Question (i):

Answer (a):

To know the number of iterations, first thing to do is knowing how many strides we must move the kernel and know the position of the kernel in the image. To do this we have to calculate length we must move in the direction of X axis and the length we must move in the direction of Y axis.

Stride_X => len(image) – len(kernel) – 1
Stride_Y => len(image [0]) – len(kernel) – 1

If both the Stride_X and Stride_Y are positive numbers, we can say that the length of image (pixel length) is less than that of the length of kernel (pixel length). This is a requirement for the convolution to happen.

Then we iterate the kernels over the image using the calculated Stride_X and Stride_Y. Next to compute the resulting value with the respective value of x and y iteration in Stride_X and Stride_Y using:

sum1 = sum1 + (kernel[k][l] * img[i + k][j + l])
(Where k and l are the positions in the kernel matrix)

Basically, for each position in the kernel, we multiply it to the corresponding position in the image with the offset (i,j). After computing the values for all the values of i and j, we append the values into each row of the corresponding result matrix.

Code:

```
def convolution(img: np.array, kernel: np.array): #Creating a function that
accepts numpy arrays

#Get the positions of kernel on image
    strides_x = len(img) - len(kernel) + 1
    strides_y = len(img[0]) - len(kernel[0]) + 1
    assert strides_x > 0 and strides_y > 0, "Error: Kernel larger than the
image"
    conv_matrix = []
    for i in range(strides_x): #Kernel Position iteration
        row = []
        for j in range(strides_y):
            sum1 = 0
            for k in range(len(kernel)): #number of columns in kernel
                for l in range(len(kernel[0])): #number of rows in kernel
                    sum1 = sum1 + (kernel[k][l] * img[i + k][j + l])
#Convolution with offset i,j values in the image
            row.append(sum1)
        conv_matrix.append(row)
    return conv_matrix
```

Example: Convolution Example

$$\begin{bmatrix} 1 & 6 & 8 & 4 & 5 \\ 2 & 1 & 6 & 5 & 7 \\ 6 & 2 & 3 & 9 & 1 \\ 3 & 5 & 5 & 6 & 7 \\ 4 & 4 & 3 & 6 & 2 \end{bmatrix} \quad \text{X} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \Rightarrow$$

Step 1: $\begin{bmatrix} 1 & 6 \\ 2 & 1 \end{bmatrix} \quad \text{X} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ **=> 1+12+6+4 =23**

Step 2: $\begin{bmatrix} 6 & 8 \\ 1 & 6 \end{bmatrix} \quad \text{X} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ **=> 6+16+3+24 = 49**

.

.

.

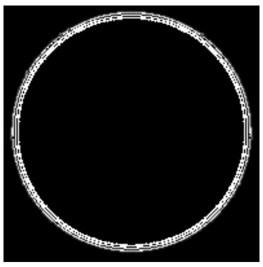
$$\begin{bmatrix} 23 & 49 & 54 & 57 \\ 30 & 32 & 61 & 51 \\ 39 & 43 & 70 & 65 \\ 41 & 49 & 50 & 46 \end{bmatrix}$$

Answer (b):

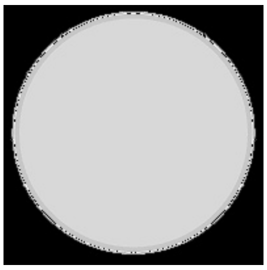
Below is the picture(circle) I chose.

Using the code given in the assignment sheet, I applied the given kernels to the image chosen. The results I obtained are below.

Kernel 1: This given kernel after convolution with the image, detects the edges in it. The output of the image is below.



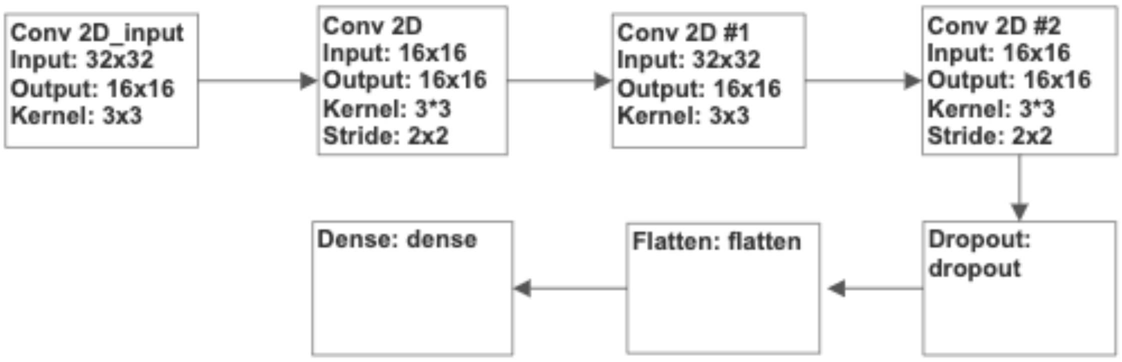
Kernel 2: The given kernel matrix after convolution with the image, sharpens it. The output of the image is below.



