CSCI 5922 Fall 2022–Lab 2

Abulitibu Tuguluke

September 29, 2022

1 Influence of Regularization

Methods

In this lab, we used CIFAR10 small images classification dataset[1], which consists of 50,000 32x32 color training images and 10,000 test images, labeled over 10 categories : airplane automobile bird cat deer dog frog horse ship truck. In order to create a separate validation set, we split the test set evenly, and dropped half of the training data, so that the ratio would be: train/validation/test(25000/5000/5000= 5/1/1), which normalized to 72/14/14. The hardware we used is: There are 3 base architectures we used in, along with 3

Amazon SageMaker G4 instance						
Instance Size	vCPUs	Instance Memory (GiB)	GPUs-T4	Network Bandwidth (Gbps)	Instance Storage (GB)	EBS Bandwidth (Gbps)
ml.g4dn.xlarge	4	16	1	Up to 25	1 x 125 NVMe SSD	Up to 3.5

regularizations+ no regularization, there are:

models	architecture	no regularization	L2	Dropout	Batch normalization
1	convolution1 → max pooling1 → convolution1 → max pooling2 → flatten → FC dense → output	no regularization	L2	Dropout	Batch normalization
2	convolution→ max pooling → flatten → FC dense → output	no regularization	L2	Dropout	Batch normalization
3	convolution1→ max pooling → convolution2→ flatten → FC dense → output	no regularization	L2	Dropout	Batch normalization

We chose the fixed hyperparameters so that they stay the same in each run, they are: 10 epoch for each run (20 for the best model for the demo), stochastic gradient descent optimizer, batch size as 64, pooling and stride sizes are 2×2, dropput rate is 0.1[2].
Model 1 is used in MNIST lab demo, we added here just to prove it will not work well for more complicated non digits images, and we can clearly see in the next section it is the case. Model 2 has more than 1 a million parameters to learn. As opposed to model 1's 48970, and is clearly having better performance. The reason for us to add model 3 at the last minute is to justify the theory of adding one more CNN layer would result in less parameter learning, yet same accuracy, we will demonstrate that as well in the next section.

Results

In our experiment, the result for model 1 is not ideal, as shown in Table 1, L2 and Dropout regularization did not improve the accuracy, also they double the run time, yet the batch normalization out-performed the

[1] `https://keras.io/api/datasets/cifar10/`
[2] 0.5 dropput rate did not work well for our experiment, contrary to most of the literature that recommend that rate.

rest, with validation accuracy almost hitting 60%, while still keeping it fast.
Model 2 2 and 3 3 with no regularization also performed better than model 1, with training accuracy approaching 80%, and validation 40%, we also can see the trend of over fitting in their L2 and batch regularization, which shows divergence between training and validation accuracy. The same can be said about Model 3 with Dropout. In my opinion Model 3 with Batch normalization performed the best (also considering the run time), hence we test it on test set while combining training and validation set, and Figure 1 is what we got, with test accuracy almost hitting 60%, with total run time of 1 minute.

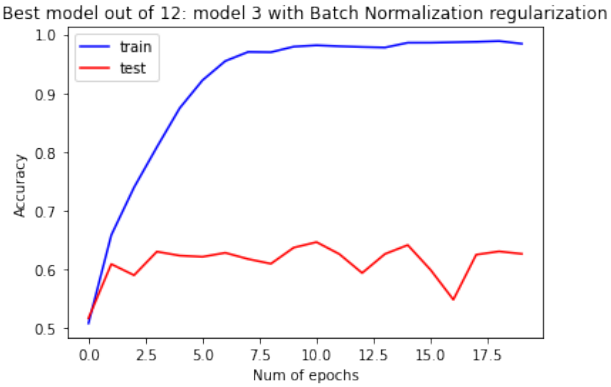


Figure 1: Model 3 with Batch normalization.

