Ritika Sharma 21334470
MSc Computer Science – Intelligent Systems sharmari@tcd.ie

ML Assignment Week 8

This assignment revolves around the concept of Convolutional Networks, a type of Neural Network.

Part i) a) In this part, it is required to implement a vanilla python function to convolve the kernel over an input array.

For this, I have created an numpy array, *inp_array* (randomly generating numbers from 0-20 by asking the size from the user) and a 3*3 size *kernel* numpy array. The user is asked to enter the size of stride as well. After taking input of these values, function `convol2d` is called with parameters "kernel, inp_array, stride". In the function, a submatrix is created to convolve the kernel over the inp_array.

```
submatrix = inp_array[i:i+k[0],j:j+k[1]]
output[i][j] = np.sum(np.multiply(submatrix, kernel))
```

This code works for all types of input array size and stride value, given the kernel is used to find the *Vertical axis.*

Example Output generated –

```
Enter value of n5
Enter value for stride2
Input matrix  [[ 1  2  3  4  5]
 [ 1  3  2  3 10]
 [ 3  2  1  4  5]
 [ 6  1  1  2  2]
 [ 3  2  1  5  4]]
Kernel  [[ 1  0 -1]
 [ 1  0 -1]
 [ 1  0 -1]]
Result of convolution  [[-1. -4.]
 [ 6. -3.]]
```

b) A geometric shape, pentagon, is selected for this part and loaded as RGB array. The kernels are provided in the assignment and these kernels and the converted RGB array with stride = 1 values are passed to the covolve2d function. The outputs are saved as output1 and output2 using the Image Library provided by Python.

```
img = Image.fromarray(output2, 'RGB')
img.save('output2.png')
```

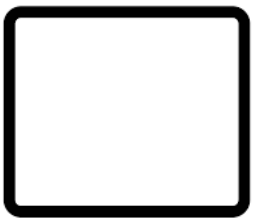

Fig. Original Image

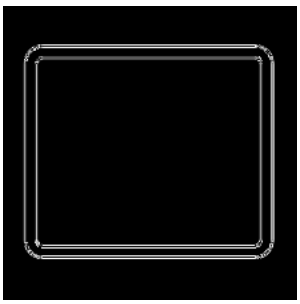

Fig. Output 1

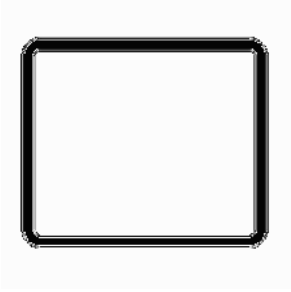

Fig. Output 2

Ritika Sharma 21334470
MSc Computer Science – Intelligent Systems sharmari@tcd.ie

Part ii) a) For this part we are using the famous CIFAR-10 dataset mainly used for Image Classification using Deep Learning. It has images belonging to 10 different classes. The size of the images used in this dataset are 32*32*3 (3- RGB format). The data is then trained by dividing every image and its pixel from 255, thus making the scale from 0-1. This is done to normalise the image and thus making our model run faster.

A model has multiple convolutional network layers, with downsampling layers, followed by Fully Connected Layer and then output.

(Convolutional Network layer → Pooling layer) *n → Flatten output → Fully Connected Layer

* *The convention followed to count the layers used in this report is – Any layer having weights or parameters will be counted as layer (ex. Strided convolution layer) and MaxPooling will be counted along with the Convolution Layer*.

Architecture of the ConvNet used in the given code–

1. For the Convolutional Network, the first layer is the Convolution layer. Here, the function used is Conv2D (used mainly for images).
2. For downsampling the image Strided Convolution is used
3. Fully connected Layer to do the prediction of the image.

Calculating Kernel Size – The 2d Kernel size is passed in the function Conv2d.

Depth of Kernel = Depth of Input Shape

Calculating Output Size –

Depth of Output = No of channels passed

If padding is passed as “Same” then the output size is same as input shape, else it can be calculated as floor((input shape – kernel /stride)+1).

Strided Convolution Layer is used to downsample the image, as we can see in the table below.

