11/1/2021

MET CS 767 Assignment 2T Neural Nets

Gaurav Tungare

Copy the implementation [here](https://colab.research.google.com/drive/1TI76Pg5RYmlA0sTNthkqDhTjZWfIoZwn?usp=sharing) to your Google drive. Modify this code in four ways, 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 MNIST training set so that the baseline implementation leaves more percentage room for improvement. You are free to combine changes 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. Note the evaluation criteria as you respond.

Please leave the gray text and the headings unchanged and honor all plagiarism rules carefully.

# The first way I modified the code to attempt improvement

## Description of what I did and reason(s) this could be an improvement

**Added – 1 Convolution Layer, 1 Max Pool Layers**

I have changed the model to add Convolution and Max Pooling Layer ,. Convolution layer is added to give the best weight of the model while the Max polling reduces the dimensions of the feature map. The detail explanation for it is given below

Following is the changed **new architecture** of my network

O/P

Fully Connected

Flatten

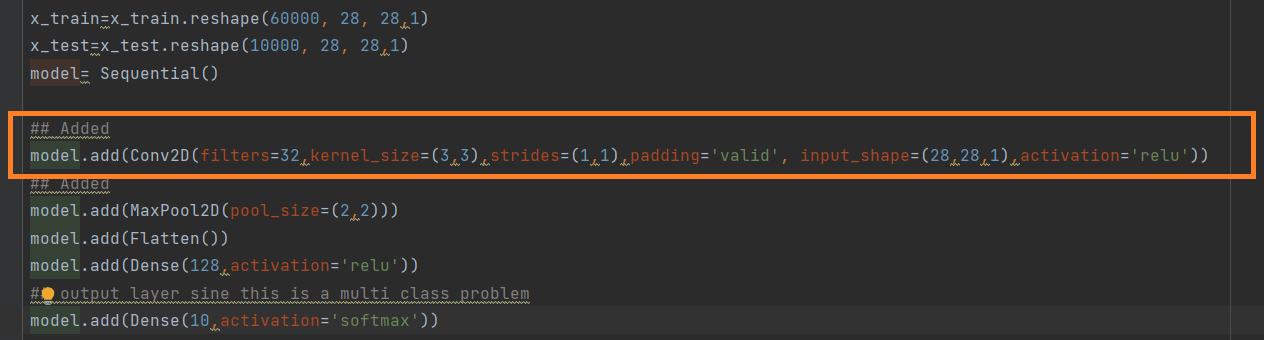
Pooling

Convolution

Input

**Explanation for Architecture**

**Added Convolution Layer**



**Why Convolution Layer is added –**

**Conv2D** is a 2D Convolution Layer, this layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs

A Convolution layer is created when we add multiple image filter to the images They layer will then be trained to figure out the best filter weight value

So with is a **filter** parameter , the more complex the data set, the larger the dataset, the more classes we are trying to classify , the more filters we should have but in our cases we only have 10 classes and the images are small so the filter value of 32 will suffice

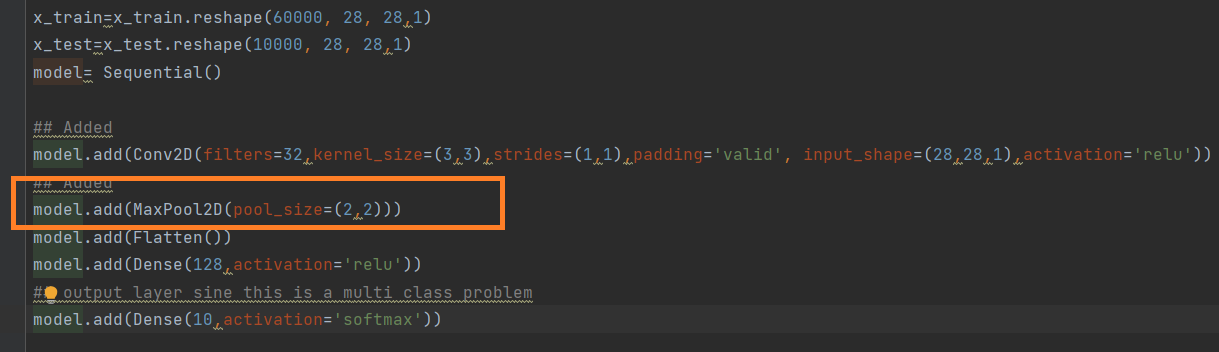
Here we are learning a total of 32 filters and then we use Max Pooling to reduce the spatial dimensions of the output volume.