Analysis

We conducted this lab to study the effect of 3 different regularization techniques effect on 3 difference architectures. Batch normalization allows us to use much higher learning rates, which resulting in increases the speed at which networks train, hence it out-performed the rests, even in model 1. The very reason model 1 is not ideal (accuracy rate approaching only 10%) is due to the fact of not enough parameters for a larger image set with more complex feature, sure it did well for image of numeric since it is relatively easy to learn. Due to the nature of convolution layers where weights are shared, there will be much fewer parameters, it may not be enough for us to training a perfect model without some other techniques.
By adding one Fully Connected Layer (1.4 million parameters) in our model 2, we already saw impressive improvement without even a regularization, reaching 70% fir training within 10 epoch. L2 takes longer to train is due to the fact it is adding penalty for the learning, and the result is better than no regularization. One thing I can not explain is the overfitting looking plot for both model 2 and 3's L2, the regularization is designed to prevent this particular phenomenon, a further study needed to be conducted in the future[3].
The purpose of model 3 of adding a Convolution layer on model 2, is to see the reduction of parameters (from 1.4 to 1.1 Millions) having effect on both accuracy and time, and the answer is the same while training time 'shrink-ed' from 0.35 Minute to 0.29, this may not seem big due to the fact of better GPU machine, but it sure will be different tuning on CPUs. I believe adding a FC layer too will improve the result, due to my limited schedule, I will leave it to the future work. One take away is FC layer is more expensive yet performance better than simple convolution.
The effect of combing training and validation set for best model (3+ batch norm) is longer training time, 1 min, as in Figure 1, it is due to large training data point, the test accuracy did not improve much from validation thanks to the diverse similarity between val and test(split from the same source).

[3]I did run all the training data and the result is still the same, so we should exclude the possibility of not enough data to represent the feature learning.

Table 1: Model 1

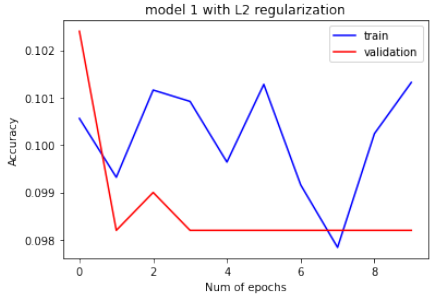
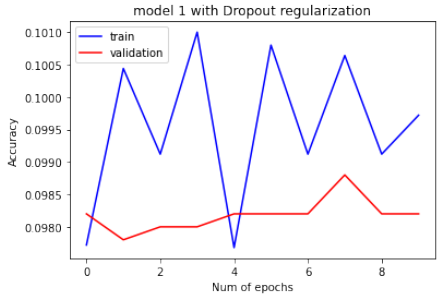
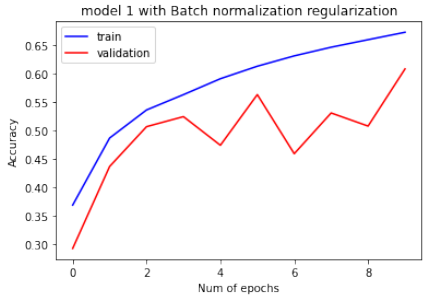
Time	Accuracy plot	Model
0.36 min		Model: "sequential" Layer (type) — Output Shape — Param # conv2d (Conv2D) — (None, 32, 32, 64) — 1792 max_pooling2d (MaxPooling2D) — (None, 16, 16, 64) — 0 conv2d_1 (Conv2D) — (None, 8, 8, 64) — 36928 max_pooling2d_1 (MaxPooling2 — (None, 4, 4, 64) — 0 flatten (Flatten) — (None, 1024) — 0 dense (Dense) — (None, 10) — 10250 Total params: 48,970 Trainable params: 48,970 Non-trainable params: 0
0.7 min		Model: "sequential_1" Layer (type) — Output Shape — Param # conv2d_2 (Conv2D) — (None, 32, 32, 64) — 1792 max_pooling2d_2 (MaxPooling2 — (None, 16, 16, 64) — 0 conv2d_3 (Conv2D) — (None, 8, 8, 64) — 36928 max_pooling2d_3 (MaxPooling2 — (None, 4, 4, 64) — 0 flatten_1 (Flatten) — (None, 1024) — 0 dense_1 (Dense) — (None, 10) — 10250 Total params: 48,970 Trainable params: 48,970 Non-trainable params: 0
0.7 min		Model: "sequential_2" Layer (type) — Output Shape — Param # conv2d_4 (Conv2D) — (None, 32, 32, 64) — 1792 max_pooling2d_4 (MaxPooling2 — (None, 16, 16, 64) — 0 dropout (Dropout) — (None, 16, 16, 64) — 0 conv2d_5 (Conv2D) — (None, 8, 8, 64) — 36928 max_pooling2d_5 (MaxPooling2 — (None, 4, 4, 64) — 0 flatten_2 (Flatten) — (None, 1024) — 0 dropout_1 (Dropout) — (None, 1024) — 0 dense_2 (Dense) — (None, 10) — 10250 Total params: 48,970 Trainable params: 48,970 Non-trainable params: 0
0.35 min		Model: "sequential_3" Layer (type) — Output Shape — Param # conv2d_6 (Conv2D) — (None, 32, 32, 64) — 1792 max_pooling2d_6 (MaxPooling2 — (None, 16, 16, 64) — 0 conv2d_7 (Conv2D) — (None, 8, 8, 64) — 36928 max_pooling2d_7 (MaxPooling2 — (None, 4, 4, 64) — 0 batch_normalization (BatchNo — (None, 4, 4, 64) — 256 flatten_3 (Flatten) — (None, 1024) — 0 batch_normalization_1 (Batch — (None, 1024) — 4096 dense_3 (Dense) — (None, 10) — 10250 Total params: 53,322 Trainable params: 51,146 Non-trainable params: 2,176

