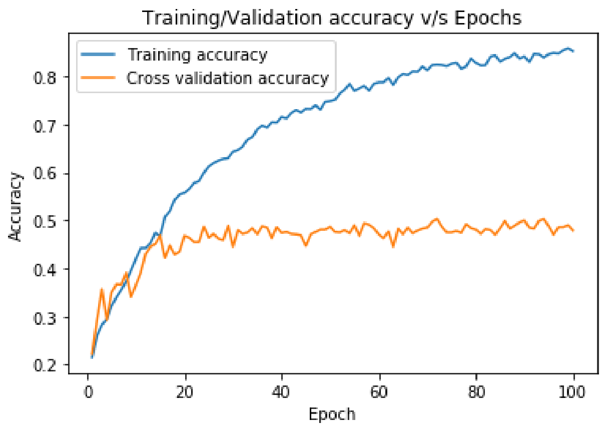
APRML : MidSem

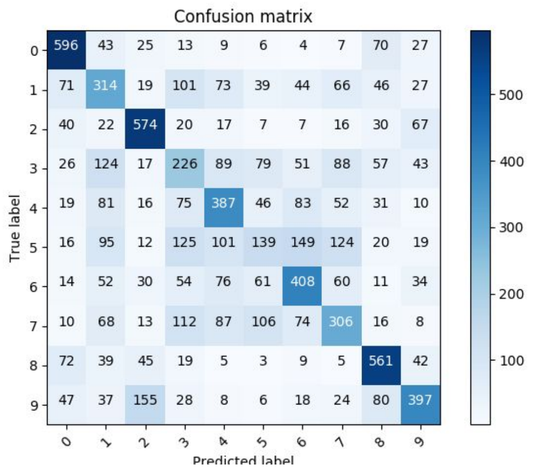
Anshuman Suri : 2014021

Note: As stated in the question, the same hyperparameters are used across different settings (100 epochs, *ADAM* optimizer, batch size of 2) across different settings.

1. For all the following parts, max-pooling, batch normalization and dropout were applied in addition to the given architecture, as otherwise the models were highly overfitting, giving near-random accuracies.

   1.1. The given CNN architecture is trained on all of the data, with a validation split

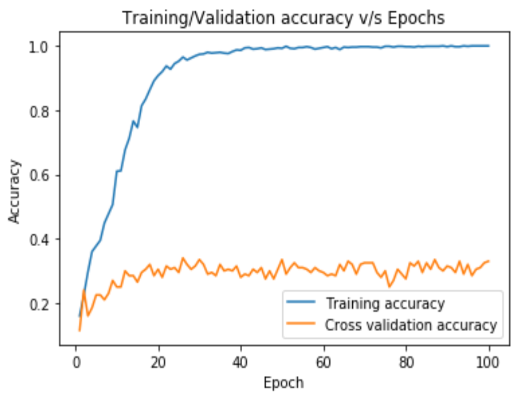




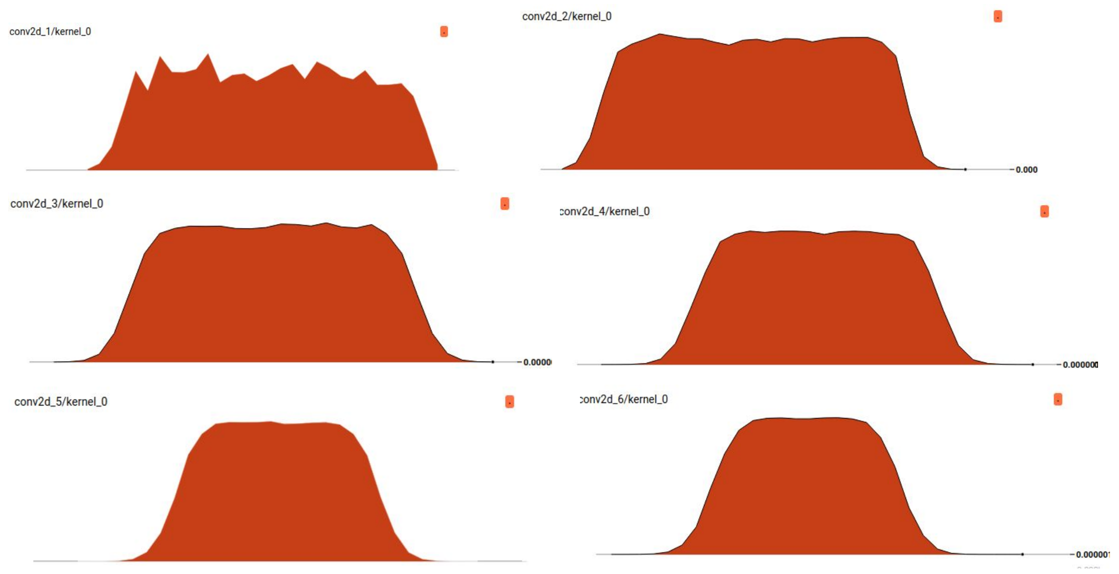
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	596	43	25	13	9	6	4	7	70	27
1	71	314	19	101	73	39	44	66	46	27
2	40	22	574	20	17	7	7	16	30	67
3	26	124	17	226	89	79	51	88	57	43
4	19	81	16	75	387	46	83	52	31	10
5	16	95	12	125	101	139	149	124	20	19
6	14	52	30	54	76	61	408	60	11	34
7	10	68	13	112	87	106	74	306	16	8
8	72	39	45	19	5	3	9	5	561	42
9	47	37	155	28	8	6	18	24	80	397