**kernel\_size** parameter determines the dimensions of the kernel, I have used (3,3) , I can change it a different value like (4,4) as well

**The strides** parameter is an integer, specifying the “step” of the convolution along with the height and width of the input volume. I have used the default value (1, 1). We are stride in both x and y direction

**Padding** “valid” parameter means that the input volume is not zero-padded, generally if we were not to loose any information of the edges we can use a padding parameter as same

**Added Max Pool Layer**



Why Max Pool Layer is added–

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map

**Why Flatten Layer**

flatten() function is used to flatten the input, without affecting the batch size. A Flatten layer flattens each batch in the inputs to 1-dimension

## 

**Fully Connected Layer**

Dense implements the operation: output = activation(dot(input, kernel)) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer

I am using **Softmax** converts a vector of values to a probability distribution.

## 

## 1.2 Comparison of the result with the original output

Following is the comparison of the output. I am using couple of graphs to show case the result

## Loss Vs Val Loss

Chart, line chart

Description automatically generated

## As expected from above plot, the loss is going down.

## As you an see from above plot, the validation loss reduces but start to settle after 1.5

## Accuracy Vs Val Accuracy

Chart, line chart

Description automatically generated

As expected from above graph, the accuracy of model is going up and touches around 98.5%

As you can see from above graph, the training accuracy has reached around 99% while the validation accuracy has reached around 98.5%.

## Confusion Matrix and Classification Report

## 

## 1.3 URL of your Colab code

Attached Code 

# The second way I modified the code to attempt improvement

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

Following is the changed new architecture of my network

**Added – 2 Convolution Layer, 2 Max Pool Layers with Early Stopping**

Adding multiple Convolution and Pool layer is standard practice in the industry, adding multiple layers gives will extract more features although  Increasing Unnecessary parameters will only overfit our network.

