Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

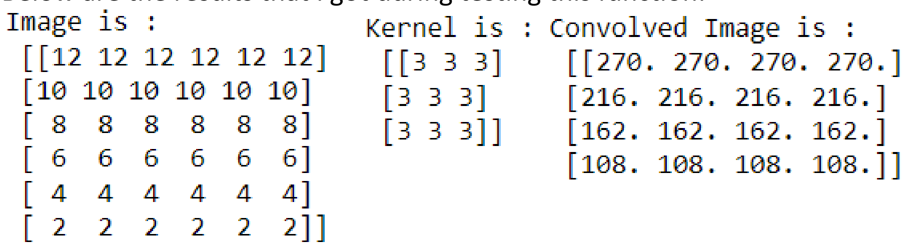
Question (i)(a)(b) Vanilla python function to convolve image and kernel, loading an image and testing 2 kernels

I have written below function that take an array and a kernel of any size as input and convolves the kernel with image and returns the array of convolved image as an output:

```
def convolve_image_kernel(image, kernel) :

  #Finding the shape of the image
  image_x_shape, image_y_shape = image.shape[0], image.shape[1]

  #Finding the shape of the kernel
  kernel_x_shape, kernel_y_shape = kernel.shape[0], kernel.shape[1]

  #Defining the output convolved image shape
  convolution_image = np.zeros((int((image_x_shape - kernel_x_shape) + 1), int((image_y_shape - kernel_y_shape) + 1)))

  #Convolution process
  for j in range(image_y_shape) :
    if j > image_y_shape - kernel_y_shape :
      break

    for i in range(image_x_shape) :
      if i > image_x_shape - kernel_x_shape :
        break
      convolution_image[i, j] = (kernel * image[i: i + kernel_x_shape, j: j + kernel_y_shape]).sum()

  return convolution_image
```

In this function I am first finding out the shape of input image and kernel. These values are then used to define the shape of final convolved image. Then, I have created a nested loop and in iteration of loop I have put in test conditions to check if the kernel has reached the end of image length wise and width wise and break the loop if this condition is met. Finally, in the innermost loop, if kernel is in right spot on the image, I have calculated the convolution of image and kernel and stored it in the output array. Once kernel iterates over the entire image then I have returned the convolved image as output.

Below are the results that I got during testing this function:



```
Image is :                    Kernel is : Convolved Image is :
 [[12 12 12 12 12 12]          [[3 3 3]    [[270. 270. 270. 270.]
 [10 10 10 10 10 10]           [3 3 3]     [216. 216. 216. 216.]
 [ 8  8  8  8  8  8]           [3 3 3]]    [162. 162. 162. 162.]
 [ 6  6  6  6  6  6]                       [108. 108. 108. 108.]]
 [ 4  4  4  4  4  4]
 [ 2  2  2  2  2  2]]
```

For Part b, I have used below image. Its size is 300X300. Then, based on code provided to us to load the image in 3 'RGB' arrays and select only one array 'R' from it, below image output was achieved:

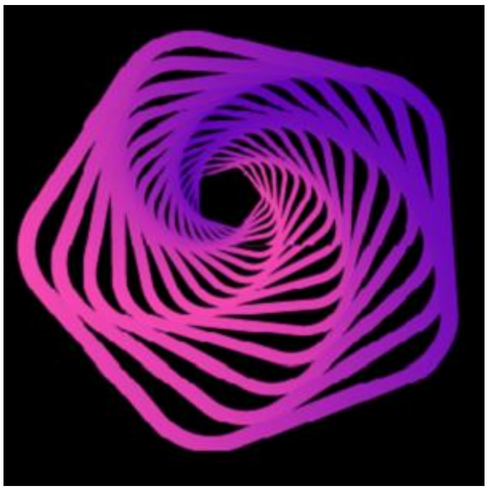


Image used for testing function

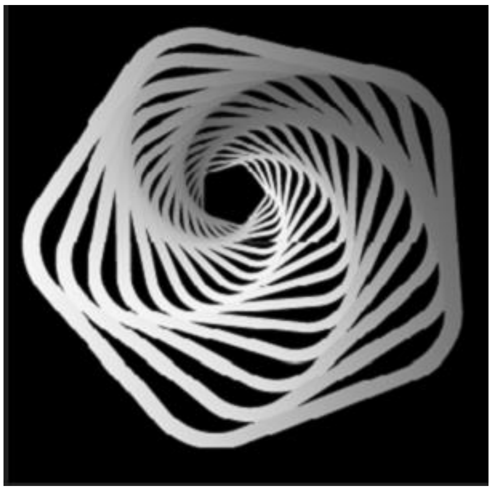


Image generated by selecting 'R' feature

from loaded 'RGB' arrays

Then, I have tested my function by passing 2 kernels and image (R array) shown above. Below images were generated for these kernels:

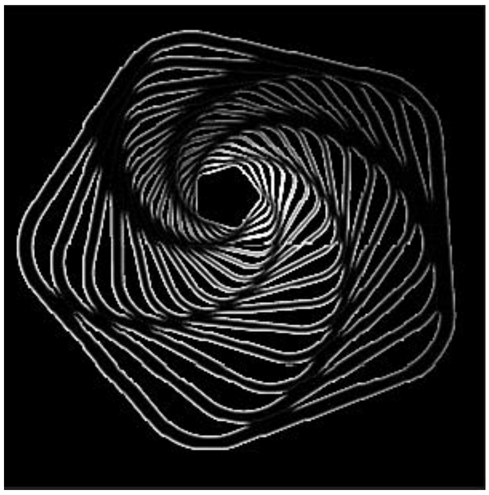


Image generated for Kernel 1

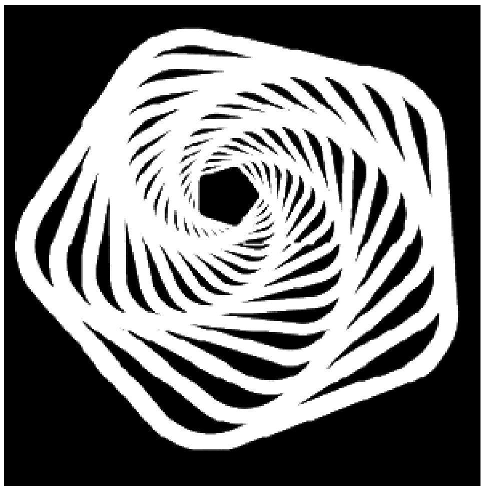


Image generated for Kernel 2

Here, I can analyse that Kernel 1 has returned all the edges of the shape and rest is not detected. Kernel 2 has brightened the image passed to it by turning all pixels to white colour.

Question (ii)(a) Architecture of ConvNet

From the code I can clearly identify that there are four different types of layers used in this model. These are Convolution layers, Dropout layer, Flatten layer and Dropout layer. Of these 4 different categories there are four Convolution layers in the network and 1 of each Flatten layer, Dense layer, and Dropout layer.

The input image to the network are images from the CIFAR10 dataset and size of the image in this dataset is 32X32. Images are split into train and test dataset and then normalised by dividing by 255. The final shape of image is 32X32X3 as one image is loaded into 3 arrays representing 'RGB' features of the image. Once images are loaded convolution is applied on each of these layers using various kernels. The ConvNet applied to the CIFAR10 dataset has below architecture:

Layer 1: model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))

The first layer of convolution applies 16 kernels of size 3X3 to the input image array. This means that at the end of this convolution layer the output image will have 16 features of size 32X32. I can clearly identify that convolution has increased feature from 3 to 16 in this layer. During convolution, there is a possibility of loss of shape (height, width) of input image. To avoid that Padding parameter is set as same. This means that additional zeroes will be padded to input image before convoluting with kernel to prevent loss of shape. Finally, the activation function used in this layer is Relu. Role of activation function is to provide non-linearity in the network. This helps the network to better fit the results as it helps the network to adapt a variety of data and fit the output. This will help to improve the overall accuracy of the network. The output shape of this layer is 32X32X16.