Table 2: Model 2

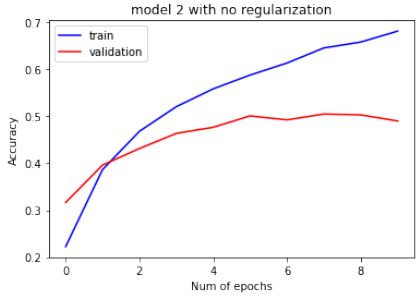
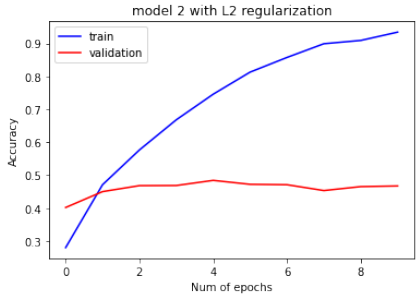
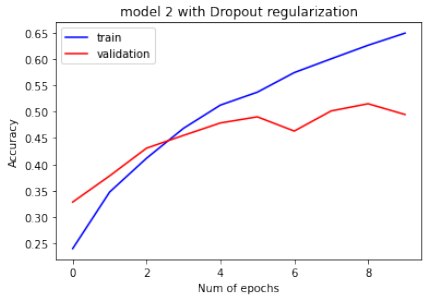
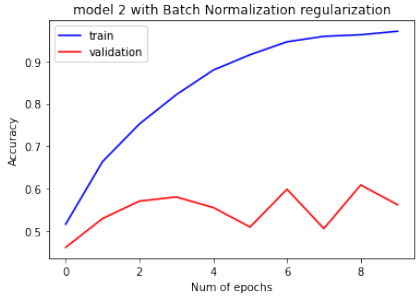
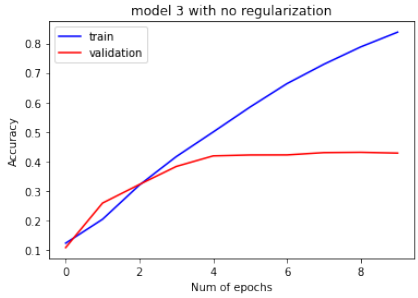
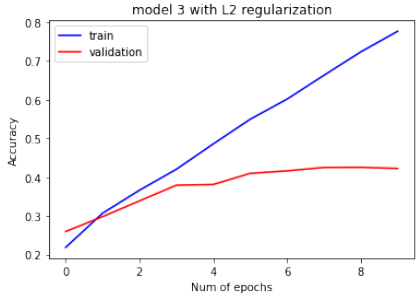
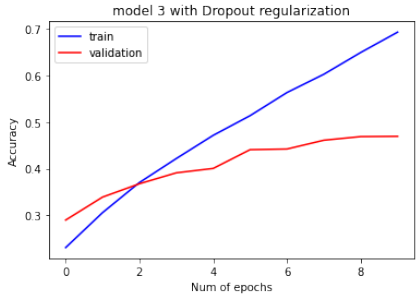
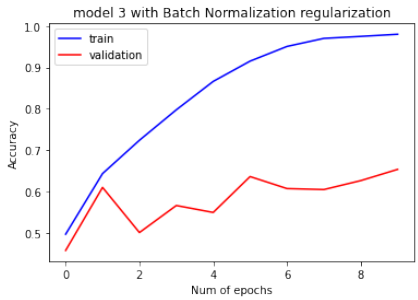
Time	Accuracy plot	Model
0.36 min		Model: "sequential_4" Layer (type) — Output Shape — Param # conv2d_8 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_8 (MaxPooling2 — (None, 15, 15, 64) — 0 flatten_4 (Flatten) — (None, 14400) — 0 dense_4 (Dense) — (None, 100) — 1440100 dense_5 (Dense) — (None, 10) — 1010 Total params: 1,442,902 Trainable params: 1,442,902 Non-trainable params: 0
0.7 min		Model: "sequential_5" Layer (type) — Output Shape — Param # conv2d_9 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_9 (MaxPooling2 — (None, 15, 15, 64) — 0 flatten_5 (Flatten) — (None, 14400) — 0 dense_6 (Dense) — (None, 100) — 1440100 dense_7 (Dense) — (None, 10) — 1010 Total params: 1,442,902 Trainable params: 1,442,902 Non-trainable params: 0
0.7 min		Model: "sequential_6" Layer (type) — Output Shape — Param # conv2d_10 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_10 (MaxPooling — (None, 15, 15, 64) — 0 dropout_2 (Dropout) — (None, 15, 15, 64) — 0 flatten_6 (Flatten) — (None, 14400) — 0 dense_8 (Dense) — (None, 100) — 1440100 dropout_3 (Dropout) — (None, 100) — 0 dense_9 (Dense) — (None, 10) — 1010 Total params: 1,442,902 Trainable params: 1,442,902 Non-trainable params: 0
0.35 min		Model: "sequential_7" Layer (type) — Output Shape — Param # conv2d_11 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_11 (MaxPooling — (None, 15, 15, 64) — 0 batch_normalization_2 (Batch — (None, 15, 15, 64) — 256 flatten_7 (Flatten) — (None, 14400) — 0 dense_10 (Dense) — (None, 100) — 1440100 batch_normalization_3 (Batch — (None, 100) — 400 dense_11 (Dense) — (None, 10) — 1010 Total params: 1,443,558 Trainable params: 1,443,230 Non-trainable params: 328

