Machine Learning Week 8 Assignment

1. **Convolution function**

   (a) Method convolve() from appendix A is the convolution function I wrote. It takes an array, a kernel, a stride, and the amount of padding on each side, and returns the convovled result. The default stride is 1 and the default amount of padding is 0, but these can be changed. The technique used is the one we were showed in class - I am going though all cells $x \in \overline{1,m} * \overline{1,m}$, and setting each of them as $x_{i,j} = \sum_{i_K \in \overline{1,K}} \sum_{j_K \in \overline{1,K}} array_{s*i+i_K, s*j+j_K}$ where $m = \lfloor \frac{n-k+s+2*p}{s} \rfloor$ is the output size, n is the size of the original array, p is the amount of padding on each side, k is the size of the kernel, and s is the stride. Figure 1 shows the method solving the example given in class.

1	2	3	4	5								
1	3	2	3	10		1	0	-1		-1	-4	-14
3	2	1	4	5	*	1	0	-1	=	6	-3	-13
6	1	1	2	2		1	0	-1		9	-6	-8
3	2	1	5	4								

Figure 1: Convolution example.

   (b) Figure 2 is the image heart.jpg that I am using.

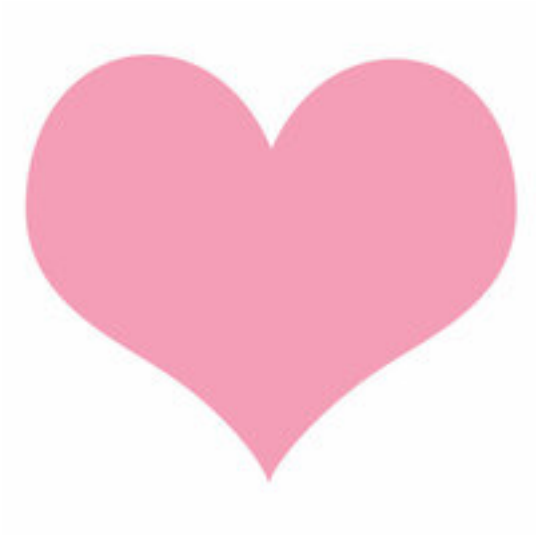


Figure 2: Image of a pink heart

   Tables 1 and 2 show the result of convolving the image with kernels 1 and 2 respectively. I am using a stride of 10 so that the resulting matrix can be legible.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	45	4	-42	-122	0	0	0	0	0	-12	15	0	5	0	0	0
0	0	7	-12	0	0	0	-3	36	16	0	21	10	-5	0	0	28	89	8	0
0	8	2	0	0	0	0	0	0	23	-2	3	0	0	0	0	0	0	-29	5
0	-56	0	0	0	0	0	0	0	0	3	14	0	0	0	0	0	0	0	-4
0	-78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
0	-20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-35
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14
0	-54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41
0	26	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18
0	0	44	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-5	-42	-29
0	0	0	6	29	0	0	0	0	0	0	0	0	0	0	2	8	17	0	0
0	0	0	0	10	36	0	0	0	0	0	0	0	0	0	-18	0	0	0	0
0	0	0	0	0	0	-8	0	0	0	0	0	0	12	16	-2	0	0	0	0
0	0	0	0	0	0	0	-13	2	0	0	11	34	-20	0	0	0	0	0	0
0	0	0	0	0	0	0	0	23	14	0	46	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	15	-29	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	38	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1. *Convolution of heart.jpg and kernel1 using stride = 10*

1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016
1016	1016	1016	1015	1021	982	940	889	1016	1016	1016	1016	1016	1003	1003	993	1018	1016	1016	1016
1016	1016	1015	965	976	976	976	984	1002	1024	1016	1021	981	965	976	976	1003	1058	1016	1016
1016	1015	981	976	976	976	976	976	976	988	1005	971	976	976	976	976	976	976	951	1023
1016	954	976	976	976	976	976	976	976	976	1003	986	976	976	976	976	976	976	976	1014
1016	919	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	1036
1016	966	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	951
1016	982	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	990
1016	952	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	1035
1016	1031	989	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	976	1030
1016	1016	1047	971	976	976	976	976	976	976	976	976	976	976	976	976	976	975	952	985
1016	1016	1016	1015	1003	976	976	976	976	976	976	976	976	976	976	976	973	986	1027	1016	1016
1016	1016	1016	1016	1023	1022	976	976	976	976	976	976	976	976	976	961	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	989	975	976	976	976	976	976	988	1026	1015	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	997	979	976	976	976	1001	995	1016	1016	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	1016	1028	986	976	1018	1016	1016	1016	1016	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	1016	1016	1022	949	1025	1016	1016	1016	1016	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1035	1016	1016	1016	1016	1016	1016	1016	1016	1016
1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016

Table 2. *Convolution of heart.jpg and kernel2 using stride = 10*

Figures 3 and 4 show the matrices resulting from using stride 1 presented as heatmaps. The first kernel captures the outline of the image, whereas the second one preserves the colour fill as well, resulting in a sharpened version of the original.

