ECE 9202: Advanced Image Processing

Deep Convolutional Neural Network Design

Chandan Chandel

2023-03-02

Section I: Dataset

I originally tried to tackle the challenge of classifying the CIFAR 100 dataset, but unfortunately I did not achieve strong enough results(50%) as the deadline approached so I simply loaded the CIFAR10 dataset and my network immediately beat the classification goals (70%). The CIFAR 10 dataset has 10 classes and 60000 images, with 6000 image per class. Each image is 32x32 and is of RGB type as opposed to a black and white image. The CIFAR10 dataset was split into train and test sets along a 5:1 ratio, therefore the training set has 50000 images.

The images (x train and test data) were normalized using min-max scaling.

$$x' = (x - \min(x))/(\max(x) - \min(x))$$

The labels were transformed into categorical data using the `keras.utils.to_categorical` command.

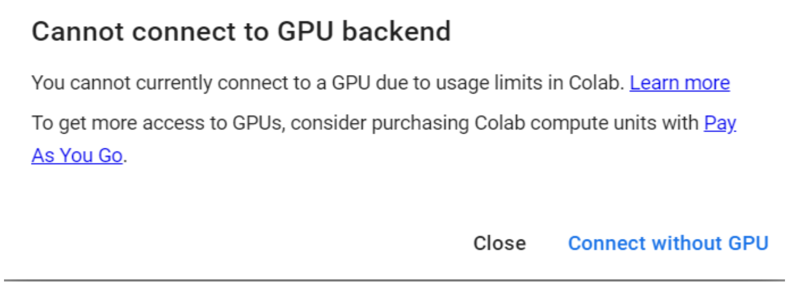
Section II: Design your Deep Convolutional Neural Network

I conducted 83 trial & error type experiments [1] in an effort to find an architecture that worked. The variable I modulated included adding/removing convolutional layers, adding/removing dense layers, adding/removing max pooling layers, using/not using the padding capability, changing the number of filters in each convolution, changing the kernel size in each convolution, modifying the regularization parameter of the dropout layer and much more and modifying the initial learning rate. After each new model was fitted the testing and training accuracies were recorded please feel free to check this trial data in the excel sheet shown in the Appendix. Also see item 2 in the Appendix to find the model that was used.

Section III: Final Performance & What Could Have Been Done Differently?

In Trial 83 the testing accuracy was 80.1% while the training accuracy was 95.9% - this suggests that there was strong overfitting. I tried to combat this by the following means; increasing the regularisation parameter value in the Dropout Layers, by decreasing the number of free parameters and by implementing early stopping. I also tried to use data augmentation for the images as well but I did not find very helpful results from this process when I used ImageDataGenerator class from tensorflow.keras.preprocessing.image, this is shown in the trial data.

I wish I had not relied on Google Collab because I occasionally got messages like this tat would not go away for up to 2 days sometimes. In the future I will make sure I have a local GPU that works. There was a considerable waste of time as a result of this issue with Google's Cloud resources. I also should have pre-trained my models as this would have yielded better results as well. I also should have used ResNet or another more established model as a benchmark to compare my results against but unfortunately Google Collab will not allow me to connect to their GPU's anymore so I am unable to completed this comparison yet.



