Lab assignment 1: Image recognition using deep networks

By Ben Harvey

In these exercises, you will explore questions of visual image processing using the Keras library for RStudio. If you do not have experience with RStudio or R, we recommend that you first complete Chris Janssen's 'Crash Course' in R (on Blackboard), which will familiarise you with the basics of R.

Keras is a high-level library for building artificial machine learning networks, specialising deep learning networks. RStudio is used to address Keras, while Keras primarily calls the open-source TensorFlow library, developed by the Google Brain team. TensorFlow in turn is written in Python and C++. As a result, Python must be installed too, which is typically done automatically when installing Keras.

Keras is therefore best viewed as an interface for TensorFlow, while TensorFlow is the underlying machine learning framework. Keras gives a more intuitive set of functions, making it easy to develop machine learning models, particularly in the context of deep learning networks. Keras, like TensorFlow and Python, has excellent cross-platform support, and importantly makes it very easy to run models on their CPUs, GPUs or clusters of either. Because artificial learning networks rely on many simple computations running in parallel, using GPUs is ideal for this application.

Because we will use relatively simple networks in this class, it is feasible to run these exercises on your own computers. However, modern laptops vary considerably in their CPUs and GPUs. A faster GPU will considerably reduce your waiting times, so if you have a choice of computers, choose the one with the better graphics card.

Remote group work: A personal, interactive process

You should work in groups of 4 on these exercises, with help from a single teacher who is assigned to your group. We expect you to work on these exercises in class time so you can work with your group and teacher. It is not acceptable to miss these classes without agreement from your group, or to repeatedly miss classes. Then you will fail the assignment, which makes it very hard to pass the course. If you won't be able to attend these lab classes, you should leave the course now: these make up most of the course. If your group members miss lab classes without agreement from your group, please inform your teacher.

We suggest all group members doing these exercises on your individual computers simultaneously: this improves (student) learning and also makes it easier to find mistakes. Don't rely on other group members' answers if you don't understand why they are correct: this is meant to be an interactive collaboration with your group, so ask your group members to explain. If your group gets stuck on a question or different group members can't agree on an answer, ask for help from your teacher. Please share your video if bandwidth allows, at least when talking with your teacher.

When your group is happy with your answer, work together to finalise your answer in a document shared with the whole group. Google docs is an excellent platform for working together on a shared document. Show these answers to your teacher as you work. You can share this document with the teacher too. It's often best to show your teacher answers for a few questions together, but you should always check your answers with your teacher after each few questions (so they can see you have completed them successfully). Your teacher will grade you as you work to monitor your progress and address problems. But we need a record of all your answers, submitted at the end of the assignment. Your teacher should be checking your final answers in this document as you work.

Many questions begin 'Discuss with your group, then describe to your teacher…'. In questions like this, it is generally best to start by asking every group member's opinion. Then work on a written answer together. Then explain your answer to your teacher. You can also ask your teacher to read what you wrote, but they will often ask questions. You may like to update this answer after talking with your teacher, but please tell your teacher what changes you made next time you talk.

Many questions build on previous questions being completed correctly, so you should be confident of your answer before using it in further questions: ask for help if you are unsure. But there are various points clearly marked (NEW TOPIC) where you do not rely on previous answers. If you get stuck and can't get help immediately, you can move on to the next topic until your teacher can help.

In the following text, explanation is plain text, instructions are underlined, questions to answer are labelled, and RStudio console commands are written in `different font`. The number of points available for each questions gives an idea of the expected depth of your answers.

Installation:
If you do not already have R installed, download it here and install it: https://cran.rstudio.com/. R is installed in many Linux distributions already.

If you do not already have RStudio installed, download it here and install it after installing R: https://www.rstudio.com/products/rstudio/download/#download

Open RStudio. In RStudio's console, install Keras using the following commands:

```
install.packages("keras")
install.packages("kerasR")
```

You may find that you this prompts you to install Python, go ahead.

Now load the Keras library. You may need to do this each time you restart RStudio.

```
library(keras)
library(kerasR)
```

Now install all the libraries that Keras relies on (like TensorFlow):

```
install_keras()
```

Exercise one: Identifying handwritten numbers

We will begin using a very simple image recognition example: classifying hand-written numbers. This is a very useful ability for computers as it allows mail carriers to read hand-written postal codes and house numbers, and thereby sort mail automatically. It also allows banks to read numbers from cheques.

Discuss with your group, then describe to your teacher, a list of applications where automatic recognition of hand-written numbers would be useful. (Question 1, 3 points)

We will use a database of labelled handwritten numbers, called MNIST.
First, download MNIST:

```
mnist <- dataset_mnist()
```

MNIST contains a training set of 60,000 grayscale images (each 28x28 pixels) of hand-written numbers (mnist$train$x) together with the numbers shown in each image, the labels (mnist$train$y). It also contains a similar test set of 10,000 images and labels (mnist$test$x and mnist$test$y).

