Brendan Jobus – 18321740

(i) (a) As the assignment specifies that we must use vanilla python, I implemented the feature without numpy, I also created a printConvolved function so that it would be easier to read the output. I also assumed that the stride would always be 1, so as to simplify the question since it was never specified.

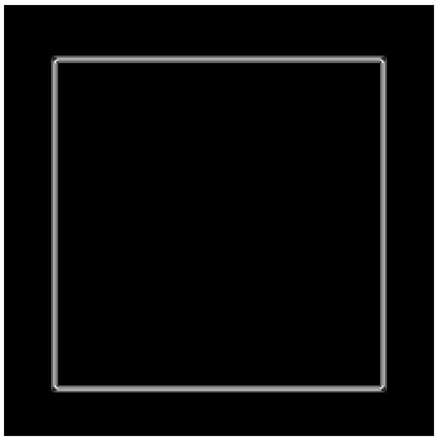
To convolve a matrix, I simply loop through all the points that could be multiplied by the top left corner element in the kernel, I then get the local matrix, a sub matrix that contains the points that will be multiplied by the kernel for this iteration, and since this is done in vanilla python I have to multiply the matrix out, then flatten it into a 1d list, then sum the values inside to get an element in the convolved array.

I also create a print function to print out the convolved matrix with walls to make things clearer, the print function will simply add walls to the ends of each row, and will add the correct number of spaces for each each input so that they will all sit at a uniform starting place for the first number, the largest number is considered to be in the thousands, and a space before any number if it is not negative.

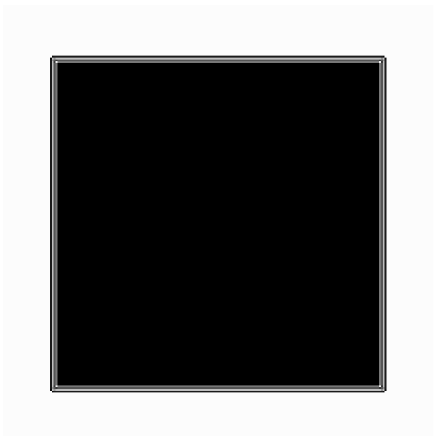
(b) The image I used for this was a black square that was placed on a 240x240 white page, the square does not take up the whole page, I chose this size because its large enough to inspect the results relatively easily, while also being small enough that it doesn't take too much time to complete.

Original Shape

First Kernel



Second Kernel



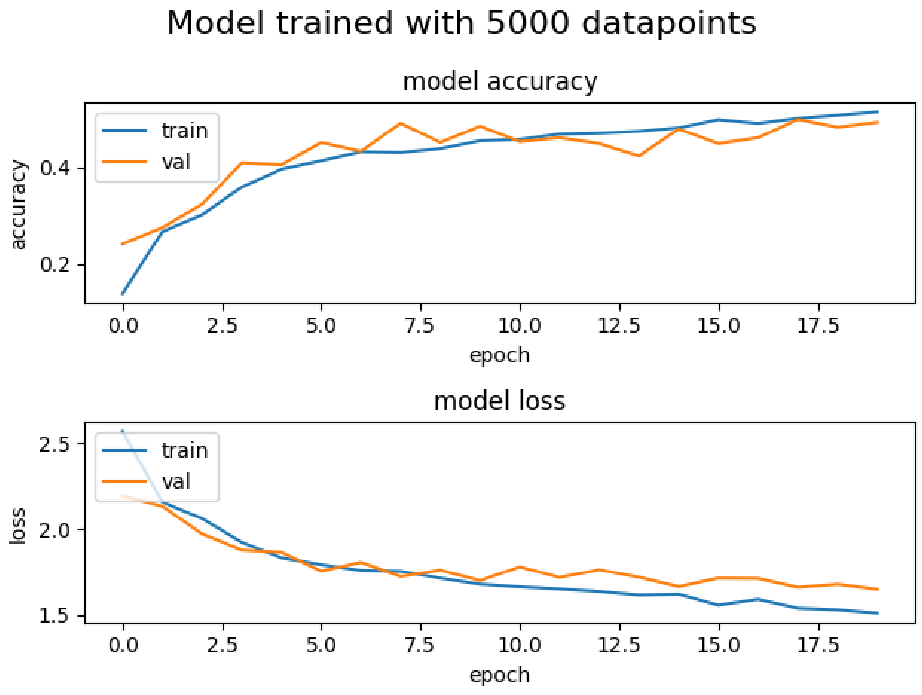
From the outputs, it looks like the first kernel holds some properties related to edge detection, and the second only affects the edges of the shape, perhaps it is doing something like changing the sharpeness of the image.

