NAME: ARNAV BHATTACHARYA

STUDENT ID: 22307812

COURSE: MACHINE LEARNING

Answers:

Answer i) a) b)

I have selected an image with distinct edges from the internet. The 3x3 kernel is convolved on the input image after adding the necessary padding to the input image to ensure that the shape of the input and output images are the same. I have taken the red channel of the input image and then convolved the respective kernels on it. The padding to the image is set in the following manner:

padded_image=np.pad(image,x_kernel_shape-2)

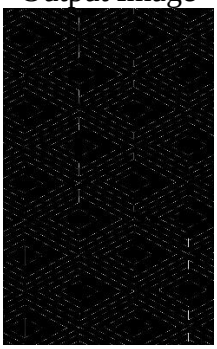
The resultant images are as follows:

Input Image

Kernel1

-1	-1	-1
-1	8	-1
-1	-1	-1

Output Image

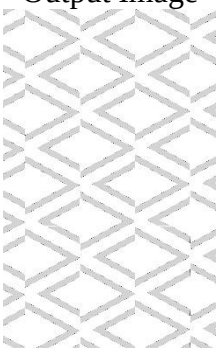


Input Image

Kernel2

0	-1	0
-1	8	-1
0	-1	0

Output Image



Answer ii) a)

The downloaded code is of ConvNet. It consists of the following sequential steps:

- A 2D Convolution Layer of 16 output filters is applied which consists of kernels each of size 3x3. The padding used for this convolution layer is of type “same”. The padding is added to the input data to ensure that the input and the output data are of

the same dimensions. The activation function used is “relu” (rectified linear activation unit). This activation function will map the negative output values to 0 and the positive ones will be mapped to their value.

- Another 2D convolution layer with 16 kernels of size 3x3 is applied. The padding used for this convolution layer is also of type “same”. The padding is added to the input image to ensure that the input and the output image are of the same dimensions. The activation function used is “relu” (rectified linear activation unit). This activation function will map the negative output values to 0 and the positive ones will be mapped to their value. The only difference between the previous step is that the “stride” is set to 2x2 which is used to downsample the image. Stride determines how much the window moves in the next iteration. The default value for stride is 1x1.
- The above two steps are repeated in the same order, but now with a convolution layer with 32 kernels of the size 3x3. The other parameters of the function are the same as before.
- A Dropout regularization is applied with the given probability. This randomly selects nodes to be dropped from the training based on the given probability which in this case is 50%. Dropout Regularization is a simple way to prevent neural networks from overfitting.
- The image is flattened using the tf.keras.layers.Flatten function. This creates a single-dimensional array of the image.
- A Dense layer is applied. It is the most basic layer in neural networks, where all the outputs from the previous layer are fed to the next layer using all of its neurons. Each neuron provides one output to the following layer. The downloaded code uses 10 classes or units in this step. The activation function used is “softmax” and the regularizer function which is applied to the kernel weights matrix is of the L1 class. The L1 penalty value is 0.0001. The L1 regularization penalty is computed as loss=L1*reduce_sum(abs(x)).

Answer ii) b) i)

On executing the downloaded code, we observe the following metrics:

- According to keras, the model has 37,146 parameters.
- The accuracy of the model for training data is observed to be 0.6 or 60%. The accuracy of the model for test data is observed to be 0.49 or 49%. Thus, the accuracy decreases by 0.11 or 11% when using the test data as compared to the training data.