Question (ii):

Answer (a):

The code given below has the following architecture.



The above is a sequential model with the following layers.

- A convolution 2D layer with 16 output channels. It has a kernel size of 3*3, padding which enables the output to be the same size as the input (this is done by adding additional zeroes to the input image, so that multiplying it by 3*3 kernel does not reduce the size of the image). The input size is 32*32. Finally, it has a 'relu' activation function (Rectified Linear Activation Function) which maps all the positive number to the same number and the negative numbers to be 0 (x<=0 => 0; x>0 => x).

- The next is also a Convolution 2D layer with 16 output filters and kernel size of 3*3, with same padding and 'relu' activation function. The only difference here is the stride which is 2*2. This determines how much the kernel window moves by (if we don't give anything, the default is 1) and is used to down sample the image.

- The next two layers are the same two layers as above respectively, the only difference being there are 32 output filters instead of 16.

- The next layer is a Dropout layer. During the training time, this randomly sets input units to be 0 with a rate of 0.5 as mentioned. This reduces over-fitting. Inputs that are not set to 0 are scaled up by 1/(1-rate). This is done so that, the sum across all the inputs doesn't change.

- The next layer is a flatten layer. This simply converts all the resultant 2D arrays into single long continuous linear vector.

- The last layer is a dense layer which means it is deeply connected with its previous layer (All the nodes in the current layer are connected with every other node in its previous layer). The num_classes are 10 which is specified in the code downloaded, it uses 'softmax' activation function which converts a vector of n real numbers into a probability distribution of n probable outcomes (gives the output of a PDF [probability distribution function]). It uses a kernel_regularizer (this is used to avoid overfitting) with L1 penalty with lambda value to be 0.0001. This penalty limits the size of coefficients by adding the absolute value of the magnitude of weights. Lasso Regression uses this penalty.

Answer (b):

I trained on 5K images with the code given. The results below.

(i)

How many parameters does keras say this model has?
The total parameters this model has given by the Keras is 37,146.

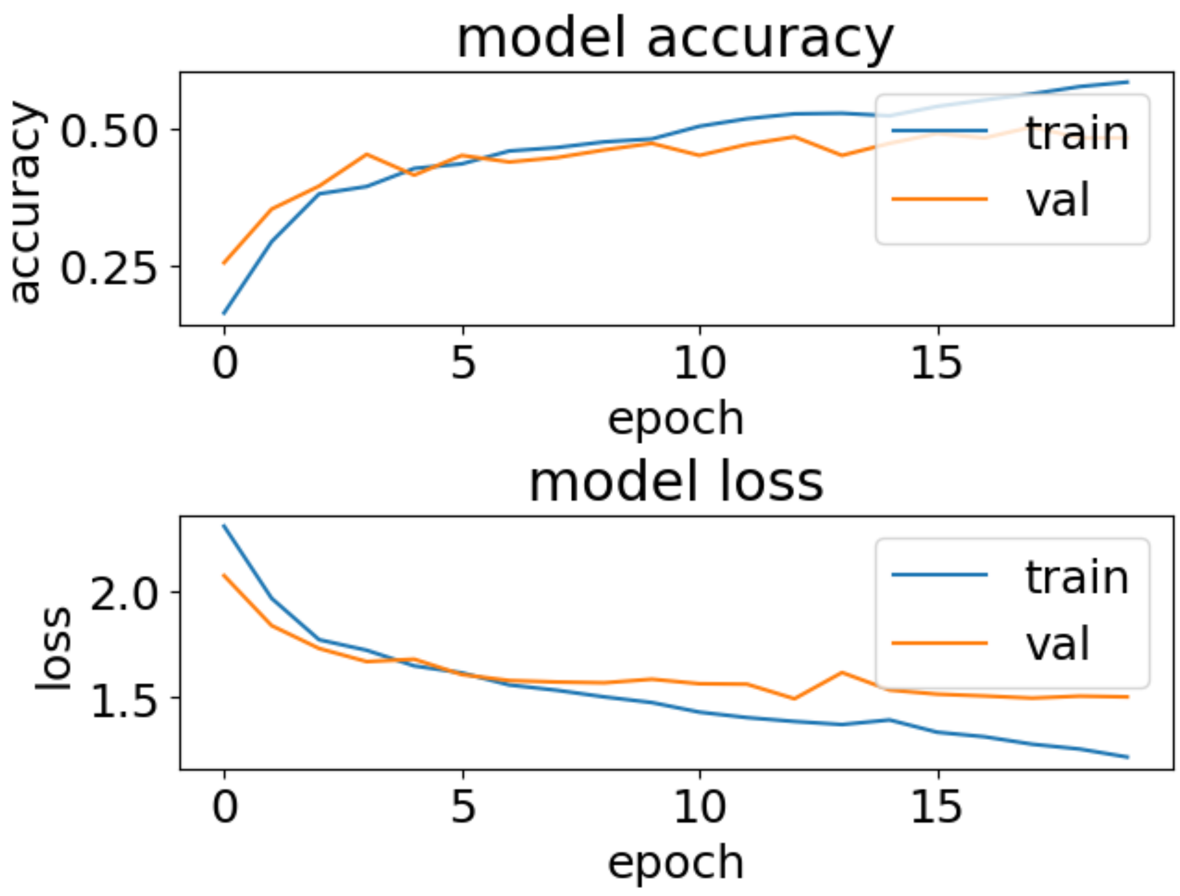
Which layer has the most number of parameters and why?
The Dense layer has the most number of parameters with 20,490 because every node in the layer is connected with every node from the previous layer (densely connected).
The number of parameters in the Dense layer is calculated by => output_channels*(input_channels+1) => So in here it becomes, 10 * (2048+1) => 20,490.

