Assignment 4 – Week 8

Pallavit Aggarwal – Ms IS
ID: 22333721

Q1)

a)

The definition of convolution varies from signal processing to basic dot product of the arrays depending on the context. The implementation here, gives an intuitive idea into how the convolution takes place, when a kernel is applied on a matrix of input array.

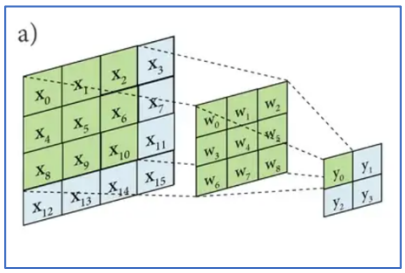


Fig: Showing convolution diagrammatically

I have considered the simple formula of

Output_Height = Image_Height – Kernel_Height + 1

and

Output_Width = Image_Width – Kernel_Width + 1

The formula with Stride and Padding considered is slightly modified, as each increment in the stride makes the input even smaller and the padding essentially keeps the information of the original image intact when convolving it with the kernel. The formula is given as:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

After going over the convolution using vanilla python, the above formula is intuitive and easy to build upon.

I have considered the same example as input arrays as provided in the covnet slides on blackboard. The output is given below:

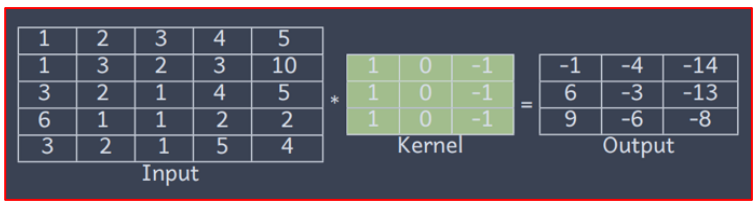


Fig: Image from Blackboard

```
M = 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]
])

K = np.array([
        [1, 0, -1],
        [1, 0, -1],
        [1, 0, -1]
    ])

print(convolution2d(M,K))
```

```
[[ -1.  -4. -14.]
 [  6.  -3. -13.]
 [  9.  -6.  -8.]]
```

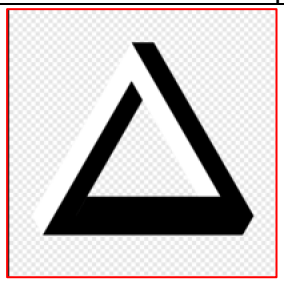
Fig: Image from python notebook

Note: Code for above is attached in the **Appendix**. I first used the brute force approach and then modified it slightly.

b)

For this part, I selected the following image and resized it online to 200x200 pixel ratio. It has all the components in it R,G and B and it helps visualize the output from the code a little better in my opinion.

The output on running convolution with R r=rgb[:,:,0] , G r=rgb[:,:,1] and B r=rgb[:,:,2] pixels gives the edge detection of those channel of pixels.

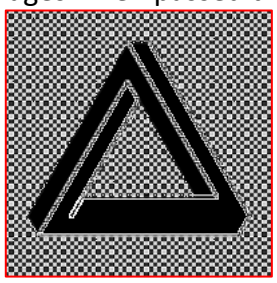


R channel is detected

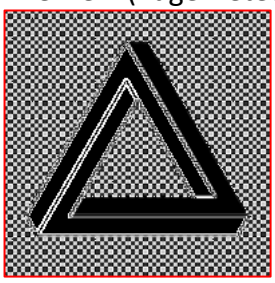
G channel is detected

B channel is detected

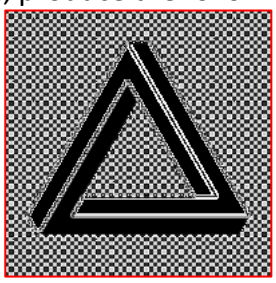
The above 3 images when passed through Kernel 1(Edge Detection), produce the following images