Layer Name	Input Size, Channels	Kernel Size	Padding/ Stride	Parameters	Output Size
Conv Layer 1	32*32*3, 16	3*3	Same	448	32*32*16
Strided 1	32*32*16, 16	3*3	2*2	2320	16*16*16
Conv Layer 2	16*16*16, 32	3*3	Same	4640	16*16*32
Strided 2	16*16*32, 32	3*3	2*2	9248	8* 8* 32
FC layer 1	8* 8* 32	-	-	20490	10

b) i) This model uses total of `37,146` parameters. The layer with the most parameter is the Dense Layer. Dense Layer is the fully connected layer, which essentially means that it feeds all the outputs from the previous layer to all the neurons of the current dense layer. Along with all the weights and bias, this layer contributes to the large number of parameters. This layer trains the model to predict.

We have 2048 parameters after flattening and 1 bias. And we have 10 filters. So,

Total = (2048 +1) *10 = 20490

The accuracy on training data is 63% and on testing data is 51%.

Ritika Sharma 21334470
MSc Computer Science – Intelligent Systems sharmari@tcd.ie

We use the most recent classifier strategy to compare the accuracy of test data. We notice that our model performs better than the baseline models.

Baseline classifier accuracy–

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

ii) For datapoint 5000, we get

```
Epoch 1/20

36/36 [==============================] - 6s 152ms/step - loss: 2.3087 -
accuracy: 0.1122 - val_loss: 2.1541 - val_accuracy: 0.2220
Epoch 2/20
36/36 [==============================] - 5s 146ms/step - loss: 2.0672 -
accuracy: 0.2483 - val_loss: 1.9615 - val_accuracy: 0.2960
Epoch 3/20
36/36 [==============================] - 5s 152ms/step - loss: 1.9232 -
accuracy: 0.3161 - val_loss: 1.7592 - val_accuracy: 0.3960
Epoch 4/20
36/36 [==============================] - 6s 155ms/step - loss: 1.7735 -
accuracy: 0.3790 - val_loss: 1.7227 - val_accuracy: 0.4100
Epoch 5/20
36/36 [==============================] - 5s 147ms/step - loss: 1.6550 -
accuracy: 0.4210 - val_loss: 1.6576 - val_accuracy: 0.4320
Epoch 6/20
36/36 [==============================] - 5s 143ms/step - loss: 1.6037 -
accuracy: 0.4417 - val_loss: 1.5897 - val_accuracy: 0.4800
Epoch 7/20
36/36 [==============================] - 5s 151ms/step - loss: 1.5620 -
accuracy: 0.4510 - val_loss: 1.5943 - val_accuracy: 0.4800
Epoch 8/20
36/36 [==============================] - 5s 147ms/step - loss: 1.5058 -
accuracy: 0.4692 - val_loss: 1.5611 - val_accuracy: 0.4740
Epoch 9/20
36/36 [==============================] - 5s 147ms/step - loss: 1.4710 -
accuracy: 0.4830 - val_loss: 1.5423 - val_accuracy: 0.4900
Epoch 10/20
36/36 [==============================] - 6s 159ms/step - loss: 1.4187 -
accuracy: 0.5003 - val_loss: 1.5442 - val_accuracy: 0.5000
Epoch 11/20
36/36 [==============================] - 6s 168ms/step - loss: 1.4124 -
accuracy: 0.5114 - val_loss: 1.5036 - val_accuracy: 0.5040
Epoch 12/20
36/36 [==============================] - 7s 198ms/step - loss: 1.3722 -
accuracy: 0.5190 - val_loss: 1.4891 - val_accuracy: 0.5200
Epoch 13/20
36/36 [==============================] - 6s 152ms/step - loss: 1.3475 -
accuracy: 0.5279 - val_loss: 1.4655 - val_accuracy: 0.5200
Epoch 14/20
36/36 [==============================] - 5s 145ms/step - loss: 1.3268 -
accuracy: 0.5423 - val_loss: 1.5082 - val_accuracy: 0.5060
Epoch 15/20
36/36 [==============================] - 6s 154ms/step - loss: 1.3108 -
accuracy: 0.5523 - val_loss: 1.4440 - val_accuracy: 0.5140
Epoch 16/20
```

```
36/36 [==============================] - 5s 148ms/step - loss: 1.2700 -
accuracy: 0.5672 - val_loss: 1.4766 - val_accuracy: 0.5080
Epoch 17/20
36/36 [==============================] - 6s 159ms/step - loss: 1.2680 -
accuracy: 0.5637 - val_loss: 1.4574 - val_accuracy: 0.5020
Epoch 18/20
36/36 [==============================] - 6s 158ms/step - loss: 1.2427 -
accuracy: 0.5750 - val_loss: 1.4139 - val_accuracy: 0.5340
Epoch 19/20
36/36 [==============================] - 6s 153ms/step - loss: 1.2097 -
accuracy: 0.5795 - val_loss: 1.4388 - val_accuracy: 0.5400
Epoch 20/20
36/36 [==============================] - 6s 159ms/step - loss: 1.2161 -
accuracy: 0.5839 - val_loss: 1.4649 - val_accuracy: 0.5160
```