ratio of 0.2. The test accuracy achieved on this was 48%, which is quite close to the cross validation accuracy. Class-wise accuracies are plotted via a confusion matrix. Analysis of classwise accuracies shows that some classes are trained much better than some others, indicating that some classes are harder to learn than others. This could have been remedied by using a weighted error function, or having more data, or over/under-sampling.

   1.2. The training-validation accuracy curve, when trained on only one fold of the data, is as seen below. Lower accuracies can be attributed to availability of lesser data. One significant difference in the curve from the one above is the huge gap between training and validation accuracies in this case. This phenomenon is called **overfitting**. Both the models have the same architecture and thus the same power of expressibility, however this case has lesser data, leading to the model overfitting on the given data, giving training accuracies close to 100%.

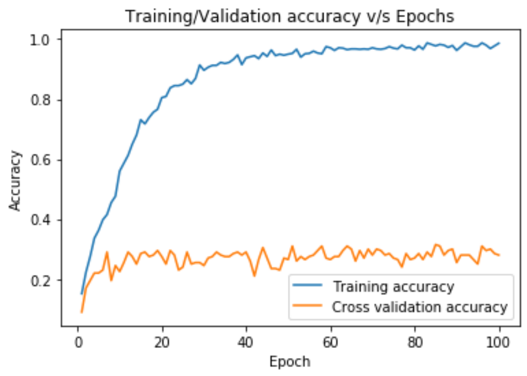


Histogram of the learned filter weights are (per convolutional layer):

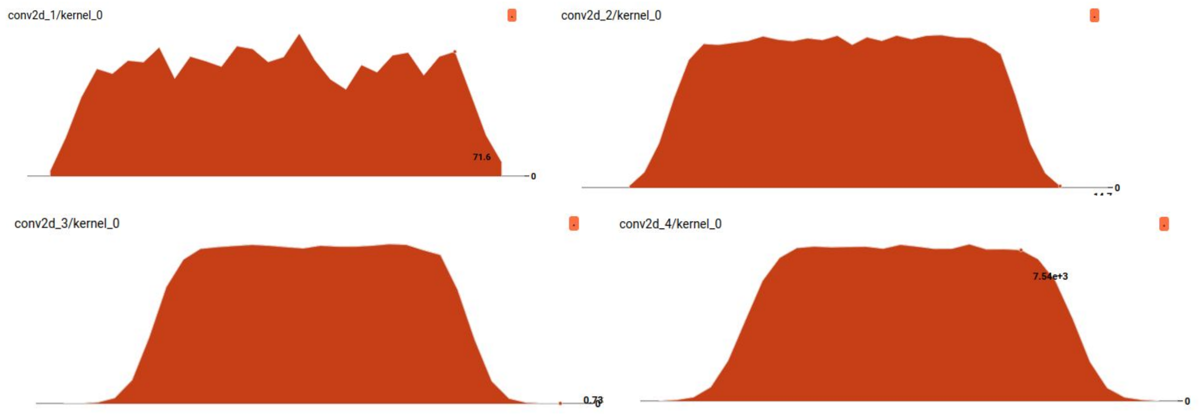


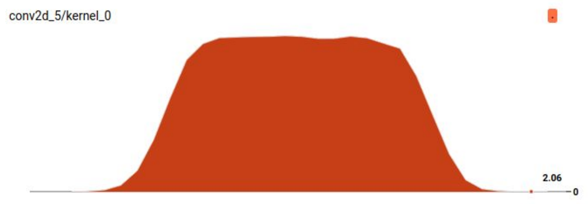
The test accuracy achieved for this setting was 35%