K1 on R Channel

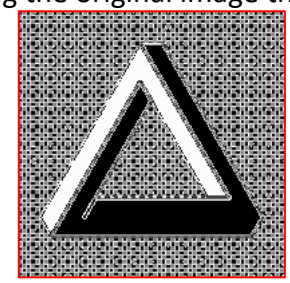


K1 on G Channel

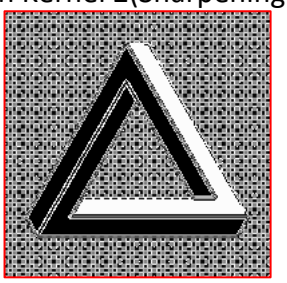


K1 on B Channel

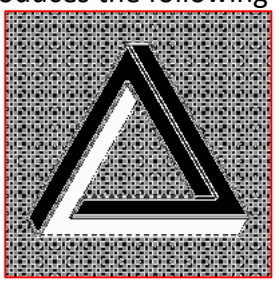
Finally, passing the original image through Kernel 2(Sharpening), produces the following images



K2 on R Channel



K2 on G Channel



K2 on B Channel

For Reference

Q2)
a)

```
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()
```

Fig: CovNet Architecture as depicted in the code

This has multiple convolutional 2d layers, appended together using Keras.Sequential.

It starts with the first layer, that has 16 filters (i.e. depth of model is 16) of size 3x3, pads the input image, so the layer's output will have the same spatial dimension as its inputs, and uses rectified linear activation function relu which will map negative outputs to 0.

Output of this is input to, Another conv2d layer of 16 filters of size 3x3 kernel. But it uses strides of 2,2 (2 pixel jump) and padding same. A quick lookup of how this works reveals that padding='same' with stride 2 is different and considered differently with another formula. **This makes the output dimensions lower.**

Next are two more conv2d layers, one having 32 filters of 3x3 kernels, and the second one same but with stride value 2,2.

After this a dropout layer is added which helps in overfitting situations for multi-layers neural nets. It helps in dropping certain neurons from the layers. It also updates the non-0 values.

A flatten layer after this, flattens the feature map into a 1-dimensional vector. This is important because we're going to insert this data into the flat neural network structure.

A Dense deeply connected layer, with 10 classes, and non-linear activation function softmax that returns the probability distribution of the vector values, and a strong regularizer l1 with alpha=0.0001 to penalize over-weighted vectors finally gets compiled with a loss function of categorical_crossentropy and adam optimizer. The output metric printed will be "accuracy".

b)
i)

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

This model has 37,146 parameters.

The layer with the most parameters is the Dense Layer with 20,490. This is because the (number of features in the previous layer + 1)*number of filters gives us, (2048+1)*10 which equals 20490. There's a parameter associated with every pair of inputs from the previous layer.

Post execution of code, the accuracy on the prediction on training data is 0.63 and on the test data is 0.50. The difference 0.13 is considerable, but it is logical, as test data is ideally data that the model has not seen, unlike train data. The recall, also drops from 63% in training data to 51% in test data.

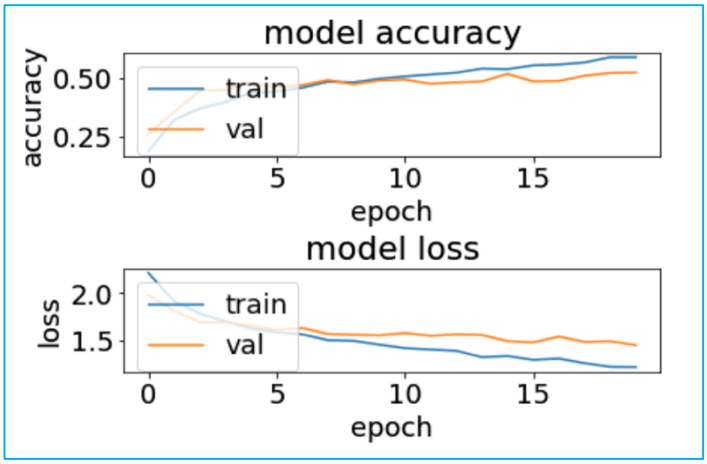
This is still a better performance than the baseline of DummyRegressor(strategy="mean") with accuracy 0.10.

```
from sklearn.dummy import DummyRegressor

dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)

preds3 = dummyModel.predict(x_train)
y_pred3 = np.argmax(preds3, axis=1)
y_train3 = np.argmax(y_train, axis=1)
print(classification_report(y_train3, y_pred3))
print(confusion_matrix(y_train3,y_pred3))

preds2 = dummyModel.predict(x_test)
y_pred2 = np.argmax(preds2, axis=1)
y_test2 = np.argmax(y_test, axis=1)
print(classification_report(y_test2, y_pred2))
print(confusion_matrix(y_test2,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       505
           1       0.00      0.00      0.00       460
           2       0.00      0.00      0.00       519
           3       0.00      0.00      0.00       486
           4       0.00      0.00      0.00       519
           5       0.00      0.00      0.00       488
           6       0.00      0.00      0.00       518
           7       0.00      0.00      0.00       486
           8       0.10      1.00      0.19       520
           9       0.00      0.00      0.00       498

    accuracy                           0.10      4999
   macro avg       0.01      0.10      0.02      4999
weighted avg       0.01      0.10      0.02      4999
```