Layer 2: model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))

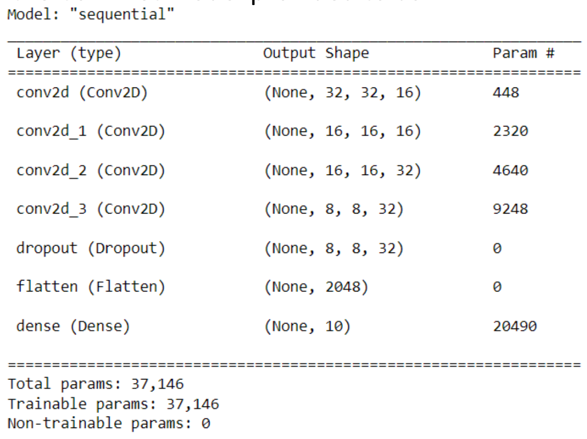
This layer has again used 16 kernels and kept padding parameter as same. However, this layer has used strides and set the stride size to 2X2. Stride is generally used in network to reduce the size of input image. Stride of length L means that once kernel has finished convolution on a part of image it will shift L steps instead of 1 step it was doing earlier. This ensures that the shape is reduced by the factor of L during convolution process itself. Since this layer used 16 kernels, therefore there will be no increase or decrease in the number of features from previous layer as it has also used 16 kernels. The output shape of this layer is 16X16X16. Input shape of size 32 is now reduced to 16 due to stride of 2 during convolution process. The key idea behind using strides in CNN is to reduce the overall number of parameters in the network.

Layer 3: model.add(Conv2D(32, (3,3), padding='same', activation='relu'))

This layer is similar to Layer 1, but this layer has increased the number of kernels from 16 to 32. The size of the kernel remains same, and padding is also applied, and same activation function is used. Therefore, the output of this layer will have 32 features instead of 16. The output shape of this layer is 16X16X32.

Layer 4: model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
This layer is similar to Layer 2. It has strides of size 2 therefore input image size i.e., 16 will be halved to 8. Again, the number of kernels remains same as previous layer i.e., 32 and padding is also to prevent loss of shape. The output shape of this layer is 8X8X32.
Layer 5: model.add(Dropout(0.5))
The dropout layer is used to prevent the overfitting of data. As we know CNN is trained using batching technique. The weight of the network is highly influenced by the batch used in previous iteration. Therefore, to prevent overfitting of weight during training, dropout is used. This layer will randomly set the weight of some neurons to 0. The number of input units to drop is based on the factor provided, in our case 0.5. This means that during iterations some neurons will be randomly deactivated i.e., dropped out to nullify their impact on the output.
Layer 6: model.add(Flatten())
This is used to convert multidimensional input from previous layers to a one-dimensional array. This means 2-D arrays of size 8X8 from previous layer will be flattened to 1-D array after this layer. The intermediary output of convolution represents the feature map in multi-dimensional array. The key idea is to flatten this array to a 1-D vector as the Fully Connected Layer uses 1-D vector of features individually to classify. This will help to reduce the size of memory used and reduce the overall time.
Layer 7: model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
This is the Fully Connected layer in our architecture which is used to predict class for an image based on its features represented in form of 1-D array. Here, we have total classes that will be predicted as 10. The activation function used in this layer is Softmax which uses a normalised exponential function to estimate the probability for multiple classes. Therefore, it is used in case of multi-class classification problems. It is like sigmoid function which is used for binomial classification. Instead Softmax produces a probability distribution for multiple classes.

Below is the model summary for the ConvNet model provided to us:


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 16)        448

 conv2d_1 (Conv2D)           (None, 16, 16, 16)        2320

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        4640

 conv2d_3 (Conv2D)           (None, 8, 8, 32)          9248

 dropout (Dropout)           (None, 8, 8, 32)          0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 37,146
Trainable params: 37,146
Non-trainable params: 0
```

Question (ii)(b)(i)(ii) Run the model, Check performance on train and test data, Compare to Baseline Classifier and Evaluation of History plots
I have executed the model for 5000 training size. Based on model summary above, this model has 37.146 parameters. The layer 'dense' which is the Fully Connected layer in the network has maximum parameters of 9248. The primary reason is this is that in a fully connected layer every input channel in the layer is connected to every output channel in the layer. Therefore, if there are 'n' inputs and 'm' output channel, there will be 'm X n' parameters in the layer.

Following is the summary of performance of the model on train data and test data:

```
              precision    recall  f1-score   support

           0       0.68      0.61      0.64       505
           1       0.82      0.60      0.69       460
           2       0.55      0.54      0.54       519
           3       0.52      0.50      0.51       486
           4       0.58      0.47      0.52       519
           5       0.58      0.59      0.59       488
           6       0.56      0.70      0.62       518
           7       0.58      0.68      0.63       486
           8       0.71      0.69      0.70       520
           9       0.64      0.77      0.70       498

    accuracy                           0.61      4999
   macro avg       0.62      0.61      0.61      4999
weighted avg       0.62      0.61      0.61      4999
```

Performance of CNN Model on Train Data

```
              precision    recall  f1-score   support

           0       0.61      0.45      0.51      1000
           1       0.65      0.66      0.65      1000
           2       0.41      0.43      0.42      1000
           3       0.36      0.37      0.36      1000
           4       0.41      0.41      0.41      1000
           5       0.41      0.42      0.41      1000
           6       0.52      0.63      0.57      1000
           7       0.62      0.50      0.55      1000
           8       0.56      0.67      0.61      1000
           9       0.59      0.54      0.56      1000

    accuracy                           0.51     10000
   macro avg       0.51      0.51      0.51     10000
weighted avg       0.51      0.51      0.51     10000
```

Performance of CNN Model on Test Data

I can clearly analyse that model has performed better on Training data then test data. The accuracy on test data is 0.61 which has significantly decreased to 0.50 on test data. The primary reason for this decrease is that model was trained on the training data, therefore weight of the network was adjusted to training images. While for test data, as new images were fed to the network to predict the class, it performed suboptimal as weight of the network were not adjusted enough to handle unseen data. Another reason for this suboptimal performance is that there were not enough samples in the training data for model to learn and adjust its weight for different variations of the image for same labels, therefore, when unseen data i.e., test data was fed into the model, its accuracy decreased significantly.

I have used the Dummy Classifier which always predicts the most frequent class i.e., dominant class in the training data. Following is the performance of Dummy Classifier on train and test data.