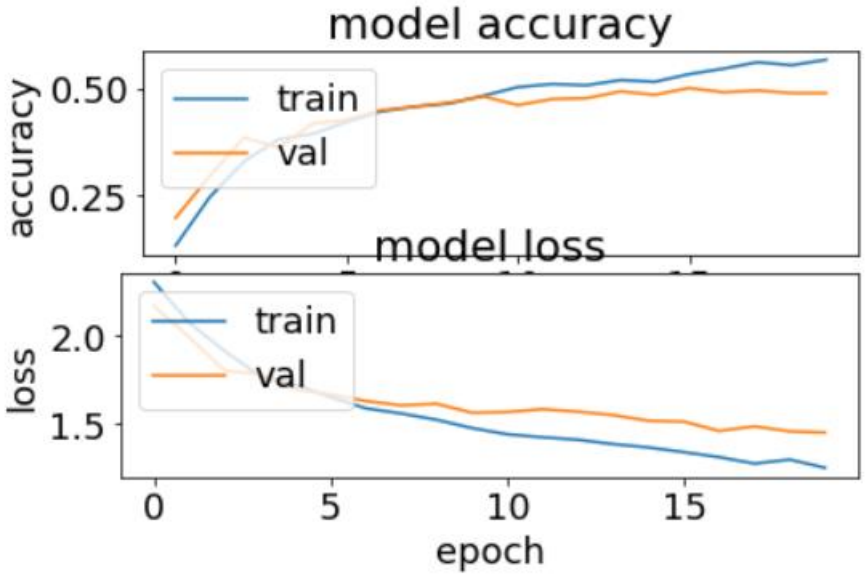
I trained a DummyClassifier model as the baseline classifier. This model predicts the most common label. On using the same code for generating the accuracy metrics, we can easily observe that the accuracy for our baseline classifier is 0.1 or 10%.

Our model when run on the test data, has an accuracy of 0.49 or 49%. Thus, it is better than the baseline classifier by 0.39 or 39%.

When our model is run on the training data, it has an accuracy of 0.6 or 60%. Thus, making it better than our baseline classifier by 0.5 or 50%.

Answer ii) b) ii)

The graph acquired for the history variable from the downloaded code is as follows:



On observing the second plot, the one with the loss on the y-axis and epoch (training iterations) on the x-axis, we notice that for smaller epoch values, the loss reduces by a significant amount quickly. Thus, it can be stated that the model under-fits for smaller values of epoch and then start tuning more parameters as the epoch value is increased and fits the data. As we keep increasing the epoch value, the loss stabilizes for a while and then starts to fluctuate as we keep on increasing the epoch value. So, for larger values of the epoch, the model becomes unpredictable as it has high variance. We notice that the model over-fits the data as we keep on increasing the epoch value, as the model's performance does not improve when we continue training and tuning parameters.

Answer ii) b) iii)

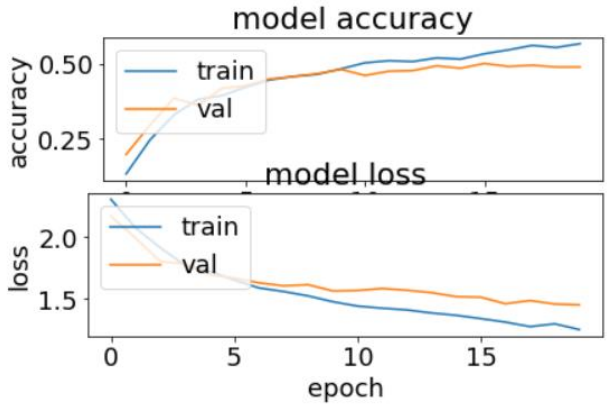
The time taken to train, the prediction accuracy of training data and testing data, when the size of the training data points are 5K, 10K, 20K, and 40K respectively are tabulated below:

Training Data Points	Training Time	Training Accuracy	Testing Accuracy
5K	172.713s	0.63	0.50
10K	220.108 s	0.64	0.55
20K	462.054 s	0.70	0.63
40K	914.919 s	0.71	0.66

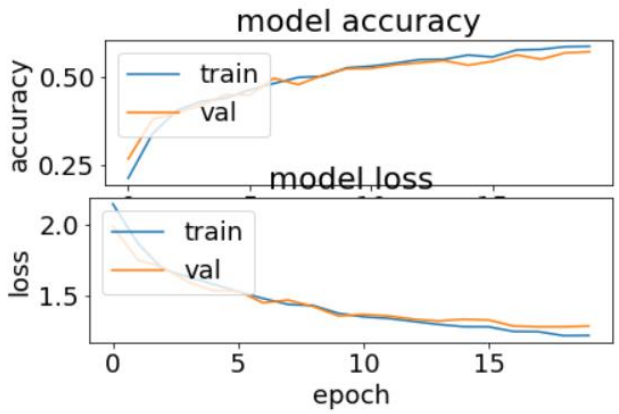
From the above table, we observe that on doubling the number of training data points, the training time approximately doubles. Also, the training and testing accuracy of the model increase by a significant amount when the number of data points is increased from 5K to 10K. On increasing the number of data points from 20K to 40K, the training and testing accuracy of the model increases by an amount lesser than in the above-mentioned case. We

note that on increasing the number of training data points, the accuracy and the time is taken increase. So, one needs to decide on how much training of data is to be performed. This depends completely on the use case.

