure 2: Output images of the text-to-image model for different test-train splits

1.4 Additional LSTM layers

In this part of the assignment, we have considered again the three models that we have previously developed (text-to-text, image-to-text and text-to-image) and we have added more LSTM layers to the encoder of each model to see the effects on the performance. In particular, for each model we have again considered the three possible test-train splits and we observed the changes in performance by adding one, two and three LSTM layers to the encoder.

1.4.1 Adding more layers to the text-to-text model

Considering the text-to-text model, the accuracy obtained for the different test-train splits by adding several LSTM layers to the encoder can be seen in Table 5. For the 50% train - 50% test split, we have observed that the addition of first one and then two LSTM layers slightly improve the model performance thanks to a resulting more complex architecture of the model. However, by inserting an additional third LSTM layers, the accuracy is decreased so we have deduced that in this case the model has over-fitted the data by adding too many layers to the encoder. In comparison, we see a small steady increase in the performance for the 10% train - 90% test split with more LSTM layers. We believe that as the model architecture becomes deeper and more complex, it becomes easier to make generalizations for a small train size and apply to the test data. Hence it is evident that a more complex text-to-text encoder architecture is most useful for a smaller training size.

Adding layers	**50% train - 50% test**	**25% train - 75% test**	**10% train - 90% test**
1 LSTM	0.9980	0.8455	0.6325
2 LSTM	0.9915	0.9187	0.6535
3 LSTM	0.8284	0.7029	0.6706

Table 5: Accuracy obtained with the text-to-text model for different train-test sets splits and adding extra LSTM layers to the encoder.

1.4.2 Adding more layers to the image-to-text model

The results that we have observed by adding more LSTM layers to the image-to-text model's encoder are reported in Table 6. We can observe that for all of the three different train-test splits, the addition of more

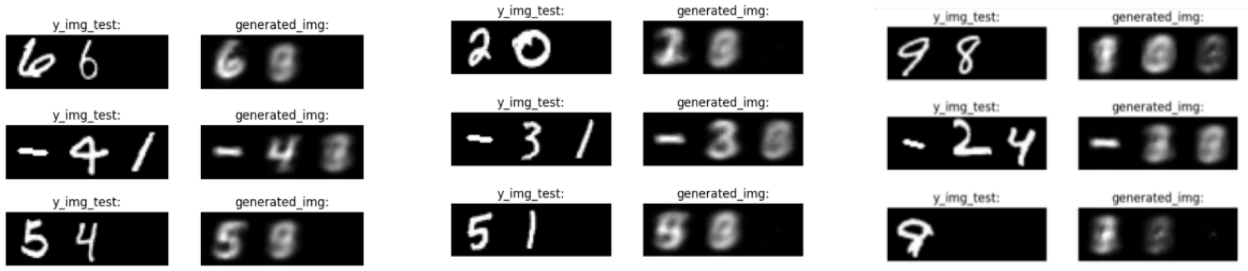
LSTM layers to the encoder architecture leads to an higher accuracy of the model; indeed a deeper model architecture allows the model to have a better understanding of the input and therefore to predict the outputs more accurately.

Adding layers	50% train - 50% test	25% train - 75% test	10% train - 90% test
1 LSTM	0.5144	0.4485	0.3736
2 LSTM	0.5616	0.4873	0.3990
3 LSTM	0.5947	0.5113	0.4047

Table 6: Accuracy obtained with the image-to-text model for different train-test sets splits and adding some LSTM layers to the encoder.

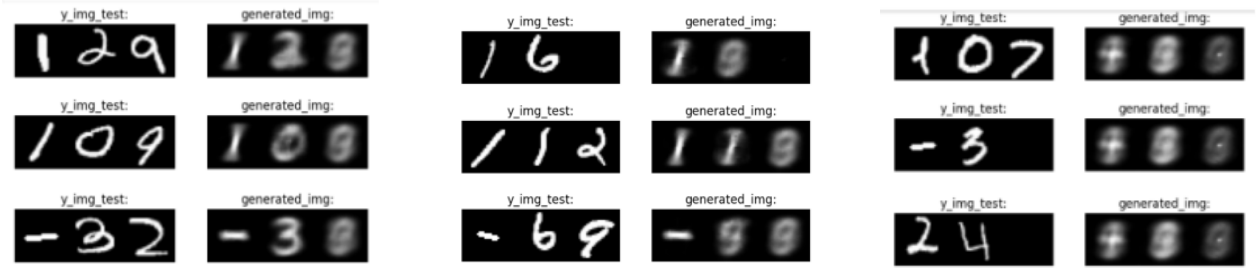
1.4.3 Adding more layers to the text-to-image model

As we have already mentioned before, looking at the accuracy obtained with the text-to-image model is not the best method to assess the performance and therefore in this case we have looked at the generated images obtained as outputs of the text-to-image model. By looking at Figure 3, we can observe that inserting one additional LSTM layer to the encoder of our model doesn't affect much the generated images for the 50% train - 50% test and 25% train - 75% test splits. However, when we have used 90% of the dataset for the test set, we can see that the addition of one LSTM layers over-fits the model, resulting in less accurate output images. The addition of two LSTM layers doen't affect much also the case for the 50% train - 50% test split as can be seen from Figure 4; maybe a slightly less accurate result can be obtained on the 25% train - 75% test split compared to the original architecture, but a significant deterioration of the output images can be clearly observed on the 10% train - 90% test split. Finally, as for the addition of three more LSTM layers to the encoder's architecture, this time both the 25% train - 75% test and 10% train - 90% test splits show less accurate output images when compared to the original architecture: probably because of overfitting, the model cannot predict the output images with the same accuracy as before. Still, no particular changes are observed for the 50% train - 50% test split (see Figure 5).



