11/19/2021

MET CS 767 Assignment 3T: CNN’s

*Nathan A. Horak*

Applied machine learning—after architecture selection—is largely a process of selecting appropriate parameters. This requires an understanding of the parameters, and a systematic approach to dealing with parameter values. The purpose of the present assignment is to give you experience with this.

Copy the implementation [here](https://colab.research.google.com/drive/1Yg-NXKlYzfvv9jI2MCxWZErovpR57_YC?usp=sharing) to your Google drive. Systematically modify this code in four ways as below, attempting to improve the output, and report the results, using this Word file as a template. Since the accuracy of the given implementation is already high, consider reducing the size of the CIFAR training set—or substituting it so that the baseline implementation leaves more percentage room for improvement. You may combine modifications, but each of the sections below should contain at least one new change. If necessary, show changes that make the result worse, with your explanation.

Please leave the gray text and the headings unchanged.

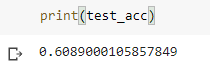
# 1. First Code Modification

## 1.1 Description of what I did and reason this *could reasonably be* an improvement (one paragraph)

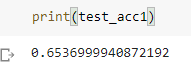
Introducing convolutional layers to a neural network can drastically improve the accuracy and performance of a neural network algorithm (see Appendix A for more on how convolutional layers work with max pooling). After initially running the code, the given test accuracy of around 0.6000 shows room for improvement. As such, the sample size will remain as is. When I first took a look at the code and results, one thing that stuck out to me (addressed in Section 1.2) is that the training accuracy is less than the validation accuracy. This means that we should not be too concerned with overfitting (if anything underfitting) and aim to address our model to fit better, according to Dipti Rohan at Medium[5]. To improve our accuracy and loss values, we want to enable the model to learn more. An easy way to do this is to increase the filter size of our Conv2D layers, according to pyimagesearch.com[1]. We will increase the filter size of our three Conv2D layers from 32, 64, and 64 to 64, 128, and 256. This change will allow our model to examine many more parameters as the algorithm is executed- allowing the neural network algorithm to recognize and learn from patterns. The number of parameters in the middle three layers changes from 18496, 36928, and 65600 to 73856, 295168, and 1048832 in the original code and first code modification.

## 1.2 Comparison of the result with the original output, with explanation

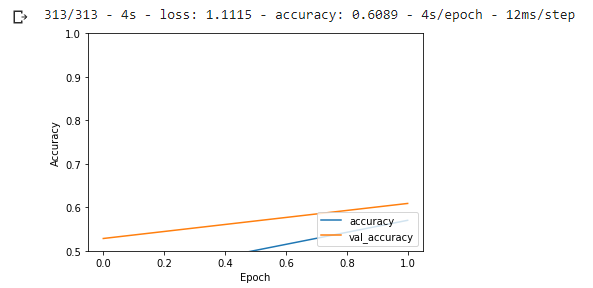
There are two measures we are interested in improving upon- loss and accuracy. Accuracy refers to the percentage of the output correctly predicted given the input while loss is a measurement that the model seeks to minimize during training, per TensorFlow Core Tutorials[7]. Accuracy and loss exist generally and for specific subsets, namely validation accuracy/loss and test accuracy/loss. Validation accuracy and validation loss are these terms applied to a subset of the training set known as the validation set, according to machinelearningmastery.com[2]. While our model will iterate upon itself using the accuracy and loss from training as well as validation, our true measuring stick of the model’s performance will be the test accuracy. The original code gave a test accuracy of 0.6089.



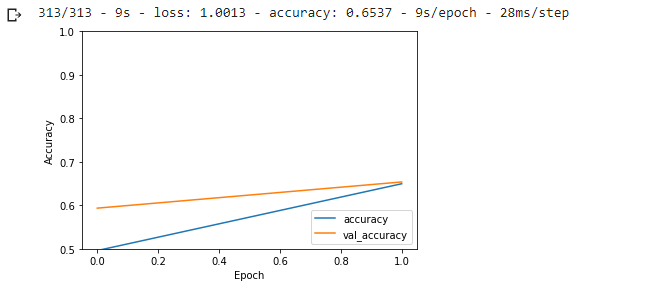
Utilizing the changes described in Section 1.1, the test accuracy improved to 0.6537.



An increase in test accuracy of 0.0448 is promising, but there is still more work to be done. As mentioned above, the model updates weights as it iterates based on training accuracy/loss and validation accuracy/loss. It is important to note what it means for training measurements to be better than validation measurements, and vice-versa. Overfitting occurs when the algorithm memorizes the dataset rather than learning from it- or when it models the training data too well, per Dipti Rohan at Medium[5]. A tell-tale sign that our model has an overfitting issue is if the training accuracy/loss are significantly better than the test or validation accuracy/loss. According to towardsdatascience.com, overfitting occurs when a model starts to memorize values from the training data rather than learning from them[6]. This is something that a machine learning engineer wants to avoid as the goal is to optimize test/validation results rather than training results. Our model starts out with the training accuracy lower than the validation accuracy:



The first code modification shows the trend line of training accuracy approaches validation accuracy. This means that currently, we should not be too concerned about overfitting but it is something to remain mindful of going forward as we apply additional changes to the code.



