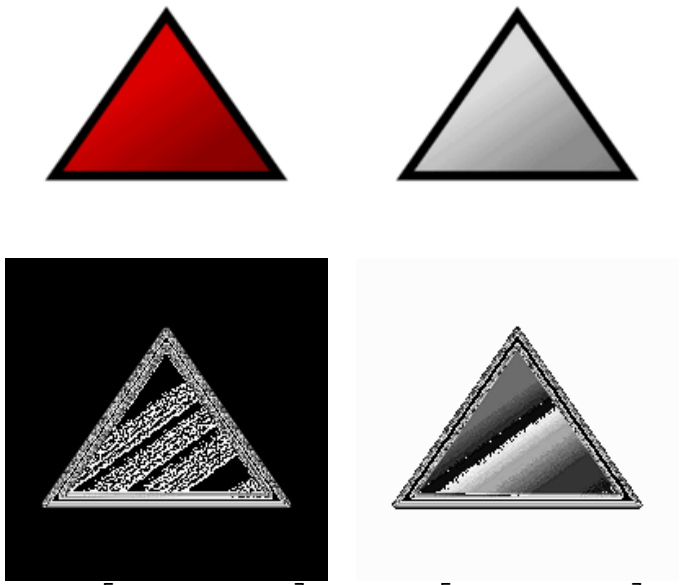

$$k_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad k_2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 1: A 200x200 RGB image (top left) has green and blue channels removed resulting in a grayscale image (top right). The image is convolved with two kernels, k_1 (bottom left), and k_1 (bottom right).

DECLARATION: I understand that this is an **individual** assessment and that collaboration is not permitted. I have read, understand and agree to abide by the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar. I understand that by returning this declaration with my work, I am agreeing with the above statement.

1 (i)

The results of a 2D convolution function applied to a single-channel image, with two different kernels, are presented in Figure 1.

2 (ii)

2.1 (ii) (a) model layers, kernels, channels

Below is the python source code for a CNN with 4 convolution layers. The input to the model is a tensor with shape (32,32,3), i.e. an RGB image with 32x32 pixels.

src/model_architecture.py Tue Nov 07 16:15:59 2023 1

```
 1: from tensorflow import keras
 2: from tensorflow.keras import layers, regularizers
 3: from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
 4: from keras.layers import Conv2D, MaxPooling2D, LeakyReLU
 5: def make_model(num_classes,x_train):
 6:     model = keras.Sequential()
 7:         model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
 8:         model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
 9:         model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
10:         model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
11:         model.add(Dropout(0.5))
12:         model.add(Flatten())
13:         model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
14:     return model
```

The CNN layers of the model have the following structures:

1. (line 7): input=(32,32,3), number of kernels=16, kernel shape=(3,3,3), output shape=(32,32,16)
2. (line 8): input=(32,32,16), number of kernels=16, kernel shape=(3,3,16), output shape=(16,16,16)
3. (line 9): input shape=(16,16,16), number of kernels=32, kernel shape=(3,3,16), output shape=(16,16,32)
4. (line 10): input shape=(16,16,32), number of kernels=32, kernel shape=(3,3,32), output shape=(8,8,32)

The next layer is a dropout layer which randomly sets on average 50% of its inputs to 0 and leaves the rest of the inputs the same. Its input and output shape is (8,8,32). The next layer simply unravels the (8,8,32) tensor into an array of length $2048 = 8 \cdot 8 \cdot 32$. The final layer consists of 10 separate linear combinations of the previous layers outputs. The output of this 'Dense' layer is just the 'softmax' function applied to the vector of ten linear combinations, $[z_1, \ldots, z_{10}]$.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}} \tag{1}$$

$$\text{output of dense layer} = [\text{softmax}(z_0), \text{softmax}(z_1), \ldots, \text{softmax}(z_{10})] \tag{2}$$

2.2 (ii) (b) (iii)

Figure 2 presents plots of the 'histories' of the training losses and accuracies on training/validation data for a sequence of models trained on 5K, 10K, 20K, and 40K training samples. Naturally, the model with most training data achievs the lowest loss and highest accuracy on the validation data. For 5K training samples the gap between training and validatin scores starts to increase after about 10 epochs, indicating over-fitting. In particular, by the 20th epoch the accuracy on the training data is higher than the accuracy on the validation data and the loss on the training data is lower than the loss on the validatin data. With 40K training samples there is a disimprovement at the 16th epoch, but the 17th, 18th, 19th and 20th epoch scores do not indicate significant over-fitting.

The general reading we can take from Figure 2 is that more training data allows us to train for more epochs without over-fitting, or without over-fitting as much.

2.3 (ii) (b) (iv)

Figure 4 presents plots of the 'histories' of the training losses and accuracies on training/validation data for a sequence of models with $L_1 \in \{0.0, 0.00001, 0.01, 100\}$.