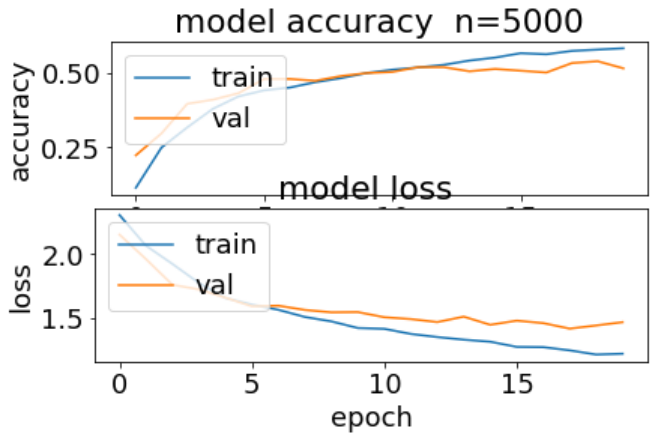

Fig. Model accuracy and loss

Here, we observe that the model accuracy for training dataset is more than testing dataset. Same goes for model loss values, the testing dataset has more data loss than training data. This data suggests that this model has overfitted on the training dataset.

Also, we observe the epoch values, after 10th epoch value, the testing data accuracy and loss values are approximately the same. But for the training data, loss keeps on decreasing and accuracy increases, which directly reflects to the fact that the model is overfitting from the 10th epoch.

iii) Accuracy for different values of weight parameter in L1 regression with 5k training data points.

Training datapoints	Training Accuracy (%)	Test Accuracy (%)	TimeTaken (s)
5k	63	51	113
10k	66	56	221
20k	67	60	362
40k	72	67	741

From the above table we can observe that the accuracy on training data lies between 60-70% for all the values of datapoints but the accuracy on testing data improves as the datapoints increases. Suggesting that overfitting reduces as we pass more training datapoints to the model. But the time taken increases as we increase the datapoints approximately twice.

Ritika Sharma 21334470
MSc Computer Science – Intelligent Systems sharmari@tcd.ie