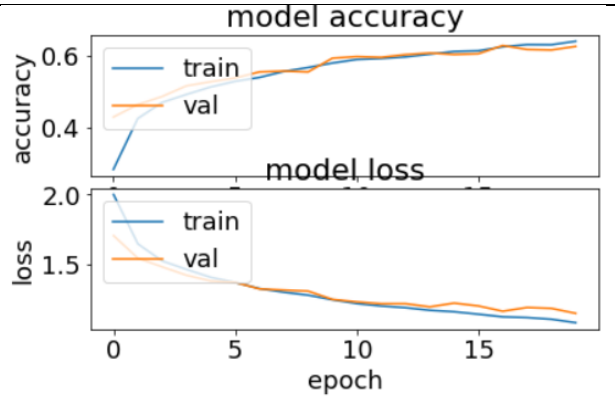
For each of the cases, the following “history” variable is obtained:



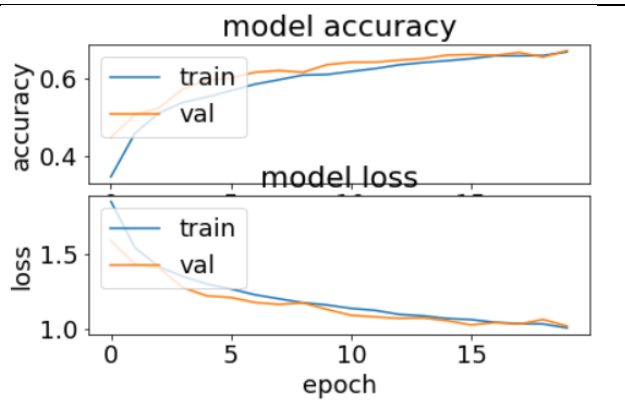
Number of Training Data Points = 5K



Number of Training Data Points = 10K



Number of Training Data Points = 20K



Number of Training Data Points = 40K

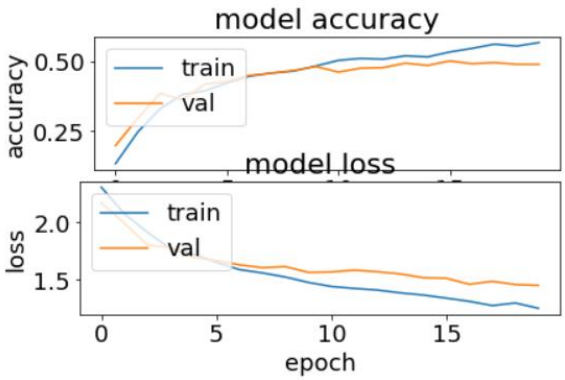
From the above plots, we notice that the accuracy inferred from the loss or loss against epoch iteration plots is much smoother for a higher number of training data points. For 5K number of training data points, there are big spikes in the plot, whereas, for 40K number of training data points, the plot is much smoother with negligible spikes.

Answer ii) b) iv)

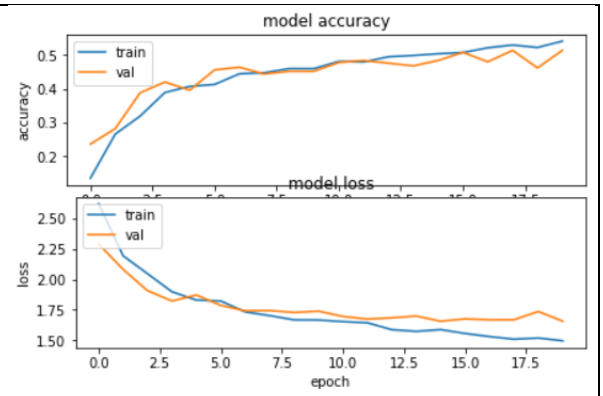
Loss Parameter	Training Accuracy	Testing Accuracy
0.0001	0.63	0.50
0.001	0.59	0.48
0	0.64	0.51
1	0.22	0.22
10	0.10	0.10
100	0.10	0.10