Here we will use two types of artificial learning network. Before using a deep convolutional network, we will test a multilayer perceptron. This is a type of artificial neural network where all nodes in each layer are connected to all nodes in the next layer. Therefore, this is not a convolutional network because there is no spatially-restricted convolutional filter, and no use of spatial relationships.

Data preparation:
So we will start by flattening the two spatial dimensions to convert from a 60000x28x28 training set to a 60000x784 training set in which spatial relationships are removed.

Use the functions `array_reshape` and perhaps `nrow` to convert the flatten training and test set images from 28x28 pixels to a column of 784 pixels for each image (row). Save the results as new variables called x_train and x_test to avoid overwriting the original images. These should have dimensions 60000x784 and 10000x784 respectively (use the function `dim` to check this). You can use the help file for `array_reshape` to see how to call this function properly, using `help("array_reshape")`.

Now rescale x_train and x_test to values between zero and one by dividing each variable by 255.

The labels are currently specified as numbers between 1 and 10. For our network, these numbers must each be separate network units in the output layer. Use the function `to_categorical` to convert the train and testset labels to two new variables, called y_train and y_test. For each label, there should be 10 elements, 9 of them zeros and 1 of them a one.

Model definition:
You will want to keep a record of the code you write here so you can easily make modifications and run it again.
For our multi-layer perceptron (MLP), we will pass our 784 input units (flattened pixels) into a 256-unit fully connected hidden layer. This in turn feeds into our label (output) layer whose activation follows a softmax function to give the probability that each image is each digit. We initialise this model using the function `keras_model_sequential`. The fully-connected layers are defined by the Keras function `layer_dense` and linked using the pipe operator `%>%`.

So, to make the MLP model described above, enter the following code:

```
model <- keras_model_sequential()
model %>%
   layer_dense(units = 256,  input_shape = c(784)) %>%
   layer_dense(units = 10, activation = 'softmax')
```

To check the resulting model is what you expect, use:

```
summary(model)
```

If this looks OK, compile the model with suitable loss functions, optimisation procedures, and performance measures, as follows:

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

Training and evaluation
Now we will fit the model to our training set, keeping a history of the performance at each stage. We will use 12 training epochs. We will set aside a random 20% of our data to check performance at each training epoch (which don't change between epochs). In each training epoch, we will use 128 image-label pairs per batch to improve computational efficiency. We will use the `verbose` argument to tell us what is happening in each epoch.

To fit the model as described above, enter the following code:
```
history <- model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 12,
  verbose = 1,
  validation_split = 0.2
)
```

Do questions 2-6 (below) and show your teacher all of the results together.

For questions where you generate a plot, table or code, you should copy this in your answer document (maybe using a screenshot).

Show your teacher the text from your console, with how long it took for each epoch to run and the training performance history. (Question 2, 5 points)

Plot the training history and show this to your teacher (Question 3, 3 points)

Discuss with your group, then describe to your teacher, how the accuracy on the training and validation sets progress differently across epochs, and what this tells us about the generalisation of the model. (Question 4, 5 points).

Evaluate the model performance on the test set using the following command:
```
score <- model %>% evaluate(
  x_test, y_test,
  verbose = 0
)
```

Show your teacher what values you get for the model's accuracy and loss. (Question 5, 2 points)

Discuss with your group, then describe to your teacher, whether this accuracy is sufficient for some uses of automatic hand-written digit classification. (Question 6, 5 points)

Changing model parameters
Do questions 7-9 (below) and show your teacher all of the results together.

In the previous model, we did not specify an activation function for our hidden layer, so it used the default linear activation. Discuss with your group, then describe to your teacher, how linear activation of units limits the possible computations this model can perform. (Question 7, 5 points)

Now make a similar model with a rectified activation in the first hidden layer, by adding the extra argument:
```
activation = “relu”
```
to the model definition for this layer. Then compile, fit and evaluate the model.

Plot the training history and show it to your teacher (Question 8, 2 points)

Discuss with your group, then describe to your teacher, how this training history differs from the previous model, for the training and validation sets. Describe what this tells us about the generalisation of the model.
(Question 9, 5 points)

Deep convolutional networks
NEW TOPIC

Our first two models used fully-connected networks for number recognition. They ran quickly, largely because of their very simple structure. They learned the relationships between our pixels and the numbers they represent fairly well. However, they had a limited ability to generalise to new data that they were not trained on.

In deep convolutional networks, each convolutional filter samples from the limited space in the previous layer's feature map. To use the whole image to determine outputs, they need more layers to allow more spatial integration. They also need multiple feature maps at each layer to capture the multiple meaningful spatial relationships that are possible. All of this greatly increases computational load.