Fig: Dummy Regressor precision, recall, f1-score, support, and avg accuracy

ii)



The two graphs plotted from the history variable are model accuracy and model loss. Now, in model loss graph, the training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

The initial epochs, have higher loss value, which may be indicative of partial underfitting, but decrease after the 5th epoch. There's a divergence between the training and validation *after the 10th epoch* which suggests slight overfitting, but not a lot as the validation trend doesn't change drastically.

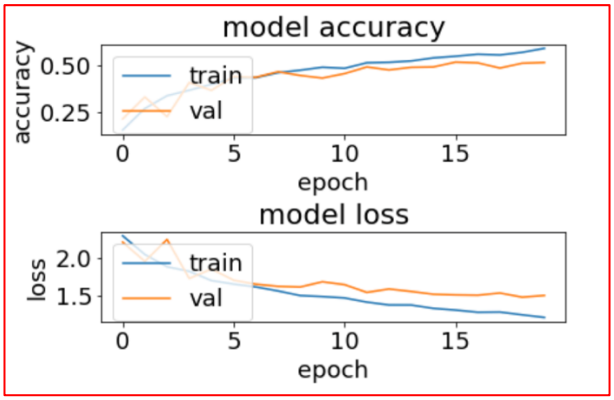
(Opinion: There is slight variance in the loss values of validation data predictions but we can't call it overfitting. A big gap may be representative of the fact that the training model is not enough for the model to capture all properties of the dataset)

The accuracy graph correlates to the stabilization of the loss graph, and accuracy is above 0.5 at the end of the iterations.

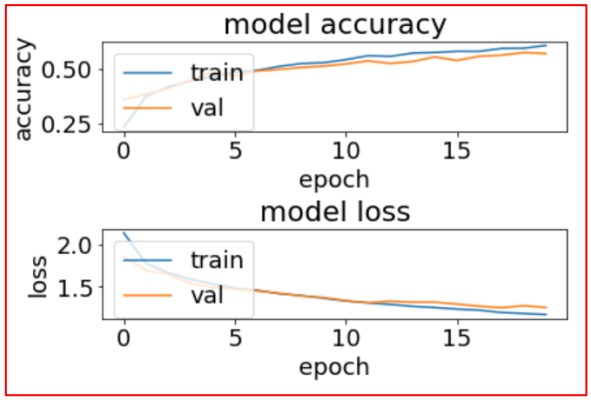
ii)

(My Config: Intel quad-core 2.4GHZ, with Intel iris graphic card – basic, LPDDR3)

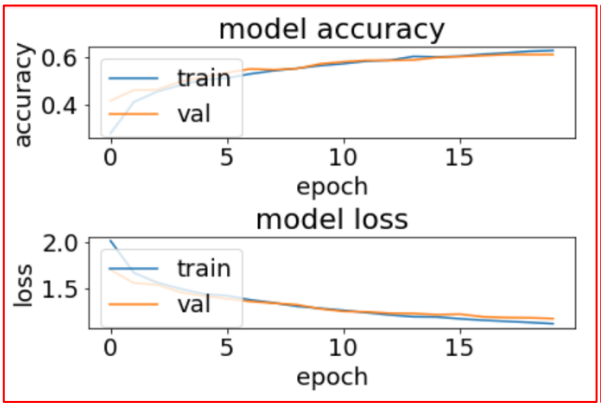
Training Dataset	Total time taken	Accuracy Train	Accuracy Test
5k	55 seconds	0.63	0.50
10k	114 seconds	0.67	0.57
20k	218 seconds	0.69	0.62
35k	377 seconds	0.70	0.66