```
              precision    recall  f1-score   support

           0       0.10      1.00      0.18       505
           1       0.00      0.00      0.00       460
           2       0.00      0.00      0.00       519
           3       0.00      0.00      0.00       486
           4       0.00      0.00      0.00       519
           5       0.00      0.00      0.00       488
           6       0.00      0.00      0.00       518
           7       0.00      0.00      0.00       486
           8       0.00      0.00      0.00       520
           9       0.00      0.00      0.00       498

    accuracy                           0.10      4999
   macro avg       0.01      0.10      0.02      4999
weighted avg       0.01      0.10      0.02      4999
```

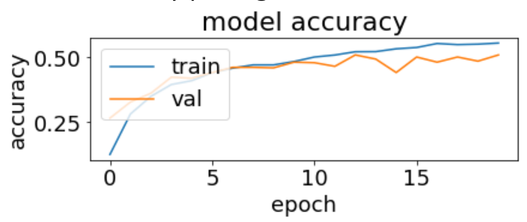
Performance of Baseline Model on Train Data

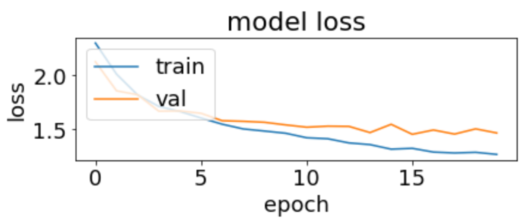
```
              precision    recall  f1-score   support

           0       0.10      1.00      0.18      1000
           1       0.00      0.00      0.00      1000
           2       0.00      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.00      0.00      0.00      1000
           5       0.00      0.00      0.00      1000
           6       0.00      0.00      0.00      1000
           7       0.00      0.00      0.00      1000
           8       0.00      0.00      0.00      1000
           9       0.00      0.00      0.00      1000