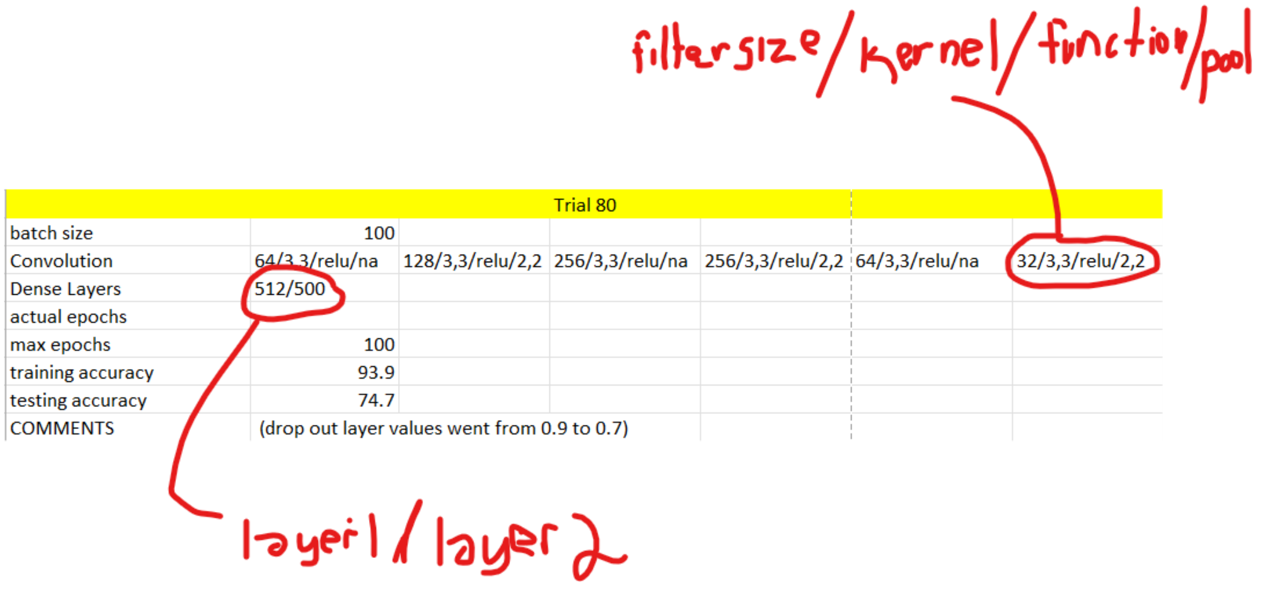
Cannot connect to GPU backend

You cannot currently connect to a GPU due to usage limits in Colab. Learn more

To get more access to GPUs, consider purchasing Colab compute units with Pay As You Go.

Appendix

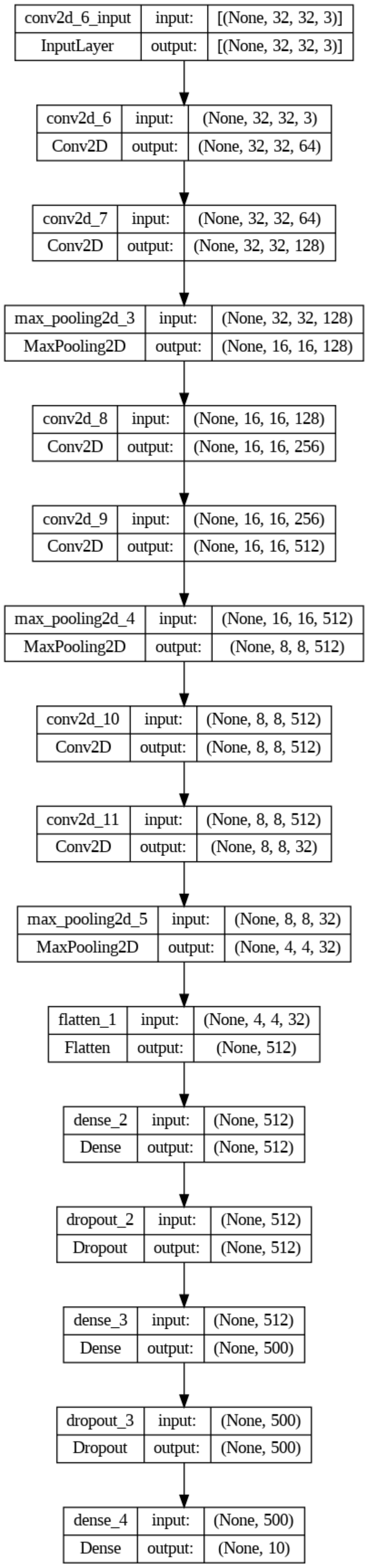
[1] Please check the attached Excel sheet for all of the trial data. The Trial data format is explained in the snippet below.



filtersize/kernel/function/pool
layer1/layer2

	Trial 80					
batch size	100					
Convolution	64/3,3/relu/na	128/3,3/relu/2,2	256/3,3/relu/na	256/3,3/relu/2,2	64/3,3/relu/na	32/3,3/relu/2,2
Dense Layers	512/500					
actual epochs						
max epochs	100					
training accuracy	93.9					
testing accuracy	74.7					
COMMENTS	(drop out layer values went from 0.9 to 0.7)					

[2] Final visualized model (on the next page)


conv2d_6_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d_6 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 64)
conv2d_7 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 128)
max_pooling2d_3 | input: | (None, 32, 32, 128)
MaxPooling2D | output: | (None, 16, 16, 128)
conv2d_8 | input: | (None, 16, 16, 128)
Conv2D | output: | (None, 16, 16, 256)
conv2d_9 | input: | (None, 16, 16, 256)
Conv2D | output: | (None, 16, 16, 512)
max_pooling2d_4 | input: | (None, 16, 16, 512)
MaxPooling2D | output: | (None, 8, 8, 512)
conv2d_10 | input: | (None, 8, 8, 512)
Conv2D | output: | (None, 8, 8, 512)
conv2d_11 | input: | (None, 8, 8, 512)
Conv2D | output: | (None, 8, 8, 32)
max_pooling2d_5 | input: | (None, 8, 8, 32)
MaxPooling2D | output: | (None, 4, 4, 32)
flatten_1 | input: | (None, 4, 4, 32)
Flatten | output: | (None, 512)
dense_2 | input: | (None, 512)
Dense | output: | (None, 512)
dropout_2 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 500)
dropout_3 | input: | (None, 500)
Dropout | output: | (None, 500)
dense_4 | input: | (None, 500)
Dense | output: | (None, 10)