How does the performance of the test data compare with the performance of the training data?
The training accuracy obtained is 0.63 which is 63%. The test data accuracy obtained was much lesser at 0.50 which is 50%. The accuracy of the test set is much lesser than the training because the model parameters are tuned to the training set, but test data is unseen to it.

Compare this model with a simple baseline model?
I compared the model with the baseline model which is a dummy classifier. The accuracy came out to be 0.10 which is 10% on both train and test sets. This is because there are 10 classes in the dataset and all are equally common in the dataset, given one class for the classifier, it would be 10% accurate on the given data.

(ii)

The graph obtained from the given code is given below.



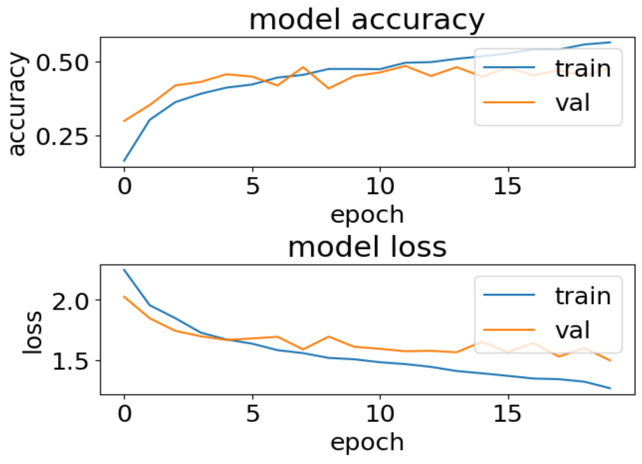
When we look at the model accuracy graph, after 10 gradient descent epochs, you can see that the validation accuracy is becoming lesser than that of the training accuracy. So, we can say that after a few epochs (>10) the model is getting over fitted. Similarly, we can see this in the model loss graph as well. The loss in validation is much lesser than that of training after few epochs. From this also we can depict that the model is overfitting.

(iii):

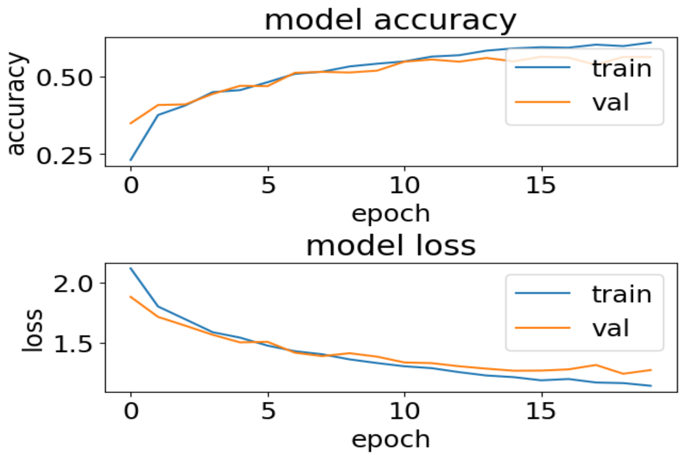
By training with larger number of images (larger training dataset), I obtain with the following accuracy and time.