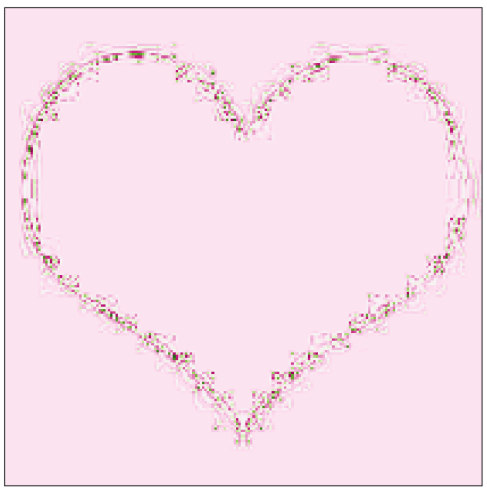


Figure 3: Convolution of heart.jpg and kernel1

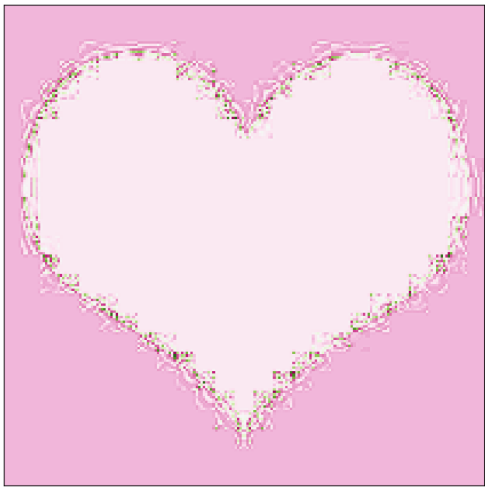


Figure 4: Convolution of heart.jpg and kernel2

2. **Convolutional network**

(a) Figure 5 shows the layers of the model and their features. The first layer is a convolutional layer with a kernel size of 3x3. There are 16 output channels. The input shape matches the image shape. The activation function is ReLU (Rectified Linear Unit), which is very common. The second layer is another convolutional layer with 3x3 kernel size, 16 output channels, 2 stride, and ReLU activation. The innput shape automatically matches the previous layer output. The third layer is another convolutional layer with 3x3 kernel size, 32 output channels and ReLU activation. The fourth layer is another convolutional layer with 3x3 kernel size, 32 output channels, and ReLU activation. All these layers use padding which makes the output have the same size as the original input. Dropout regularisation is then added. Then the image is flattened, then the fully connected Dense layer is applied. The Dense layer has softmax activation and uses L_1 regularisation with weight 0.001.

```
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)))
```

Figure 5: Model architecture

(b) i. Figure 6 shows the keras model summary shown when running the code. There are 37146 total parameters. The dense layer has the most - 20490. This is because it is a fully connected layer, which tends to be very expensive. The accuracy is almost 50% which is much better than a dummy classifier always predicting the mode, which achieves an accuracy of 10%.

```
_________________________________________________________________
 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
_________________________________________________________________
```

Figure 6: Model summary

ii. Figure 7 shows the plot of training vs testing accuracy during training. It can be seen how both training and testing accuracy increase together up to a point, after which training accuracy continues to increase and surpasses testing accuracy, which tapers. This can mean that the model is beginning to overfit, as it gets better at predicting training data, but this does not have a positive impact on its test performance.