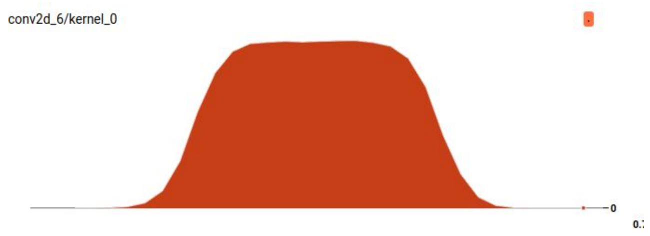
1.3. For both *L1* and *L2* regularization, alpha values of 0.01 (default) were used.



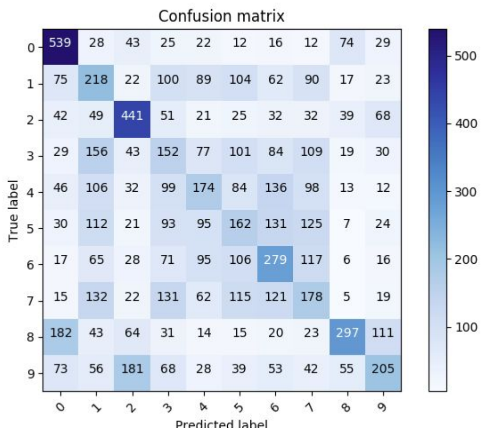
1.3.1. Even though regularization is used, it does not work out so well and is unable to prevent the model from overfitting: the training accuracy tends towards 100%, while the validation accuracy stagnates around 30-35%.
An analysis of classwise accuracies shows that the model ends up highly overfitting: one class has a great 1 v/s rest accuracy, while almost all other classes have low accuracies. Thus, in this case, *L2* regularization is unable to stop the model from overfitting.
Histograms of kernel weights for all convolutional layers:







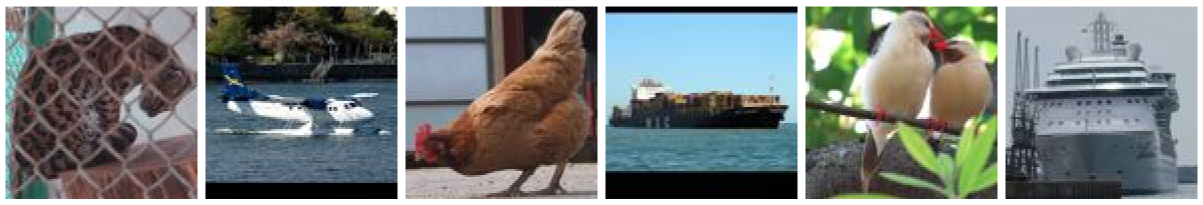
Comparison with the case of no-regularization shows that the kernel weights learnt are more spread/broadened, but nonetheless fail to enforce proper regularization on the given model. The confusion matrix for this configuration is much better than the previous case: the model does not overfit over a single class. However, the improvement is not quite significant.


Confusion matrix
True label
Predicted label
500
400
300
200
100

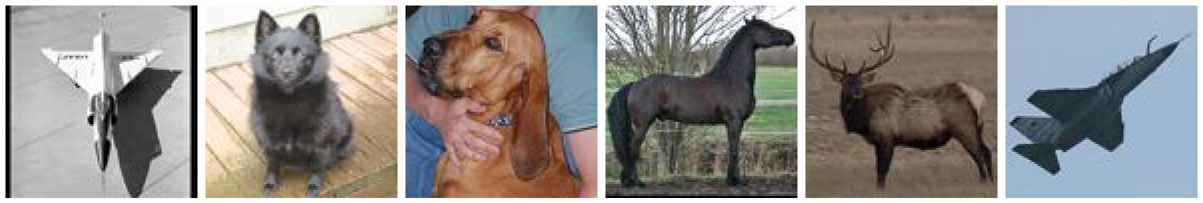
	0	1	2	3	4	5	6	7	8	9
0	539	28	43	25	22	12	16	12	74	29
1	75	218	22	100	89	104	62	90	17	23
2	42	49	441	51	21	25	32	32	39	68
3	29	156	43	152	77	101	84	109	19	30
4	46	106	32	99	174	84	136	98	13	12
5	30	112	21	93	95	162	131	125	7	24
6	17	65	28	71	95	106	279	117	6	16
7	15	132	22	131	62	115	121	178	5	19
8	182	43	64	31	14	15	20	23	297	111
9	73	56	181	68	28	39	53	42	55	205

For *L2* regularization, a testing accuracy of 33% was achieved. It is comparable to the setting of no regularization, indicating that this regularization is not of much help.