## 1.3 URL of your Colab code

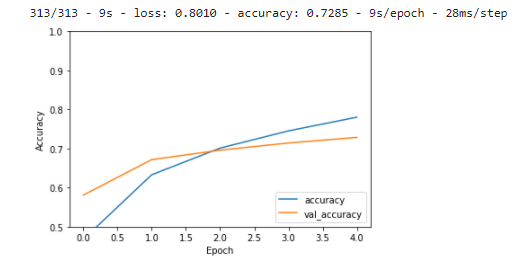
<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=AYFvqz9b6eFR>

# 2. Second Code Modification

## 2.1 Description of what I did and reason this *could reasonably be* an improvement (one paragraph)

After executing the first code modification, there is a second modification I want to incorporate and another change that I want to consider. The graphs that show training accuracy and validation accuracy by epoch were very helpful in examining what type of changes I wanted to make to the code as well as making sure that overfitting was not a serious problem. However, much more can be made of a graphed line with multiple points on it rather than a line with just two. As such, I was set on expanding the number of epochs to not only improve the accuracy of the model but also to see how the training accuracy and validation accuracy changed over each successive epoch. This does come with a downside- each additional epoch will increase the running time. According to Dipti Rohan at Medium, increasing the number of epochs is a proven way to increase the performance of a convoluted neural network[8]. I first tried with 8 epochs (see Appendix B), but after analyzing the training and validation accuracy over epochs, I determined that 8 is too high, at least at this stage when I am experimenting with the model. Around 4 or 5 epochs, the validation accuracy gain ceased to increase while the running time continued to. Given this, I 8 is too large of a number for epochs (at least at this point) and I decided to move forward with 5 epochs. I am open to increasing the number of epochs in the future as I iterate further on the code, but the flat-lining of validation accuracy leads me to believe that epochs 6 through 8 are not currently necessary given the increase in training time. Secondly, I want to introduce some dropout instances to the layers to prevent overfitting as I increase the model’s complexity. As addressed in Section 1.1, I want to closely monitor training accuracy versus validation accuracy over epochs.

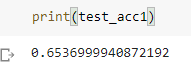
*Training accuracy and validation accuracy over epochs with no dropout:*



Given the difference between training accuracy and validation accuracy, I want to introduce dropout to allow the model to learn rather than simply memorize the training set. I settled on adding four instances of dropout- an initial dropout with a parameter value of 0.20 following the first convolutional layer and max pooling followed by subsequent dropouts with parameter value of 0.30 following the next Conv2D/MaxPooling2D layers and then before the final output layer, after the convolutional layers were flattened (see more in Appendix C).

## 2.2 Comparison of the result with the original output, with explanation

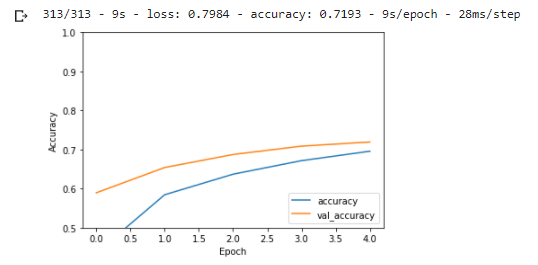
As mentioned in Section 1.2, we are interested in improving the test accuracy. After the first code modification changes, the resulting test accuracy is 0.6537.



Following the second code modification changes, the test accuracy improved to 0.7193.



As mentioned in Section 1.1 (also Appendix B), it is important to note the change of training accuracy compared to validation accuracy over each successive epoch. Given the below results, with both lines trending upward but slowing in growth and with validation accuracy greater than training accuracy, it seems as though our results are improving and that the changes made in the second code modification were beneficial.



Given that validation and training accuracy seem to be still increasing, there is an argument for adding a 6th epoch- this is something that I’ll keep in mind going forward to the third code modification.