After 20 epochs the model with $L_1 = 0.00001$ had a train loss below val loss, and train accuracy above val accuracy, which indicates over-fitting, but the model trained with $L_1 = 0.01$ exhibits an opposite pattern, where val score is better than thet train score. However, while the model with $L_1 = 0.00001$ shows more signs of being fitted too closely to the training data, the accuracy of the model on validation set is better than the accuracy of the more regularized model, $L_1 = 0.01$.

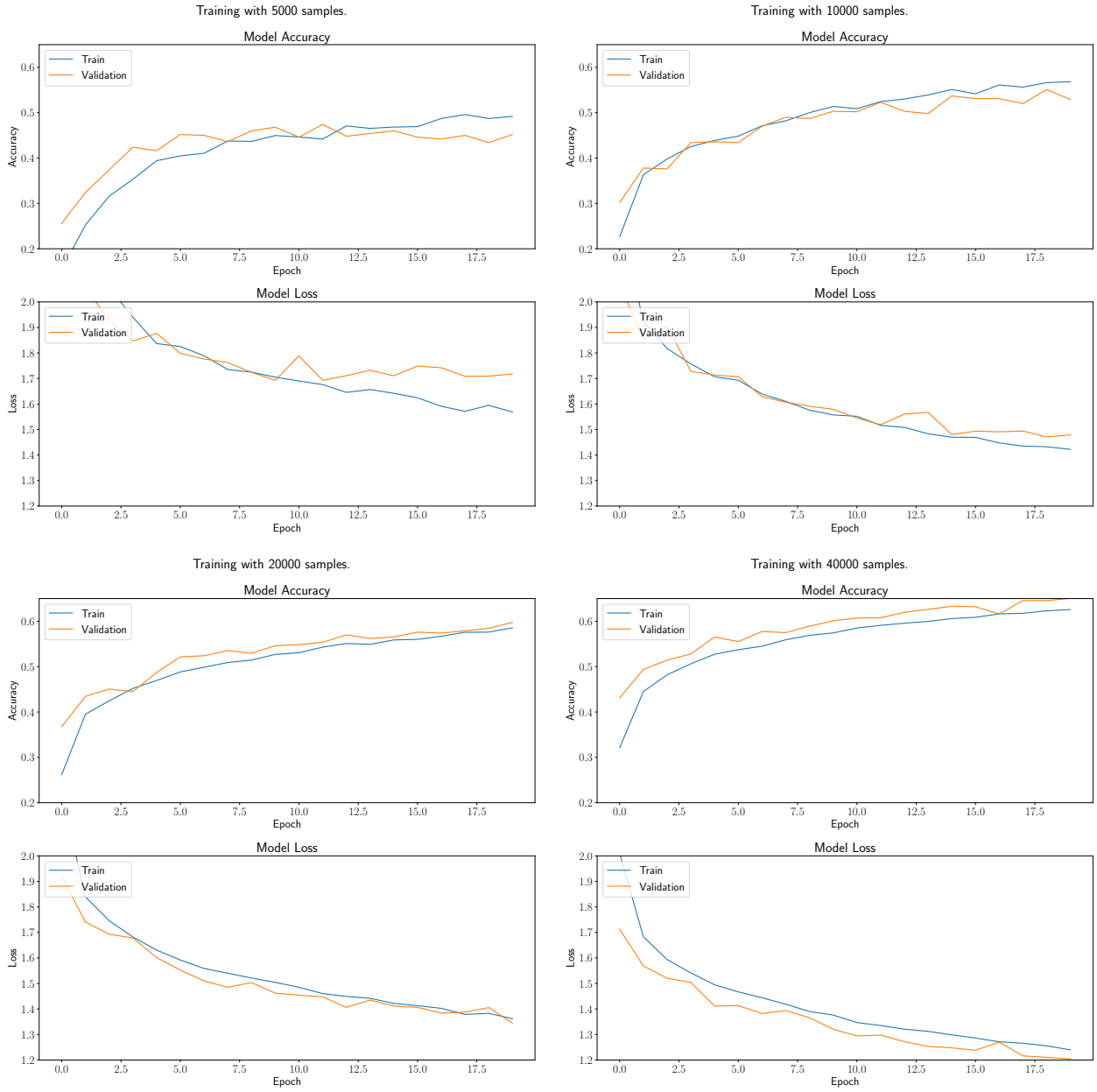

Figure 2: A comparison of accuracy/loss on training/test data from epochs 1 to 20 for different quantities of training data, 5K, 10K, 20K and 40K. Each model is trained with $L_1 = 0.001$.