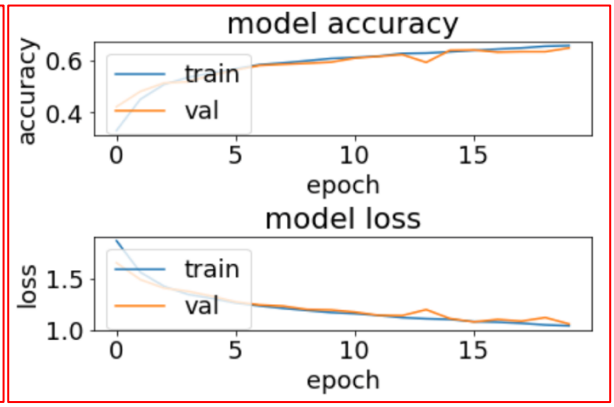
Model with 5k Train Set



Model with 10K Train Set



Model with 20K Train Set



Model with 35K Train Set

How does the time taken to train the network vary with the amount of training data used?
We can observe that the time almost doubles as the training data doubles.

How does the prediction accuracy on the training and test data vary with the amount of training data?

There is roughly a 10% jump in the prediction accuracies initially for both training and test, but eventually the accuracy seems to be plateauing, as without any significant improvement to the model, we're simply increasing the training set, and then finally running prediction on top of that. This is a tradeoff and depending on the use case this decision can be taken forward, so if it's for a business case, we can spend extra time and improve even the slightest accuracy, but in current scenario a jump from 0.65 to 0.70 for double the time, doesn't really make much of a difference.

Look at the plot of the "history" variable for each run, how does it vary with the amount of training data used and what does this indicate about over/underfitting?
There is definitely a smoothening observed in the model training and validation loss as the training set is increased. This means, that the model is able to correctly predict from the parameters learned in the training set. Though this is not indicative of overfitting, but merely increasing the training dataset, does however expose the model to some vulnerability towards overfitting.

iii)

L1 Parameter	Accuracy Train	Accuracy Test	Recall Test
0.00001	0.65	0.51	0.51
0.0001 (Initial)	0.65	0.51	0.51
0.001	0.60	0.51	0.50
0.01	0.46	0.42	0.42
0.05	0.42	0.40	0.40
0.1	0.36	0.35	0.35
0	0.51	0.44	0.44
1	0.19	0.18	0.18

I iterate over different L1 values and report the accuracy of train and test and recall for 5k training data set. While a strong regularization (lower value of L1) yields a better accuracy, we have to be wary for the weights not to be disregarded completely.
As the value is increased the accuracy decreases and I stop as it tends towards the dummy regressor with strategy mean as shown above. The interesting aspect is when the L1 has value 0 (no regularizer) and the accuracy is same as with a regularizer with value 0.01 to 0.001 range.
Increasing the training data from 5k to 20k improves the accuracy even for higher regularizer value like 0.01.
For 0.01 with 20k, the reported accuracy is : 0.51 Train and 0.47 Test and 0.47 Test Recall.

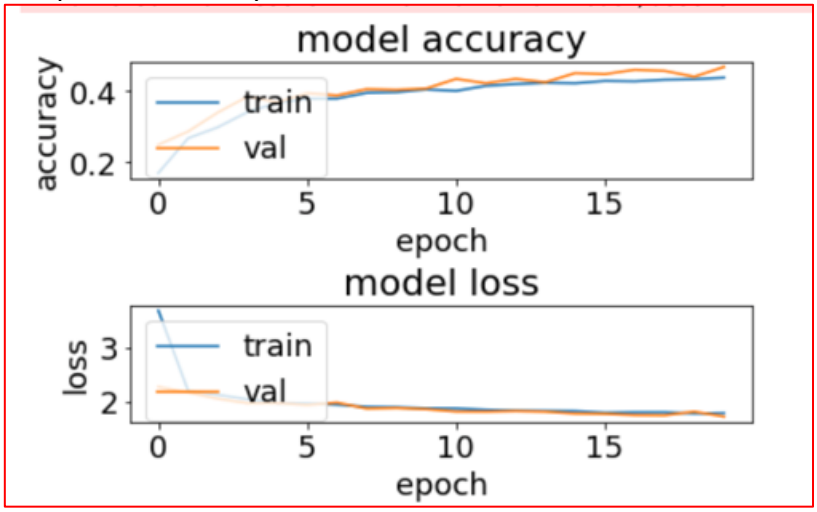


Fig: L1 0.01, 20K training data

This improvement however cannot be due to overfitting, as the model loss graph shows a tight weave between training loss and validation loss with validation loss even going under training loss! at times, indicating slight underfitting. So overall, the more the training data, the better the model, but it definitely is not overfitting it.