Table 3: Model 3

Time	Accuracy plot	Model
0.36 min		Model: "sequential_6" Layer (type) — Output Shape — Param # conv2d_8 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_6 (MaxPooling2 — (None, 15, 15, 64) — 0 conv2d_9 (Conv2D) — (None, 13, 13, 64) — 36928 flatten_6 (Flatten) — (None, 10816) — 0 dense_12 (Dense) — (None, 100) — 1081700 dense_13 (Dense) — (None, 10) — 1010 Total params: 1,121,430 Trainable params: 1,121,430 Non-trainable params: 0
0.7 min		Model: "sequential_7" Layer (type) — Output Shape — Param # conv2d_10 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_7 (MaxPooling2 — (None, 15, 15, 64) — 0 conv2d_11 (Conv2D) — (None, 13, 13, 64) — 36928 flatten_7 (Flatten) — (None, 10816) — 0 dense_14 (Dense) — (None, 100) — 1081700 dense_15 (Dense) — (None, 10) — 1010 Total params: 1,121,430 Trainable params: 1,121,430 Non-trainable params: 0
0.7 min		Model: "sequential_8" Layer (type) — Output Shape — Param # conv2d_12 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_8 (MaxPooling2 — (None, 15, 15, 64) — 0 conv2d_13 (Conv2D) — (None, 13, 13, 64) — 36928 dropout_10 (Dropout) — (None, 13, 13, 64) — 0 flatten_8 (Flatten) — (None, 10816) — 0 dense_16 (Dense) — (None, 100) — 1081700 dropout_11 (Dropout) — (None, 100) — 0 dense_17 (Dense) — (None, 10) — 1010 Total params: 1,121,430 Trainable params: 1,121,430 Non-trainable params: 0
0.29 min		Model: "sequential_9" Layer (type) — Output Shape — Param # conv2d_14 (Conv2D) — (None, 30, 30, 64) — 1792 max_pooling2d_9 (MaxPooling2 — (None, 15, 15, 64) — 0 conv2d_15 (Conv2D) — (None, 13, 13, 64) — 36928 batch_normalization (BatchNo — (None, 13, 13, 64) — 256 flatten_9 (Flatten) — (None, 10816) — 0 dense_18 (Dense) — (None, 100) — 1081700 batch_normalization_1 (Batch — (None, 100) — 400 dense_19 (Dense) — (None, 10) — 1010 Total params: 1,122,086 Trainable params: 1,121,758 Non-trainable params: 328