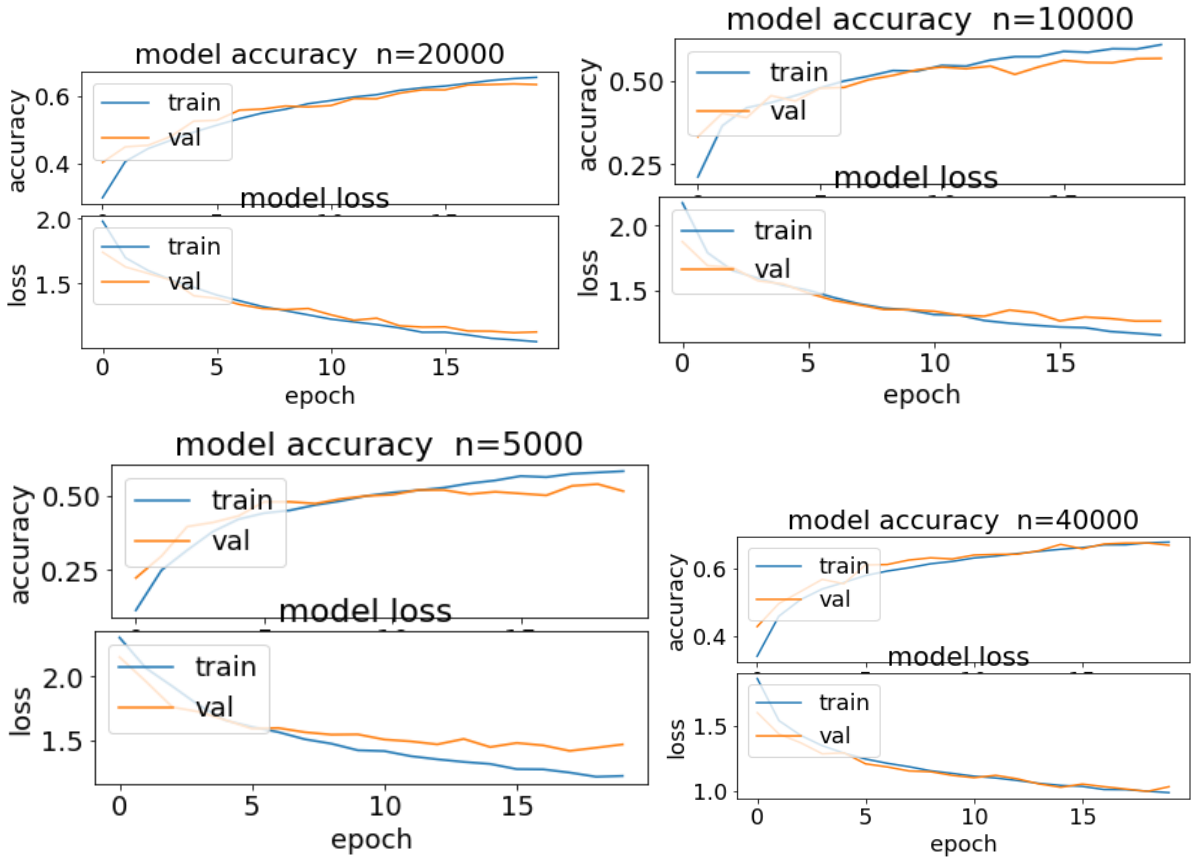

Fig. Model Accuracy and Loss for different datapoints

iv) Accuracy for different values of weight parameter in L1 regression with 5k training data points.

Alpha	Training Accuracy (%)	Test Accuracy (%)
0	63	51
0.1	37	36
10	10	10
0.001	57	50
0.0001	63	51
100	10	10