Training Images	Train Time	Train Accuracy	Validation Accuracy
5000	14.51	0.61	0.49
10000	27.90	0.65	0.55
20000	53.63	0.70	0.62
40000	106.39	0.72	0.68

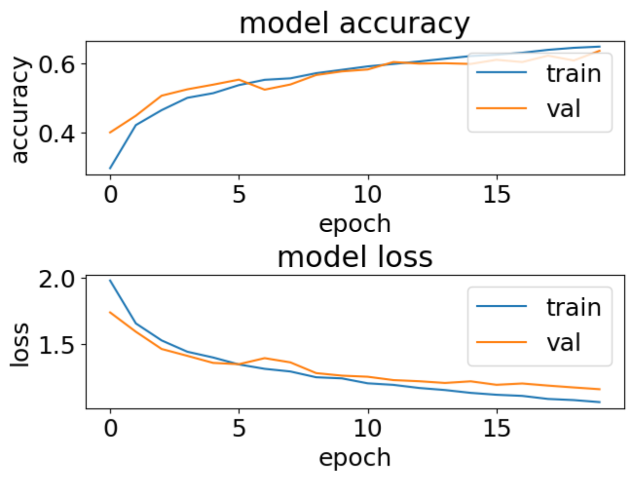
Training set = 5K



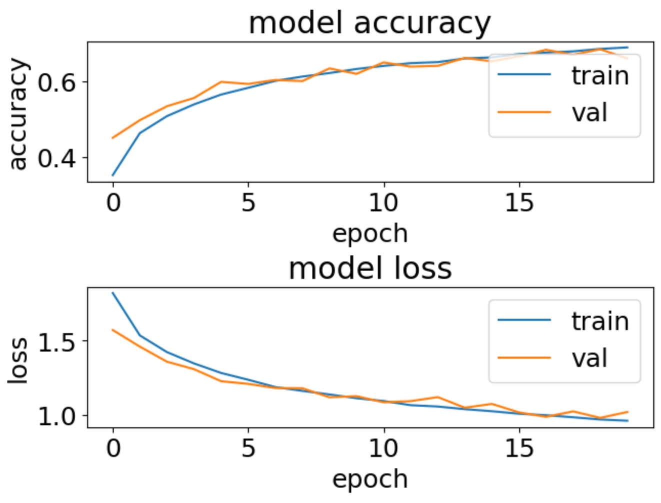
Training set = 10K



Training set = 20K



Training set = 40K



The validation accuracy increases with the increase of the training set. This is because, there are a lot more data points used, and the model has more data to be trained. Increase in the data reduces the chance the model is overfitted. This comes at a cost that this increases training time. So, there is a trade-off between training time and validation accuracy. When the training dataset is set to 40000, we can see that the training accuracy and the testing accuracy is very similar. Also, the loss and accuracy graphs are much smoother in contrast with the plots when number of images is 5000 which shows a lot of variances at higher epochs (high variance) which is a case for overfitting.

(iv):

Below are the results for the training accuracy and the validation accuracy with increasing value of L1 Regularization Parameter on 5K images. We can see that as the penalty term increases, both the training and the validation accuracy is reduced. After the term value is greater than 1, the accuracy is equal to the dummy model which predicts the most common class (0.1).

L1 Regularization Parameter	Training Accuracy	Validation Accuracy
0	0.64	0.51
0.0001	0.61	0.49
0.001	0.57	0.47
0.01	0.46	0.42
0.1	0.38	0.6
1	0.19	0.18
10	0.10	0.10
100	0.10	0.10

When the regularization parameter is lesser, we can see that there is a bigger difference between training accuracy and the validation accuracy. From this we can say that the model is on the onset of over fitting. As the parameter value increases, we can see that the difference between these two accuracies is reduced. However, from the above results of accuracies of the model when dataset is larger than 5K, we can deduce that, increasing the amount of training data is likely much larger on managing overfitting.

Question (c):

(i)

Instead of using stride to downsample, I added a maxpooling layer after the previously strided layer. Then remove the strided layer. The rest of the layers like Dense, Flatten and Dropout layers are not changed anything.