(ii) (a) The code here creates a convolutional neural network where each hidden layer uses relu for the activation function, 3x3 kernels and same padding. The inputs are 32 x 32 rgb images, so our inputs are 3 channel 32 x 32 matrices. The first layer makes 16 output channels, the second layer has 16 output channels and uses a stride of 2 to halve our output matrices size, the third layer outputs 32 channels, and the 4th hidden layer outputs 32 channels and uses a stride of 2 again to reduce the size of the output once more. We use a dropout regularizer of 50% to reduce over fitting and then flatten the outputed matrices to a vector so that they can be put into the final layer, a soft max dense layer with an l1 regularizer where c = 0.0001.

It uses batch sizes of 128 and will train the model over 20 epochs.

(b) (i) The model has 37,146 parameters, and the layer with the most parameters is the final layer implementing the softmax algorithm with 20,490 parameters, this is because this is a fully connected layer which connects all of the parameters from the final convolutional layer to a number of neurons, each of which is checking the likelihood that our input corresponds to some class, this means that our 2048 parameters from our last convoutional layer(8 x 8 x 32) will be connected to 10 different output nodes each, giving us 20480 parameters, and each output node also has a bias, giving us 20,490 parameters. With a baseline model that only predicts the most frequent result, which for me was the 8th class, would give us an accuracy of 16.66666% on the training data and 14.28571% on the test data.

(ii)



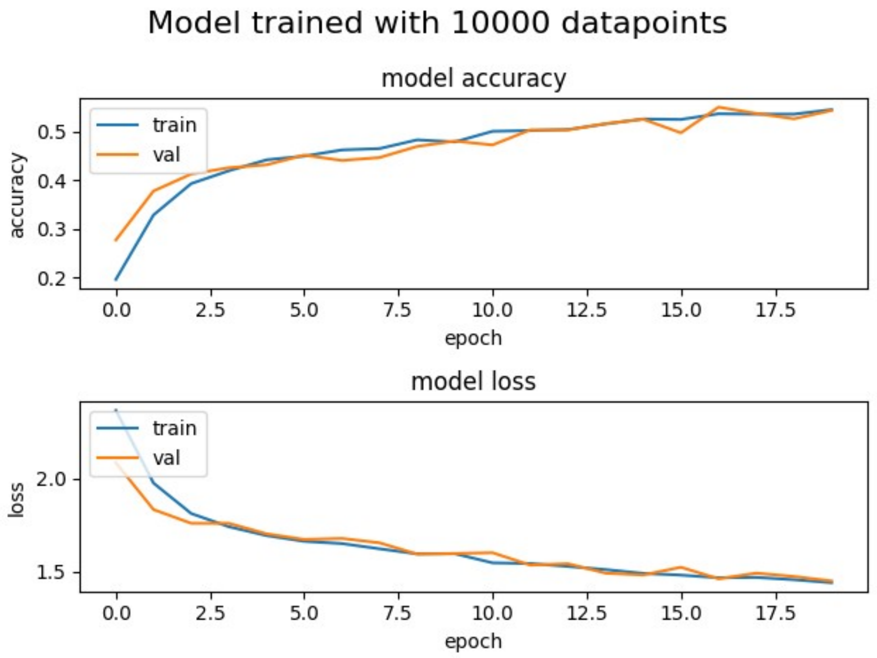
From the plots we can see that the model does not appear to be overfitting very much even towards the 20th epoch, though it does look like the loss' are starting to diverge, it is still not yet a problem, though if we were to continue training our model over more epochs we would want to keep an eye on it.