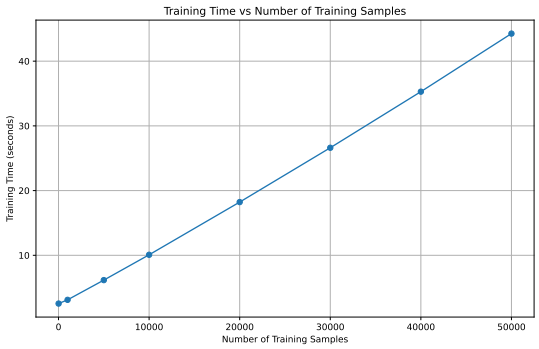

Figure 3: The amount of time needed to train the ConvNet for 20 epochs is plotted against thet number of training samples used. The relationship is linear.

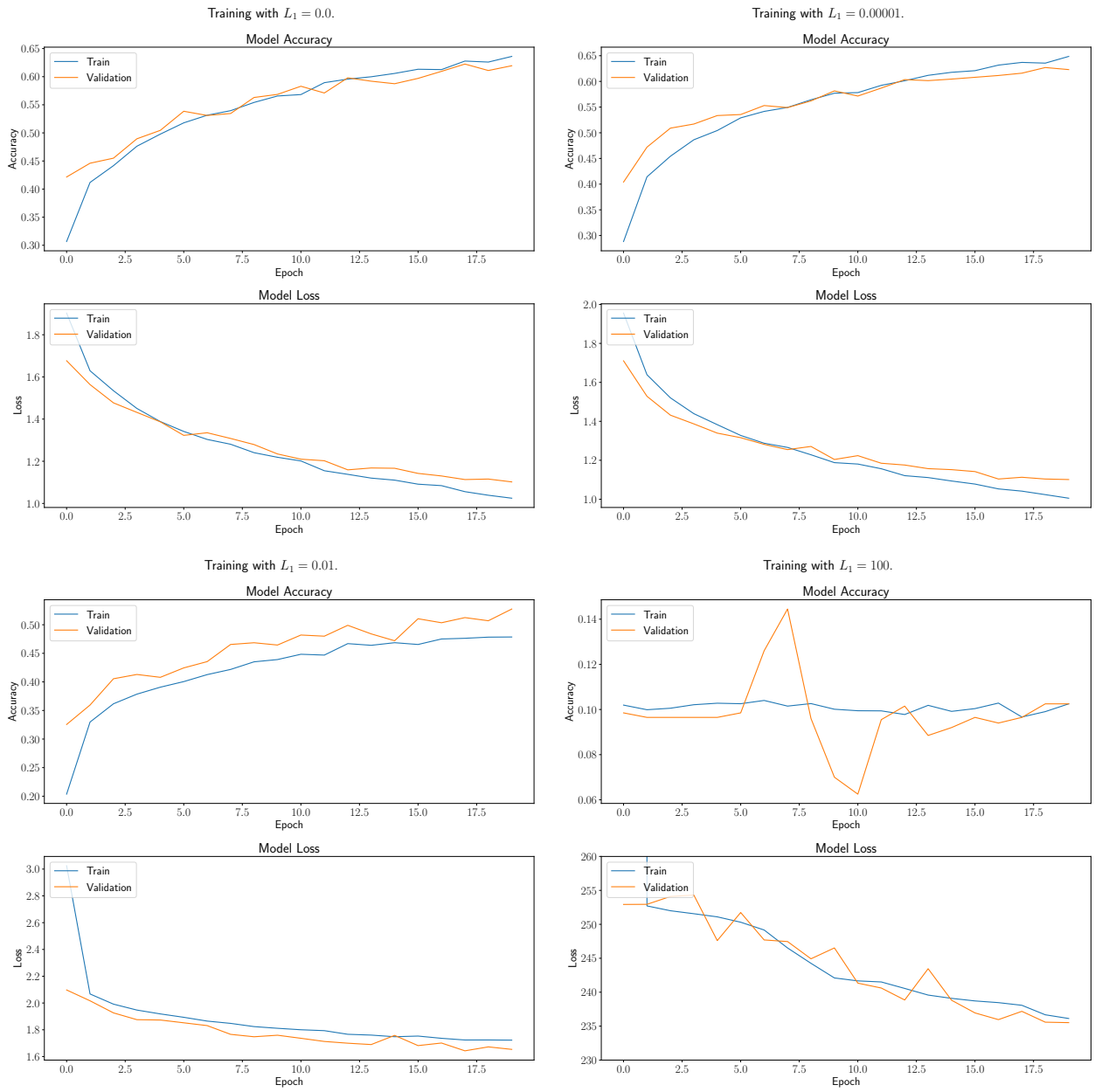

Figure 4: A comparison of accuracy/loss on training/test data from epochs 1 to 20 for different L_1 regularization terms, $0.0, 0.00001, 0.01, 1000$. Each model is trained on 5K training samples.