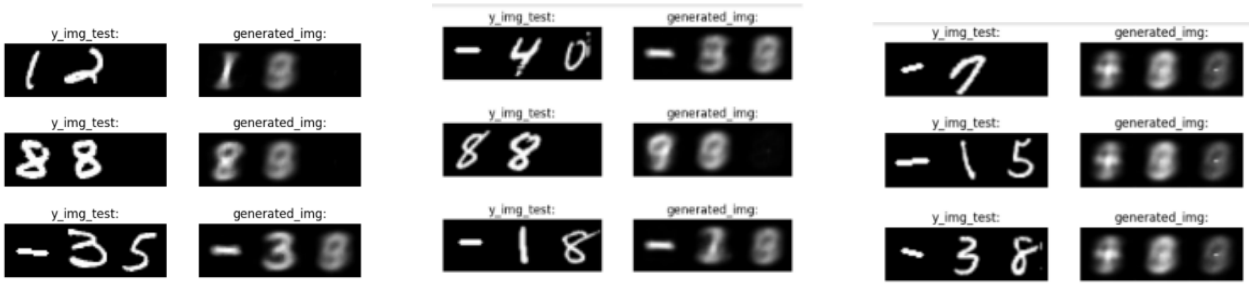
(a) 50% test - 50% train (b) 75% test - 25% train (c) 90% test - 10% train

Figure 3: Output images of the text-to-image model for different test-train splits when inserting one additional LSTM layer to the encoder



(a) 50% test - 50% train (b) 75% test - 25% train (c) 90% test - 10% train

Figure 4: Output images of the text-to-image model for different test-train splits when inserting two additional LSTM layers to the encoder



(a) 50% test - 50% train (b) 75% test - 25% train (c) 90% test - 10% train

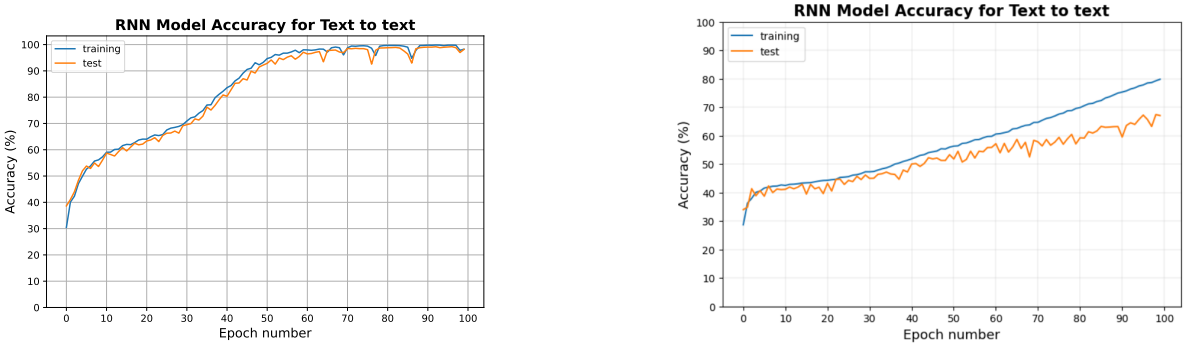
Figure 5: Output images of the text-to-image model for different test-train splits when inserting three additional LSTM layers to the encoder

2 Multiplication

In this section, we now take two of the models we have worked with above and apply them to the multiplication of MNIST digits in the range of [0,99]. We now find the expected output range to be [0,9801] since there will no longer be negative digits. For all text strings, "hot encoded" vectors are used as input and extracted from the model as output just as before. We compare our results to that obtained from the addition/subtraction dataset, and how each operation differs in such models.

2.1 Text to text model

Adapting our text to text model for addition and subtraction to multiplication was relatively simple. For the encoder the maximum length of digits had to be increased to 4 to accommodate the increase in expected output range, any considerations made for negative values was also removed. Below is a comparison of the accuracy's for the model on the different datasets:



(a) Addition and Subtraction (b) Multiplication

Figure 6: Comparison of text-to-text model accuracy for addition/subtraction and multiplication datasets. In both cases, a train-test split of 50:50 was used.

From the figures above, it is clear that the generalization capabilities for the model drop significantly when applied to the multiplication dataset from 92% (On the addition and subtraction dataset) to 76%. Furthermore it is apparent that there is greater divergence of the training and test accuracies for multiplication compared to that for addition and subtraction. This is all of course expected as generalizing multiplication introduces another level of complexity compared to addition and subtraction. In the multiplication dataset, the model needs to handle a much wider range of output values (close to 10,000) and the relationships between input and output become more complicated.