    accuracy                           0.10     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.10      0.02     10000
```

Performance of Baseline Model on Test Data

The baseline model has performed exactly as I expected i.e., it has an accuracy of 0.10 as we have 10 classes that is to be predicted and it can only predict only one class i.e., the most dominant class correctly. The performance remains same on both test and training data. We can comfortably say that our CNN model has performed better than the baseline model therefore, this model should be preferred over the baseline model.

Following are the history plots generated for this model:





The plot of model's accuracy vs. epochs is plotted for train and validation set. Validation set is used during training in epochs. At the end of epoch, model's accuracy is calculated against a small data kept apart from training set at the beginning of epoch called validation set. This is used to check how model is performing during training like if it has started to overfit the data and do hyperparameter tuning if required. This is also used to set early stopping condition during training itself. Ideally, the gap between the lines of validation and train accuracy or loss in plot should be negligible like it should be overlapping for most of the epochs and should not be wide. From this plot, I can clearly analyse that for epoch 1 and 2, model's accuracy on training data is very less as compared to validation set. At this stage model is underfitting the data. After around 9 epochs, model's accuracy on training set has superseded that of validation set and after epoch 14 this margin has grown significantly and it has continued till the end of all epochs. This is where model has started to overfit the training data. Similarly from the plot of model's loss vs. epochs we can see that difference between loss on training set and validation set has grown significantly after around epoch 14. This is also highlighting the model is overfitting the training data. I can now understand the reason for this model to fair poorly on the test data as it was overfitted for the training data.

Question (ii)(b)(iii) Run the model on different training size, Calculate training time and Check performance of model for varying training size, Evaluation of History plots

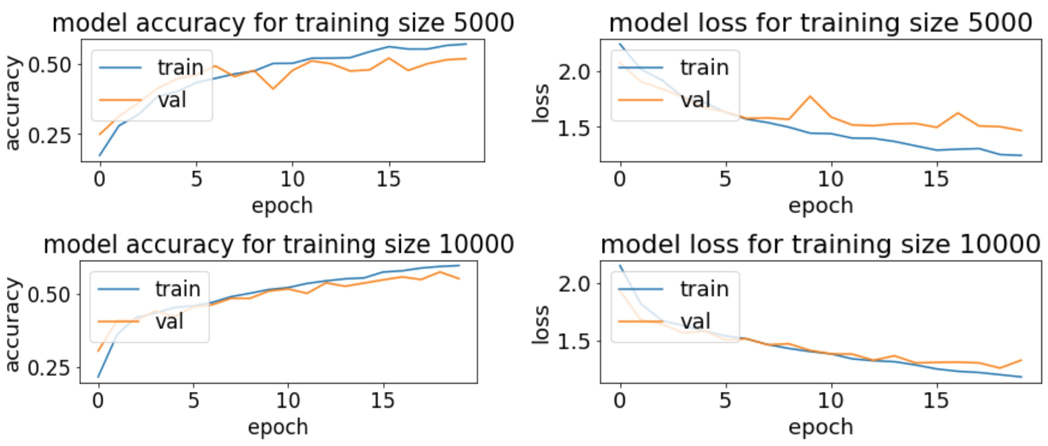
I have generated following summary table for time taken during training, accracy on train data and accuracy on test data for varying training size:

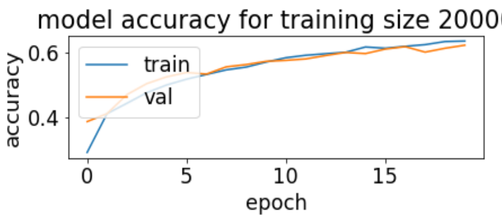
	Training Size	Training Time	Accuracy on Training Data	Accuracy on Test Data
0	5000	12.3 seconds	62.01%	49.86%
1	10000	14.55 seconds	63.01%	55.17%
2	20000	26.91 seconds	69.12%	61.47%
3	40000	54.29 seconds	72.14%	67.74%

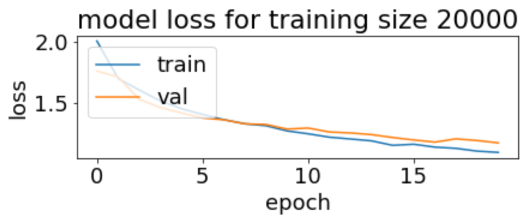
I can clearly analyse that time taken to train the model has increased significantly as we increase the size of training data. The primary reason for this behaviour is that as training size increase the number of images that model needs to be trained with increase in every epoch. This means that more images will go through the convolution process and weights will be adjusted in each convolution process. Therefore, as time taken during convolution has increased due to increase in number of images in each epoch, the overall training time has increased.

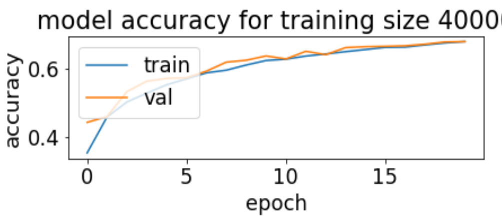
The model accuracy on both training and test data has also increased as we increase the training size. Primarily the model's accuracy on test data has increased significantly from 49.86% for 5K training size to 67.74% for 40K training size. The primary reason for this is that as more number of images are passed during epochs for increased training size, the more model goes throguh convolution process and it has more opportunity to adjust the weights of the network as it analyses more and more unseen data during training. Thus, passing more data will help the model to generalise efficiently as it will understand more and more how input features of an image are mapped to its labels and this will help the model to adapt to unseen data in test data.

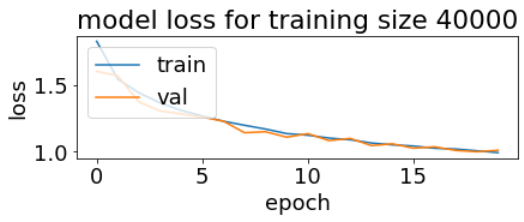
Following are the history plots generated for varying training size:







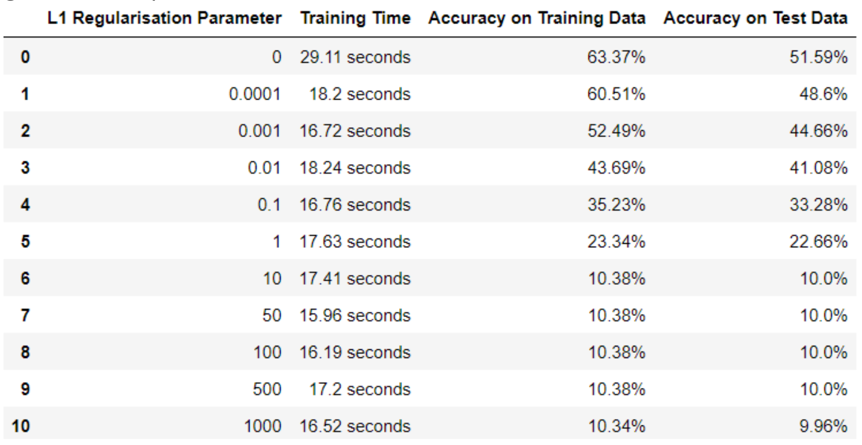




From the history plots I can now understannd the reason for increase in model's accuracy on test data as we increase the training size as I can see that overfitting of the training data has reduced significantly as we increase the training size. I can see that as epochs progresses, the gap was widening between train and validation line in the plot for both accuracy and loss, with model performing better on training data, when training size was 5K. This means that model has started to overfit the train data after some epochs and that is the reason it faired poorly on the test data for train size 5K. However, as we increased the train size, the gap between these line started to decrease highlighting the reduction in overfitting on more generalisation of the model. For training size 40K, I can analyse that the gap is almost negligible and after a few epochs, these lines almost overlap each other for most of the epochs in both accuracy and loss plots. Also, the overall loss of the model has reduced when compared to other training sizes. This means that model has generalised very well to adapt to unseen data and overfitting has significantly reduced. This is the primary reason we see a significant increase in accuracy of the model and less difference between train and test accuracy.

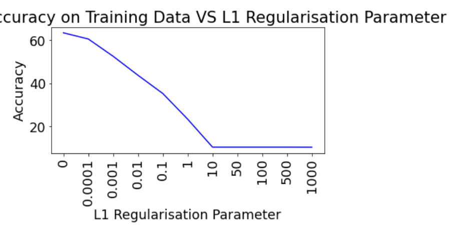
Question (ii)(b)(iv) Run the model on different L1 regularisation parameter, Check performance of model for varying L1 regularisation parameter and compare to model trained with varying training size

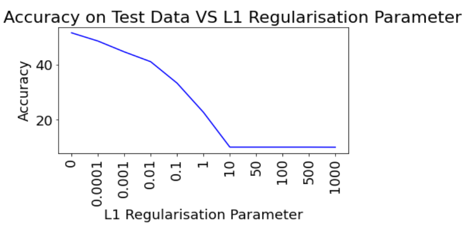
I have generated following summary table for time taken during training, accracy on train data and accuracy on test data for varying L1 regularisation parameter:



	L1 Regularisation Parameter	Training Time	Accuracy on Training Data	Accuracy on Test Data
0	0	29.11 seconds	63.37%	51.59%
1	0.0001	18.2 seconds	60.51%	48.6%
2	0.001	16.72 seconds	52.49%	44.66%
3	0.01	18.24 seconds	43.69%	41.08%
4	0.1	16.76 seconds	35.23%	33.28%
5	1	17.63 seconds	23.34%	22.66%
6	10	17.41 seconds	10.38%	10.0%
7	50	15.96 seconds	10.38%	10.0%
8	100	16.19 seconds	10.38%	10.0%
9	500	17.2 seconds	10.38%	10.0%
10	1000	16.52 seconds	10.34%	9.96%

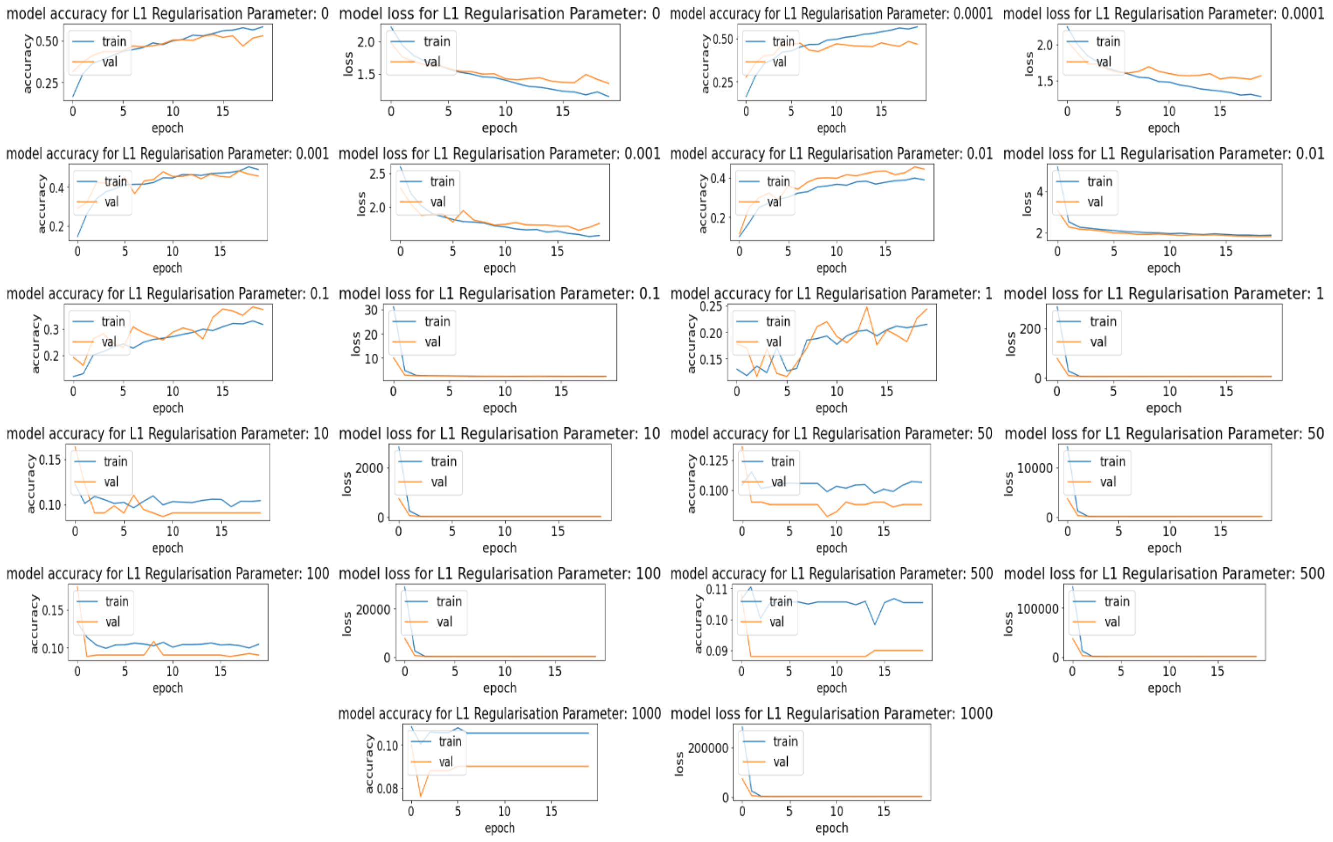
Following are the plots generated I have generated using above data:





From the table and plots above, I can clearly analyse that as we increase L1 the regularisation parameter, the accuracy on both train and test data start to decrease and as the value of this parameter changes from less than 1 to multiples of 10, the accuracy has decreased significantly and is close to 10%. The primary reason for this behaviour is that L1 regularisation parameter set the weight of the model to 0. Higher the weight of this penalty parameter, more and more weights in the network are set to 0 and hence, we see a sharp decrease in the overall accuracy of the model on both test and train data. In CNN, weights play a very important role in accurately predicting the class of the image as these are adjusted for various features of the image during epochs and setting them to 0 will negatively impact the model's performance as we can see here. This is the reason that we have the highest accuracy on both train and test data when the L1 parameter was set to 0. Once we increase the L1 parameter value even by a fraction of 0.0001, the accuracy has started to decrease.

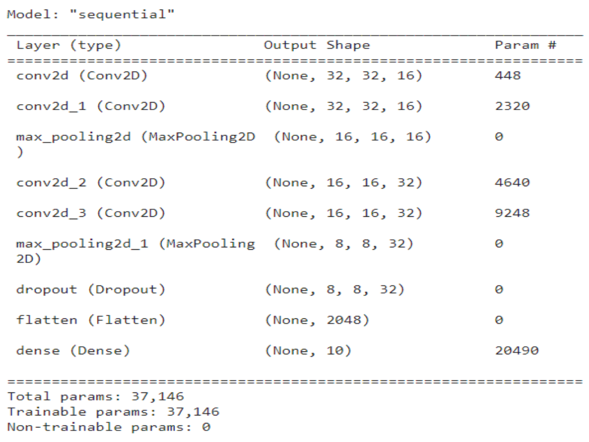
For analysing it role in overfitting the data, I have generated below history plots:



I can clearly see that regulating L1 parameter does not help in reducing overfitting problem significantly as in almost all the plots for various L1 regularisation parameters, I can see that there is gap between train and validation lines and this gap widens as epochs progresses. Infact, for L1 parameter in range more than 100, this gap is significantly higher and loss has also increased to mutiples of 10^5. Also, the overall accuracy has also decreased significantly. When the range of L1 regualrisation paramter is small i.e. less than 1, we see the some overfitting in some cases, but accuracy has not improved on test data, therefore, when compared to increasing train size instead of L1 regularisation parameter, I can clearly say that increasing training size help model to fit better and reduce the overfitting problem, and increase the overall accuracy of the model. The primary reason for this is that increasing the training size helps the model to generalise the weight of the model and adapt to unseen data. The more images it uses during training, the more feautres it recognise, the more it adjust its weight, the more it reduces the impact of overfitting the data and improve the accuracy on test data. However, L1 parameter just penalises the model by setting the weights to 0 which leads to decrease in accuracy as model weights are not adjusted according to features.

Question (ii)(c)(i)(ii) Run the model with Max-pool layer, calculate training time and check performance on train and test data, compare with strides architecture

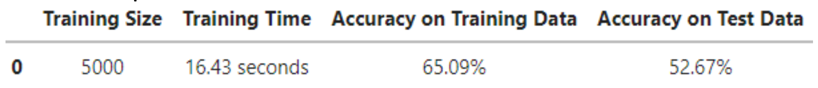
I have changed the code provided to us to include the max-pooling layer and removed stride parameter from the convolution layer. Below is the architecture summary of the new model:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 16)        448

 conv2d_1 (Conv2D)           (None, 32, 32, 16)        2320

 max_pooling2d (MaxPooling2D  (None, 16, 16, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        4640

 conv2d_3 (Conv2D)           (None, 16, 16, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout (Dropout)           (None, 8, 8, 32)          0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 37,146
Trainable params: 37,146
Non-trainable params: 0
```

