CSC2516: PA #2

Due on Feb 28

Roger Grosse, Jimmy Ba

Thomas Hollis

Problem A.1

Describe the model `RegressionCNN`. How many convolution layers does it have? What are the filter sizes and number of filters at each layer? Construct a table or draw a diagram.

Solution

Using the values set in the final cell, the `RegressionCNN` model has 6 convolutional layers with the following filter sizes and filter numbers:

Layer Name	Filter Size	Number Of Filters
downconv1	$3 \times 3 \times 1$	32
downconv2	$3 \times 3 \times 32$	64
rfconv	$3 \times 3 \times 64$	64
upconv1	$3 \times 3 \times 64$	32
upconv2	$3 \times 3 \times 32$	3
finalconv	$3 \times 3 \times 3$	3

Problem A.2

Run all the notebook cells in `colour_regression.ipynb` on Colab (No coding involved). You will train a CNN, and generate some images showing validation outputs. How many epochs are we training the CNN model in the given setting?

Solution

The `colour_regression.ipynb` cells were all run successfully.
The number of epochs we are training the CNN model in the given setting is: 25.

Problem A.3

Re-train a couple of new models using a different number of training epochs. You may train each new models in a new code cell by copying and modifying the code from the last notebook cell. Comment on how the results (output images, training loss) change as we increase or decrease the number of epochs.

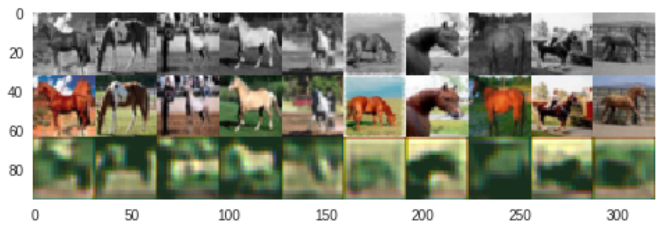
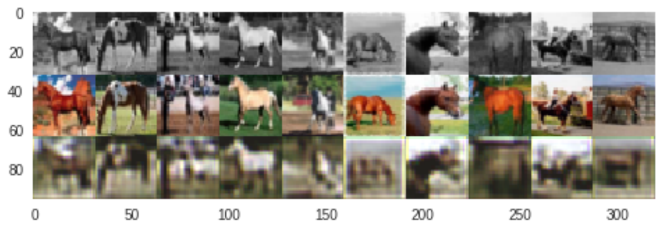
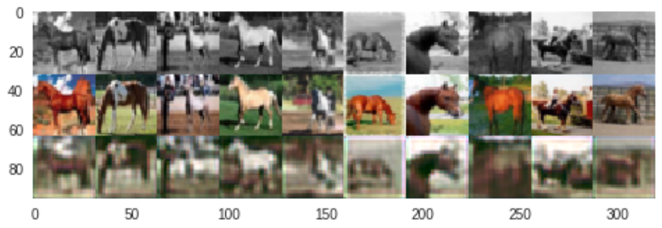
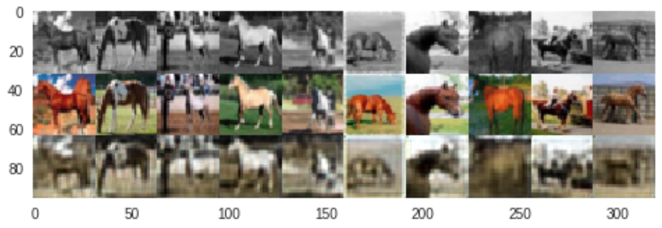
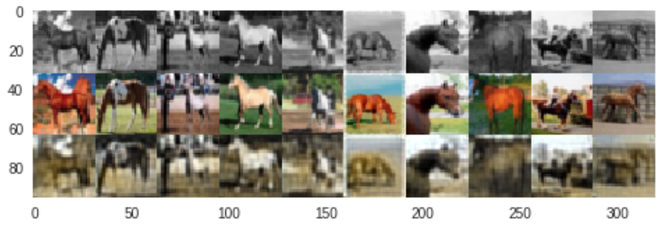
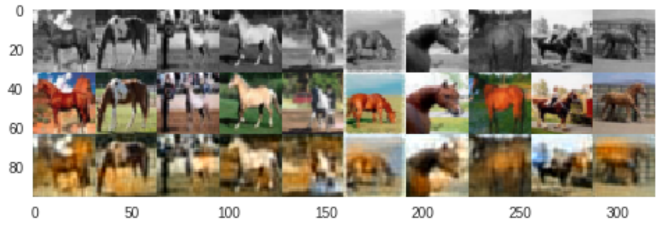
Solution