## 2.3 URL of your Colab code

<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=alftEwIEh_pW>

# 3. Third Code Modification

## 3.1 Description of what I did and reason this *could reasonably be* an improvement (one paragraph)

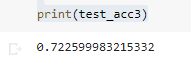
In Section 2.2, I mentioned a positive trend line of the training accuracy and validation accuracy and how I would approach improving the model further. I entertained the addition of a 6th epoch but I decided against including it for this step given the increased iteration time from associated changes. I want to see if I could further improve upon the algorithm before I add an additional epoch. I first decided that I wanted to lower the dropout parameter from 0.30 to 0.20 for all hidden layers, to match the input layers (see Appendix C). I also decided to increase the size of the penultimate dense layer (after being flattened) from 256 to 512. I wanted to increase the ability of the model to learn, and increasing the filter size of the second-to-last layer would allow the layer to contain more information from the previous Conv2D and MaxPooling2D layers. I also slightly adjusted the code (but not the functionality) by moving the layer activation out to a separate line. According to pyimagesearch.com, this maintains the same functionality[1]. I wanted the activation method to be separate in this way to keep the code clean with short lines and to allow the activation method to be changed easily. I attempted to add batch normalization to the model (for more on batch normalization, see Appendix D) but I was not impressed with the results. This makes sense given that batch normalization and dropout combined can be an overall negative in a model, according to machinelearningmastery.com[2]. The main change I went forward with was increasing the batch\_size parameter (for more see Appendix E). According to towardsdatascience.com, trying different values for batch size is a good way to increase the performance of a convolutional neural network[6]. Continuing with the pattern of increasing or dividing values by 2, I set the batch\_size to 64 up from 32. After seeing some erratic movement on the training accuracy and validation accuracy graphs, I wanted to add some stability to the learning process such that my algorithm could operate with minimal noise. According to machinelearningmastery.com, increasing batch size is a good way of introducing stability[2].

## 3.2 Comparison of the result with the original output, with explanation

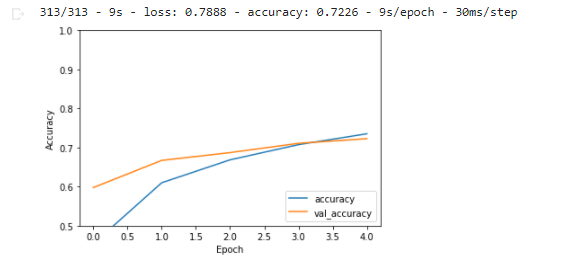
The main measure we are interested in improving is our test accuracy. After the first and second code modification changes, the resulting test accuracy is 0.7193.



The third code modification increases the accuracy to 0.7226.



Although this is not as large of an increase in test accuracy as there was in previous steps, it is an increase nonetheless- without increasing the number of epochs. Examining the training and validation accuracy over epochs, these trendlines keep increasing and don’t show much sign of slowing down. Given this, I see no reason to not increase the number of epochs for the final code modification.



## 3.3 URL of your Colab code

<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=Zw3ZXDarJmj_>

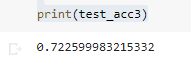
# 4. Fourth Code Modification

## 4.1 Description of what I did and reason this *could reasonably be* an improvement (one paragraph)

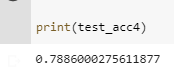
For the last code modification, I knew that I wanted to increase the number of epochs. From the graph in Section 3.2, the training accuracy and validation accuracy seem to be steadily increasing as the number of epochs increase. I wanted to see if I could continue that trend. I also wanted to add more complexity to the model. At first, I attempted to add another Conv2D layer with dropout and MaxPooling2D before flattening into the penultimate Dense layer. However, given how MaxPooling works, this would not be possible given the output shape (see more in Appendix F). I settled on lowering the filter size of the input Conv2D layer and then adding a secondary Conv2D layer immediately after the first at twice its filter size. This would maintain the same escalation of filter size going through the model, although the model would start from a filter size of 32 rather than 64. Along with increasing the number of epochs, I wanted to also increase the model’s complexity. Increasing the number of epochs allows for additional modeling time as well as increased batch size for stability, per machinelearningmastery.com[2]. Another benefit of having the initial layer have a filter of size 32 is that this matches the dimensions of the input shape given the nature of the data. In the first step of the algorithm, there will initially be a Conv2D input layer with filter size equal to the dimensions of the original images (32x32). Then after a ‘ReLU’ activation, there is another Conv2D with filter size 64. After a MaxPooling2D layer followed by dropout, the model continues. The addition of another Conv2D at the front of the algorithm gives another step for the model to learn expanding from a filter size of 32 to 64 before pooling and dropout.

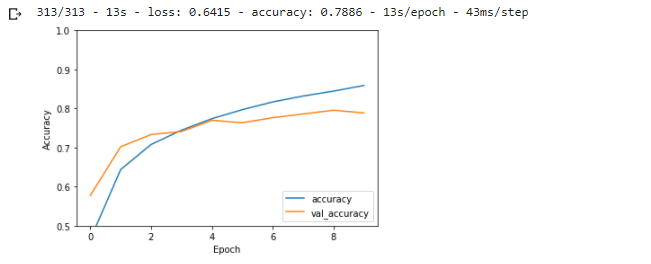
## 4.2 Comparison of the result with the original output, with explanation

As mentioned in Section 1.2, the metric that we want to improve is test accuracy. Following the first three code modification changes, the test accuracy is 0.7226.



Following the fourth and last code modification change the test accuracy improved to 0.7886.



The fourth code modification substantially improved the test accuracy. When examining the training accuracy and validation accuracy epoch graph, there seems to be a meaningful gap between training accuracy and validation accuracy in later epochs, with a larger training accuracy. [2]The validation accuracy line even dips at the very end, although it seems unlikely that this would be a permanent trend or if it is simply an anomaly. This does, however, lend credence to the idea that the model can be further improved upon.