c)

i + ii)

```
else:
    model = keras.Sequential()

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

    # 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(MaxPooling2D(2,2))

    # 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(0.01)))
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    model.summary()
```

Here we pass the convolution feature map, to the max pooling function. In this we go over the feature map in a 2x2 window, and pick the max value in the window. Just like the convolution step, the creation of the pooled map disposes information and down samples. The reason we extract the maximum value, which is actually the point from the whole pooling step, is to account for distortions. In addition to that, pooling serves to minimize the size of the images as well as the number of parameters which, in turn, prevents an issue of "overfitting" from coming up.

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

Fig: Before MaxPooling and Commenting

------->

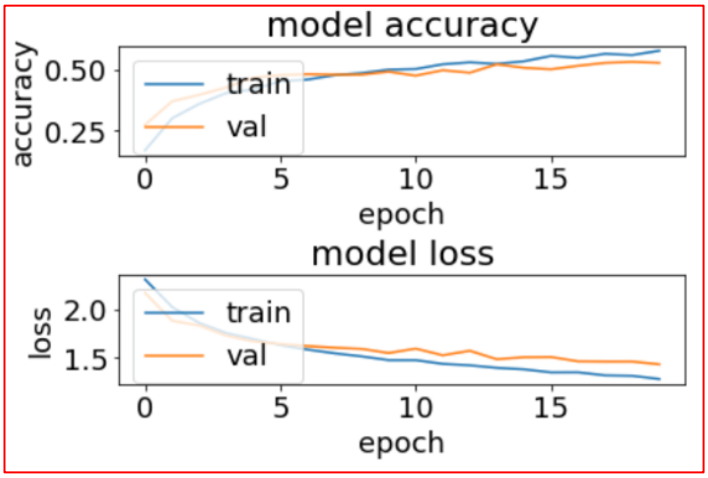
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 16)	448
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
dropout (Dropout)	(None, 8, 8, 32)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20490

Total params: 25,578
Trainable params: 25,578
Non-trainable params: 0

Fig: After MaxPooling and Commenting

The total parameters of the model have reduced and the model takes 43 seconds totally to train. The accuracy however, doesn't get affected. The train accuracy is 0.62 and the test accuracy is 0.53. The total params have changed from 37,146 to 25,578.

The train accuracy has decreased by 1% which could be due to reducing the features in the total model, but the test accuracy has increased by 2%. This could be very subjective to the test dataset as MaxPooling could be teaching the model to value the right features, that ultimately makes sense in this case.



If we increase the training set to 20k, the train accuracy is 0.66 as compared to 0.69 initially, and a test accuracy of 0.62 which is the same as before. The total time taken is 165 seconds as compared to 218 seconds. Thus, tuning a CovNet is really important. Adding more layers doesn't always mean increasing the accuracy. A good understanding of the data at hand, and applying the appropriate filters, helps us judge our model better.

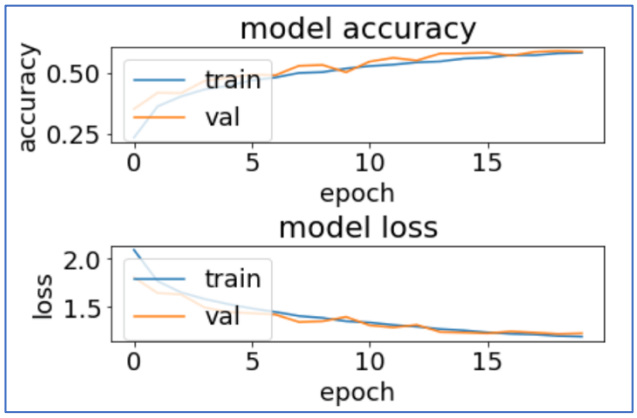
d) (optional)

The model here, has 2 conv2d layers with 8 filters of size 3x3 and one strided with a 2x2 window jump. Next is 16 filters , and finally 32 filters. A dropout of p=0.5 and finally flattening and inputting into the Dense layer.