New Maxpooling layer is added using: model.add(Maxpooling2D(2,2))

(ii)

Using the previous Regularization parameter of 0.0001 and the training size with 5000 images, the model is trained. Below are the output accuracies with the above parameters.

Approach	Train Time	Train Accuracy	Validation Accuracy	Parameters
Old Approach with Stride	14.51s	0.61	0.49	37,146
New Approach with Max Pooling	18.63s	0.68	0.55	37,146

The New approach has longer computation time because we are not down sampling which reduces the computation time. Also, we are adding a new maxpooling step in between. So, adding the new layer and removing the down sampling during the convolution is causing the training time to increase a bit. The training and test accuracies are not changed much improving with training accuracy improving from 0.61 to 0.68 and validation accuracy improving from 0.55 from 0.49.

<u>Question (d) optional:</u>

I added the given architecture which is thinner and deeper instead of the previous architecture and trained it for 50,000 images which is the size of the dataset and 40 epochs.

Training Accuracy: 0.76
Testing Accuracy: 0.70
Time: 260.91s

By increasing the depth of model (by adding new layers to the model), there is more chance to over fit. So, we must use larger datasets for the model to avoid overfitting. This will increase the training and validation accuracy. At the end of 40 epochs, we can see that the training and validation accuracy is 76% and 70% respectively. So, we can say that the model here is not overfitting. But this further increases the training time, with 20 epochs taking 125 seconds and 40 epochs taking 260.91 seconds.

<u>Appendix:</u>

<u>Question (i):</u>

<u>Part A:</u>

```
from PIL import Image
import numpy as np
def convolution(img: np.array, kernel: np.array):
    strides_x = len(img) - len(kernel) + 1
    strides_y = len(img[0]) - len(kernel[0]) + 1
    assert strides_x > 0 and strides_y > 0, "Error: Kernel larger than the
image"
    conv_matrix = []
    for i in range(strides_x):
        row = []
        for j in range(strides_y):
            sum1 = 0
            for k in range(len(kernel)):
                for l in range(len(kernel[0])):
                    sum1 = sum1 + (kernel[k][l] * img[i + k][j + l])
            row.append(sum1)
        conv_matrix.append(row)
    return conv_matrix
```

<u>Part B:</u>

```
im = Image.open('circle.png')
newsize = (200, 200)
```

```
im1 = im.resize(newsize)

rgb = np.array(im1.convert('RGB'))
r = rgb[:, :, 0]

kernel1 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
kernel2 = np.array([[0, -1, 0], [-1, 8, -1], [0, -1, 0]])

im_k1 = convolution(r, kernel1)
im_k2 = convolution(r, kernel2)
Image.fromarray(np.uint8(im_k1)).show()
Image.fromarray(np.uint8(im_k2)).show()
```

Question (ii):

Part (a,b):

```
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 time
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys

# Model / data parameters
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 #change it to 10k,20k,40k for part iii
x_train = x_train[1:n]; y_train=y_train[1:n]
#x_test=x_test[1:500]; y_test=y_test[1:500]

# 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))) #change 
it to 0,0.001,0.01,0.1,1,10,100 for part iv
      model.compile(loss="categorical_crossentropy", optimizer='adam', 
metrics=["accuracy"])
      model.summary()

      batch_size = 128
      epochs = 20
      start = time.time()
      history = model.fit(x_train, y_train, batch_size=batch_size, 
epochs=epochs, validation_split=0.1)
      end = time.time()
      print("time spent training: ", end - start)
      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 right')
      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 right')
      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))
```

Part (c):

```
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 time
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys

# Model / data parameters
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]
#x_test=x_test[1:500]; y_test=y_test[1:500]

# 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), padding='same', activation='relu'))
      model.add(MaxPooling2D((2,2)))
      model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
      model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
      model.add(MaxPooling2D((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
      start = time.time()
      history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.1)
      end = time.time()
      print("time spent training: ", end - start)
      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 right')
      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 right')
      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))
```

Part (d) optional:

```
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 time
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys

# Model / data parameters
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=50000
x_train = x_train[1:n]; y_train=y_train[1:n]
#x_test=x_test[1:500]; y_test=y_test[1:500]

# 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(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 = 40
```

```
	start = time.time()
	history = model.fit(x_train, y_train, batch_size=batch_size, 
epochs=epochs, validation_split=0.1)
	end = time.time()
	print("time spent training: ", end - start)
	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 right')
	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 right')
	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))
```