1.1.a

```
RegressionCNN(
  (downconv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (downconv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (rfconv): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
  )
  (upconv1): Sequential(
    (0): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Upsample(scale_factor=2, mode=nearest)
  )
  (upconv2): Sequential(
    (0): Conv2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Upsample(scale_factor=2, mode=nearest)
  )
  (finalconv): MyConv2d()
)
```

It has 5 Conv layer :

Convolution	number_of_filter	kernel_size
1: Conv2d()	32	(3, 3)
2: Conv2d()	64	(3, 3)
3: Conv2d()	64	(3, 3)
4: Conv2d()	32	(3, 3)
5: Conv2d()	3	(3, 3)

1.1.b We are training for 25 epochs

1.1.c With [25,15,50] epochs, we are getting loss around [0.0096, 0.0102,0.0068] respectively. Images are having better color at 50 epochs, followed by 25 and 15 epochs.

1.1.d

- Pure blue and pure green in RGB space have very different lightness
- If RGB space corresponded to what our eyes saw, that would be a perfect hexagon. It isn't. It's warped, and the warping isn't subtle.
- Distances in the RGB colorspace doesn’t have approximate perceptual distance (how different two colors look).
- From a design point-of-view, RGB actually has some problems, as it is difficult to manipulate in a reliable manner. However, from a data point-of-view if is useful because it can scale up or down depending on how much room you have to store information. Also, it relates to the way TV and many video standards transmit information.

https://news.ycombinator.com/item?id=13882105

1.1.e

Instead of trying to predict the colors directly via a regression task, we split all the colors into bins, with a classification task," Marc Thibault, another researcher involved in the study, told TechXplore. "Formulating the problem as a classification task allows us to have better control over how colorful we want our output to look, by fine-tuning how we predict color from the output of the network.

Ref:

https://techxplore.com/news/2018-11-colorunet-deep-cnn-classification-approach.html

1.2.b

The loss of this model is more than the previous model i.e. 1.55 compared to 0.0096. The resultant images are far better than the regression model even though the regression model has less loss.

1.3.b

The loss of this model is better than the previous model i.e. 1.25 and giving much better results due to skipping connection as a flow of gradient to an earlier layer is good in the skipped network.

Yes, Skip connection improved the validation accuracy and loss.

Skip connections also improved the quality of results.

Give at least two reasons why skip connections might improve the performance of our CNN models.

- Better flow of gradient.
- Eliminating the singularities inherent in the loss landscapes of deep networks. These singularities are caused by the non-identifiability of subsets of parameters when nodes in the network either get eliminated (elimination singularities), collapse into each other (overlap singularities).
  Ref https://arxiv.org/pdf/1701.09175.pdf
- There was some information that was captured in the initial layers and was required for reconstruction during the up-sampling done using the FCN layer. If we would not have used the skip architecture that information would have been lost (or should say would have turned too abstract for it to be used further). So the information that we had in the primary layers can be fed explicitly to the later layers using the skip architecture.
- Better model convergence.
- Advantages of skip connection are they pass feature information to lower layers so using it to classify minute details becomes easier. As you do max-pooling some amount of spatial information is lost. skip connections can help increase the accuracy of classification because the final layer feature will have more information.

1.3.c

200 MiniBatch

```
Epoch [25/25], Loss: 1.3450, Time (s): 56
Epoch [25/25], Val Loss: 1.4070, Val Acc: 46.9%, Time(s): 57
```

100 MiniBatch

```
Epoch [25/25], Loss: 1.2606, Time (s): 58
```

```
Epoch [25/25], Val Loss: 1.3347, Val Acc: 49.4%, Time(s): 59
```

50 MiniBatch

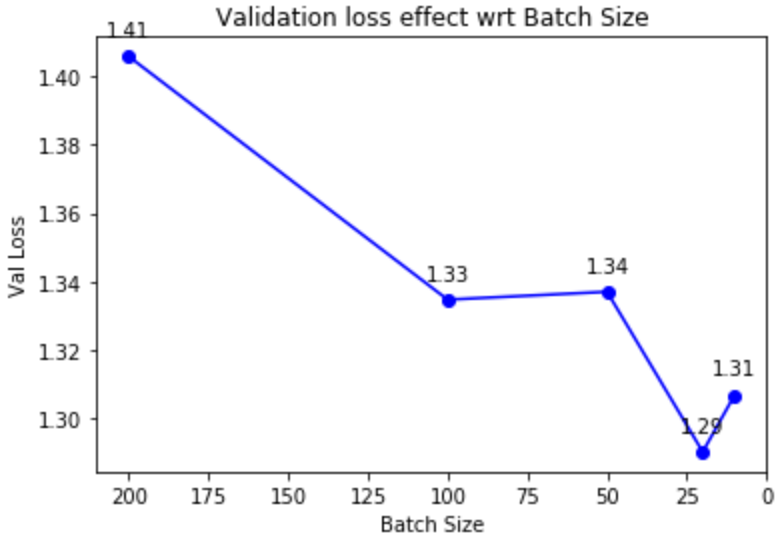
```
Epoch [25/25], Loss: 1.2194, Time (s): 71
Epoch [25/25], Val Loss: 1.3370, Val Acc: 49.5%, Time(s): 72
Validation l
```