Following is the **changed new architecture** of my network

Pooling

Convolution

Pooling

Convolution

Input

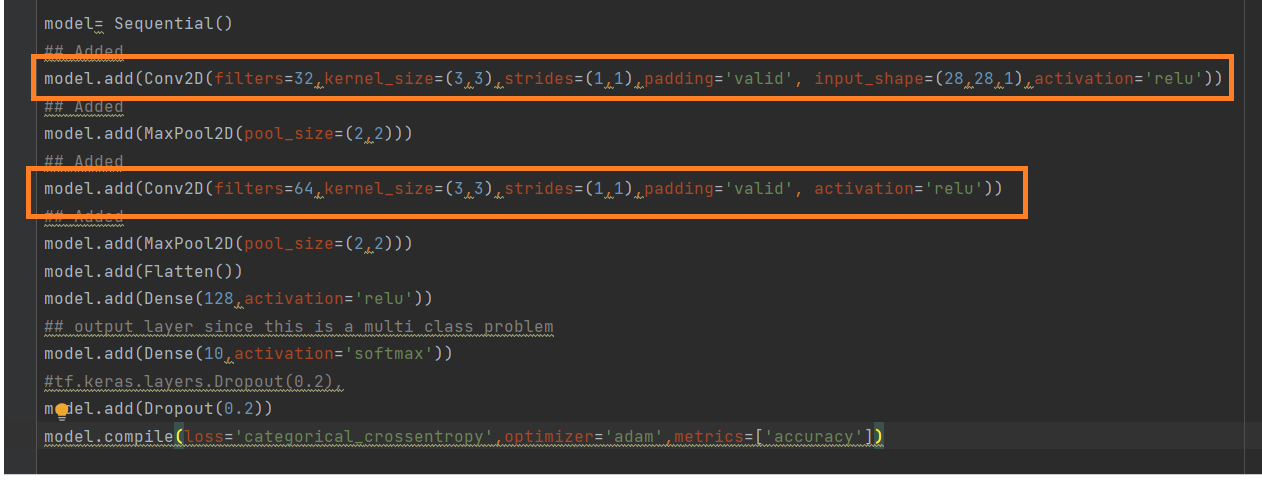
O/P

Fully Connected

Flatten

**Explanation for Architecture**

**Added 2 Convolution Layer**



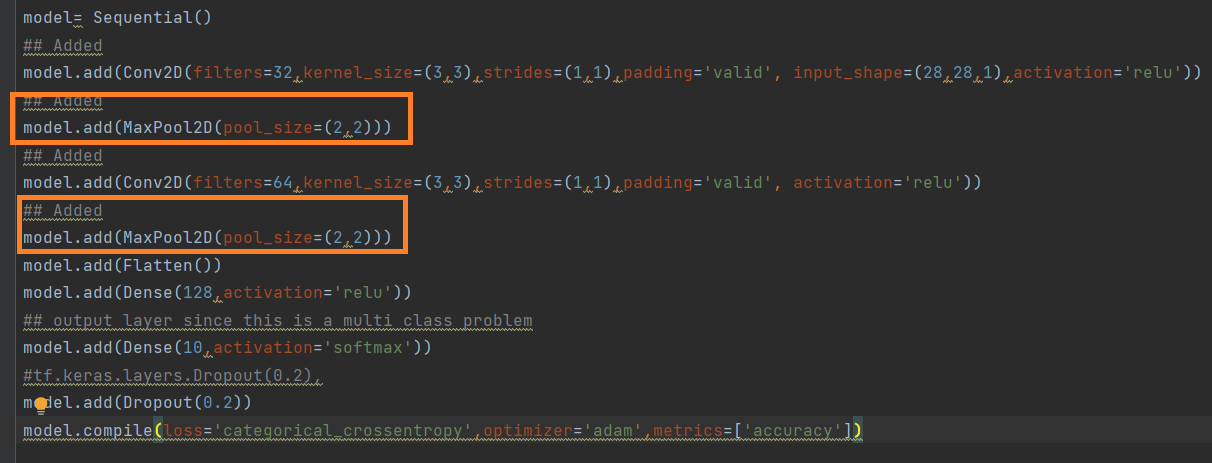
Why Convolution Layer id added –

A Convolution layer is created when we add multiple image filter to the images They layer will then be trained to figure out the best filter weight value

So with a **filter** parameter , the more complex the data set, the larger the dataset, the more classes we are trying to classify , the more filters we should have but in our cases we only have 10 classes and the images are small so the filter value of 32 will suffice

Here we are learning a total of 32 filters and second one with 64 and then we use Max Pooling to reduce the spatial dimensions of the output volume.

**Added 2 Max Pool Layer**



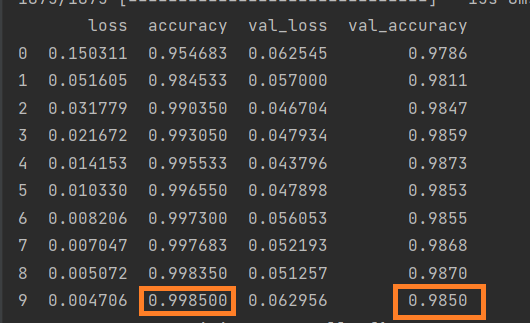
Why Max Pool Layer –

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network

## 2.2 Comparison of the result with the original output

Following is the result for accuracy – Training accuracy of around 99.95% while validation accuracy is around 98.50%

Conclusion – Adding multiple Convolution and Max Polling **has not necessarily been effective at increasing the accuracy of model**



A screenshot of a computer

Description automatically generated with medium confidence

## 2.3 URL of your Colab code



# The third way I modified the code to attempt improvement

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

**Added –Early Stopping**

**Why Early Stopping**

The **EarlyStopping callback** monitors a user-specified metric, I am using “validation loss” and ends training when it stops improving. Assuming the goal of a training is to minimize the loss. I am using Early Stopping which will monitor the “**validation\_loss**”. Early stopping will help to optimize the performance of the model

Here I am using the validation loss for early stopping.