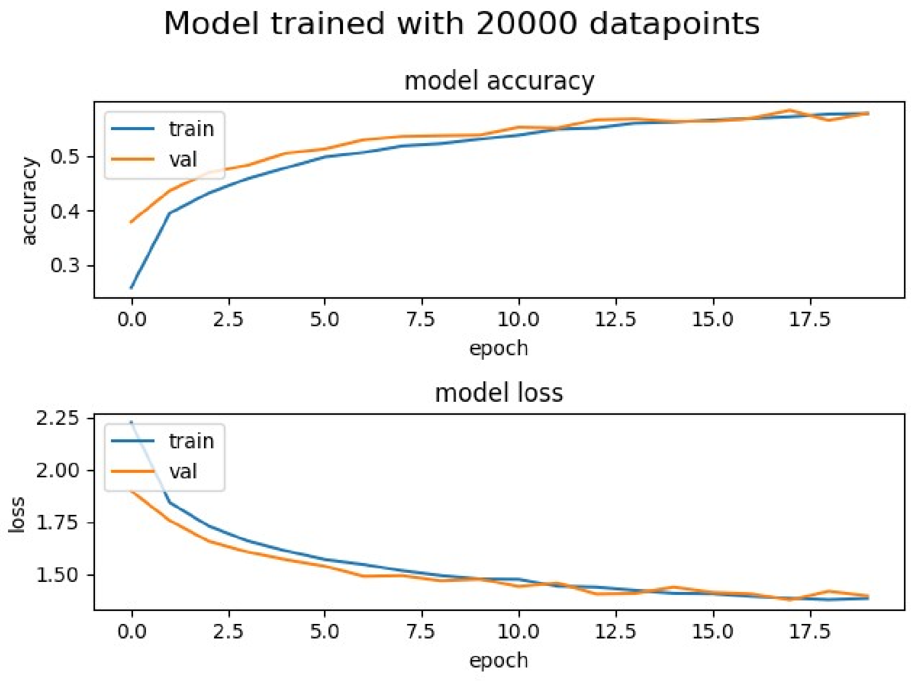
(iii) For the time taken to train each model I got the following data:

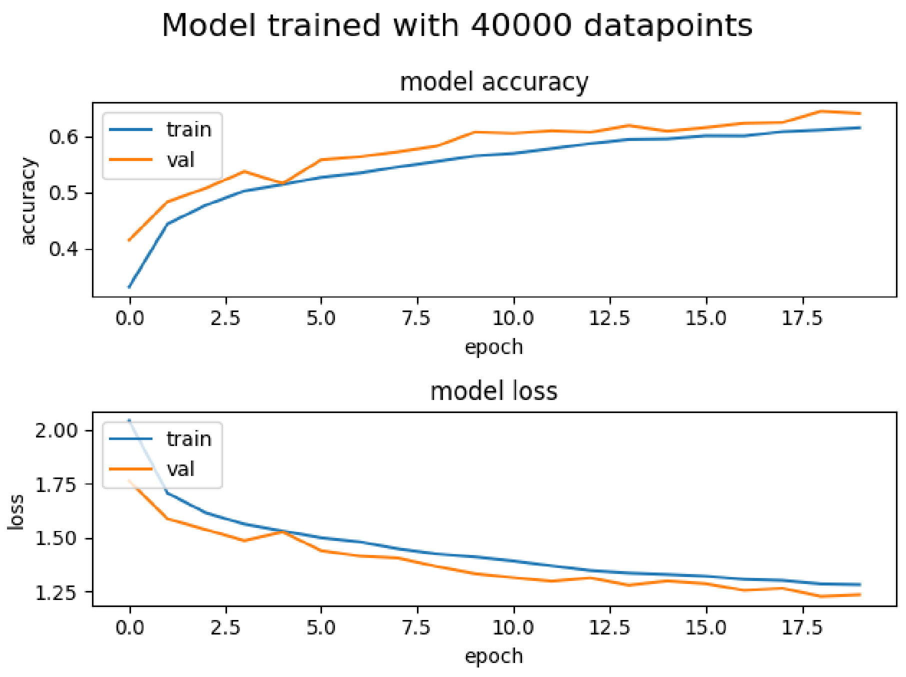
Dataset Size	Time Taken	Training Accuracy	Test Accuracy
5K	24 seconds	0.52	0.50
10K	60 seconds	0.58	0.58
20K	121 seconds	0.59	0.59
40K	224 seconds	0.59	0.62