First, we need to prepare our data differently. Convolutional layers don't flatten x and y spatial dimensions, and need an extra dimension for colour channels (in the input image) or multiple feature maps (from previous convolution steps). Reshape mnist$train$x to a new variable (x_train) of size 60000, 28, 28, 1. Reshape mnist$test$x to a new variable (x_test) of size 10000, 28, 28, 1. Rescale both results to values between zero and one as before. y_train and y_test are categorical units, as before.

Now we will define a convolutional learning model with 2 convolutional layers that result from 32 convolutional filters into the first layer and 64 filters into the second. We will use 3x3 pixel filters to sample from the image to the first layer, and the same to sample from the first layer to the second. We will use rectified activation functions for both convolutional layers. We will use pooling to downsample the second convolutional layer to half its size in both spatial dimensions (so one quarter of the pixels). We will flatten the resulting feature map to one dimension, then use one fully-connected layer to link our network to the labels.

To make the (slightly deep) convolutional network model described above, enter the following code:

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3),
      activation = 'relu', input_shape = c(28,28,1)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3),
      activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

Now summarise the model to check it is what you want.
When compiling the model, we will use a slightly different backpropagation of error procedure, so change the 'optimizer' part of the compile command to:

```
optimizer = optimizer_adadelta(),
```

Fit the model as before, but using only 6 epochs. Expect that model fitting will take far longer.

Do questions 10-15 (below) and show your teacher all of the results together.

Plot the training history and show this to your teacher. (Question 10, 2 points)

Discuss with your group, then describe to your teacher, how the training history differs from the previous model, for the training and validation sets. What does this tell us about the generalisation of the model?
(Question 11, 5 points)

Show your teacher what values you get for the model's accuracy and loss. (Question 12, 2 points)

Discuss with your group, then describe to your teacher, whether this accuracy is sufficient for some uses of automatic hand-written digit classification. (Question 13, 5 points)

'Dropout' is a method used in deep network training to prevent overfitting of training date and focus on aspects of the learning model that will generalise to new data. Your Individual Assignment includes a question about this, so you may like to research that answer now to better understand this process.

Add dropout layers (`layer_dropout()`) after the max pooling stage (rate = 0.25) and after the fully-connected (dense) layer (rate = 0.5). Compile and train the resulting model as before.
Discuss with your group, then describe to your teacher, how the training history differs from the previous (convolutional) model, for both the training and validation sets, and for the time taken to run each model epoch (Question 14, 3 points)

Discuss with your group, then describe to your teacher, what this tells us about the generalisation of the two models. (Question 15, 3 points)

Exercise two: Identifying objects from images

NEW TOPIC
Identifying objects from images with a high variation in object position, size and viewing angle is a particularly difficult problem, and a major application of artificial deep learning networks. Here we will build a network to do this. This network has more layers and is more computationally-intensive than previous exercises, even using the low-resolution images we will use here.

The CIFAR-10 dataset contain colour images of objects, each 32x32x3 pixels (for the three colour channels). These have 10 categories (or classes) of object (airplane, automobile, bird, cat, deer, dog, frog, horse, ship & truck) with 5,000 images in each, making a total of 50,000 images in the training set (cifar10$train$x), randomly ordered with numerical labels for each (1=airplane, 2=automobile etc.). The test set (cifar10$test) contains 10,000 images ordered by their label.

First, download and load the data set using:
```
cifar10 <- dataset_cifar10()
```

Prepare the test and training images by dividing their values by 255, storing the result in variables x_train and x_test. The shapes of the image matrices are already correct for input into Keras. Convert the training and test labels to categorical variables, as before with the handwritten digits, storing the result in variables y_train and y_test.

Define the model using the convolutional network with dropout (from Questions 10 and 15) as a template.
-For the first convolutional layer, add the arguments:
`input_shape = c(32, 32, 3), padding = “same”`
to the layer_conv_2d call.
- In the second convolutional layer, use 32 filters instead of 64 to reduce computational load.
-After max pooling and dropout layers, repeat these layers again (add conv, conv, pool, dropout, after the existing conv, conv, pool, dropout). There is no need to define input_shape here.
-Flatten the result and link it to a larger fully-connected layer than before, using 512 units instead of 128, with dropout as before.
-Link this to a 10-unit output layer as before.

Before fitting the model, show your teacher the code you used to define the model described here. (Question 16, 6 points)

In compiling the model, we will use a specialised optimizer module:
`optimizer = optimizer_rmsprop(lr = 0.0001, decay = 1e-6)`

Next we will fit the model, which **will take a couple of hours at least**. During fitting, there are other exercises to do (Exercise three, below, and your individual assignment), so you don’t need to wait. You may choose to run this overnight, particularly if each training epoch lasts more than 10 minutes.