## 4.3 URL of your Colab code

<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=1PGIGMJVQjT4>

For more modifications, see Appendix G, Appendix H, & Appendix I.

# References

Show that you used a wide variety of resources by listing them below and clearly indicating in the body above where you used. Make sure to use proper referencing in your paper. We suggest using the APA format, but other formats are fine as long as they clearly distinguish your work from the work of others in your response. In general, observe the stated plagiarism rules.

[1] Rosebrock, Adrian. (2018, December 31). *Keras Conv2D and Convolutional Layers*. pyimagesearch.com. Retrieved November 20, 2021. <https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/>

[2] Brownlee, Jason. (2020, April 17). *Machine Learning Mastery*. machinelearningmastery.com. Retrieved November 20, 2021. [https://machinelearningmastery.com/](https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/)

[3] @MRINALWALIA. (2020, May 18). *Keras.Conv2D Class*. geeksforgeeks.org. Retrieved November 20, 2021. <https://www.geeksforgeeks.org/keras-conv2d-class/>

[4] deeplizard. (2018, February 18). *Machine Learning & Deep Learning Fundamentals*. deeplizard.com. Retrieved November 20, 2021. <https://deeplizard.com/learn/video/ZjM_XQa5s6s>

[5] Pawar, Dipti. (2018, August 14). *Improving Performance of Convolutional Neural Network!*. medium.com/@dipti.rohan.pawar. Retrieved November 20, 2021. <https://medium.com/@dipti.rohan.pawar/improving-performance-of-convolutional-neural-network-2ecfe0207de7>

[6] Gandhi, Rohith. (2018, May 17). *Improving the Performance of a Neural Network.* towardsdatascience.com. Retrieved November 20, 2021. <https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d>

[7] *TensorFlow Tutorials*. tensorflow.org. Retrieved November 21, 2021. <https://www.tensorflow.org/tutorials/>

[8] Ghouzam, Yassine. *Introduction to CNN Keras - 0.977 (top 6%)*. Retrieved November 23, 2021. <https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6>

# Evaluation

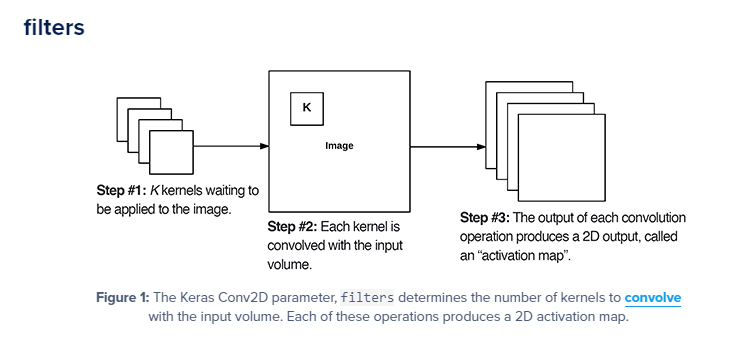


# Appendix

[A]

<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=_ujkOs1LUA1q&uniqifier=1>

The addition of convolutional and max pooling layers can improve the performance of a neural network. A convolution is the simple application of a filter to an input that results in an activation, according to machinelearningmastery.com[2]. Per pyimagesearch.com, kernels are convolved with the input volume and then each convolution produces a 2D output (in the case of a Conv2D layer) called an “activation map”[1].



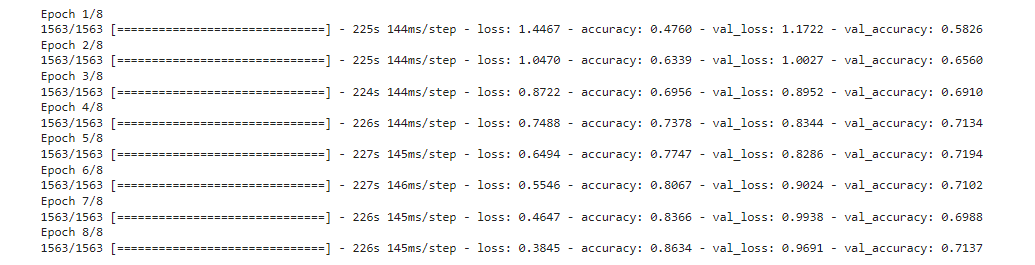
*Image courtesy of pyimagesearch.com*

According to Jason Brownlee at machinelearningmastery.com, convolutional layers grant the ability to automatically learn a large number of filters under constraints of a specific predictive modeling problem, such as image classification [2]. From a code aspect, a Keras Conv2D is a 2D Convolution Layer wound with layers input which helps produce a tensor of outputs, according to geeksforgeeks.org[3]. Given that the problem we are aiming to address is related to image classification, use of convolutional layers seems like a solid choice.