Images which were classified correctly by the regularized model and not the vanilla one are mostly of cars, ships and animals with very detailed patterns. Some example images are:

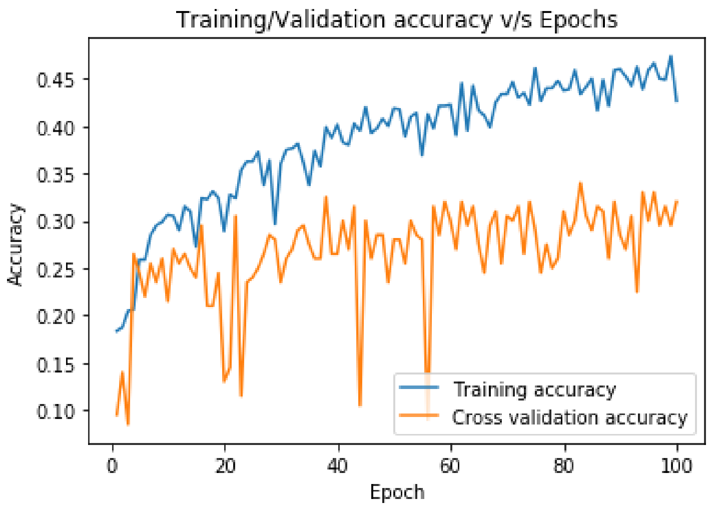


The images which were classified incorrectly by the regularized model and correctly by the vanilla one are mostly of planes and animals with plain patterns. Some example images are:



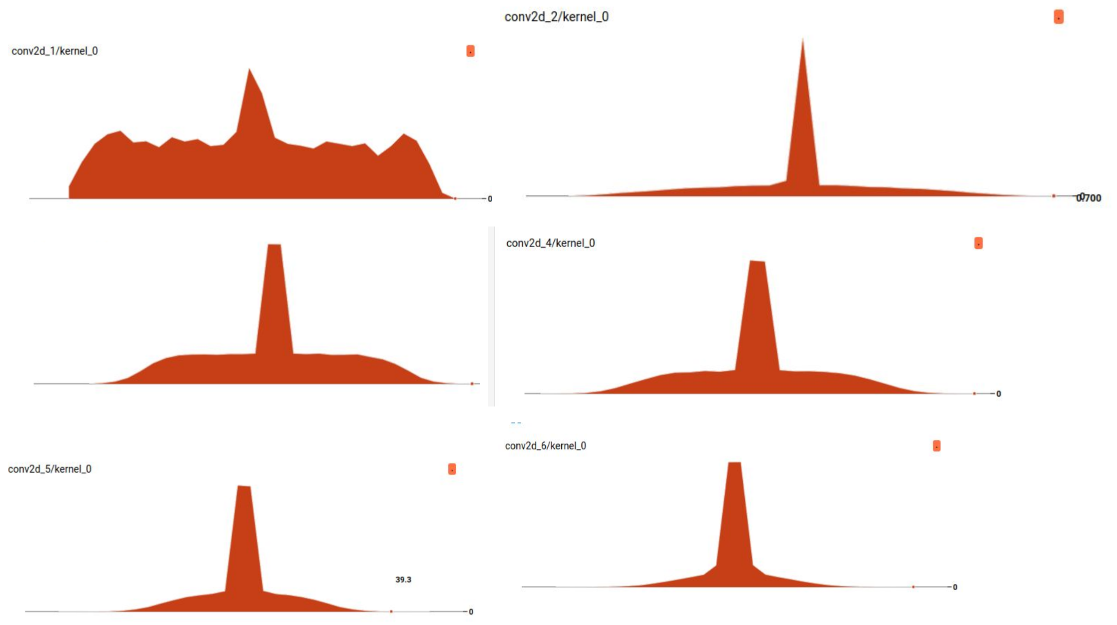
A possible explanation for this could be the model's ability to generalize getting better, thus it is better able to identify animals even when they have different patterns. However, this same constraint also seems to be hindering its ability to correctly classify subsets of some other classes.