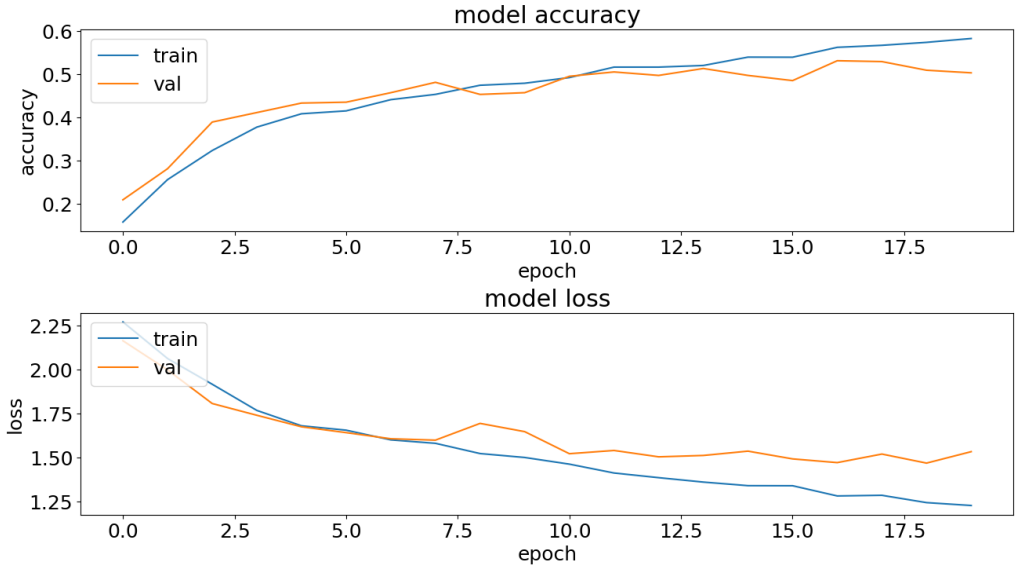


Figure 7: History plot with 5000 points

iii. Figure 7 shows the 5000 points plot. The time is recorded at the bottom left, and is around 10 sec. Figures 8,9, and 10 show the plots for the 10K, 20K, and 40K points history. All of them have the number of points and the time taken to train network listed in the bottom left corners. These times increase from 12 sec to 19 sec to 31 sec respectively. As the number of points increases, the gap between training and testing accuracy closes. This is because when there are more points in the training set, the model can be trained for more epochs to discover the patterns in the data, improving accuracy, whereas with a small dataset, training for too many epochs won't uncover new, valuable information. Instead, the model will just pick up the errors in the small training set.