The model has 37,146 parameters. The number of parameters has not changed when compared to stride model since we have not added any new layer that performs the convolution, we have just changed the way to downsize the image during convolution. Downsizing image helps us to reduce the amount of time taken to train the model as it reduces the number of calculations performed during the convolution process. It has no impact on the number of parameters in the network.

Below is the summary of model's performance on 5K data.



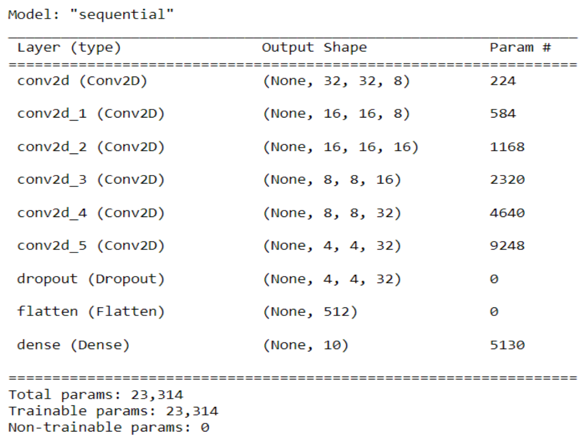
	Training Size	Training Time	Accuracy on Training Data	Accuracy on Test Data
0	5000	16.43 seconds	65.09%	52.67%

I can clearly identfy that time taken by maxpool model (16.43 seconds) is more than time taken by stride model (12.3 seconds) for same amount of training data. The primary reason for this increase in time is that strides were happening during the convolution process itself. As soon as kernel has finished convolving over a part of image, it strided to next section of image based on the stride length given, thereby skipping some columns/rows during convlution process itself. On the other hand, max-pool layer works after the convolution process has completed. Therefore time needed to calculate values for entire image in the max-pool process is added after convolution has happened, thereby increasing the time needed for overall training.

I can also see a slight increase in the accuracy on test data of the max-pool model (52.67%) when compared to stride model(49.86%). The primary reason for this is that during stride, we skip some part of image from calculation to downsize it during convolution, but during max-pool we do not skip any part of the image while downsizing, thereby increasing the overall accuracy of the model. Therefore, we have to decide on trade-off between training time and overall acuuracy of the model. Since, in our case, we do not have hude dataset to train (only 5K images), we should prefer max-pool model over stride model as it results in better overall accuracy on test data.

Question (iii)(a) Thinner and Deeper ConvNet Architecture

Below is the architecture of this model:

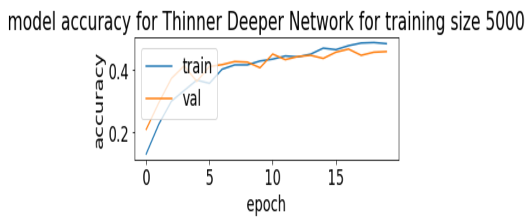


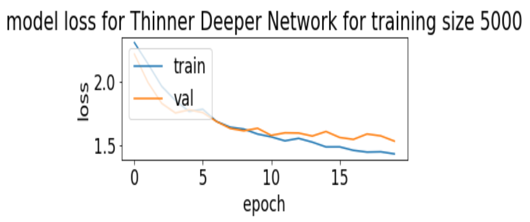
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 8)         224

 conv2d_1 (Conv2D)           (None, 16, 16, 8)         584

 conv2d_2 (Conv2D)           (None, 16, 16, 16)        1168

 conv2d_3 (Conv2D)           (None, 8, 8, 16)          2320

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          4640

 conv2d_5 (Conv2D)           (None, 4, 4, 32)          9248

 dropout (Dropout)           (None, 4, 4, 32)          0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 10)                5130