Now this model does take longer to train, owing to additional layers getting added. It has 23,314 features in total.

```
orig x_train shape: (19999, 32, 32, 3)
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 32, 32, 8)         224

 conv2d_19 (Conv2D)          (None, 16, 16, 8)         584

 conv2d_20 (Conv2D)          (None, 16, 16, 16)        1168

 conv2d_21 (Conv2D)          (None, 8, 8, 16)          2320

 conv2d_22 (Conv2D)          (None, 8, 8, 32)          4640

 conv2d_23 (Conv2D)          (None, 4, 4, 32)          9248

 dropout_5 (Dropout)         (None, 4, 4, 32)          0

 flatten_5 (Flatten)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 23,314
Trainable params: 23,314
Non-trainable params: 0
_________________________________________________________________
```

The accuracy I got with 20000 images as the training dataset was - train accuracy 0.63, test accuracy 0.58.



The model should definitely benefit from training for more epochs, and decreasing the regularization from 0.0001 to 0.001 or 0.005. A max pooling layer instead of the second and third strided layers, would also improve the accuracy even further as evident from the above analysis and observation on the dataset.

End.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt



def convolution2d(image, kernel):
    kernel_height, kernel_width = kernel.shape
    padded_height, padded_width = image.shape

    output_height = (padded_height - kernel_height) + 1
    output_width = (padded_width - kernel_width) + 1

    new_image = np.zeros((output_height, output_width)).astype(np.float32)

    for y in range(0, output_height):
        for x in range(0, output_width):
            new_image[y][x] = np.sum(image[y:y+kernel_height,x:x+kernel_width]*kernel).astype(np.uint8)
    return new_image

M = 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]
])

K = np.array([
        [1, 0, -1],
        [1, 0, -1],
        [1, 0, -1]
    ])

print(convolution2d(M,K))
```

```
# def con(array, kernel):
#     #find new size
#     new_width = len(array[0]) - len(kernel[0])+1
#     new_height = len(array) - len(kernel)+1
#     result = []
#     #for each row in result
#     for row in range(new_height):
#         new_res_row = []
#         #fro each column in result
#         for col in range(new_width):
#             new_mat = []
#             #get appropriate slice of array
#             for i in range(len(kernel) ):
#                 new_mat_row = []
#                 for j in range(len(kernel[0])):
#                     new_mat_row.append(array[row+i][col+j])
#                 new_mat.append(new_mat_row)
#             sum = 0
#             #for row of new array
#             for res_row in range(len(kernel)):
#                 #for column of new array
#                 for res_col in range(len(kernel[0])):
#                 #convolve this new array
#                     sum += (new_mat[res_row][res_col] * kernel[res_row][res_col])
#             #append reult to row
#             new_res_row.append(sum)
#         #append row to result
#         result.append(new_res_row)
#     return result
```

```
from PIL import Image
im = Image.open('tri.png')
rgb = np.array(im.convert('RGB'))
r=rgb[:,:,2] #array of R pixels
# r=r.tolist()
# M=Image.fromarray(np.uint8(r)).show()
K = np.array([
        [-1, -1, -1],
        [-1, 8, -1],
        [-1, -1, -1]
    ])
Image.fromarray(np.uint8(np.array(convolution2d(r,K)))).show()
# print(convolution2d(r,K))
```

```
from PIL import Image
im = Image.open('tri.png')
rgb = np.array(im.convert('RGB'))
r=rgb[:,:,2] #array of R pixels
# M=Image.fromarray(np.uint8(r)).show()
K = np.array([
        [0, -1, 0],
        [-1, 8, -1],
        [0, -1, 0]
    ])
Image.fromarray(np.uint8(np.array(convolution2d(r,K)))).show()
```

```
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
import time
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
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(2,2))

#     # 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(MaxPooling2D(2,2))

#     # 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.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()

    start = time.time()
    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.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))

stop = time.time()
duration = stop-start
print("Total Time Taken for 5k Training dataset: "+ str(duration))
```