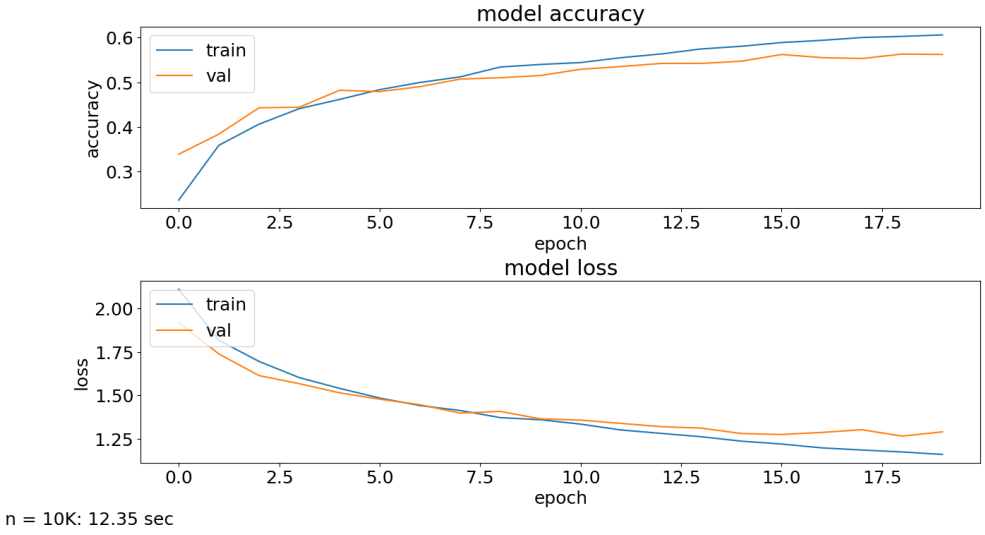


Figure 8: History plot with 10000 points

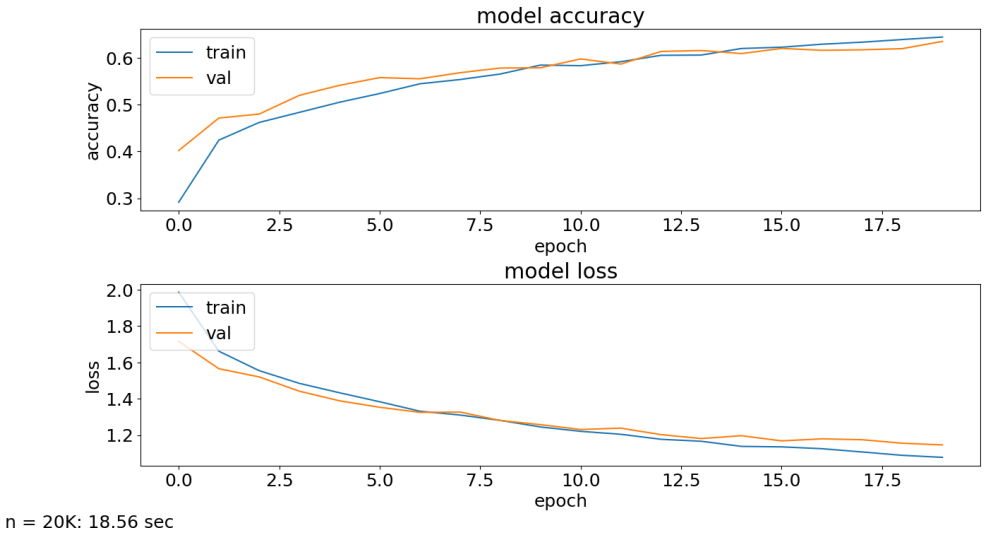


Figure 9: History plot with 20000 points

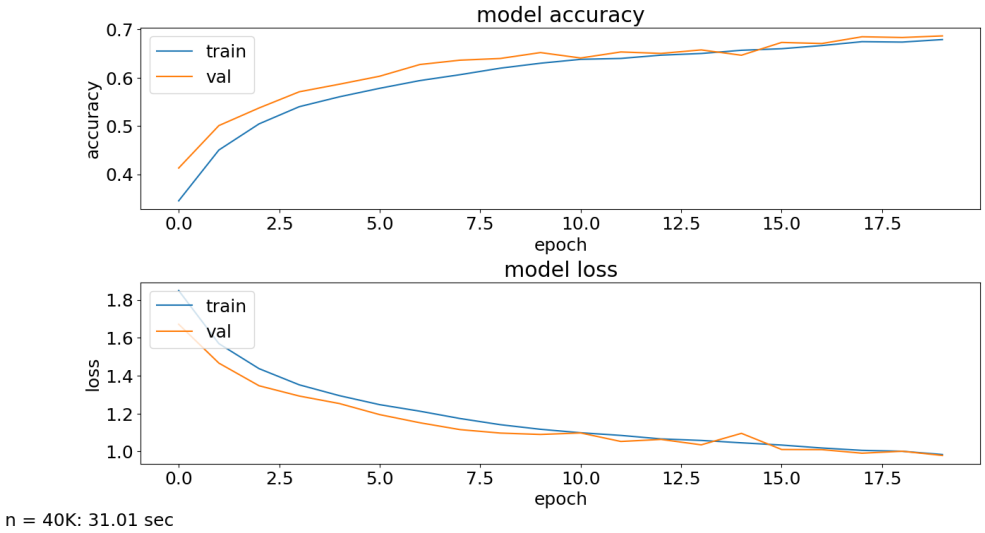


Figure 10: History plot with 40000 points

iv. Figures 11, 12, 13, 14, and 15 show the history plot when setting the L1 regularisation weights equal to 0, 0.0001, 0.001, and 0.01 respectively. The weight and the training time is recorded in the bottom left corner. Setting a higher weight does indeed lead to less overfitting. Moreover, setting a higher weight allows the training to be done is the even less time than a lower non-zero weight. In this sense, modifying the regularisation weight can be considered an effective way of avoiding overfitting. That being said, the performance of these models decreases when the penalty weight is larger. This is in contrast to the strategy explored at (2)(b)(iii) of increasing the number of points, which in spite of increasing running times, also increases testing accuracy. Thus, when addressing overfitting, there is a tradeoff between time and accuracy. In this case, even with 40K points, the model runs in under a minute. So a 40% increase in accuracy is worth a 20 second increase in training time. However, this might not be the case for different models, different datasets, or even different computers.