L1 regularization will make the parameters with sparse values. The values of parameters will be close to zeros. From the above table, we can observe how the training and testing accuracies vary with the loss parameter. As the loss parameter is increased from 0.0001 to 0, the training and testing accuracy have less to negligible variance. When the loss parameter is set to 1, there is a drastic reduction in the training and testing accuracies of 0.42 or 42% and 0.29 or 29% respectively. When the loss parameter is set to 10 and 100, the training and testing accuracy is constant at 0.10 or 10%. As we keep increasing the loss parameter for L1 regularization, the training and testing accuracies vary almost negligibly till the loss parameter is 0. On increasing the loss parameter value from 0, we note that there is a drastic reduction in the training and testing accuracy when the loss parameter is set to 1. On further increment of loss parameter, we observe that the training and testing accuracies are constant at 0.1 or 10%.

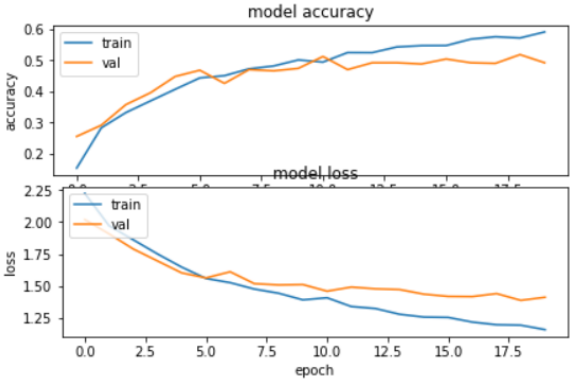
The following plots depict the accuracy and the loss of the model with 5K data points and varying loss parameter values of L1 Regularization.



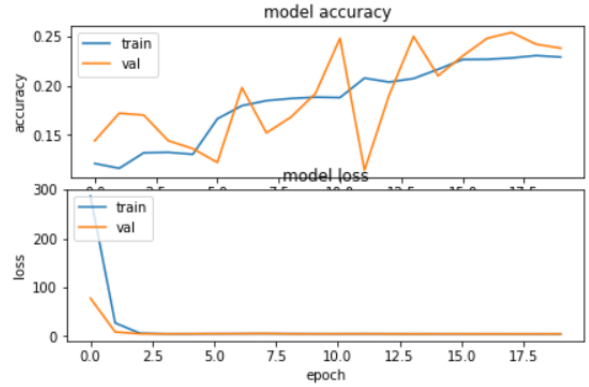
Loss Parameter = 0.0001



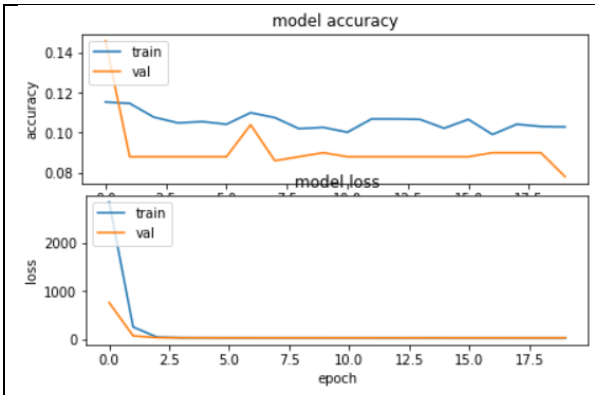
Loss Parameter = 0.001



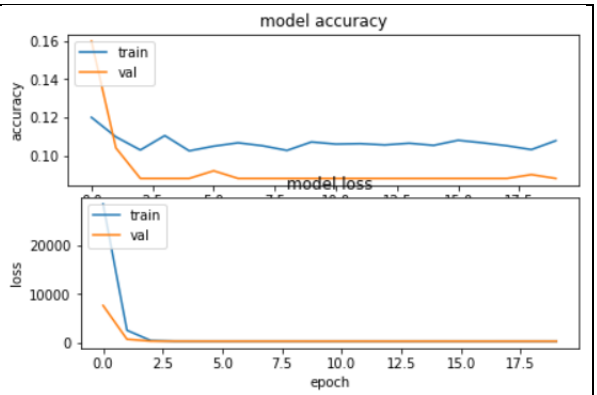
Loss Parameter = 0



Loss Parameter = 1



Loss Parameter = 10



Loss Parameter = 100

Answer ii) c) i)

In the downloaded code of ConvNet, I added the "max-pooling" layer after the 2 convolution layers of 16 and 32 layers of kernels with padding as "same". I also removed the layers which used strides. The remaining layers(density, dropout, and/or flatten layer) are left as it is.