By running the training procedure using different epochs (1 to 1000 in 7 steps with 1 cell per step) we can see the change in the output images. Similarly, in the last cell, by running the training procedure over 1000 epochs we can see the trend in training and validation loss. These loss trends are summarised in the following table:

Number of Epochs	Training Loss	Validation Loss
1	0.1080	0.1330
5	0.0115	0.0114
10	0.0094	0.0095
50	0.0060	0.0061
100	0.0052	0.0064
500	0.0040	0.0068
1000	0.0037	0.0123

From the table on the previous page we can clearly see that the CNN's performance continues to improve from epoch 1 to epoch 50 (approximately trend continues to epoch 80). Indeed, the CNN starts to overfit after approximately epoch 80. This can be seen as the training loss continues to decrease while the validation loss increases thereafter for epochs 100, 500 and 1000.

As for the change in images, we can observe the following:

The above images show the output when running over 1, 5, 10, 50, 100 and 1000 epochs respectively. These images show that for an epoch of 1 the CNN can barely infer a different colour for the horses and for their surroundings. Indeed, due to the inherent stochasticity, the inferred colour (here green) changes from run to run. With subsequent epochs up to epoch 50 approximately, as training error and validation error both decrease, the colour inference is increasingly realistic and coherent with training data and its labels. Beyond epoch 50 as the CNN starts to overfit, the inferred colours get increasingly more unrealistic as validation error begins to rise. This is best seen in epoch 1000 where orange overlays are observed in places where there is no horse.

It is worth noting that, here, even the best colourisations fail to reproduce something close to the ground-truth colours.

Problem A.4

A colour space is a choice of mapping of colours into three-dimensional coordinates. Some colours could be close together in one colour space, but further apart in others. The RGB colour space is probably the most familiar to you, the model used in `colour_regression.py` computes squared error in RGB colour space. But, most state of the art colourization models do not use RGB colour space. How could using the RGB colour space be problematic? Your answer should relate how human perception of color is different than the squared distance. You may use the Wikipedia article on color space to help you answer the question.

Solution

Using the RGB colour space could be problematic as a unit change in the RGB colour space does not correspond to a unit change in human perceived colour. This is because human eyes have three types of cone receptors that allow us to decide what colour pieces of incoming light are. Indeed, the CIELAB colour space allows us to map this colour range according to how they are perceived by human receptor cells but no digital display can truly show such a wide colour gamut.

Therefore, because human's have a non-uniform perception of colour, using the RGB colour space to do a regression with squared error loss is bound to be problematic. Thus, when using the RGB colour space, we missing out information from certain colours that cannot be shown using the RGB gamut.

To make matters worse, optimising for squared error will result in harshly penalising outliers due to the squared cost function thus we will end up with a narrow range of similar colours (as seen by the dull colours in the images of problem A.3).

Problem A.5

How does framing colourization as a classification problem alleviate the above problem?

Solution

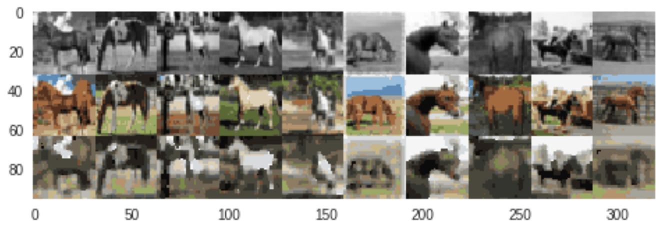
Framing colourisation as a classification problem alleviates the above problem because samples are no longer subject to the issues of regression using squared distances. This is because for classification we end up with a cross-entropy objective which uses the output probability of each given colour rather than its distance to the correct colour. Thus, we can at least expect to overcome the issue of dull colours.