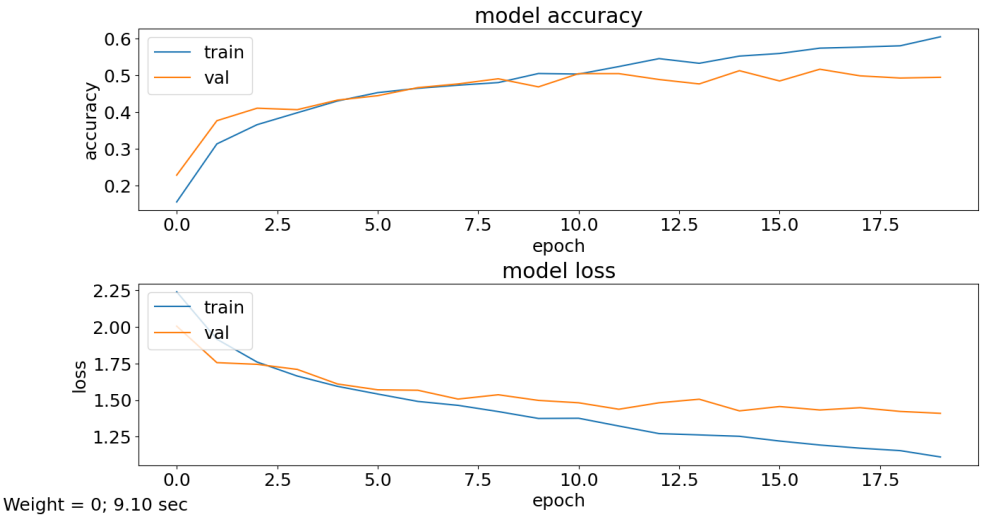


Figure 11: History plot with L1 regularisation weight = 0

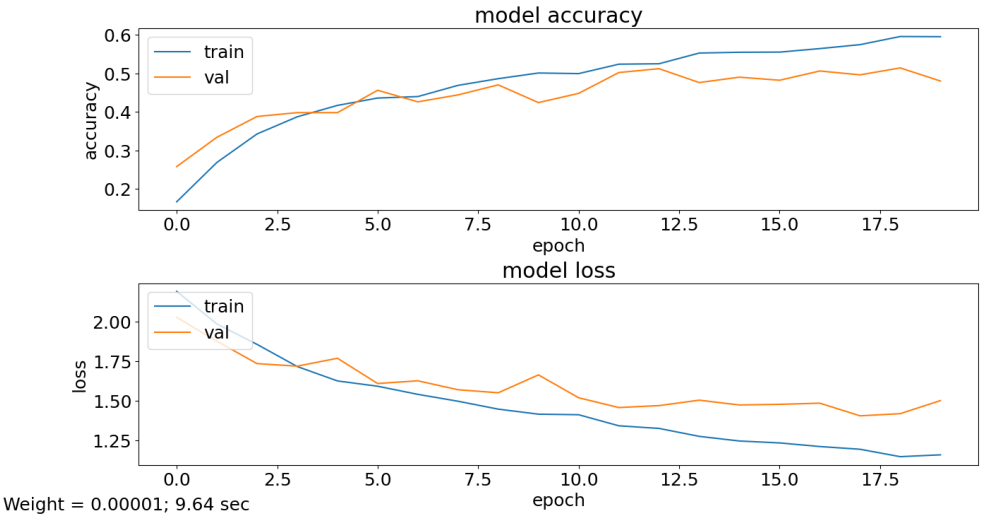


Figure 12: History plot with L1 regularisation weight = 0.00001

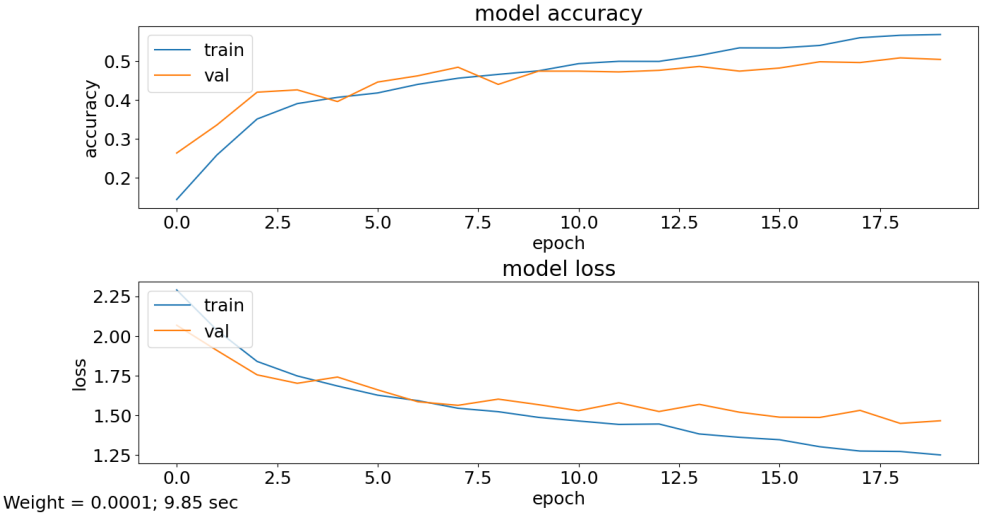


Figure 13: History plot with L1 regularisation weight = 0.0001 (default)