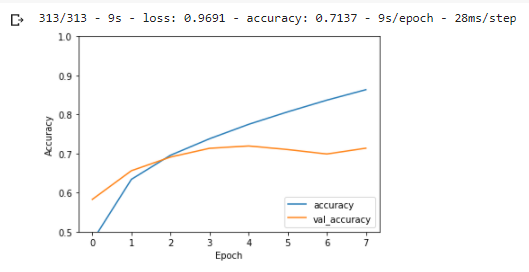
[B]

As mentioned above, when I was experimenting with the number of epochs in the second code modification I first settled on 8 epochs. However, after analyzing the resulting graphic, I determined that at this point in the code modification process, 5 would be a much better value for the number of epochs. I determined this after analyzing the results and viewing the graph of training accuracy and validation accuracy over epochs. Around the 4th or 5th epoch, the increase in validation accuracy seems to stop increasing (and indeed halt/decrease) while the training accuracy steadily increases.

*Training loss, training accuracy, validation loss, and validation accuracy over 8 epochs:*



*Training accuracy and validation accuracy over 8 epochs:*



Given the additional processing time per additional epoch, I decided to set the number of epochs to 5 for the second code modification. I’m open to the possibility of increasing the number of epochs further if I can change the code such that the validation accuracy continues to increase from the 3rd to the 4th and then to the 5th epoch.

[C]

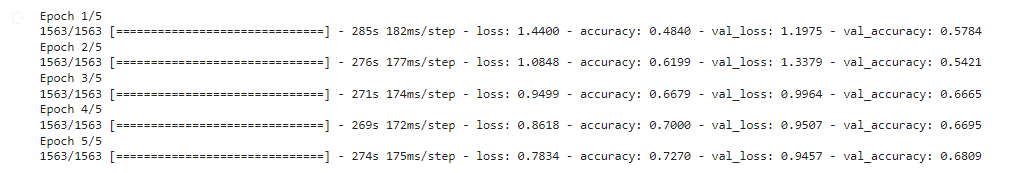
Dropout is an important element of neural networks. According to machinelearningmastery.com, dropout is a technique used to prevent overfitting the training data[2]. It works by randomly skipping neurons during training with the goal of encouraging the model to learn from the data rather than simply memorize the training data. From Jason Brownlee of machinelearningmastery.com, a common value is a probability of 0.50 (dropout parameter of 0.50) for hidden layers and a value close to 1.0, such as 0.80 (dropout parameter of 0.20) for the input layer[2].

[D]

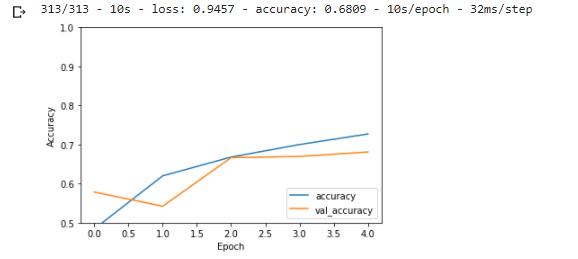
<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=lLEKoc7YfUVX&uniqifier=1>

According to Jason Brownlee of machinelearningmastery.com, training deep neural networks can be challenging as they are sensitive to initial random weights[2]. One way to address this is to incorporate batch normalization. Per the afore-mentioned source, batch normalization works by standardizing the inputs to a layer for each batch and stabilizing the learning process. Batch normalization allows a user to dramatically reduce the number of epochs used. Given the additional processing time associated with each epoch, the last point is especially beneficial. Per machinelearningmastery.com, it is generally best to use before the activation function, especially when using activation methods such as the rectified linear activation function (ReLU). However, there are some issues with incorporating batch normalization- specifically, that it is not the best idea to use batch normalization in conjunction with dropout. The statistics used to normalize the activations of the previous layer may be subject to noise given the random dropping of nodes during dropout, from machinelearningmastery.com[2]. See the results below showing training accuracy, validation accuracy, and test accuracy with dropout as well as batch normalization.

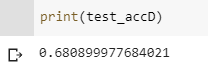
*Training loss, training accuracy, validation loss, and validation accuracy with dropout and batch normalization:*

**

*Training accuracy and validation accuracy with dropout and batch normalization over 5 epochs:*