Answer ii) c) ii)

Using the above model with the number of data points set to 5K and the loss parameter set to 0.0001, we train a new model including max-pooling. From the output in the console, we can take note of the following:

Parameters	**Model Without MaxPooling**	**Model With MaxPooling**
Train Time	124.008 s	504.75 s
Total Number of Params	37146	98586
Accuracy of Training Data	0.63	0.81
Accuracy of Testing Data	0.50	0.53

Let's see how the added parameters do in terms of accuracy, we notice a small improvement in test accuracy (improvement of $0.53-0.5 = 0.03$) - it isn't a big improvement but still worth noting. In terms of the training data though the improvement is slightly more considerable: $0.75 - 0.63 = 0.12$.

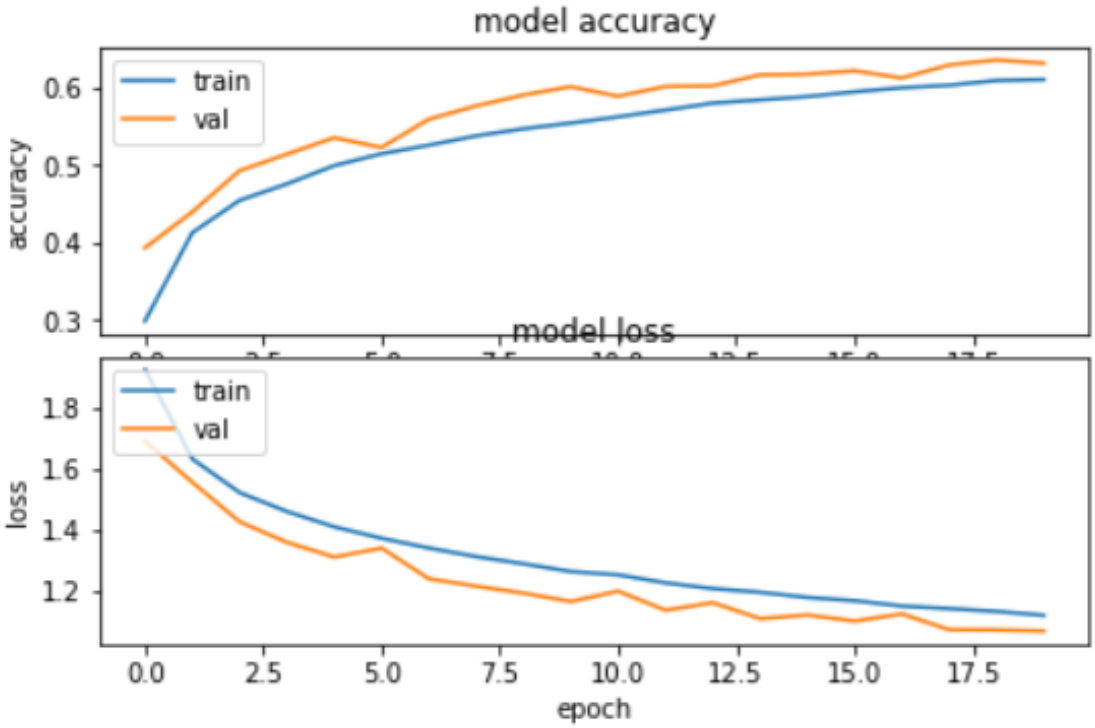
From the above table, we note that there is a huge increase in the time taken for training. The original model took around 124 seconds whereas the model with max-pooling took around 505 seconds. In the new model, we are no longer downsampling the data points as we have removed strides. This explains the increase in the time taken to train the data points.