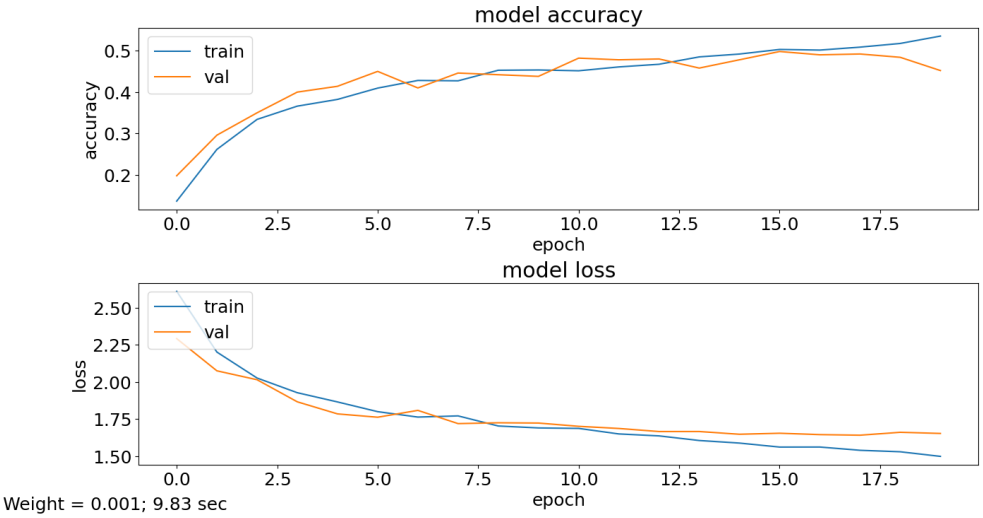


Figure 14: History plot with L1 regularisation weight = 0.001

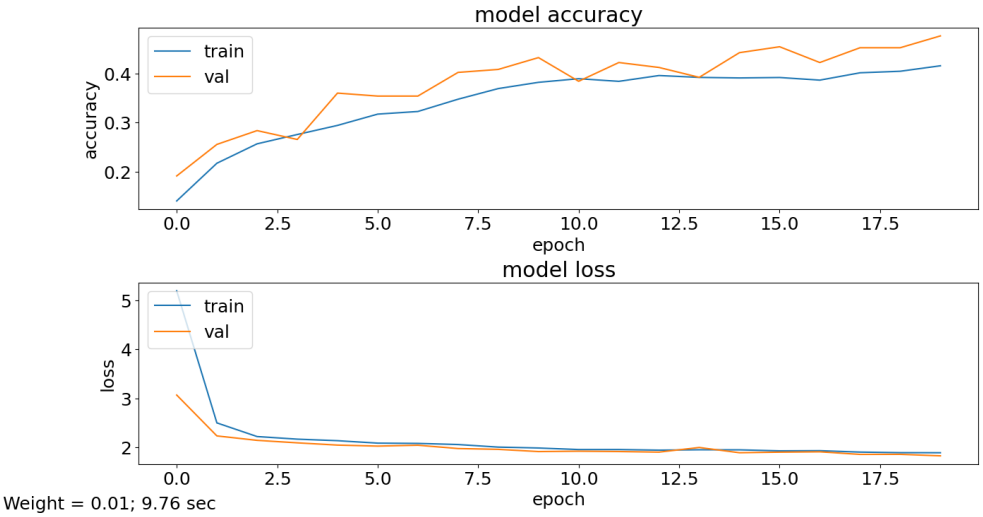


Figure 15: History plot with L1 regularisation weight = 0.01

(c) i. Figure 16 shows the model architecture with max-pooling instead of striding.

```
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(pool_size=(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(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax', kernel_regularizer=regularizers.l1(weight)))
```

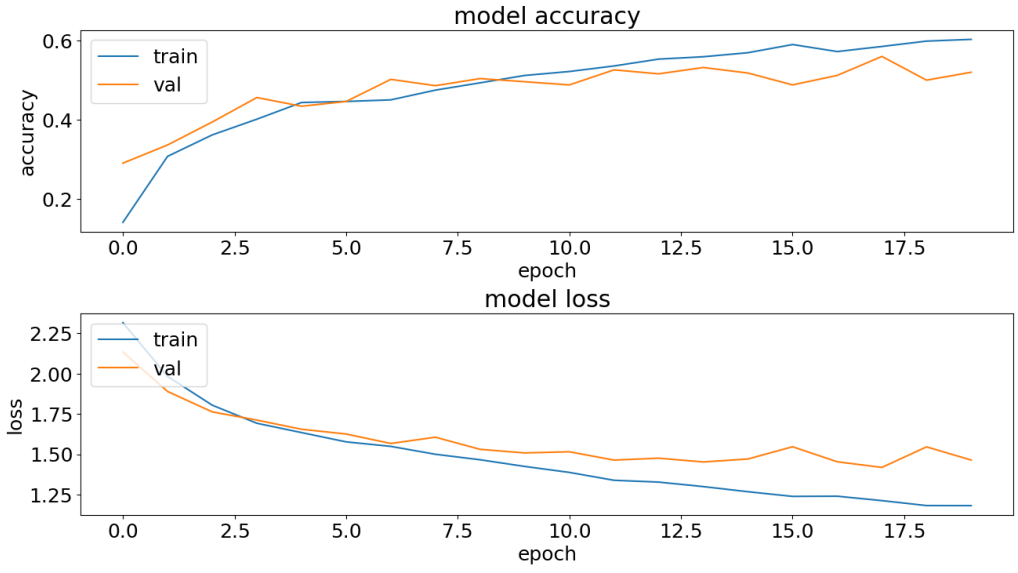
Figure 16: Model architecture with max-pooling

ii. Figure 17 shows the model summary. The number of parameters is the same as that of the striding model.

```
_________________________________________________________________
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
```

Figure 17: Model summary with max-pooling

Figure 18 shows the history plot, with the training time marked in the lower left corner. In this case, maxpooling does not improve the model performance, but leads to increased training times. The increased training times are because the max-pooling model needs to calculate the additional intermediate convolutions compared to the striding model.