**

*Test accuracy with dropout and batch normalization:*

**

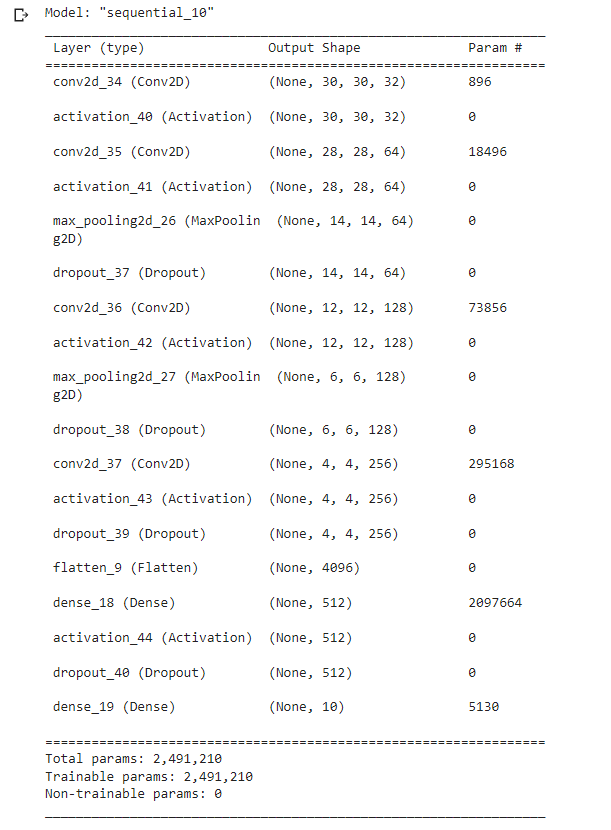
As seen from the above graphs and test accuracy, incorporating use of both a dropout and batch normalization resulted in lower training accuracy, validation accuracy, and most importantly, test accuracy. Given the delicate nature of tweaking machine learning parameters, it seems most efficient to either use a dropout instance or incorporate batch normalization rather than try to fine-tune both values for an optimal result.

[E]

According to Jason Brownlee of machinelearningmastery.com, the batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model’s internal parameters are updated[2]. Per the above-mentioned source, a larger value of the batch\_size hyperparameter gives a learning process that converges more slowly when estimating the error gradient. This can create less noise during the training process[2].

[F]

Per deeplizard.com, MaxPooling is an operation used following individual convolutional layers to reduce the number of pixels from the previous layer[4]. MaxPooling is mainly used to reduce computational load and limit overfitting. By reducing the resolution of the output from a convolutional layer, the number of parameters are reduced and thus computational load is decreased- resulting in a faster runtime. Incorporating MaxPooling can give a machine learning engineer the option of additional Conv2D layers without drastically increasing running time[4]. Although some of the information gained from learning with an additional layer is lost by nature of reduced dimensions, MaxPooling is an extremely useful tool when used with multiple Conv2D layers. However, one cannot keep adding Conv2D and MaxPooling2D layers indefinitely. Given that we are using a TensorFlow MaxPooling2D instance with pool\_size of (2,2), the first two dimensions of the output’s shape are reduced by a factor of 2 for each MaxPooling2D instance. Furthermore, the output shape’s first two dimensions are subtracted by 2 (one minus the kernel size- [3,3]) for every Conv2D layer. The nature of these changing dimensions can be seen below:



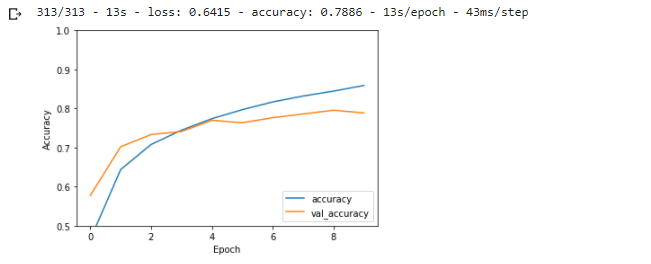
Given that the algorithm cannot function when the output shape has a ‘0’ in one of its three dimensions, both the number of Conv2D layers but especially the number of MaxPooling2D layers are intrinsically limited in number by the dimensions of the input shape (32,32,3 for this problem).

[G]

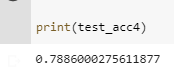
<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=jaEC__a4fgAz&uniqifier=1>

An additional change I wanted to try was adjusting the dropout rates for the hidden layers. As referenced in Appendix C, it is not uncommon to increase the dropout parameter and thus lower the percentage of the data to run through the algorithm for the subsequent layers beyond input. In the fourth code modification, I had the dropout rate parameter constant at 0.20 for all instances of dropout. Given the difference in training accuracy and validation accuracy after the fourth code modification in the higher epochs, I wanted to raise the dropout parameter for instances beyond the first. I decided to test the dropout parameter at 0.40.

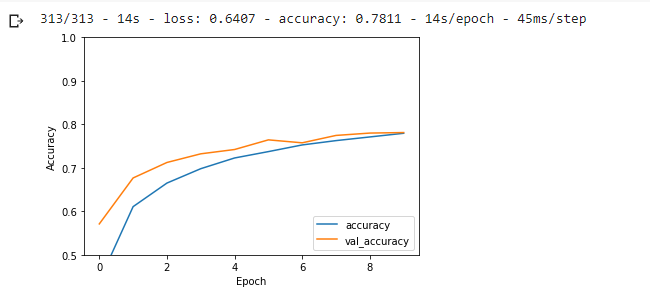
*Training accuracy and validation accuracy for Fourth Code Modification (20% Dropout for subsequent instances) over 10 epochs:*



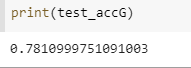
*Test accuracy for Fourth Code Modification (20% Dropout for subsequent instances) over 10 epochs:*