The increase in maximum length for digits in the encoder to 4 is necessary to account for the expanded range of possible results. Additionally, the removal of considerations for negative values reflects the specific nature of multiplication, where the result is inherently non-negative. This decrease indicates the need for

further adaptation to enhance its ability to capture the intricacies of multiplication operations. Table 7 below also shows the consistent decrease in accuracy amongst the different train/test data splits.

Split (train:test)	Text-to-text Accuracy
50:50	0.7684
25:75	0.6310
10:90	0.5112

Table 7: Accuracy obtained with the text-to-text model on the multiplication dataset for different splits.

2.2 Image to text model

For this section, we have taken the image-to-text RNN model discussed previously, and once again applied it to images which now represent queries of multiplication operations. The model then attempts to output the answer to the given operation in a text format. The architecture for the model remains the same as with the text-to-text model as the data is relatively similar to addition and subtraction. The encoders used for the multiplication text-to-text model were again used, which only differ in terms of maximum output length. The results for each split are displayed in Table 8, and compared to the results obtained for the same model on the addition and subtraction dataset.

Split (train:test)	Addition/Subtraction accuracy	Multiplication Accuracy
50:50	0.4638	0.3173
25:75	0.4110	0.2740
10:90	0.3592	0.2303

Table 8: Accuracy obtained with the image-to-text model on the addition/subtraction and multiplication datasets for different splits.

As observed in Table 8 above, much like when comparing the text-to-text model with the two different datasets previously, the results obtained using the multiplication dataset were found to be much poorer relative to that for addition and subtraction. The drop in the accuracy value is however consistent amongst the different train/test splits, likely indicating the model is consistent in its training regardless of the dataset chosen and would again need further adaptation to generalise higher accuracies for the more complex multiplication dataset.

2.3 Additional LSTM layers

As in Section 1.4, we discuss the impact on the performance of both the text-to-text and image-to-text models we have built for the multiplication dataset when more LSTM layers are added to the encoder of each model. For consistency, we apply this to the same three train-test splits we considered previously. The results of this can be observed in Tables 9 and 10 for each of the text-to-text and image-to-text models respectively.

Comparing to Table 7, we see that adding additional LSTM layer to the text-to-text model causes the performance to decline for both 50-50 and 25-75 splits. This implies that the model begins to overfit as an added complexity to the architecture is introduced. On the other hand, for the 10%-90% train-test split, we find a small boost in the performance which is similar to what we observed previously for the addition/subtraction dataset. This again indicates that adding more LSTM layers allows the model to improve its generalization capabilities for a limited training set compared to the same applied to a larger training set where the model begins to overfit. Hence the best text-to-text model performance (when applied to the test set) is found for a 50-50 train-test split when no additional LSTM layers are included.

When adding to the image-to-text model, we find a substantial increase in the model performance for all splits with extra encoder layers relative to what we found in Table 8. Since in this case we have image inputs, it follows that additional layers would allow the model to discern more difficult features from such images and classify them appropriately, enhancing the overall performance. Therefore, compared to the text-to-text

model, the best image-to-text model performance is found for a 50-50 train-test split when three additional LSTM layers are included.

Adding layers	**50% train - 50% test**	**25% train - 75% test**	**10% train - 90% test**
1 LSTM	0.7548	0.6106	0.5284
2 LSTM	0.7086	0.5849	0.5321
3 LSTM	0.6691	0.5832	0.5354

Table 9: Accuracy obtained with the multiplication text-to-text model for different train-test sets splits and adding extra LSTM layers to the encoder.

Adding layers	**50% train - 50% test**	**25% train - 75% test**	**10% train - 90% test**
1 LSTM	0.4774	0.4350	0.3877
2 LSTM	0.5028	0.4466	0.3960
3 LSTM	0.5119	0.4492	0.3959

Table 10: Accuracy obtained with the multiplication image-to-text model for different train-test sets splits and adding extra LSTM layers to the encoder.

Concluding remarks

We have investigated using RNN model architectures to compute operations such as addition, subtraction and multiplication on both text and image representations of MNIST digits. We found that the simplest model is evidently the text-to-text model, as the encoded representation of digit characters is quite simple for the model to understand. The complexity grows as we introduce images as inputs, and the model performance begins to decline. The same is true for text-to-image models for which we must discern the discrepancies by eye. In addition, we found that adding extra LSTM layers to the encoder of the network consistently improved the generalization capability when applied to a small training set (aside from text-to-image).

When attempting to apply the models used on the addition and subtraction dataset to the multiplication dataset, we were at first surprised at how little adaptation was required to apply the same models to the multiplication dataset. However we realised that since the data is at its basis the same, the only difference would be the slightly longer training time and the decrease in performance due the higher complexity of the data which is reflected in our results. The performance improved for the image-to-text model on this dataset with the addition of LSTM layers to the encoder part of the network. This illustrates that for image-based RNN models, a more complex architecture is necessary to achieve better performance while text-based RNN models work incredibly well using a very simple architecture.