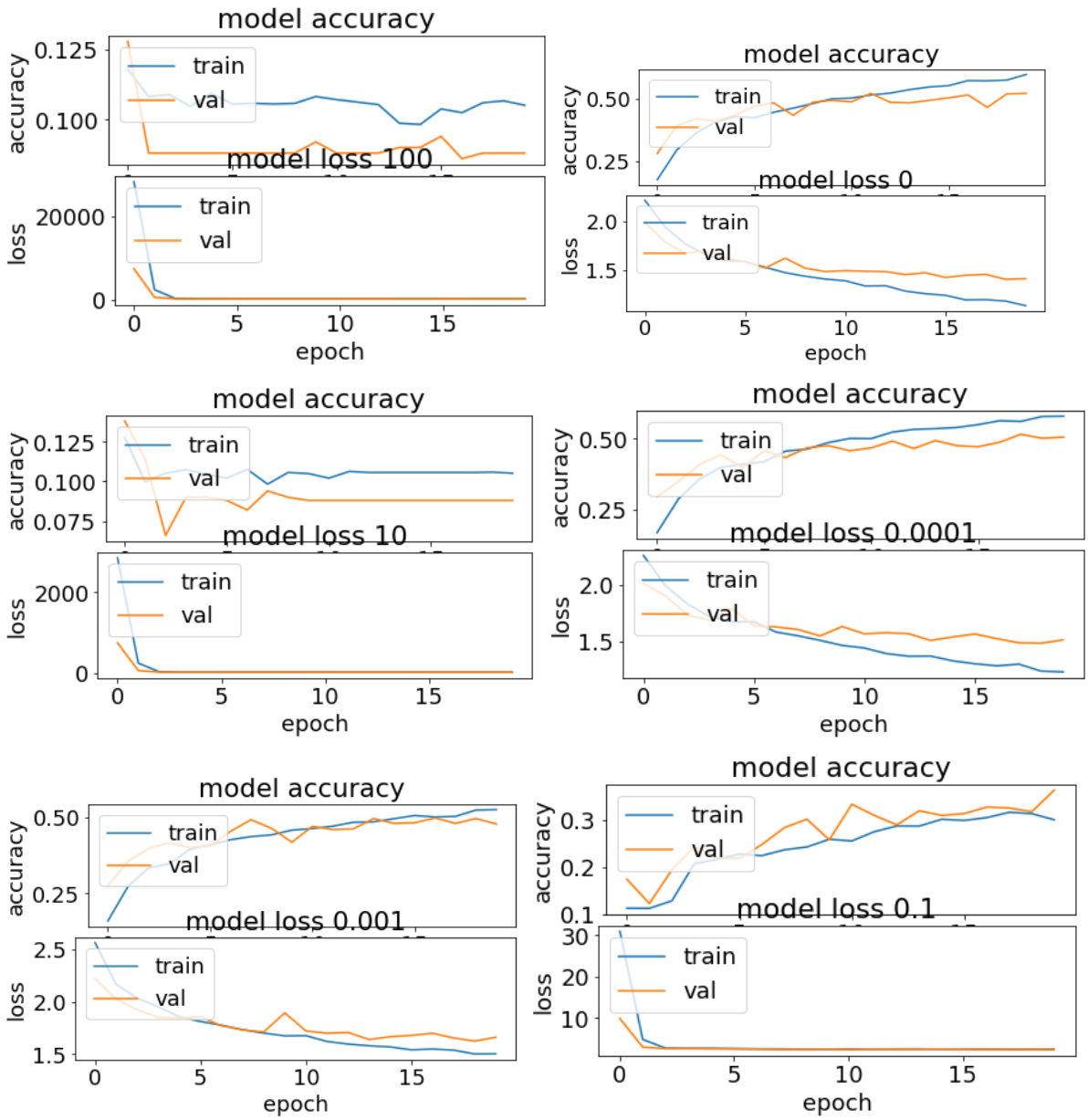

Fig. Model accuracy and loss for different weight parameters

The accuracy of the training data improves as the parameter value decreases but the accuracy for testing data remains constant from 0 downwards, so we can assume that the model is overfitting.

And for values above 0, the model is underfitting for both training and testing data.

If we increase the value of training data points to 10k, then we observe that the accuracy for weight>0 has approximately 10% accuracy for both training and testing data, but for weight<0 the accuracy for training data accuracy is better that 5k, indicating that model performs better for 10k data points. But the issue of overfitting persists with decreasing value of alpha.

Part c) i) MaxPooling is another way to downsample the input. To use MaxPooling, replace the strided convolution layer with Max Pooling. To reduce the size to half, use the pool_size =(2,2).

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

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

ii) Pooling is a way to downsample the input size, while also keeping important parameters. Max pooling takes the max value of the submatrix.

For 5k training dataset -

Total no of Parameters - 25578

This layer doesnot add any parameter to the computation thus the no of parameters is less than Strided Convolution Layer.

Accuracy on training data – 58%

Accuracy on testing data – 50%

Time taken – 81 s

The time taken is a little less than strided since it has less parameters to compute on. It helps in decreasing the computation of the model. The prediction accuracy for test data is same as that of strided. But the accuracy on training dataset is less than strided. MaxPooling allows only important features thus reducing overfitting in the model, as seen by the accuracy score.