Problem B.2

Run `main training loop of CNN` in `colourization.ipynb` on Colab. This will train a CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. How do the results compare to the previous regression model?

Solution

The `main training loop of CNN` in `colourization.ipynb` was run successfully after writing the missing code. The output image is as follows:



After 25 epochs using classification rather than regression, we can see that the issue of dull colours indeed is no longer present in the classification approach with limited colour palette. There remains however some more improvement to be done as the colourisation is still quite inaccurate in some areas, even with the drastically reduced colour palette.

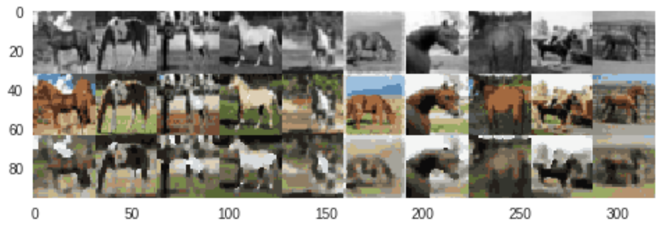
It is worth noting however that any direct comparison with the regression image of part A is an unfair one as the labels used to train the part B models are different (the classification labels are the original images but with the reduced colour palette).

Problem C.2

Train the "UNet" model for the same amount of epochs as the previous CNN and plot the training curve using a batch size of 100. How does the result compare to the previous model? Did skip connections improve the validation loss and accuracy? Did the skip connections improve the output qualitatively? How? Give at least two reasons why skip connections might improve the performance of our CNN models.

Solution

After writing the necessary code, the U-Net model was trained using a batch size of 100. This resulted in the validation accuracy increasing from 36% (CNN) to 41% (UNet). In addition, this also improved the loss from 1.7907 (CNN) to 1.6152 (UNet). Thus skip connections did indeed improve validation loss and accuracy.
As for the output result of using a UNet, we get the following colourisation:



This image is very slightly better than the regular CNN output seen in the previous question. The improvement seems to occur as a slightly more accurate colourisation. For example, there is no longer any green pixels on the face of the horse.

Skip connections might improve the performance of CNN models as:
- Information captured in the initial layers required for reconstruction that would be lost throughout the CNN may be explicitly fed into later layers thus increasing classification accuracy of final output as it will have more information.
- More parameters from skip connections means skip architectures can fit more complex training data better as information is allowed to traverse the deep neural network faster (thus optimisation is more effective) preventing the loss of some gradient information over multiple layers (vanishing gradients) which results in better classification of minute details.

Problem C.3

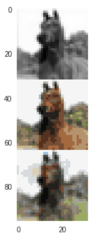
Re-train a few more "UNet" models using different mini batch sizes with a fixed number of epochs. Describe the effect of batch sizes on the training/validation loss, and the final image output.

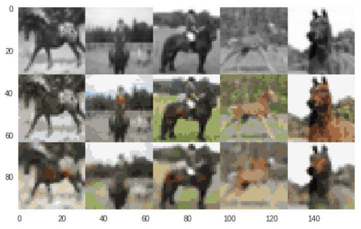
Solution

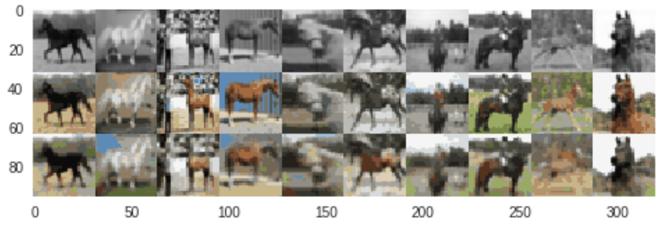
The effect of batch size on training/validation loss can be seen by running over different batch sizes (fixed over 25 epochs) as follows:

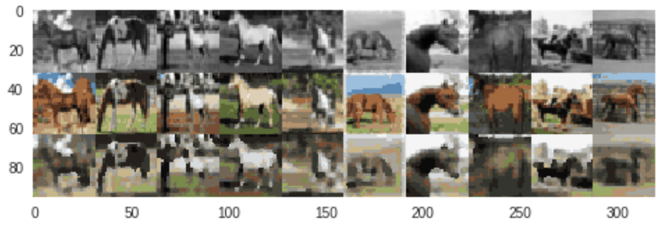
Batch size	Training Loss	Validation Loss	Accuracy
1	1.5291	1.5238	45.0%
5	1.5059	1.5012	44.0%
10	1.4385	1.4334	46.0%
100	1.6521	1.6442	40.0%
500	2.4074	2.3888	22.0%
1000	2.3741	2.3865	22.0%

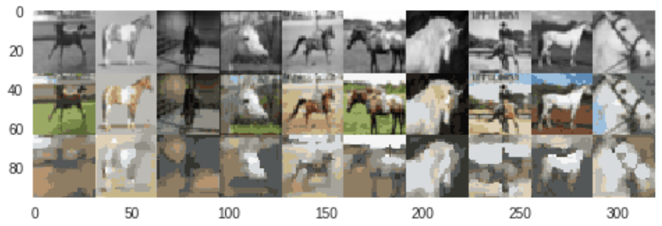
The corresponding images for these batches sizes are, respectively, as follows:

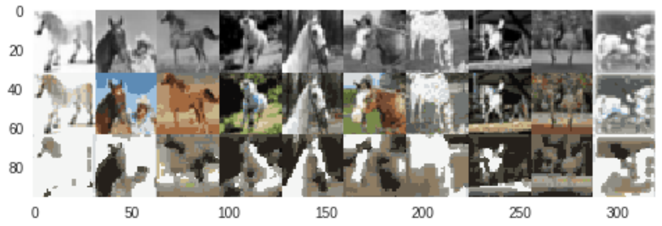












The above images show that small batch sizes (up to around 50) make consistently good colourisations but beyond batches of size 100, loss increases both for training and validation. As a consequence, colourisations get worse which results in a worse accuracy for those runs as shown in the table.

It is worth noting that smaller batch sizes are much slower to run as the GPU cannot exploit as well the parallelised optimisations.

Problem D.1

Take a look at the data process function `process`. What is the resolution difference between the downsized input image and output image?

Solution

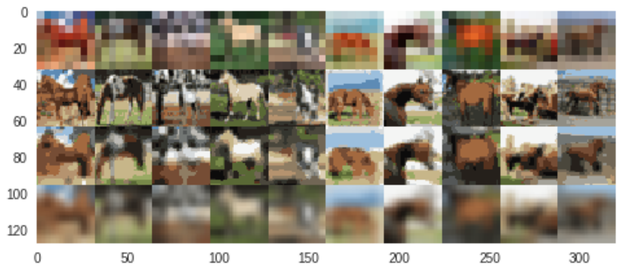
The downsized input image is of size 8×8. By contrast the resolution of the output image is 32×32. Thus the resolution difference is $d = 32 \times 32 - 8 \times 8 = 1024 - 64 = 960$. There is therefore a resolution difference of 960 pixels. Alternatively, you could say the image has been scaled down by a factor of 16 as $\frac{32 \times 32}{8 \times 8} = 16$.

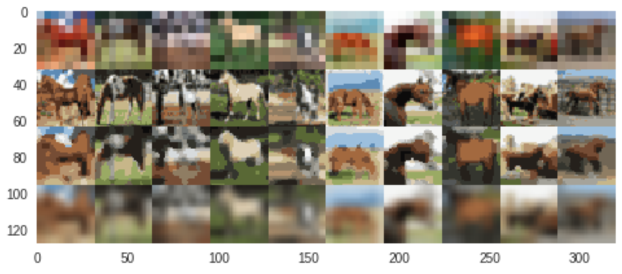
Problem D.2

Bilinear interpolation is one of the basic but widely used resampling techniques in image processing. Run `super-resolution` with both CNN and UNet. Are there any difference in the model outputs? Also, comment on how the neural network results (images from the third row) differ from the bilinear interpolation results (images from the fourth row). Give at least two reasons why conv nets are better than bilinear interpolation.