The option for Early Stopping comes under **“callbacks”**

The parameter input to EarlyStopping function is **monitor,** we can pass different parameter value to monitor the loss

Input

Pooling

Convolution

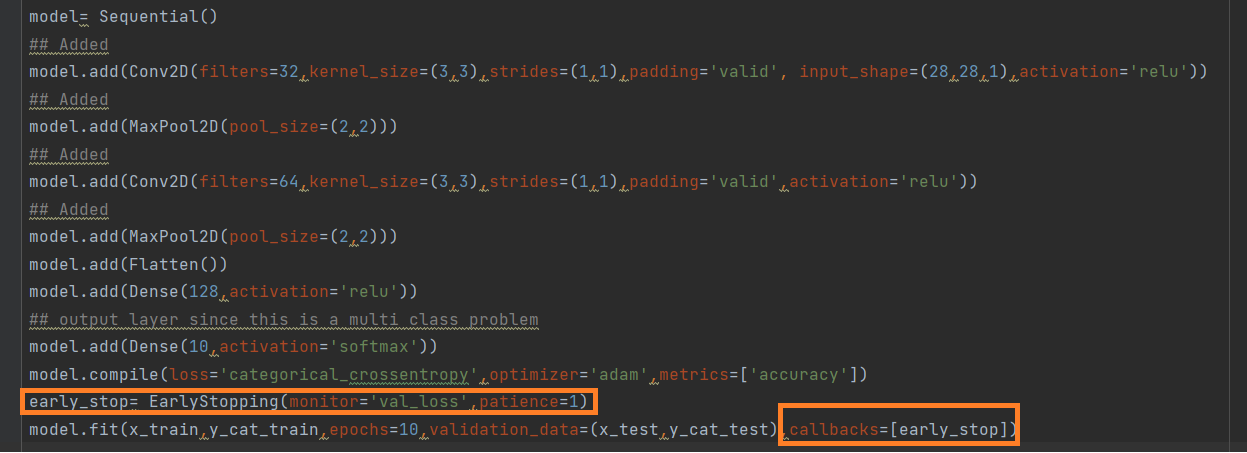
Pooling

Convolution

Flatten

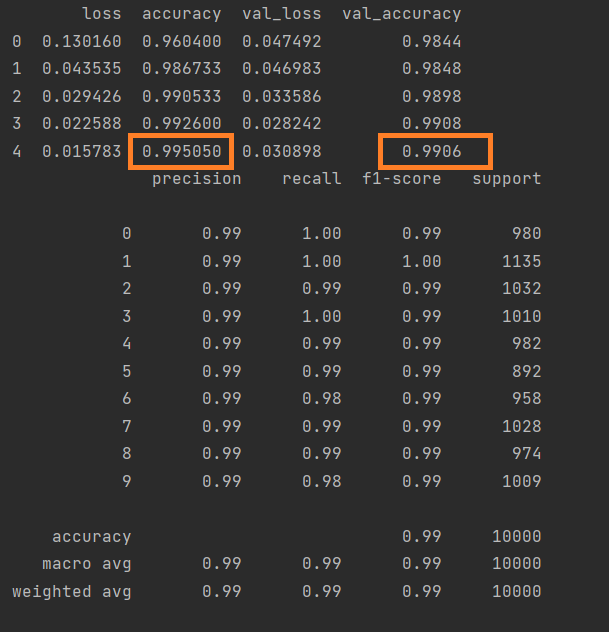
Output

Fully Connected



## 3.2 Comparison of the result with the original output

Please see the matrix below for model evaluation



Conclusion – **Early stopping is very effective and accuracy of the model on the test set has definitely improved.** The validation accuracy has almost reached 99.06% which is very good estimate of the model .