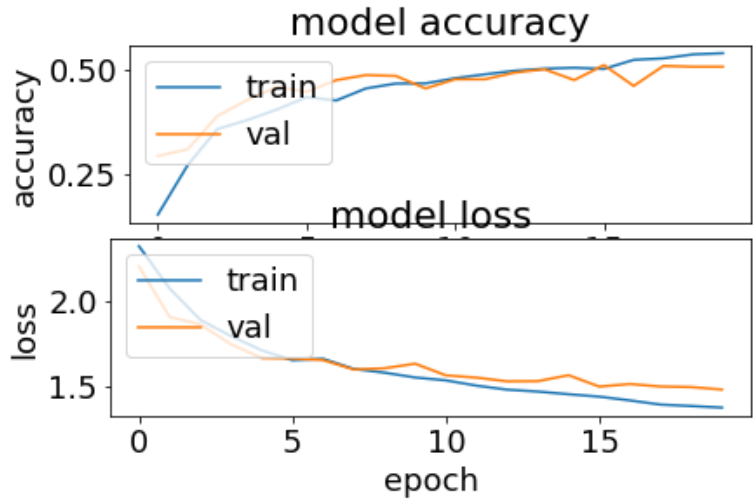

Fig – Model accuracy and loss for max pooling

Ritika Sharma 21334470
MSc Computer Science – Intelligent Systems sharmari@tcd.ie

Appendix

Part i)

```
########## part i) ##########
def convol2d(kernel, inp_array, stride):
  k = kernel.shape
  n = inp_array.shape
  output = np.empty(shape=((n[0]-k[0])//stride + 1,(n[1]-k[1])//stride + 1))
  #output = np.empty(shape=(n[0]-stride-1,n[1]-stride-1))
  for i in range(output.shape[0]):
    for j in range(output.shape[1]):
      submatrix = inp_array[i:i+k[0],j:j+k[1]]
      output[i][j] = np.sum(np.multiply(submatrix, kernel))

  return output

##########  i) a) ##########

n = int(input('Enter value of n'))
stride = int(input('Enter value for stride'))

import numpy as np
from numpy import random
inp_array = random.randint(20, size=(n,n))
# inp_array = np.array([[1, 2, 3, 4, 5],
#                       [1, 3, 2, 3, 10],
#                       [3, 2, 1, 4, 5],
#                       [6, 1, 1, 2, 2],
#                       [3, 2, 1, 5, 4]])
kernel = np.array([[1,0,-1],[1,0,-1],[1,0,-1]])

print("Input matrix ", inp_array)
print("Kernel ",kernel)
output = convol2d(kernel, inp_array, stride)
print("Result of convolution ", output)

##########  i) b) ##########
import numpy as np
from PIL import Image
im = Image.open('download1.png')
rgb = np.array(im.convert('RGB'))
r=rgb [ : , : , 0 ] # a r r a y o f R p i x e l s

img = Image.fromarray(np.uint8(r))
img.save('input.png')
kernel1 = np.array([[-1, -1, -1],[-1, 8, -1], [-1, -1, -1]])
kernel2 = np.array([[0, -1, 0],[-1, 8, -1],[0, -1, 0]])
```

```
output1 = convol2d(kernel1, r, 1)
print("Output with Kernel 1", output1)
img = Image.fromarray(np.uint8(output1))
img.save('output1.png')
img.show()

output2 = convol2d(kernel2, r, 1)
print("Output with Kernel 2", output2)

img = Image.fromarray(np.uint8(output2))
img.save('output2.png')
img.show()
```

Part ii)

```
# part ii)
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
from sklearn.dummy import DummyClassifier
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()
# change for different data points
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)

c = 0.0001
# c = [0,0.1,0.001,0.0001,100,10]
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(c)))
    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))


############ baseline classifier #################
baseline = DummyClassifier(strategy = "most_frequent")
baseline.fit(x_train, y_train)
preds = baseline.predict(x_test)
ypred = np.argmax(preds, axis=1)
ytest = np.argmax(y_test, axis=1)
print(classification_report(ytest, ypred))
```

Part iii)

```
########## iii) a) ##########
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
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(MaxPooling2D(pool_size=(2,2)))
    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    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
    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))
```