2 Interpreting CNN Representations

Methods

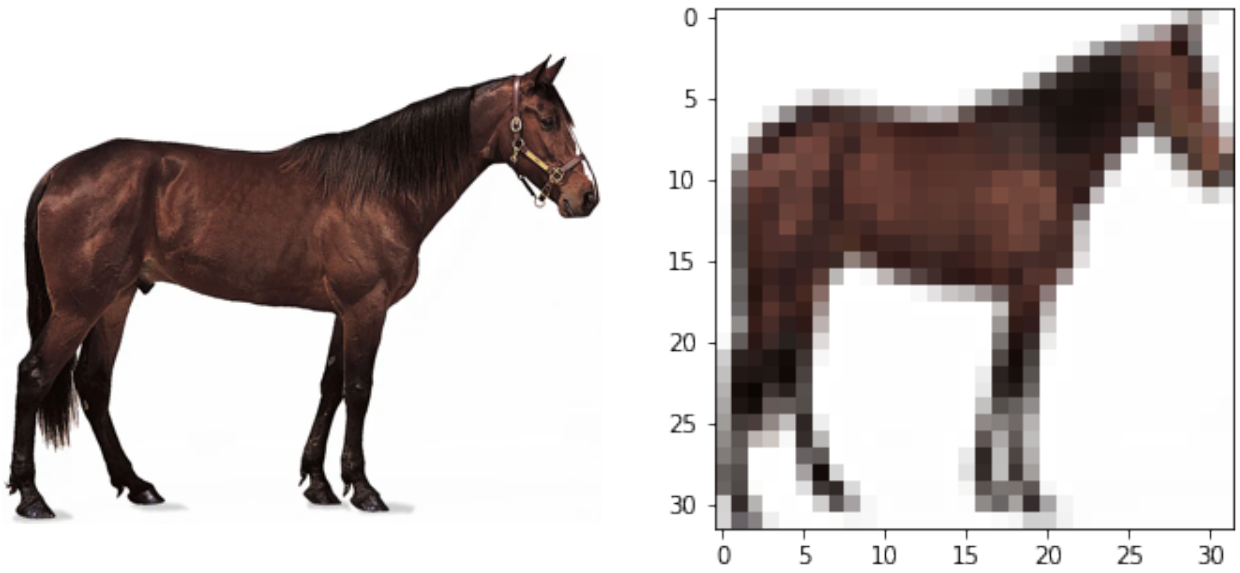


Figure 2: Images for obs.

in this experiments, we apply the technique of visualizing the feature maps of the first 4 layers from our best model: 3rd with batch normalization. The image[4] we used is shown in Figure 2, we reside the original left one to 32×32 on the right to fit our model. The layers are as follow (we only investigate the first four):