=================================================================
Total params: 23,314
Trainable params: 23,314
Non-trainable params: 0
```

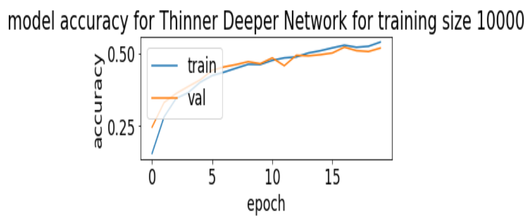
To understand the trade-off between prediction performance and overfitting for this thinner and deeper network, I have trained this model for multiple training sizes and below is the result for different iterations:

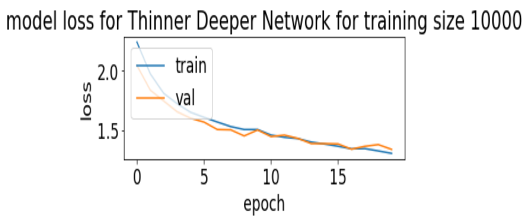
	Training Size	Training Time	Accuracy on Training Data	Accuracy on Test Data
0	5000	18.1 seconds	54.35%	46.01%
1	10000	19.49 seconds	58.91%	52.25%
2	20000	41.26 seconds	62.84%	56.39%
3	30000	49.25 seconds	65.03%	61.05%
4	40000	66.76 seconds	64.52%	60.88%
5	50000 (Full Data Set)	81.0 seconds	68.7%	65.16%

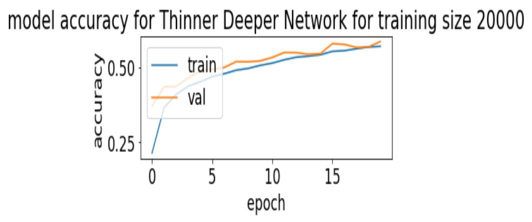
We can see that the number of parameter in this model(23,314) are less compared to our earlier model(37,146). As we increase the training size the time to train model also increases as more images go through convolution process during training epochs and more convolutions required more processing time. We also see increase in the training accuracy when we increase the training size as model becomes more generalised and adapts more to unseen data. For full data set, there is a sharp increase in the accuracy as expected as model was already trained for all the images in test data so it's weight were already adjusted to it. Below are history plots generated for this model:

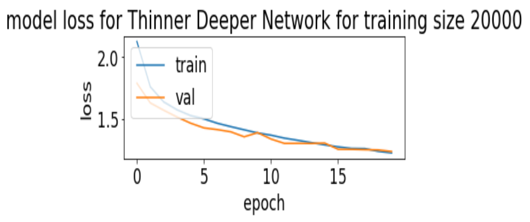


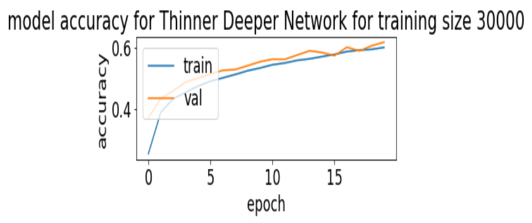


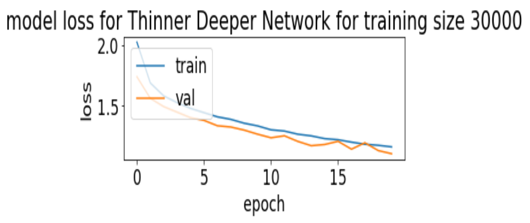


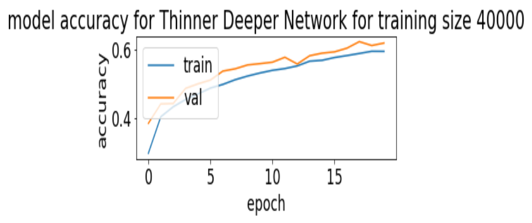


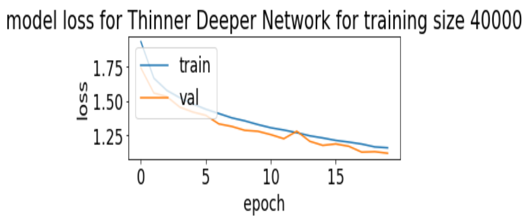


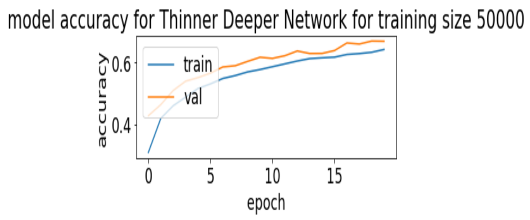


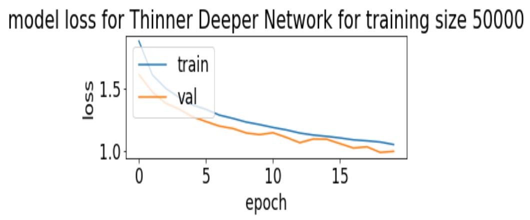












I can analyse that as we increase the training size from 5,000 to 30,000 the model overfitting has decreased as the gap between train and validation line has decreased and it started to overlap. Therefore, we can see that we can improve the performance of the model by increasing the training size but training time also increase. However passign the lower training size also results in overfitted models. Therefore, it is important to understand the sweet point where the model is not overfitted and it does not take huge time train for a considerable training size.

Appendix: Code for Question 1 - All Parts

```
import numpy as np
from PIL import Image
def convolve_image_kernel(image, kernel) :

   #Finding the shape of the image
   image_x_shape, image_y_shape = image.shape[0], image.shape[1]

   #Finding the shape of the kernel
   kernel_x_shape, kernel_y_shape = kernel.shape[0], kernel.shape[1]

   #Defining the output convolved image shape
   convolution_image = np.zeros((int((image_x_shape - kernel_x_shape) + 1), int((image_y_shape - kernel_y_shape) + 1)))

   #Convolution process
   for j in range(image_y_shape) :

      if j > image_y_shape - kernel_y_shape :
         break

      for i in range(image_x_shape) :

         if i > image_x_shape - kernel_x_shape :
            break

         convolution_image[i, j] = (kernel * image[i: i + kernel_x_shape, j: j + kernel_y_shape]).sum()

   return convolution_image

image = np.array([[12, 12, 12, 12, 12, 12], [10, 10, 10, 10, 10, 10], [8, 8, 8, 8, 8, 8],
             [6, 6, 6, 6, 6, 6], [4, 4, 4, 4, 4, 4], [2, 2, 2, 2, 2, 2]])
kernel = np.array([[3, 3, 3], [3, 3, 3], [3, 3, 3]])
convolution_image = convolve_image_kernel(image, kernel)

print(f'Image is :\n {image}')
print(f'Kernel is :\n {kernel}')
print(f'Convolved Image is :\n {convolution_image}')

im = Image.open('Test_Image.png')
rgb = np.array(im.convert('RGB'))
r = rgb [:, :, 0]
Image.fromarray(np.uint8(r)).show( )

kernel_1 = np.array([[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]])
convolution_image_1 = convolve_image_kernel(r, kernel_1)
Image.fromarray(np.uint(convolution_image_1)).show()

kernel_2 = np.array([[0, -1, 0],[-1, 8, -1],[0, -1, 0]])
convolution_image_2 = convolve_image_kernel(r, kernel_2)
Image.fromarray(np.uint(convolution_image_2)).show()
```

Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

Appendix: Code for Question 2 - All Parts

```
import numpy as np
import tensorflow as tf
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
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.dummy import DummyClassifier
import warnings
warnings.filterwarnings("ignore")

num_classes = 10
input_shape = (32, 32, 3)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
n=5000
x_train = x_train[1:n]; y_train=y_train[1:n]

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

use_saved_model = False
if use_saved_model:
   model = keras.models.load_model("cifar.model")
else:
   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)))
   model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
   model.summary()
   batch_size = 128
   epochs = 20
   history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
   model.save("cifar.model")

plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss'); plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))
print(confusion_matrix(y_train1,y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
print(confusion_matrix(y_test1,y_pred))

Most_Frequent_Dummy_Model = DummyClassifier(strategy = "most_frequent")
Most_Frequent_Dummy_Model.fit(x_train, y_train)

Most_Frequent_Dummy_Predictions_Train = Most_Frequent_Dummy_Model.predict(x_train)

y_Most_Frequent_Dummy_Predictions_Train = np.argmax(Most_Frequent_Dummy_Predictions_Train, axis=1)
y_Train_1 = np.argmax(y_train, axis=1)
print(classification_report(y_Train_1, y_Most_Frequent_Dummy_Predictions_Train))
print(confusion_matrix(y_Train_1, y_Most_Frequent_Dummy_Predictions_Train))

Most_Frequent_Dummy_Predictions_Test = Most_Frequent_Dummy_Model.predict(x_test)

y_Most_Frequent_Dummy_Predictions_Test = np.argmax(Most_Frequent_Dummy_Predictions_Test, axis=1)
y_Test_1 = np.argmax(y_test, axis=1)
print(classification_report(y_Test_1, y_Most_Frequent_Dummy_Predictions_Test))
print(confusion_matrix(y_Test_1, y_Most_Frequent_Dummy_Predictions_Test))

import numpy as np
import tensorflow as tf
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
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.dummy import DummyClassifier
import warnings
warnings.filterwarnings("ignore")
import time
import pandas as pd

num_classes = 10
input_shape = (32, 32, 3)

def Create_Conv_Model(n = 5000) :