Also, the number of parameters in the model with max-pooling is almost three times the number of parameters in the model without max-pooling. Mathematically, the model with max-pooling has 61440 more parameters.

The accuracy of the model with max-pooling is more than the model without max-pooling for both training and testing data. Mathematically, the accuracy of training data for the model with max-pooling is 18% more than the accuracy of training data for the model without max-pooling. Whereas, the accuracy of testing data for the model with max-pooling is 3% more than the accuracy of testing data for the model without max-pooling.

Answer ii) d)

The number of data points is set to 50K which is the entire size of the dataset. The accuracy for training data points is 0.65 or 65% whereas for testing data points are 0.62 or 62%. The time taken for the code to run is 829.813 seconds. The plots of the accuracy and the loss of the model are depicted as follows:



Appendix:

```
img1=plt.imread("/content/drive/MyDrive/Machine Learning/Week 8/images/featured-geometric.jpg")

img2=plt.imread("/content/drive/MyDrive/Machine Learning/Week 8/images/mosaic.jpg")

path="/content/drive/MyDrive/Machine Learning/Week 8"

def convolution(image,kernel):

  x_image_shape=image.shape[0]

  y_image_shape=image.shape[1]

  x_kernel_shape=kernel.shape[0]

  y_kernel_shape=kernel.shape[1]

  output_image=np.zeros((x_image_shape,y_image_shape))

  padded_image=np.pad(image,x_kernel_shape-2)

  # print(out.shape)

  for x in range(0,image.shape[0]):

    for y in range(0,image.shape[1]):
```

```
      temp=padded_image[x:x+x_kernel_shape,y:y+y_kernel_shape]
      # print(temp.shape)
      output_image[x,y]=np.sum(np.multiply(temp,kernel))

  return output_image
kernel1=np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
kernel2=np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])
# Taking into consideration only the red channel:
conv1=convolution(img1[::,::,0],kernel1)
conv2=convolution(img2[::,::,0],kernel1)
# Taking into consideration only the red channel:
conv3=convolution(img1[::,::,0],kernel2)
conv4=convolution(img2[::,::,0],kernel2)
cv2.imwrite(os.path.join(path,'result1_k1.jpg'),conv1)
cv2.imwrite(os.path.join(path,'result2_k1.jpg'),conv2)
cv2.imwrite(os.path.join(path,'result1_k2.jpg'),conv3)
cv2.imwrite(os.path.join(path,'result2_k2.jpg'),conv4)

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

# 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 # this selects the size of the training data. Change this to 5K
10K 20K 40K

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)))
```

```
  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))


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


# 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=10000 # this selects the size of the training data. Change this to 5K
10K 20K 40K

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)))

  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))

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

# 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=20000 # this selects the size of the training data. Change this to 5K
10K 20K 40K

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)))
```

```
  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))


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


# 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=40000 # this selects the size of the training data. Change this to 5K
10K 20K 40K

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)))

  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))

n_values=[0.001,0,1,10,100]
for n_value in n_values:

print("=================================================================
")
  print(f'n: {n_value}')
  plt.rc('font', size=10)
  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), 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(MaxPooling2D((2,2)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(n_value))) #
uses n here as 0.001,0,1,100
    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))



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=10)
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
  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))



from sklearn.dummy import DummyClassifier
# Dummy model
dummy_model = DummyClassifier(strategy="most_frequent").fit(x_train,
y_train)
print("Dummy model")
preds_dummy = dummy_model.predict(x_train)
y_pred_dummy = np.argmax(preds_dummy, axis=1)
y_train1_dummy = np.argmax(y_train, axis=1)
print(classification_report(y_train1_dummy, y_pred_dummy))
print(confusion_matrix(y_train1_dummy, y_pred_dummy))

preds_dummy = dummy_model.predict(x_test)
y_pred_dummy = np.argmax(preds_dummy, axis=1)
y_test1_dummy = np.argmax(y_test, axis=1)
print(classification_report(y_test1_dummy, y_pred_dummy))
print(confusion_matrix(y_test1_dummy, y_pred_dummy))
```

```
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=10)
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 = keras.Sequential()

  model.add(Conv2D(8, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))

  model.add(Conv2D(8, (2,2),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

  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))
```