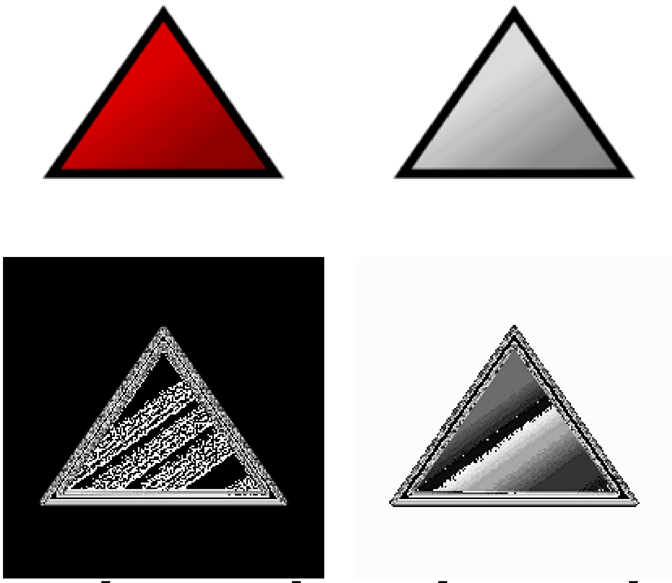


$$k_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad k_2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

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

In Figure 2 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.

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, $[z_1, \ldots, z_{10}]$.

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

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

Figure 2: Source code of a ConvNet keras model.

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

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

2.2 (ii) (b) (i)

Keras reports that model given by the code in Figure 2 has 37146 total parameters, all of which a trainable. The final Dense layer has the most parameters, namely $2048 \cdot 10 + 10 = 20490$. The number of parameters in a convolution layer is determined by the kernel size and the number of filters and the number of channels, no. params $= k_w \cdot k_h \cdot c \cdot f$, whereas the number of paramaters in a Dense layer is determined by the input and output sizes. Since the input dimension for the Dense layer is quite large (2048), this layer ends up having more parameters than any of the convolution layers.

The models evaluation scores are significantly better on the training data than they are on the test data. On the training data the model has an accuracy of 57%, and on the the test data is 48%. The average F_1-score is 0.48. A simple baseline which always predicts the most frequent class achieves an accuracy of 10%, naturally considering the test set is balanced and there are ten classes, and the average F_1-score across the classes in 0.018. The ConvNet is much better than the ‘most_frequent’ baseline.

2.3 (ii) (b) (ii)

The history of loss and accuracy of the model trained on 5K samples over 20 epochs is presented in Figure 3.

2.4 (ii) (b) (iii)

Figure 4 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 4 is that more training data allows us to train for more epochs without over-fitting, or without over-fitting as much.

2.5 (ii) (b) (iv)

Figure 6 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

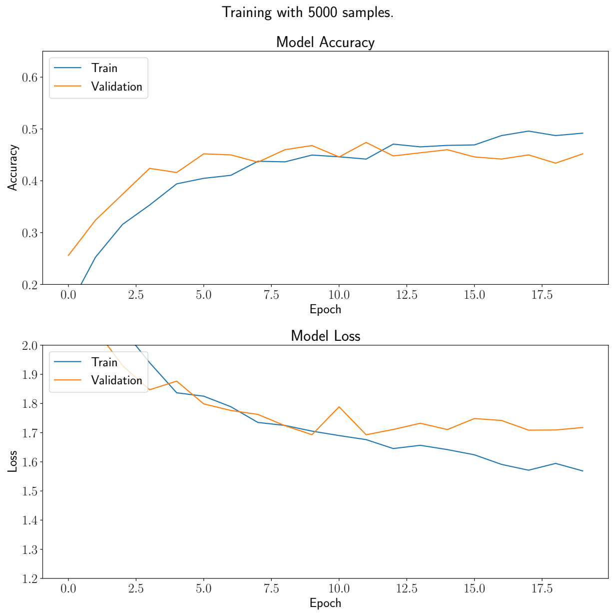


Figure 3: A comparison of accuracy/loss on training/test data from epochs 1 to 20 when trained on 5K training samples with $L_1 = 0.001$.

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, $L_1 = 0.01$.

2.6 (ii) (c)

The model described for parts (ii) (b) (i)-(iv) above used layers alternating between stride of (1,1) and (2,2). The effect of the (2,2) stride layers is that the output tensor's width and height are halved (while the 'depth' of the tensor is determined by the number of filters). In this section we compare the stride technique to a model that instead uses max-pooling layers to reduce the dimensionality. Each layer that had a stride of (2,2) is given a stride of (1,1) and another MaxPooling2D layer is added to follow that layer. The MaxPooling2D model has 37146 parameters, the same number as the model using strided layers. There is no change to the number of paramaters because; 1. the Conv2D layer's number of parameters is independent of the stride 2. the MaxPooling2D layer has no parameters.