  ### Loading the data and splitting data in train and test split. Also normalising the data

  # the data, split between train and test sets
  (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
  x_train = x_train[1:n]; y_train=y_train[1:n]

  # Scale images to the [0, 1] range
  x_train = x_train.astype("float32") / 255
  x_test = x_test.astype("float32") / 255
  print("orig x_train shape:", x_train.shape)
```

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)


### Setting the ConvNet parameters and Training the ConvNet model
model_name = "cifar_" + str(n) + ".model"
use_saved_model = False
if use_saved_model:
    model = keras.models.load_model(model_name)
else:
    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)))
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    model.summary()
    batch_size = 128
    epochs = 20

    ##Saving the time to train the CNN
    start_time = time.time()
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
    model.save(model_name)
    train_time = time.time() - start_time

    ### Plot of Model Accuracy Vs.Epochs
    plt.figure()
    plt.subplot(211)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'model accuracy for training size {n}')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')


    ### Plot of Model Loss Vs.Epochs
    plt.figure()
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'model loss for training size {n}')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()


### Testing model accuracy on Train Data
preds_train = model.predict(x_train)
y_pred_train = np.argmax(preds_train, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(f'Model Accuracy on Train Data for Training Size: {n}')
print(classification_report(y_train1, y_pred_train))
print(confusion_matrix(y_train1, y_pred_train))


### Testing model accracy on Test data
preds_test = model.predict(x_test)
y_pred_test = np.argmax(preds_test, axis=1)
y_test1 = np.argmax(y_test, axis=1)
```

```
    print(f'\n\nModel Accuracy on Test Data for Training Size: {n}')
    print(classification_report(y_test1, y_pred_test))
    print(confusion_matrix(y_test1, y_pred_test))

    training_size_str = str(n)
    training_time_str = str(round(train_time, 2)) + ' seconds'
    accuracy_score_train = round(accuracy_score(y_train1, y_pred_train) * 100, 2)
    accuracy_score_train_str = str(accuracy_score_train) + '%'
    accuracy_score_test = round(accuracy_score(y_test1, y_pred_test) * 100, 2)
    accuracy_score_test_str = str(accuracy_score_test) + '%'


    CNN_model_Params_array = pd.array([training_size_str, training_time_str, accuracy_score_train_str, accuracy_score_test_str])

    return CNN_model_Params_array

n = 5000
CNN_model_array_5K = Create_Conv_Model(n)

n = 10000
CNN_model_array_10K = Create_Conv_Model(n)

n = 20000
CNN_model_array_20K = Create_Conv_Model(n)

n = 40000
CNN_model_array_40K = Create_Conv_Model(n)

final_data_array = np.vstack((CNN_model_array_5K, CNN_model_array_10K))
final_data_array = np.vstack((final_data_array, CNN_model_array_20K))
final_data_array = np.vstack((final_data_array, CNN_model_array_40K))

CNN_Models_DF = pd.DataFrame()
CNN_Models_DF = CNN_Models_DF.append(pd.DataFrame(final_data_array, columns = ["Training Size", "Training Time", "Accuracy on Training Data", "Accuracy on Test Data"]), ignore_index = True)

display(CNN_Models_DF)

import numpy as np
import tensorflow as tf
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
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.dummy import DummyClassifier
import warnings
warnings.filterwarnings("ignore")
import time
import pandas as pd

num_classes = 10
input_shape = (32, 32, 3)

def Create_Conv_Model(l1_regularisation_parameter = 0.0001) :

    ### Loading the data and splitting data in train and test split. Also normalising the data

    # the data, split between train and test sets
    n = 5000
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

```
x_train = x_train[1:n]; y_train=y_train[1:n]

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)


### Setting the ConvNet parameters and Training the ConvNet model
model_name = "cifar_" + str(n) + ".model"
use_saved_model = False
if use_saved_model:
    model = keras.models.load_model(model_name)
else:
    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(l1_regularisation_parameter)))
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    model.summary()
    batch_size = 128
    epochs = 20

    ##Saving the time to train the CNN
    start_time = time.time()
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
    model.save(model_name)
    train_time = time.time() - start_time

    ### Plot of Model Accuracy Vs.Epochs
    plt.figure()
    plt.subplot(211)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'model accuracy for L1 Regularisation Parameter: {l1_regularisation_parameter}')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')


    ### Plot of Model Loss Vs.Epochs
    plt.figure()
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'model loss for L1 Regularisation Parameter: {l1_regularisation_parameter}')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()


### Testing model accuracy on Train Data
preds_train = model.predict(x_train)
y_pred_train = np.argmax(preds_train, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(f'Model Accuracy on Train Data for L1 Regularisation Parameter: {l1_regularisation_parameter}')
print(classification_report(y_train1, y_pred_train))
print(confusion_matrix(y_train1, y_pred_train))
```

```
    ### Testing model accracy on Test data
    preds_test = model.predict(x_test)
    y_pred_test = np.argmax(preds_test, axis=1)
    y_test1 = np.argmax(y_test, axis=1)
    print(f'\n\nModel Accuracy on Test Data for L1 Regularisation Parameter: {l1_regularisation_parameter}')
    print(classification_report(y_test1, y_pred_test))
    print(confusion_matrix(y_test1, y_pred_test))

    l1_regularisation_parameter_str = str(l1_regularisation_parameter)
    training_time_str = str(round(train_time, 2)) + ' seconds'
    accuracy_score_train = round(accuracy_score(y_train1, y_pred_train) * 100, 2)
    accuracy_score_train_str = str(accuracy_score_train) + '%'
    accuracy_score_test = round(accuracy_score(y_test1, y_pred_test) * 100, 2)
    accuracy_score_test_str = str(accuracy_score_test) + '%'


    CNN_model_Params_array = pd.array([l1_regularisation_parameter_str, training_time_str, accuracy_score_train_str,
accuracy_score_test_str])

    return CNN_model_Params_array

l1_regularisation_parameter = 0
CNN_model_array_0 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 0.0001
CNN_model_array_1 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 0.001
CNN_model_array_2 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 0.01
CNN_model_array_3 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 0.1
CNN_model_array_4 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 1
CNN_model_array_5 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 10
CNN_model_array_6 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 50
CNN_model_array_7 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 100
CNN_model_array_8 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 500
CNN_model_array_9 = Create_Conv_Model(l1_regularisation_parameter)

l1_regularisation_parameter = 1000
CNN_model_array_10 = Create_Conv_Model(l1_regularisation_parameter)

final_data_array = np.vstack((CNN_model_array_0, CNN_model_array_1))
final_data_array = np.vstack((final_data_array, CNN_model_array_2))
final_data_array = np.vstack((final_data_array, CNN_model_array_3))
final_data_array = np.vstack((final_data_array, CNN_model_array_4))
final_data_array = np.vstack((final_data_array, CNN_model_array_5))
final_data_array = np.vstack((final_data_array, CNN_model_array_6))
final_data_array = np.vstack((final_data_array, CNN_model_array_7))
final_data_array = np.vstack((final_data_array, CNN_model_array_8))
final_data_array = np.vstack((final_data_array, CNN_model_array_9))
final_data_array = np.vstack((final_data_array, CNN_model_array_10))
```

```
CNN_Models_DF = pd.DataFrame()
CNN_Models_DF = CNN_Models_DF.append(pd.DataFrame(final_data_array, columns = ["L1 Regularisation Parameter", "Training Time", "Accuracy on Training Data", "Accuracy on Test Data"]), ignore_index = True)

display(CNN_Models_DF_style)
```

Appendix: Code for Question 3 - All Parts

```
import numpy as np
import tensorflow as tf
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
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.dummy import DummyClassifier
import warnings
warnings.filterwarnings("ignore")
import time
import pandas as pd

num_classes = 10
input_shape = (32, 32, 3)


def Create_Conv_Model(n = 5000) :