11.12 sec

Figure 18: History plot with 5K points and max-pooling

(d) Figure 19 shows the results of training the model with the new set of layers on 5K points and all default settings. The training time is longer by almost 2 seconds (almost 20%) compared to the original model with the same settings. The accuracy is not improved.

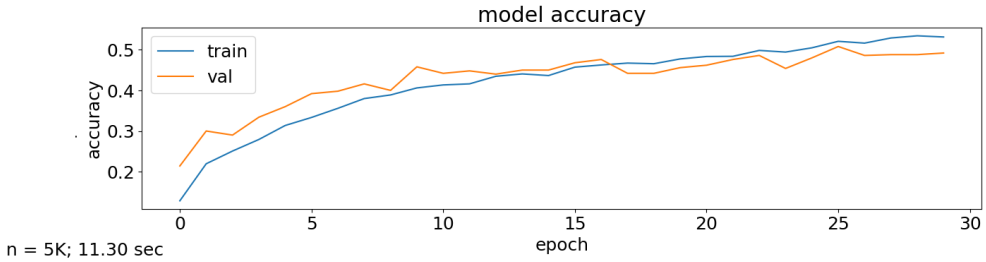


Figure 19: New layers

Figure 20 shows the results of training the model with the new set of layers on the full dataset with all other settings unchanged. The training time is almost 5 times longer than the previous setting with 5K points. The accuracy is significantly improved, achieving around 65%. It should be noted that the model with the original layers achieved a higher accuracy, close to 70%, by training on only 40K points, which took only 30 seconds.

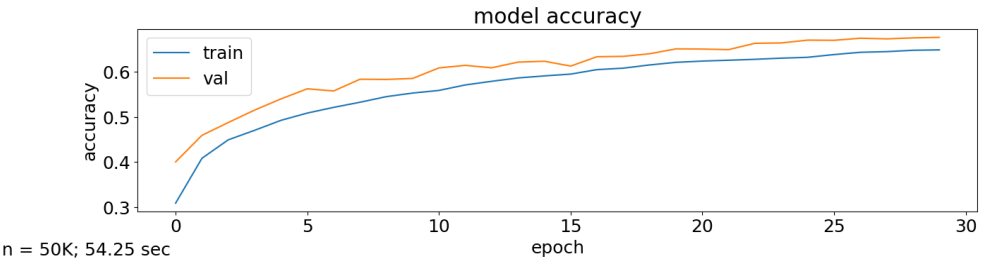


Figure 20: New layers and full dataset

Figure 21 shows the results of training on the full 50K dataset for 60 epochs. An accuracy of over 70% is achieved in 98 seconds. The increase in accuracy is very small compared to training on 40K points for 30 epochs, but the training takes three times as long.

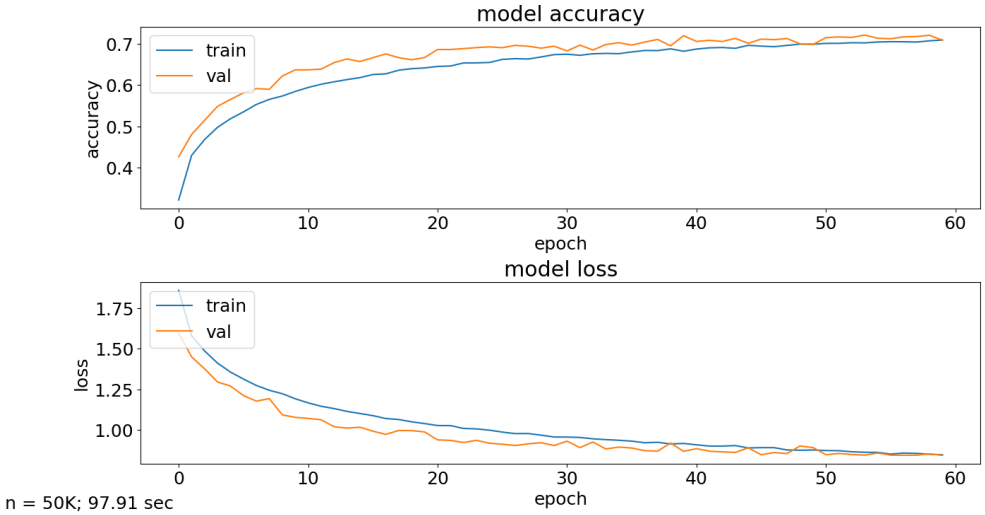


Figure 21: New layers and full dataset with 80 epochs

Appendix A - Convolution function

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt


def convolve(array, kernel, s, p):
    # kernel, input, and output sizes
    k = kernel.__len__()
    n = array.__len__()
    m = (n - k + s + 2 * p) // s