*Training accuracy and validation accuracy for Appendix G Modification (40% Dropout for subsequent instances)over 10 epochs:*



*Test accuracy for Appendix G Modification (40% Dropout for subsequent instances) over 10 epochs:*

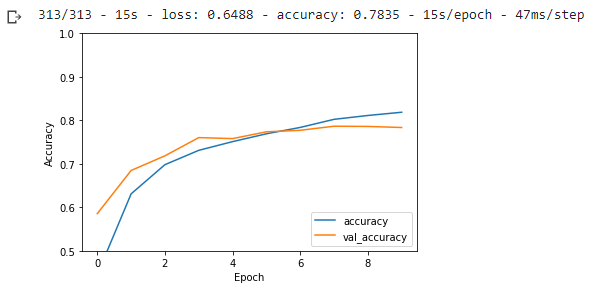


Comparing the two test accuracy values, the change of 0.20 to 0.40 dropout parameter did not increase the test accuracy, in fact, it slightly reduced it. Comparing the two graphs, the validation accuracy is not higher for the changed dropout parameter while the training accuracy is lower. Given this and the nature between overfitting and training/validation accuracy discussed in Section 1.1 and 1.2 I wanted to try the model with a dropout parameter of 0.30 for subsequent dropout instances past the first (see Appendix H).

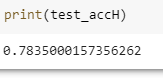
[H]

<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=wWAv7rTI7fTy&uniqifier=1>

As mentioned in Appendix G, I wanted to run the fourth code modification model except with a dropout parameter of 0.30 for dropout instances past the first. Interestingly enough, these are the dropout values I used initially for the Second Code Modification. The results for the fourth code modification with 0.30 dropout parameter for subsequent dropout instances past the input is below.

*Training accuracy and validation accuracy for Appendix H modification (30% Dropout for subsequent instances) over 10 epochs:*

*Test accuracy for Appendix H Modification (30% Dropout for subsequent instances) over 10 epochs:*

**

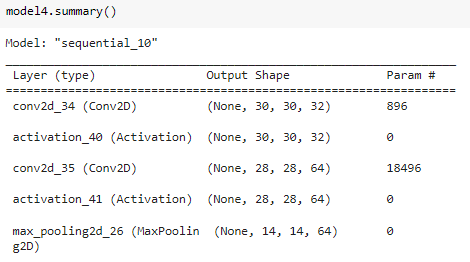
Comparing the output with the results from the fourth code modification, the training accuracy is lower, the validation accuracy around the same value, and the test accuracy slightly lower. While the results seem to be better than the modifications from Appendix G (0.40 dropout parameter) it seems that any further iteration of the fourth code modification model to improve the test accuracy should not involve tweaking with the dropout parameters- at least at this point.

[I]

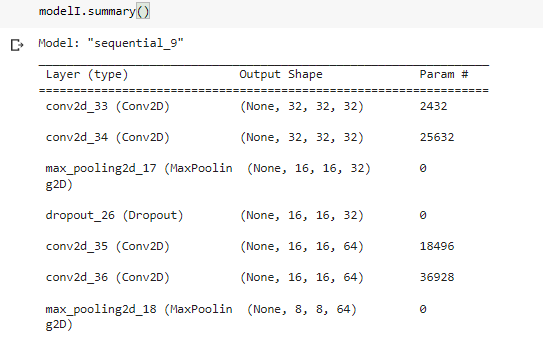
<https://colab.research.google.com/drive/1Ks8Q26DYOXBX5ElFPz0BGlmBQTvzPzlO#scrollTo=oZ8TNA8ABS-2&uniqifier=1>

Creating neural networks, particularly ones using Conv2D layers with MaxPooling2D provide a user with a lot of opportunities for customization. There are so many parameters to tweak- this can be seen going over the TensorFlow tutorials[7]. After I spent a lot of time iterating on the code, I wanted to see what other individuals came up with in regards to their convolutional neural network. One I stumbled across was via kaggle, from a user named Yassine Ghouzam[8]. Yassine Ghouzam has a PhD and likely much more experience creating convolutional neural networks than I do. The code was written originally for a different dataset (but also image classification) and I wanted to examine the differences between my code, incorporate his code, and then compare results. Starting from the top of the model, a noticeable difference between our models is that the first two Conv2D layers, including the initial layer, have a kernel size of (5,5) while the next two have a kernel size of (3,3)[8]. Per pyimagesearch.com, it is often prudent to use (5,5) or even (7,7) kernel size to learn larger features and reduce dimensions for subsequent Conv2D layers[1]. This is something that I could’ve easily incorporated into my model, likely to it’s benefit. Another change I noticed is that the Appendix I model Conv2D layers incorporate padding where mine did not, with the default setting. Conv2D layer padding incorporates the same output dimensions with the addition of zeros from each Conv2D layer.