In the model fit command, set the batch size to 32 and the number of epochs to 20. Rather than splitting the training data to give a validation set, we will pass in the test set as follows:
`validation_data = list(x_test, y_test),`
Finally, set:
`shuffle = TRUE`
to run through the images in a different order of batches each epoch.

Execute this model fit command. After your fitting is finished, do questions 17-19 (below) and show your teacher all of the results together.

Plot the training history and show it to your teacher (Question 17, 2 points)

Discuss with your group, then describe to your teacher, how the training history differs from the convolutional model for digit recognition and why. (Question 18, 5 points)

Discuss with your group, then describe to your teacher, how the time taken for each training epoch differs from the convolutional model for digit recognition. Give several factors that may contribute to this difference (Question 19, 4 points)

Exercise three: Low-level functions

NEW TOPIC
Deep learning relies on a few basic operations: convolution, nonlinear activation, pooling, and normalisation. Backpropagation of error is used to learn connection weights, and a fully-connected layer linked to output nodes with a softmax function are also required. So far, we have used calls to Keras's high-level libraries to do these operations, as these are efficiently coded and easy to use. But it is also important that you understand the basic functions at a low level.

In each of the following answers, try to make your code efficient, using matrix operations where possible. You will lose a few marks for code that works through each element in turn, though if you are finding the matrix operations difficult you can use element-by-element calculations to show you understand the principles involved.

Show the first answer to your teacher, as their feedback often helps with later questions.

Write a simple function that achieves the convolution operation efficiently for two-dimensional and three-dimensional inputs. This should allow you to input a set of convolutional filters ('kernels' in Keras's terminology) and an input layer (or image) as inputs. The input layer should have a third dimension, representing a stack of feature maps, and each filter should have a third dimension of corresponding size. The function should output a number of two-dimensional feature maps corresponding to the number of input filters, though these can be stacked into a third dimensional like the input layer. After agreeing on a common function with your group members, show this to your teacher. (Question 20, 5 points)

Write a simple function that achieves rectified linear (relu) activation over a whole feature map, with a threshold at zero. After agreeing on a common function with your group members, show this to your teacher. (Question 21, 2 points)

Write a simple function that achieves max pooling. This should allow you to specify the spatial extent of the pooling, with the size of the output feature map changing accordingly. After agreeing on a common function with your group members, show this to your teacher. (Question 22, 3 points)

Write a simple function that achieves normalisation within each feature map, modifying the feature map so that its mean value is zero and its standard deviation is one. After agreeing on a common function with your group members, show this to your teacher. (Question 23, 2 points)

Write a function that produces a fully-connected layer. This should allow you to specify the number of output nodes, and link each of these to every node a stack of feature maps. The stack of feature maps will typically be flattened into a 1-dimensional matrix first. After agreeing on a common function with your group members, show this to your teacher. (Question 24, 5 points)

Write a function that converts the activation of a 1-dimensional matrix (such as the output of a fully-connected layer) into a set of probabilities that each matrix element is the most likely classification. This should include the algorithmic expression of a softmax (normalised exponential) function. After agreeing on a common function with your group members, show this to your teacher. (Question 25, 2 points)

Backpropagation of error:
So far, we have avoided explaining backpropagation in detail. We have discussed what backpropagation does, and used Keras's implementation to train our networks. But we have not looked at how it works because the mathematics are complex and do not fit well with the goals of this course. AFTER completing the Individual Assignment's questions on backpropagation, you may choose to answer the bonus questions below.
In these questions, it is common for students to start making functions, but be unable to produce an effective function: they are meant to be challenging. If you get part way through ad give up, show your teacher your incomplete answers.
Your teacher will prioritise help for students still working on the main assignment, so don't expect much help with the bonus questions.

BONUS QUESTION: Write a function to achieve backpropagation of error to affect the convolutional filter (kernel) structure used in Question 20. Modify your function from Question 20 so you can input the convolutional filters, allowing you to modify these filters using backpropagation (outside the convolution function). Initialise the network with random weights in the filters. After agreeing on common functions for convolution with your group members, show your teacher how you changed this from the answer given in Question 20. Show your teacher your code for backpropagation (Question 26, 5 points)

BONUS QUESTION: Write a piece of code that uses all of these functions (from Questions 22-28) together to make a convolutional neural network with two convolutional layers, a fully connected layer, and an output layer (pooling is optional, but thresholding and normalisation are required). This should give the accuracy of the labels as an output. Show the resulting code to your teacher. (Question 27, 5 points)

BONUS QUESTION: Use the resulting function to learn to classify the mnist data set, as you did in question 10. Plot the progression of the classification accuracy over 100 cycles. Show the resulting plots to your teacher. (Question 28, 5 points).