    # padding with zeroes
    if p > 0:
        pad = []
        for i in range(n):
            pad.insert(i, 0)
        for i in range(0, p):
            array.insert(i, pad)
        for i in range(n, n - p):
            array.insert(i, pad)
        for i in range(n + p):
            for j in range(0, p):
                array[i].insert(j, 0)
            for j in range(n, n - p):
                array[i].insert(j, 0)

    # adding elements to the output matrix using the formula
    output = []
    for i in range(m):
        row = []
        for j in range(m):
            x = 0
            for iK in range(k):
                for jK in range(k):
                    x = x + array[s * i + iK][s * j + jK] * kernel[iK][jK]
            row.insert(j, x)
        output.insert(i, row)
    return output


def printMatrix(conv):
    matrix = '\n'.join(['' .join(['{:8}'.format(item) for item in row]) for row in conv])
    print(matrix)


# this method only works with input, kernel ,output sets of size 5, 3, 3
def prettyPrint(input, kernel, output):
    pad = [' ', ' ', ' ']
    kernel.insert(3, pad)
    output.insert(3, pad)
    kernel.insert(0, pad)
    output.insert(0, pad)

    times = [' ', ' ', '*', ' ', ' ']
    equal = [' ', ' ', '=', ' ', ' ']
    all = []
```

```
    for i in range(5):
        row = list(map(str, input[i]))
        row.extend(list(map(str, times[i])))
        row.extend(list(map(str, kernel[i])))
        row.extend(list(map(str, equal[i])))
        row.extend(list(map(str, output[i])))
        all.insert(i, row)
    printMatrix(all)


def heatmap(conv):
    fig = plt.imshow(conv, cmap='PiYG_r', interpolation='nearest')
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    plt.show()


# testing the convolve function
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 = [[1, 0, -1],
          [1, 0, -1],
          [1, 0, -1]]
stride = 1
padding = 0
conv = convolve(array, kernel, stride, padding)

prettyPrint(array, kernel, conv)
# printMatrix(convolve(array, kernel, stride, padding))

# image convolution
im = Image.open('heart.jpg')
rgb = np.array(im.convert('RGB'))
r = rgb[:, :, 0]
# Image.fromarray(np.uint8(r)).show()

kernel1 = [[-1, -1, -1],
           [-1, 8, -1],
           [-1, -1, -1]]
kernel2 = [[0, -1, 0],
           [-1, 8, -1],
           [0, -1, 0]]

# choose the desired kernel and stride
conv = convolve(r, kernel2, 1, 0)

# show the convolution heatmap
# heatmap(conv)

# show the convolution matrix
# printMatrix(conv)
```

Appendix B - Edited image classifier

```
import numpy as np
import statistics as statistics
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
import time
from sklearn.dummy import DummyClassifier

tic = time.perf_counter()
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
weight = 0.0001
x_train = x_train[1:n]
y_train = y_train[1:n]
# x_test=x_test[1:500]; y_test=y_test[1:500]

with tf.device('/device:GPU:1'):
    # 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(weight)))
        model.compile(loss="categorical_crossentropy", optimizer='adam',
```

```
                metrics=["accuracy"])
    model.summary()'''
    '''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(pool_size=(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(pool_size=(2, 2)))
     model.add(Dropout(0.5))
     model.add(Flatten())
     model.add(Dense(num_classes, activation='softmax',
                kernel_regularizer=regularizers.l1(weight)))
     model.compile(loss="categorical_crossentropy", optimizer='adam',
                metrics=["accuracy"])
     model.summary()'''
    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(weight)))
    model.compile(loss="categorical_crossentropy", optimizer='adam',
                metrics=["accuracy"])
    model.summary()
batch_size = 128
epochs = 60
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
        validation_split=0.1)
model.save("cifar.model")
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss: % f accuracy: % f" % (score[0], score[1]))
score = model.evaluate(x_train, y_train, verbose=0)
print("Train loss: % f accuracy: % f" % (score[0], score[1]))
fig = plt.figure()
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')
    toc = time.perf_counter()
    text = 'n = ' + str(n // 1000) + 'K; ' + '%.2f' % (toc - tic) + ' sec'
    # text = 'Weight = ' + str(weight) + '; ' + '%.2f' % (toc - tic) + ' sec'
    # text = '%.2f' % (toc - tic) + ' sec'
    fig.text(0, 0.03, text)
    plt.show()
    dummy = DummyClassifier(strategy='most_frequent')
    dummy.fit(x_train, y_train)

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))

preds = dummy.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))
```