Solution

The `super-resolution` was successfully run with both CNN (train loss of 1.7083, val loss of 1.7102, val accuracy of 41%) and UNet (train loss of 1.6724, val loss of 1.6913, val accuracy of 43.0%). The results are, respectively, as follows:





There indeed are very small differences in the model outputs. As shown above, on the penultimate image, the UNet interpolates some white pixels in the middle of the horses which the CNN does not. The CNN results differ from the bilinear interpolations as they are much sharper and more accurate. Indeed, the bilinear interpolation is more blurry as the inserted pixels are direct interpolations of their four surrounding pixels. The opposite effect is true for the neural network approach as we see a beige pixel in the middle of the gross in the fourth image, for example.

Two reasons why CNNs are better than bilinear interpolation are:
- CNNs can make more informed guesses as they consider much more information in their prediction than bilinear interpolation which only considers the four surrounding pixel values.
- CNNs can represent nonlinear mappings whereas bilinear interpolation is constrained to linear mappings

Problem E.1

Visualize the activations of the CNN for a few test examples. How are the activation in the first few layers different from the later layers? You do not need to attach the output images to your writeup, only descriptions of what you see.

Solution

Activations of the first few layers are mostly specific (e.g. vertical lines of activation for the horse's legs) while activations in later layers are more high-level (e.g. activation of the whole top part of the image for the sky). This pattern seems to confirm a similar trend to those observed for the hidden activations of MNIST digit classifiers. It is however worth noting that due to the stochasticity of the code, different runs will lead to different activations.

Problem E.2

Visualize the activations of the colourization UNet for a few test examples. How do the activations differ from the CNN activations?

Solution

It seems like the UNet activations are in general much more scattered across the image rather than in particular lines, curves or areas like in the CNN activations. This makes the UNet activations look more 'noisy' and less 'focused'. I believe that this might be because the skip connections allow focus on both generic high-level features and in the same time low level details from original image. This leads to a mix of high and low level activations compared to the reference CNN activations.

Problem E.3

Visualize the activations of the super-resolution UNet for a few test examples. Describe how the activations differ from the colourization models?

Solution

The super resolution activations often focus on very specific areas, i.e. two or three pixels of the image. This occasionally leads to very sparse activations. This happens a lot less in the UNet and CNN activations. Indeed, there is also much less 'pattern' in these activations than in the colourization models. I believe that this is because the super-resolution task is much less interested with the concepts of the 'horse' and 'sky' for example.

Problem F.1

We also did not tune any hyperparameters for this assignment other than the number of epochs and batch size. What are some hyperparameters that could be tuned? List five.

Solution

Some hyperparameters that could be tuned are:
- dropout rate
- learning rate
- stride
- number of filters
- kernel size

Problem F.2

In the `RegressionCNN` model, `nn.MaxPool2d` layers are applied after `nn.ReLU` activations, comment on how the output of CNN changes if we switch the order of the max-pooling and ReLU?

Solution

The output of the CNN does not change if we switch the order of the max-pooling and ReLU because the maxpool of the ReLU is equivalent to the ReLU of the maxpool.

Problem F.3

The loss functions and the evaluation metrics in this assignment are defined at pixel-level. In general, these pixel-level measures correlate poorly with human assessment of visual quality. How can we improve the evaluation to match with human assessment better?

Solution

We can improve the evaluation to match with human assessment better by using perceptual loss functions for training rather than per-pixel loss. This is because perceptual loss functions are designed to correlate better with human assessment of visual quality compared to per-pixel loss functions, as discussed in the referenced paper.

Problem F.4

In `colourization.ipynb`, we have trained a few different image processing convolutional neural networks on input and output image size of 32x32. In the test time, the desired output size is often different than the one used in training. Describe how we can modify the trained models in this assignment to colourize test images that are larger than 32x32.

Solution

We actually do not need to modify the trained models in this assignment to colourise test images larger than 32×32 because we are dealing with a fully convolutional network.

Note that my implementation of `colourization.ipynb` is attached as a separate file.