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 13, 13, 64)        36928
_________________________________________________________________
batch_normalization_8 (Batch (None, 13, 13, 64)        256
_________________________________________________________________
```

We then plot the 64 filter of each layer and observe the results shown in Figure 3

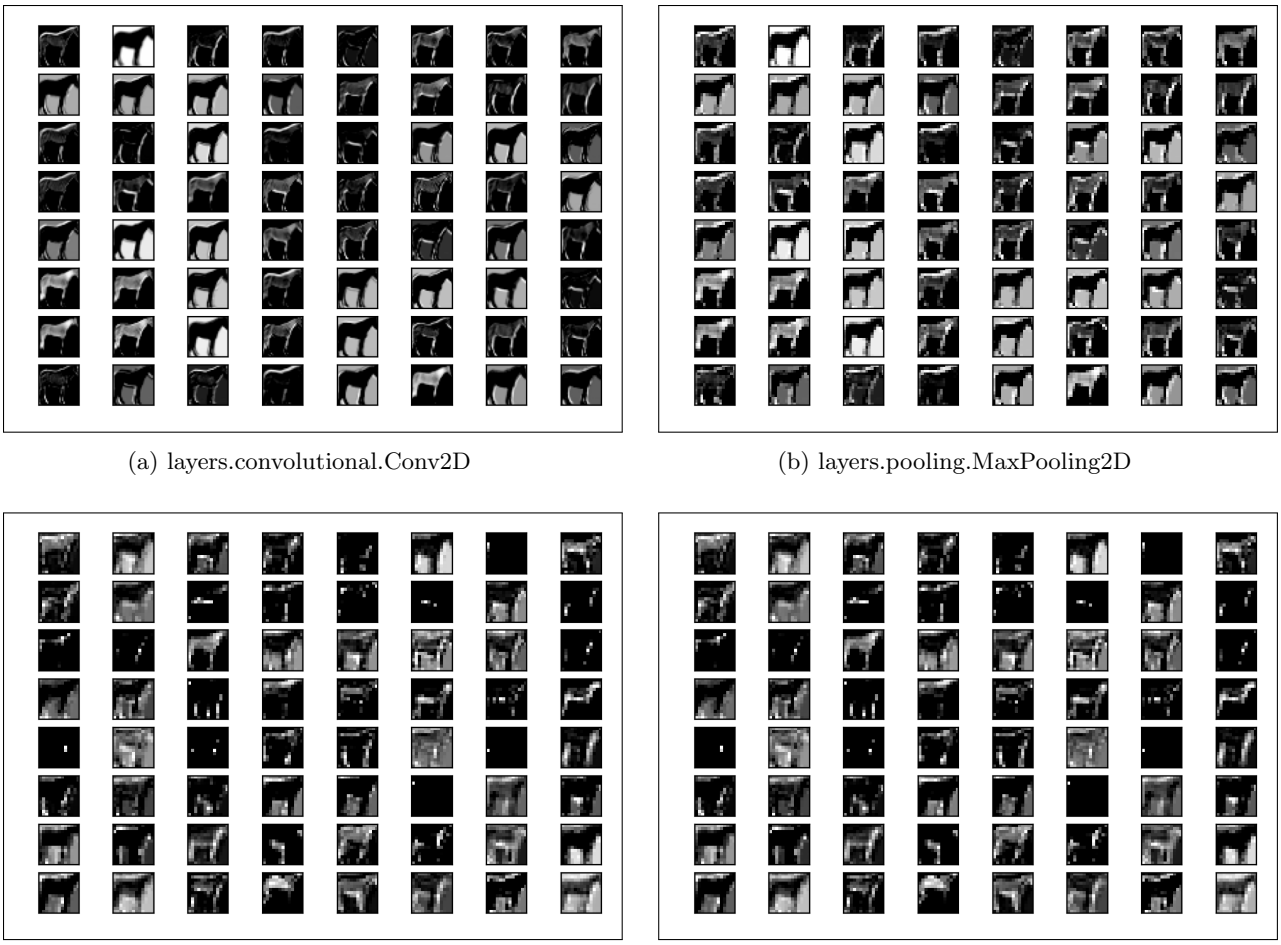
Results

There are no numerical solutions or accuracy plots in this experiment, only visualizations of each layer's feature map, and the result is shown Figure 3, where we can clearly see the different between first 3 layers. The batch norm has less parameters (256) than the second convolution later(36928), yet the output are the same size, so we can not see the big difference between these two. Next, we will try to analyze what we see.

Analysis

Part (a) in Figure 3 shows us exactly what each filter is learning, location (1,2) is the overall 'shape' of the horse, and we can see (1,8) has horse tail in it. This due to how the filter was design in each. The matrix

[4] https://cdn.britannica.com/96/1296-050-4A65097D/gelding-bay-coat.jpg