    ### Loading the data and splitting data in train and test split. Also normalising the data

    # the data, split between train and test sets
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
    x_train = x_train[1:n]; y_train=y_train[1:n]

    # Scale images to the [0, 1] range
    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
    print("orig x_train shape:", x_train.shape)

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)


    ### Setting the ConvNet parameters and Training the ConvNet model
    model_name = "cifar_" + str(n) + "_MaxPool.model"
    use_saved_model = False
    if use_saved_model:
        model = keras.models.load_model(model_name)
    else:
        model = keras.Sequential()

        model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))

        #Removed Strides from here
        model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
        #Adding Maxpool layer here
        model.add(MaxPooling2D(pool_size = (2, 2)))


        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))

        #Removed Strides from here
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        #Adding Maxpool layer here
        model.add(MaxPooling2D(pool_size = (2, 2)))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
        model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
        model.summary()
        batch_size = 128
        epochs = 20

        ##Saving the time to train the CNN
        start_time = time.time()
        history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
        model.save(model_name)
        train_time = time.time() - start_time

        ### Plot of Model Accuracy Vs.Epochs
        plt.figure()
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title(f'model accuracy for training size {n} with Maxpool')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')


        ### Plot of Model Loss Vs.Epochs
        plt.figure()
        plt.subplot(212)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title(f'model loss for training size {n}  with Maxpool')
        plt.ylabel('loss'); plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.show()


    ### Testing model accuracy on Train Data
    preds_train = model.predict(x_train)
    y_pred_train = np.argmax(preds_train, axis=1)
    y_train1 = np.argmax(y_train, axis=1)
    print(f'Model Accuracy on Train Data for Training Size: {n}  with Maxpool')
    print(classification_report(y_train1, y_pred_train))
    print(confusion_matrix(y_train1, y_pred_train))


    ### Testing model accracy on Test data
    preds_test = model.predict(x_test)
    y_pred_test = np.argmax(preds_test, axis=1)
    y_test1 = np.argmax(y_test, axis=1)
    print(f'\n\nModel Accuracy on Test Data for Training Size: {n}  with Maxpool')
    print(classification_report(y_test1, y_pred_test))
    print(confusion_matrix(y_test1, y_pred_test))

    training_size_str = str(n)
    training_time_str = str(round(train_time, 2)) + ' seconds'
    accuracy_score_train = round(accuracy_score(y_train1, y_pred_train) * 100, 2)
    accuracy_score_train_str = str(accuracy_score_train) + '%'
    accuracy_score_test = round(accuracy_score(y_test1, y_pred_test) * 100, 2)
    accuracy_score_test_str = str(accuracy_score_test) + '%'


    CNN_model_Params_array = pd.array([training_size_str, training_time_str, accuracy_score_train_str, accuracy_score_test_str])

    return CNN_model_Params_array

n = 5000
```

```
CNN_model_array_5K = Create_Conv_Model(n)


CNN_Models_DF = pd.DataFrame()

CNN_Models_DF = CNN_Models_DF.append(pd.DataFrame([CNN_model_array_5K], columns = ["Training Size", "Training Time", "Accuracy
on Training Data", "Accuracy on Test Data"]), ignore_index = True)

CNN_Models_DF = CNN_Models_DF.style.set_properties(**{'text-align': 'center'})
display(CNN_Models_DF)
```

Appendix: Code for Question 4 - All Parts

```
import numpy as np
import tensorflow as tf
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
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
from sklearn.dummy import DummyClassifier
import warnings
warnings.filterwarnings("ignore")
import time
import pandas as pd

num_classes = 10
input_shape = (32, 32, 3)

def Create_Conv_Model(n = 5000) :

    ### Loading the data and splitting data in train and test split. Also normalising the data

    # the data, split between train and test sets
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
    x_train = x_train[1:n]; y_train=y_train[1:n]

    # Scale images to the [0, 1] range
    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
    print("orig x_train shape:", x_train.shape)

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)


    ### Setting the ConvNet parameters and Training the ConvNet model
    model_name = "cifar_TD.model"
    use_saved_model = False
    if use_saved_model:
        model = keras.models.load_model(model_name)
    else:
        model = keras.Sequential()

        model.add(Conv2D(8, (3, 3), padding= 'same', input_shape= x_train.shape[1: ], activation = 'relu' ) )
        model.add(Conv2D(8, (3, 3), strides = (2,2) , padding= 'same', activation = 'relu' ) )
        model.add(Conv2D(16, (3, 3), padding= 'same',  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)))
        model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
        model.summary()

        batch_size = 128
        epochs = 20

        ##Saving the time to train the CNN
        start_time = time.time()
        history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
        model.save(model_name)
        train_time = time.time() - start_time

        ### Plot of Model Accuracy Vs.Epochs
        plt.figure()
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title(f'model accuracy for Thinner Deeper Network for training size {n}')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')


        ### Plot of Model Loss Vs.Epochs
        plt.figure()
        plt.subplot(212)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title(f'model loss for Thinner Deeper Network for training size {n}')
        plt.ylabel('loss'); plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.show()


    ### Testing model accuracy on Train Data
    preds_train = model.predict(x_train)
    y_pred_train = np.argmax(preds_train, axis=1)
    y_train1 = np.argmax(y_train, axis=1)
    print(f'Model Accuracy on Train Data for Thinner Deeper Network for training size {n}')
    print(classification_report(y_train1, y_pred_train))
    print(confusion_matrix(y_train1, y_pred_train))


    ### Testing model accracy on Test data
    preds_test = model.predict(x_test)
    y_pred_test = np.argmax(preds_test, axis=1)
    y_test1 = np.argmax(y_test, axis=1)
    print(f'\n\nModel Accuracy on Test Data Thinner Deeper Network for training size {n}')
    print(classification_report(y_test1, y_pred_test))
    print(confusion_matrix(y_test1, y_pred_test))


    if n == 50000 :
        training_size_str = '50000 (Full Data Set)'
    else :
        training_size_str = str(n)
    training_time_str = str(round(train_time, 2)) + ' seconds'
    accuracy_score_train = round(accuracy_score(y_train1, y_pred_train) * 100, 2)
    accuracy_score_train_str = str(accuracy_score_train) + '%'
    accuracy_score_test = round(accuracy_score(y_test1, y_pred_test) * 100, 2)
    accuracy_score_test_str = str(accuracy_score_test) + '%'


    CNN_model_Params_array = pd.array([training_size_str, training_time_str, accuracy_score_train_str, accuracy_score_test_str])

    return CNN_model_Params_array

n = 5000
CNN_model_array_5K = Create_Conv_Model(n)

n = 10000
CNN_model_array_10K = Create_Conv_Model(n)

n = 20000
```

```
CNN_model_array_20K = Create_Conv_Model(n)

n = 30000
CNN_model_array_30K = Create_Conv_Model(n)

n = 40000
CNN_model_array_40K = Create_Conv_Model(n)

n = 50000
CNN_model_array_50K = Create_Conv_Model(n)

final_data_array = np.vstack((CNN_model_array_5K, CNN_model_array_10K))
final_data_array = np.vstack((final_data_array, CNN_model_array_20K))
final_data_array = np.vstack((final_data_array, CNN_model_array_30K))
final_data_array = np.vstack((final_data_array, CNN_model_array_40K))
final_data_array = np.vstack((final_data_array, CNN_model_array_50K))

CNN_Models_DF = pd.DataFrame()
CNN_Models_DF = CNN_Models_DF.append(pd.DataFrame(final_data_array, columns = ["Training Size", "Training Time", "Accuracy on
Training Data", "Accuracy on Test Data"]), ignore_index = True)

display(CNN_Models_DF)
```