Our accuracy increases each time we increase the dataset size, however, the accuracy gains we get are decreasing, meaning we can't just increase the dataset size to get an accuracy of 1. When it comes to overfitting, it appears that the 20K dataset is the perfect size for this problem as we have our model appears to generalize better than both the 5K and 40K dataset trained model, while having a better accuracy than the 10K dataset which also generalizes quite well. The rate at which the time to train the models increases appears to be about linear with the rate of increase in dataset size, that is, if we double the dataset size, we appear to require twice as long to train the model as well.

For all of the models I trained, the same trend occured, for the first 2 or 3 epochs, the model correctly predicts the test data better than the training data, but as we get through more and more epochs, it gets closer and closer, and in some cases they get so close together that they will swap as to which is more accurate. We also see that the majority of the increase in accuracy occurs in the first 5 or so epochs, and then the rate of increase of the accuracy significantly decreases, however, it is still rising steadily.





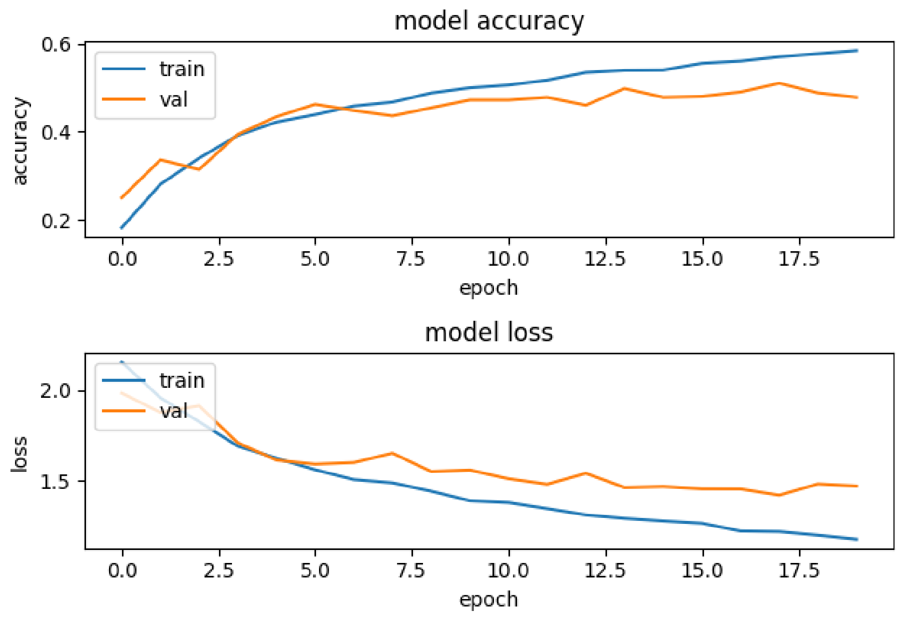


(iv) I got the following data for part 4:

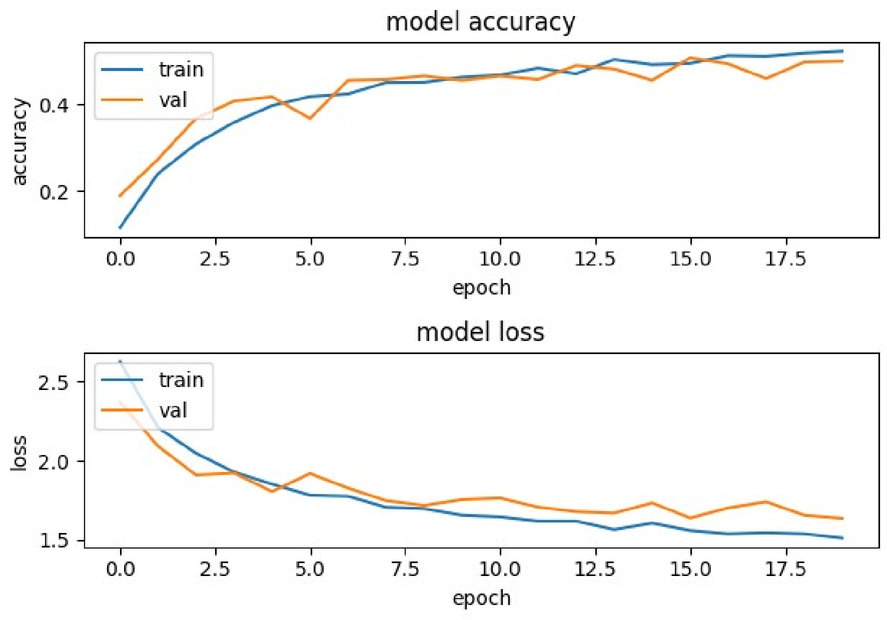
C	Training Accuracy	Test Accuracy
0	0.60	0.48
0.001	0.49	0.48
0.1	0.29	0.37
1	0.1	0.08
10	0.1	0.05
100	0.1	0.09

We can clearly see that by increasing the value of C we are significantly decreasing the accuracy of our model, as well as this, of the values with accuracies that were not horrible, only one of them dealt with overfitting well, because of this, I would say that both are essential for dealing with overfitting, however, I would say that the value of C is more important for dealing with overfitting as a bad value of C gives us results that overfit far worse than if we were to use a smaller dataset.