(a) layers.convolutional.Conv2D

(b) layers.pooling.MaxPooling2D

(c) layers.convolutional.Conv2D

(d) layers.normalization.BatchNormalization

Figure 3: feature map visulization on 4 layers

operator is mapping each pixel while 'cherry-picking' the part it needs to mapped into the next layer.
For part (b) max pooling layer, we can see the images are less clear , yet the attentions are the same as layer one, this is due to pooling is changing the size of 30×30 into 15×15. Without it, it will take longer time to train the model. Part (c) of second conv layer again learning different feature than first one, if we compare location (8,2), we noticed the shades are different, again, thanks to linear mapping, this time with more parameters (36928). Figure 4 are the last 8 3×3 filter of first conv and second, if they look the same, then the part(a) and part(c) would be no much difference, but they obviously not (linear mapping to separate vector space), hence the attention.
Part(d) batch normalization layer is re-scaling the image, hence the look-a-like plot.

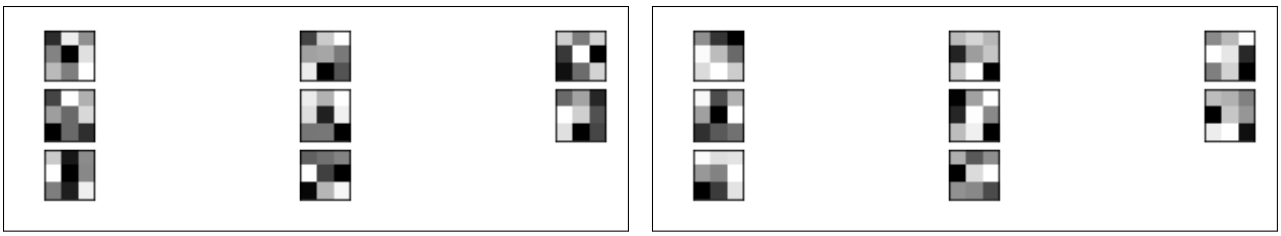


Figure 4: The last 8 filters of conv1 vs conv2.