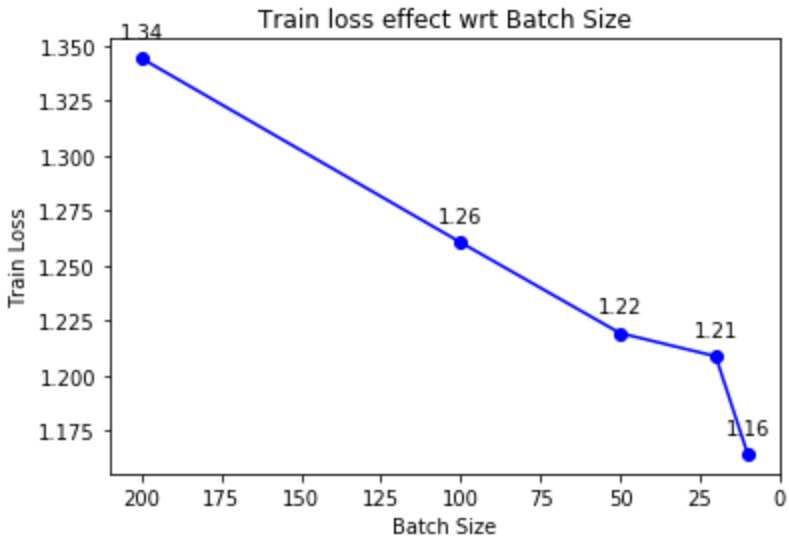
20 MiniBatch

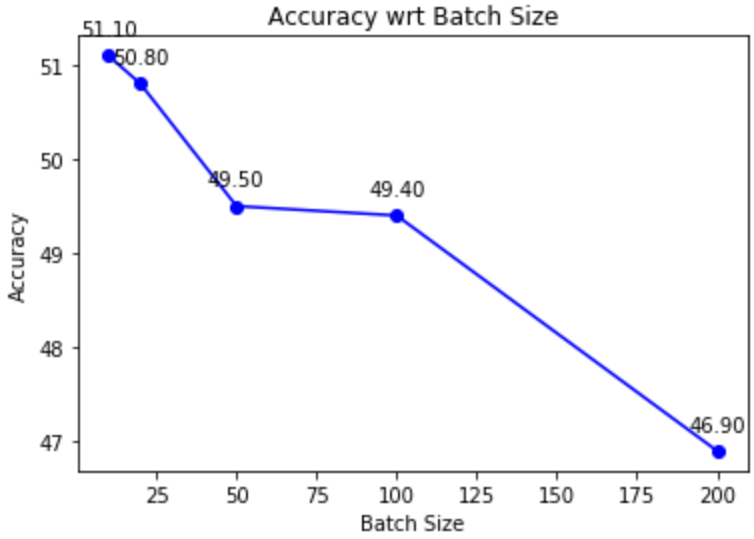
```
Epoch [25/25], Loss: 1.2089, Time (s): 113
Epoch [25/25], Val Loss: 1.2902, Val Acc: 50.8%, Time(s): 114
```

10 MiniBatch

```
Epoch [25/25], Loss: 1.1646, Time (s): 161
Epoch [25/25], Val Loss: 1.3068, Val Acc: 51.1%, Time(s): 161
```