The MaxPooling2D version of the model took about 54 seconds to train on my laptop, whereas the strided version took about 25 seconds. The reason the strided version is faster to train is that the kernel's; stride results in skipping a proportion of the calculations needed for the forward pass. With a stride of 2 in both directions the number of times the kernel is multiplied elementwise by the corresponding slice of the input tensor is $1/4$ so many as with a stride of 1 in both directions. The MaxPooling2D version also happens to have 2 additional layers but this is *not* a significant factor in the increased training time.

The accuracy of the two versions whe

Metric	MaxPooling2D	stride=(2,2)
train accuracy	66%	62%
test accuracy	54%	50%

Table 1: Comparison of model performance with MaxPooling2D and stride=(2,2), both using $L_1 = 0.0001$

2.7 (ii) (d)

With the two extra ConvNet layers the model now has a total of 23314 trainable parameters.

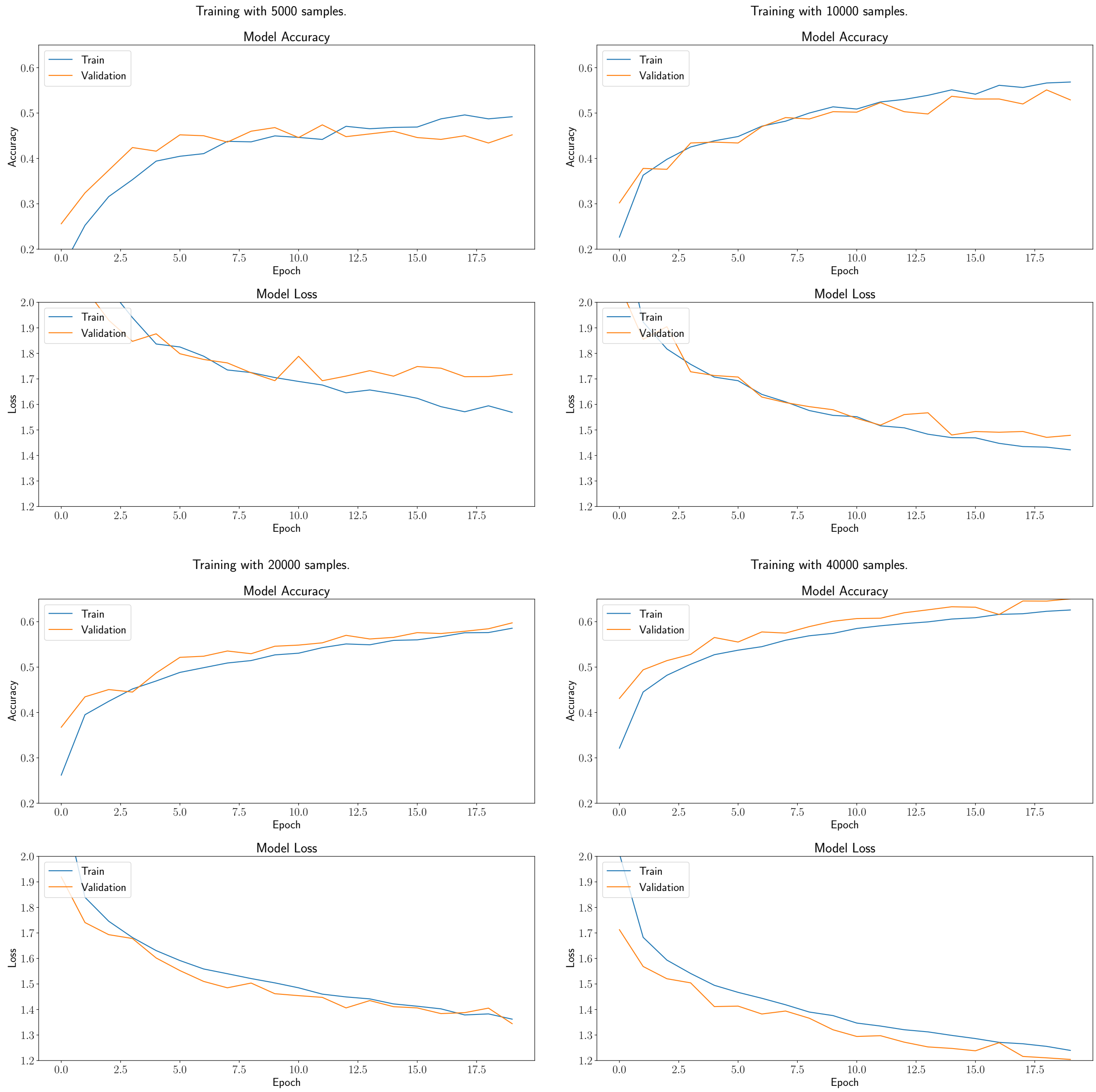
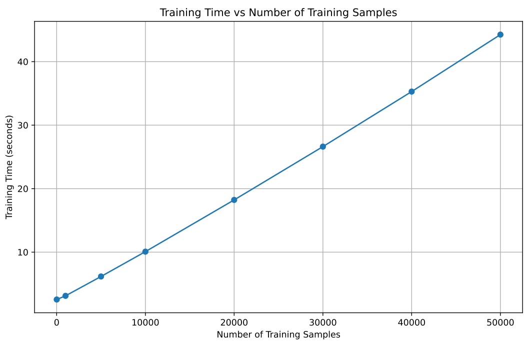


Figure 4: A comparison of accuracy/loss on training/test data from epochs 1 to 20 for different quantities of training data, 5K, 10K, 20K and 40K. Each model is trained with $L_1 = 0.001$.