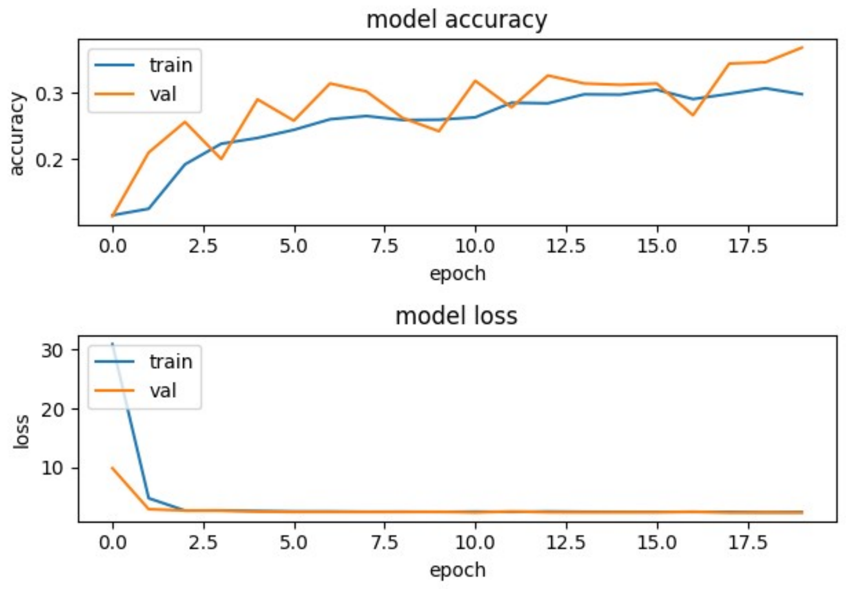
Model with c value of 0



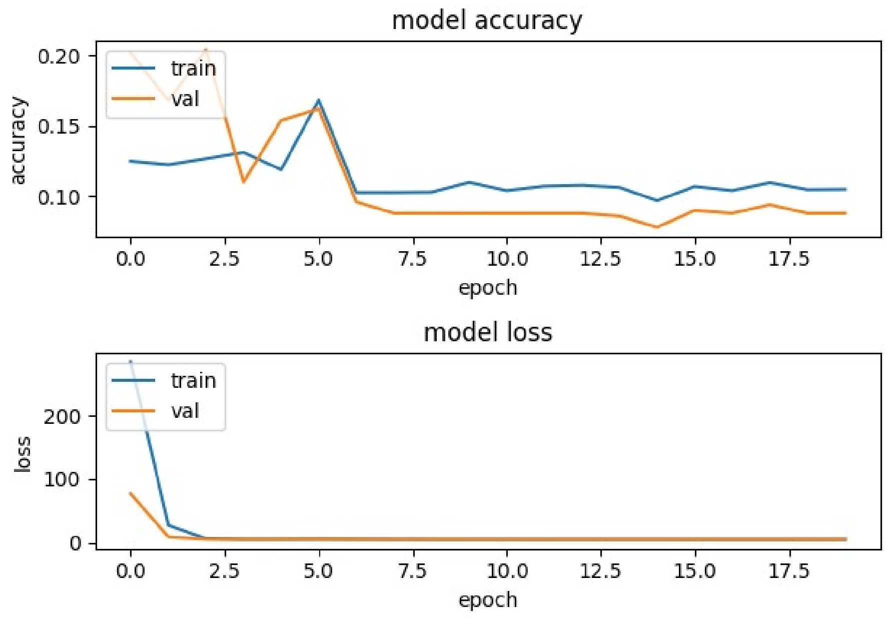
Model with c value of 0.001

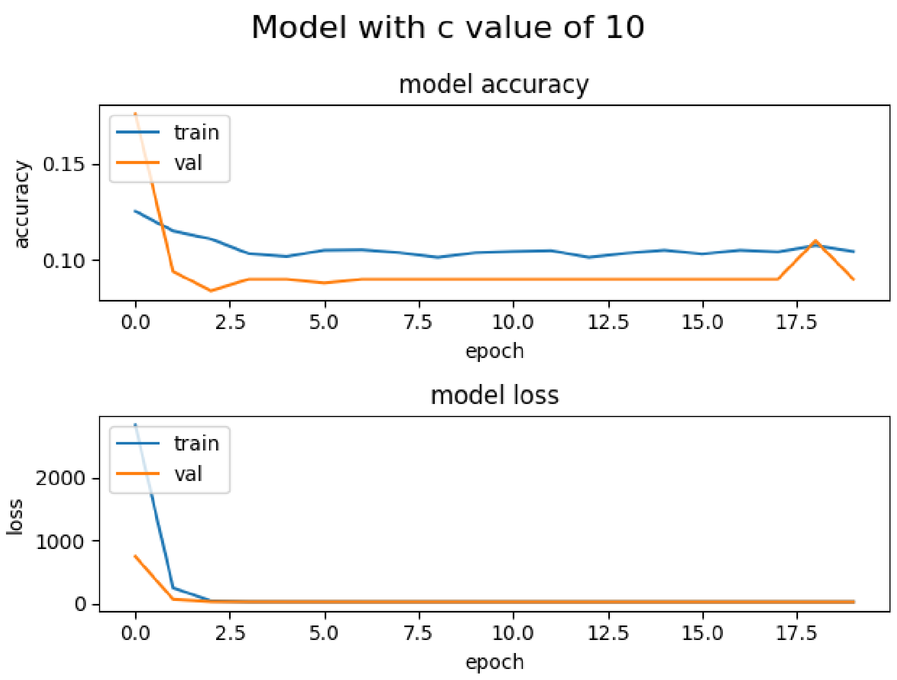


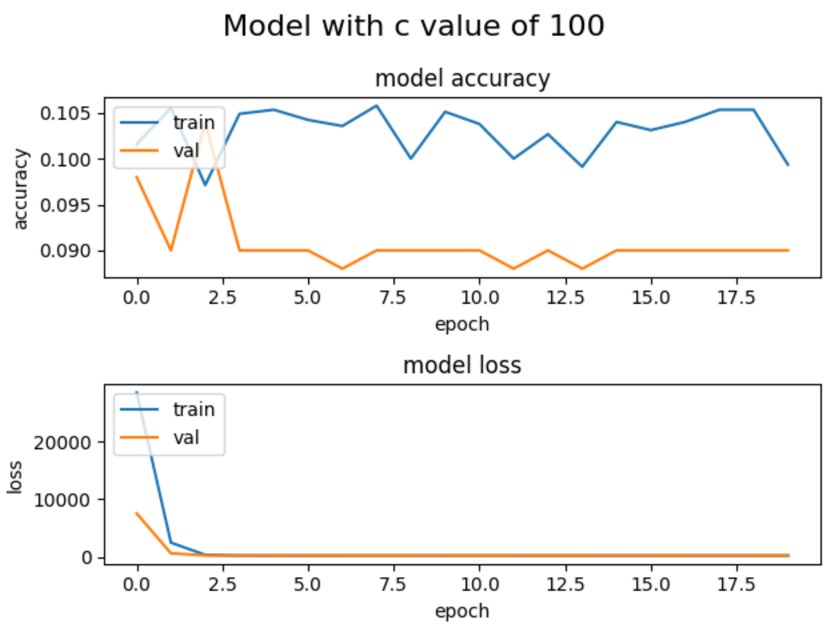
Model with c value of 0.1



Model with c value of 1







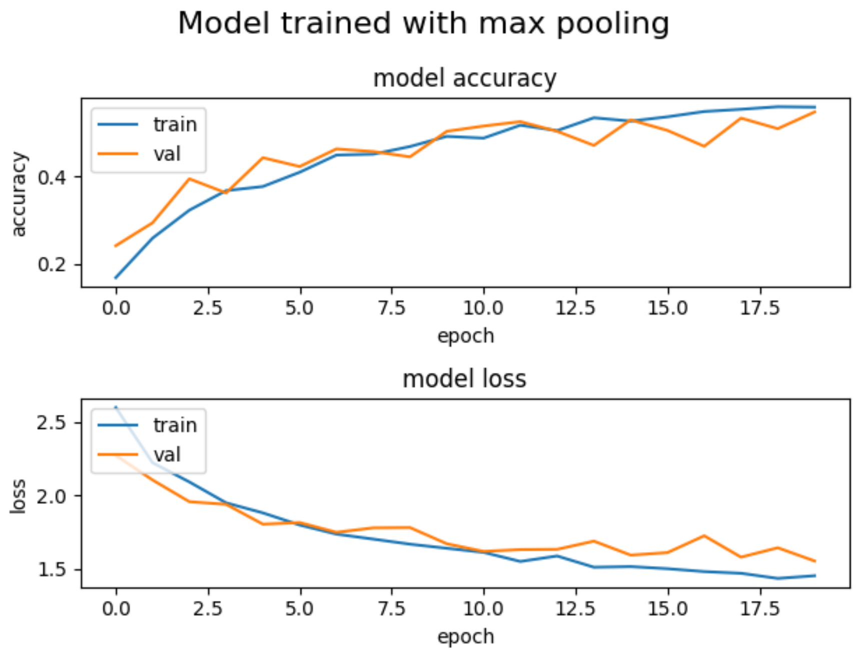
Compared to increasing the number of data points, changing the value of c is far less effective at managing overfitting, with larger values of c exasperating the problem and with

(c) (i) Not much needs to be changed in order to make our model use max pooling instead of strides all we have to do is replace the lines that implemented the striding with 2 lines, one is just a normal line identical to the one that was above the stride line, and also a seperate line that simply does max pooling:

```
model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
model.add(Conv2D(16, kernel_size=(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, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

We keep the rest of the lines that finished creating the model.

The results are as follows:



(ii) Keras says that this model has 37,146 parameters, compared to the original network, it took 63 seconds, almost 3 times longer to train, as well as this, our accuracies this time are both 50%, just 1 and 2 percent above what we got using striding, because of this combined with the fact that our model takes nearly three times longer to train, we would likely say that the architecture of this model is probably not the right one for this problem . The training time has increased because before, we would shrink the size of the matrix we were working on with the strides, and then convolve, whereas this time, we are convolving first and then reducing the size of the matrix with max pooling afterwords.

Appendix:

```
import matplotlib.pyplot as plt
import math
import numpy as np
import PIL.Image as Image
from sklearn import dummy
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow import keras
from tensorflow.keras import regularizers
from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D

def convolve(n, k):
outputLen = len(n) - len(k) + 1

convMatrix = []
localMatrixSizeModifier = len(k)
for row in range(outputLen):
convRow = []
for col in range(outputLen):
localMatrix = [i[col : col + localMatrixSizeModifier] for i in n[row : row +
localMatrixSizeModifier]]
multMatrices = [[x * y for x, y in zip(a, b)] for a, b in zip(localMatrix, k)]
flattenedMatrices = sum(multMatrices, [])
convRow.append(sum(flattenedMatrices))
convMatrix.append(convRow)
return convMatrix

def printConvolved(c):
for row in c:
toPrint = '|'
for elem in row:
if elem != 0:
newElem = ''
logged = math.log10(abs(elem))
if logged % 1 != 0:
spaces = 5 - (math.ceil(logged))
else:
spaces = 4 - (math.ceil(logged))
newElem = str(elem) + (' ' * spaces)
if elem > 0:
newElem = ' ' + newElem
toPrint += newElem
else:
toPrint += ' ' + str(elem) + (' ' * 4)
```

```
print(toPrint + '|')

def modelCreation(x_train, num_classes, c=0.001, pool=False, deeper=False):
model = keras.Sequential()
if not deeper:
if pool:
model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
model.add(Conv2D(16, kernel_size=(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, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
else:
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()
else:
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()
return model

def runModel(n=5000, c=0.001, pool=False, deeper=False, title=''):
# 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()
```

```
# takes the first 5000 datapoints in x_train and y_train
x_train = x_train[1:n]; y_train=y_train[1:n]
#x_test=x_test[1:500]; y_test=y_test[1:500]

# images are originaly in 0-255 values, regularize them to 0-1
# 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)

# data currently saved as a vector with numbers 0-9, instead, turn each y into a list of binary
numbers, with 9 elements in each list
# 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)

model = modelCreation(x_train, num_classes, c, pool, deeper)

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.suptitle(title, fontsize=16)
plt.tight_layout()
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))

def baseline():
num_classes = 10
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
# takes the first 5000 datapoints in x_train and y_train
x_train = x_train[1:5000]; y_train=y_train[1:5000]

# images are originaly in 0-255 values, regularize them to 0-1
# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

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

preds = model.predict(x_train)
y_train = sum(y_train.tolist(), [])
totalVals = 0
correctVals = 0
for v, i in enumerate(y_train):
if v == preds[i]:
correctVals += 1
totalVals = i
acc = correctVals / totalVals
print(acc)

preds = model.predict(x_test)
y_test = sum(y_test.tolist(), [])
totalVals = 0
correctVals = 0
for v, i in enumerate(y_test):
if v == preds[i]:
correctVals += 1
totalVals = i
acc = correctVals / totalVals
print(acc)

def part1():
# (a)
n = [[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]]
k = [[1, 0, -1],
[1, 0, -1],
[1, 0, -1]]
printConvolved(convolve(n, k))

# (b)
im = Image.open('shape.png')
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]]
con1 = convolve(r, kernel1)
Image.fromarray(np.uint8(con1)).show()
#printConvolved(con1)
print("\n\n")
con2 = convolve(r, kernel2)
Image.fromarray(np.uint8(con2)).show()
#printConvolved(con2)

def part2():
# (b) (i)
baseline()

# (b) (iii)
N = [5000, 10000, 20000, 40000]
for n in N:
runModel(n=n, title=f'Model trained with {n} datapoints')

# (b) (iv)
Ci = [0, 0.001, 0.1, 1, 10, 100]
for c in Ci:
runModel(c=c, title=f'Model with c value of {c}')

# (c) (i)
runModel(pool=True, title=f'Model trained with max pooling')

# (d)
runModel(deeper=True, n=40000, title=f'Thinner and deeper model')
```

part1()
part2()