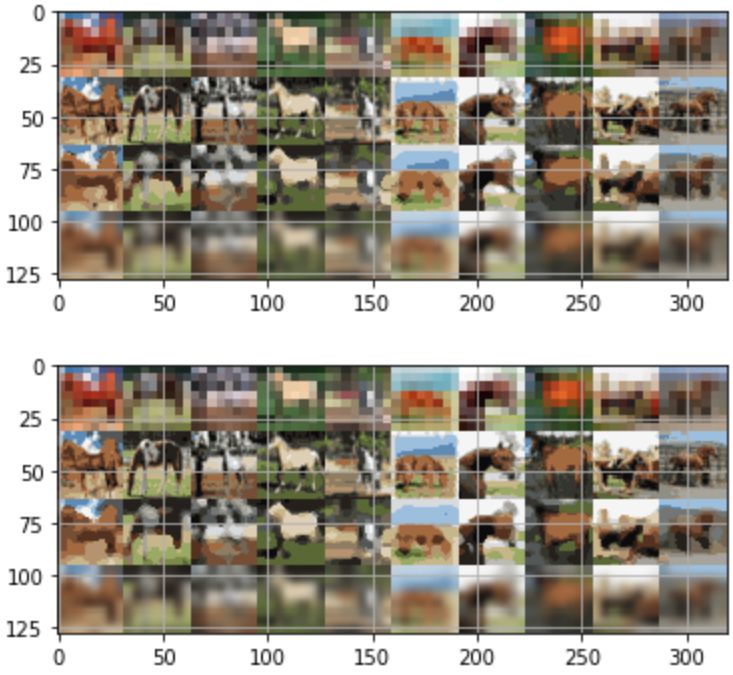
From the above graph, we can see that validation loss is decreasing till mini-batch size = 20 and on 10 mini-batch sizes is increased slightly. Training loss is decreasing as we decrease batch size while accuracy keeps on increasing.

1.4.a

The difference between the downsized input image and the output image is half of the input images.

1.4.b

UNet model is slightly better than the CNN model as we can see the improve n accuracy, the difference between two is almost the same. CNN has more noise and UNet has a higher contrast ratio. Not as smooth as UNet(sharp edges)



We can see a better effect on the UNet model than the CNN model. We can see better images than the CNN model.

We consider a convolutional neural network that directly learns an end-to-end mapping between low- and high-resolution images. These are implicitly achieved via hidden layers. Furthermore, patch extraction and aggregation are also formulated as convolutional layers, so they are involved in the optimization. The entire SR pipeline is fully obtained through learning, with little pre/postprocessing.

Its advantages of simplicity and robustness could be applied to other low-level vision problems, such as image deblurring or simultaneous SR+denoising
Ref https://arxiv.org/pdf/1501.00092.pdf

1.5.a

When we compare test0 test5, test15 we can clearly see the difference in test0 we can see that images results are like clustering/segmentation. After 5 epochs we can see better results and some of the edges that too smooth, not fine. After 15 epochs we can see better results.

1.5.b

This one is much better than the above one due to the better flow of gradient we can see a lot better results from the first layer. After 15 epochs the improvement is tremendous. Even after 1 epoch, we can see the object unlike in the above method.

1.5.c

The results are really bad compared to the colorization model. As we can see from the results that they need a lot of training to get a better result as UNET shows exciting results as compared to the CNN colorization model. This one performs worst among all. After 15 epochs we can just see the structure that too not in all the images which we can see in the above colorization model.

1.6.a

The hyperparameters that we can tune are:

- Learning Rate
- Optimizer
- The number of features maps.
- The number of layers.
- Activation Function.

With changed Learning rate (0.01):

We can see the validation loss is jittering around while the validation loss with lr = 0.001 is quite smoothly reducing. Validation accuracy with 0.001 is slightly better than 0.01. Training loss of 0.001 is more than 0.01 because of higher steps.

Epoch [25/25], Loss: 1.5579, Time (s): 57
Epoch [25/25], Val Loss: 1.5745, Val Acc: 41.6%, Time(s): 58 with lr 0.001

Epoch [25/25], Loss: 1.5182, Time (s): 57
Epoch [25/25], Val Loss: 1.5861, Val Acc: 40.8%, Time(s): 57 with lr 0.01

1.6.b

Well, max-pooling and monotonously increasing non-linearities commute. This means that MaxPool(Relu(x)) = Relu(MaxPool(x)) for any input. So the result is the same in that case. So it is technically better to first subsample through max-pooling and then apply the non-linearity (if it is costly, such as the sigmoid). In practice, we do Maxpool first then ReLU(less computation).

As for conv2D, it does *not* flip the kernel. It implements exactly the definition of convolution. This is a linear operation. In conclusion, we can save a lot of running time if we do put max-pooling before the non-linear layers like ReLu.

1.6.c

Replacing the per-pixel loss with a perceptual loss gives visually pleasing results. As optimizing perceptual loss functions based on high-level features that most humans consider. the use of perceptual loss functions allows the transfer of semantic knowledge from the loss network to the transformation network

1.6.d.

We can do upsampling to increase the size of output images. We can also increase the size by averaging the nearby pixel to increase the size of the output image.