Please find the plot showing the same for this model Chart, line chart

Description automatically generated

## 3.3 URL of your Colab code

## 

# The fourth way I modified the code to attempt improvement

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

Following are the things which I have changed/added in the code, What I have changed

**Changed –Optimizer to SGD**

**Changed –epoch=15**

**Added – Dense Layer**

**Why SCD**

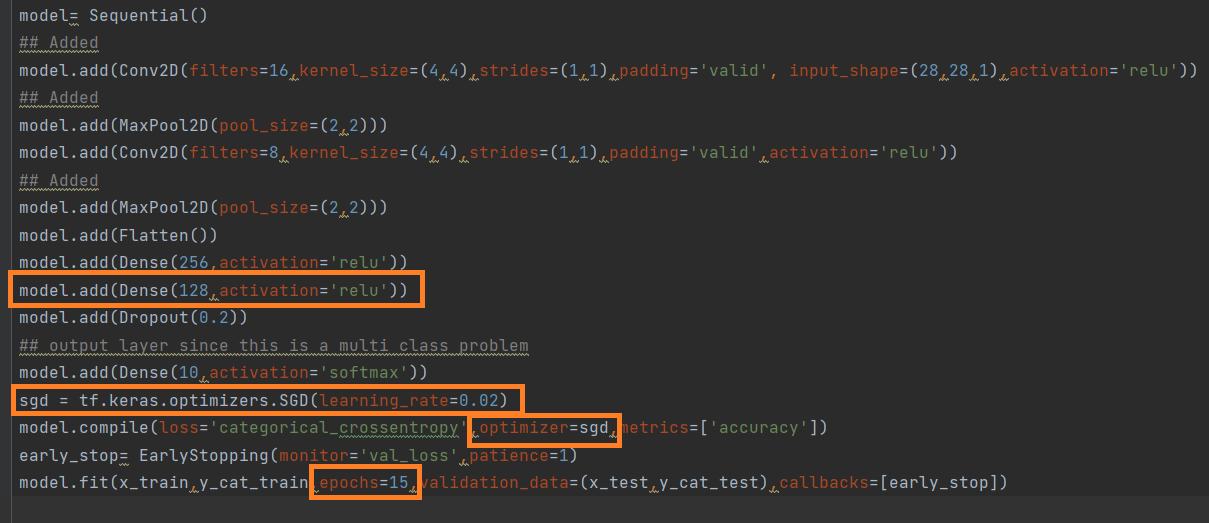
Gradient descent (with momentum) optimizer is one of the optimizer which I have learnt in the previous class, it one of the most widely use optimizer and I wanted to try out the tenserflow code with SGC to measure the performance of the model.   
I have used a **learning rate** parameter which adjust the gradient parameters , in my model I have used a learning rate of 0.02

**Why Dense Layer**

I have used 2 Dense Layer to make the model more resilient with one Layer256 Units while other with 128 units .

The output Dense layer has 10 units and the softmax activation function.

Changed –epoch to 15 to give the model enough iteration for learning

\

## 4.2 Comparison of the result with the original output

## Observation - The model is giving a accuracy of 98.57% on the validation set. This accuracy is still less than my previous model but considering that I am using SGD, I would say the model is performing pretty good

## 

## 4.3 URL of your Colab code



# 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 APA format, but other formats are fine if they clearly distinguish your work from work of others in your response. Each oh your references should occur within the text; so for example [1] should occur below *and* within the body of your response at the relevant location. You can include specific sections of the notes and the textbooks.

I have used following reference below

[1] For Tenser Flow Activation

[Module: tf.keras.activations  |  TensorFlow Core v2.7.0](https://www.tensorflow.org/api_docs/python/tf/keras/activations)

[2] For Tensor Flow Optimizer

[tf.keras.optimizers.Adam  |  TensorFlow Core v2.7.0](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam)

3 . SGD

[SGD (keras.io)](https://keras.io/api/optimizers/sgd/)

# Evaluation