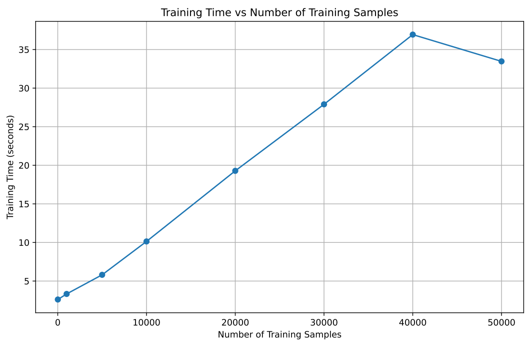


Figure 5: The amount of time needed to train the ConvNet for 20 epochs is plotted against thet number of training samples used. The relationship is linear.

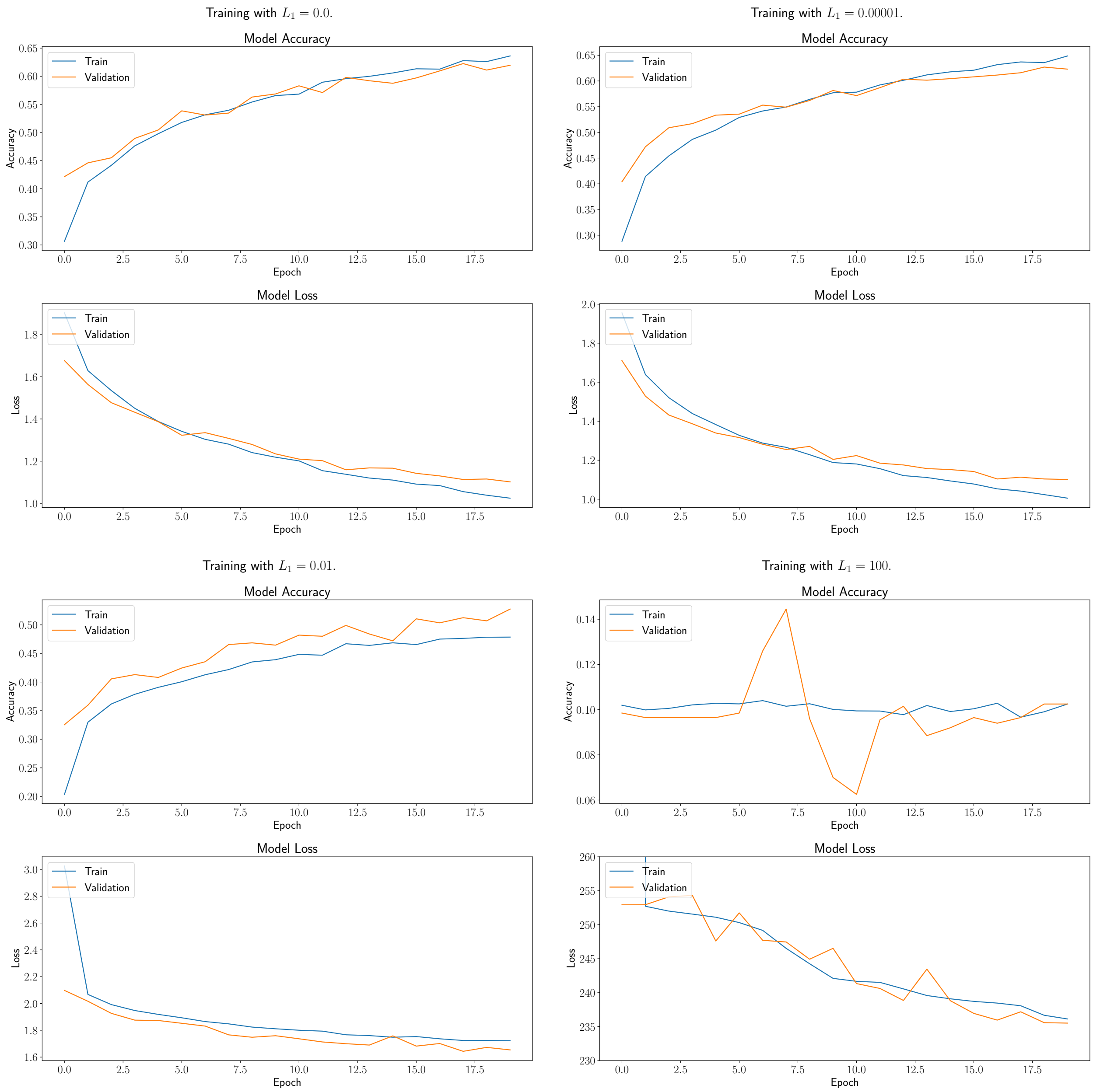


Figure 6: A comparison of accuracy/loss on training/test data from epochs 1 to 20 for different L_1 regularization terms, $0.0, 0.00001, 0.01, 1000$. Each model is trained on 5K training samples.