*First few layers of Fourth Code Modification model (no padding):*

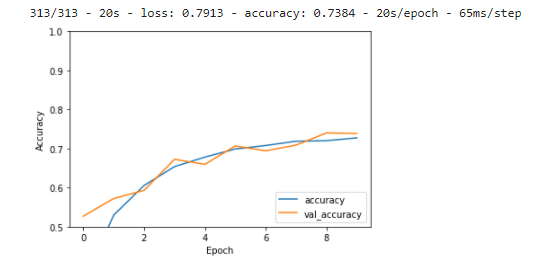


*First few layers of Appendix I Code Modification (padding):*



Similar to my model, there are 2 Conv2D layers before pooling and dropout. The aforementioned kaggle user maintains the filter size at 32 before being halved to 16 after a MaxPooling2D layer whereas the first two dimensions of my model go from 30 to 28 to 14. Padding introduces uniformity and is helpful in model design. By keeping the first two dimensions of size base 2 (2, 4, 8, 16, 32, 64) it is much cleaner to incorporate in MaxPooling2D layers. However, the introduction of additional zeroes on the perimeter locations of a dimensional shape can introduce noise into a model. It is possible my model would have benefitted from this as well, but it is not certain. While the size doubling to 64 was similar to the progression in my model, the next two Conv2D layers (following a dropout instance) were stacked together much like the first two were- with no MaxPooling2D layer in between[8]. The strides parameter was also changed for the next instance of MaxPooling2D increasing it from the default (1,1) to (2,2). Per the TensorFlow tutorial, strides is a tuple or list of 2 integers that specifies the gap of convolution along the height and width of the dimensional object[7]. A stride value of (2,2) indicates that the pooling mechanism moves two units per execution rather than one. Lastly, after dropout and flattening there is a final Dense layer of size 256- twice 64 2 times. In addition to missing the 128 filter size step, Yassine Ghouzam’s model also had a smaller filter size for the penultimate layer. It is possible that the kaggle user’s model loses some complexity here with a smaller filter size for the Dense layer, but it's likely that the model gained complexity in the second Conv2D layer section. After a dropout instance, the final layer is an output layer of size 10 with ‘softmax’ activation (similar in functionality to the code I used). One last big difference with regard to the model is the dropout parameter. Instead of differentiating the dropout parameter by initial and subsequent input, the author differentiates it by Conv2D layer section and Dense layer section[8]. Yassine Ghouzam has a dropout parameter of 0.25 for the Conv2D layers (similar to 0.20 for my input layer) but the Dense layer has a dropout instance with a dropout parameter of 0.50. As addressed in Appendix C, hidden layers can have a larger dropout parameter than input layers. The value used is greater than the values I tried (0.20, 0.30, and 0.40) but reasonable for a hidden layer, according to machinelearningmastery.com[2]. Functionally, the lower dropout rate after the Conv2D layers pairs allows the model to incorporate the training data into its learning while the larger dropout parameter following the Dense layer likely prevents significant overfitting, from Dipti Rohan at Medium[5]. In addition to a different optimizer (‘RMSprop’ instead of ‘adam’), Yassine Ghouzam had different values for epochs and batch size. From examining the figures in Section 4.2, Appendix G, and Appendix H it seems likely that the models I used would continue to improve its validation and test accuracy as the number of epochs increased. I decided to keep the number of epochs constant at 10 to not give an unfair advantage, as well as being mindful of computation time concerns. I also opted to change the batch size from 64 to 86 to match the kaggle model[8]. Here are the results as follows given the changes described above and written here: <https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6> :

*Training accuracy and validation accuracy for Appendix I Modification (kaggle framework) over 10 epochs:*

******

*Test accuracy for Appendix I Modification (kaggle framework) over 10 epochs:*

**

Examining the training/validation accuracy graph and the test accuracy value, it seems as though the model underperformed. It is around a point and a half percentage points better with regard to test accuracy for the third code modification (with more epochs), but with a drastically lower test accuracy when compared to the fourth code modification (with the same number of epochs). It is possible that Appendix I code would outperform the fourth code modification with 30 epochs (as the kaggle user’s code specified)[8]. However, it seems likely that both models would perform better and there is no reason to suspect that Appendix I code would outperform the fourth modification code, especially examining both training/validation accuracy graphs (See Section 4.2).

[J]

The kaggle user previously mentioned in Appendix I does incorporate a final change to the code- introducing data augmentation[8]. Per towardsdatascience.com, a proven way to increase the performance of a neural network is to increase the amount of training data and given restricted amounts of training data, data augmentation can be a good substitute[6]. According to Yassine Ghouzam, by utilizing data augmentation and randomly applying transformations to our training data, we can expand our dataset and prevent possible overfitting[8]. The kaggle user also incorporates learning rate reduction into fitting the second fit of the model. Per the kaggle user, the changes will lead to a changing learning rate that decreases as accuracy remains unimproved and remains high (thus lowering computation time) when the model’s accuracy is improving. When considering additional ways to improve on the model from the fourth code modification, it would be pertinent to consider incorporating data augmentation and introducing a learning rate annealer.