1.3.2. Using *L1* regularization seems to be helping reduce overfitting, as is indicated by the training v/s validation accuracy curves. Though the training is not stable (peaks and downfalls happening very often), it overall seems to be increasing for both, with the gap more or less constant. This is indicative of the fact that continuing training for more epochs may have led to better accuracies (however, we were restricted to using same hyperparameters for comparison across configurations) . For the given configuration, a testing accuracy of 38% was achieved, better than the two cases.



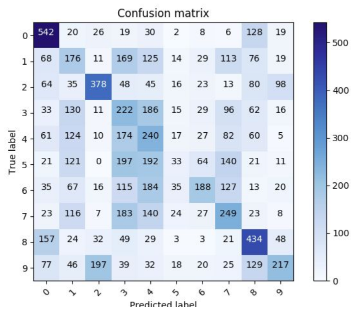
Thus, we may conclude that *L1* regularization performs better than *L2* regularization in this case. A possible explanation for this can be the sparsity enforced by *L1*, effectively reducing the density of the model and thus removing overfitting to an extent.

Histograms of kernel weights for all convolutional layers:



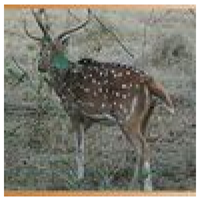
There is a significant difference between the kernel weights learnt here and the ones learnt in the previous two iterations. The weight histograms have peaks, as one one would expect because of the sparsity constraint. Weights of the first convolution too are less scattered.

The classwise accuracies are also a little better than the

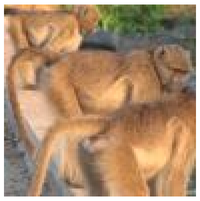


	0	1	2	3	4	5	6	7	8	9
0	542	20	26	19	30	2	8	6	128	19
1	68	176	11	169	125	14	29	113	76	19
2	64	35	378	48	45	16	23	13	80	98
3	33	130	11	222	186	15	29	96	62	16
4	61	124	10	174	240	17	27	82	60	5
5	21	121	0	197	192	33	64	140	21	11
6	35	67	16	115	184	35	188	127	13	20
7	23	116	7	183	140	24	27	249	23	8
8	157	24	32	49	29	3	3	21	434	48
9	77	46	197	39	32	18	20	25	129	217

other two cases, but nonetheless far from ideal.
Images which were classified correctly by the regularized model and not the vanilla one follow behavior similar to the case of *L2* regularization. Some example images are:

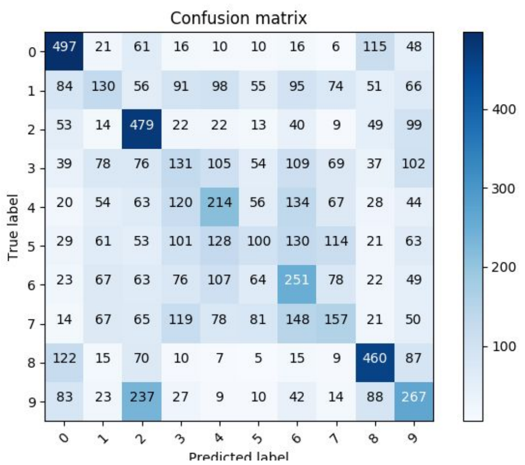


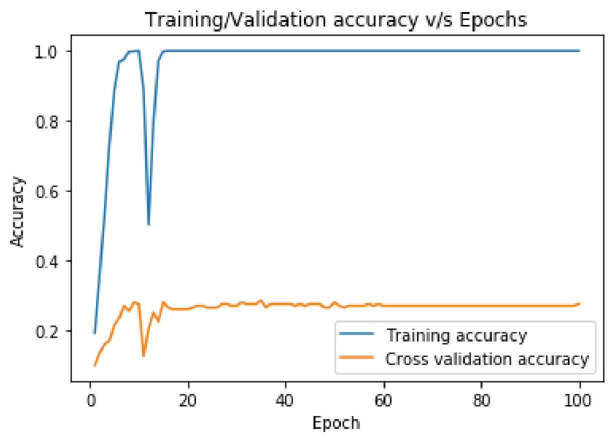
The images which were classified incorrectly by the regularized model and correctly by the vanilla one follow behavior similar to the case of *L2* regularization. Some example images are:



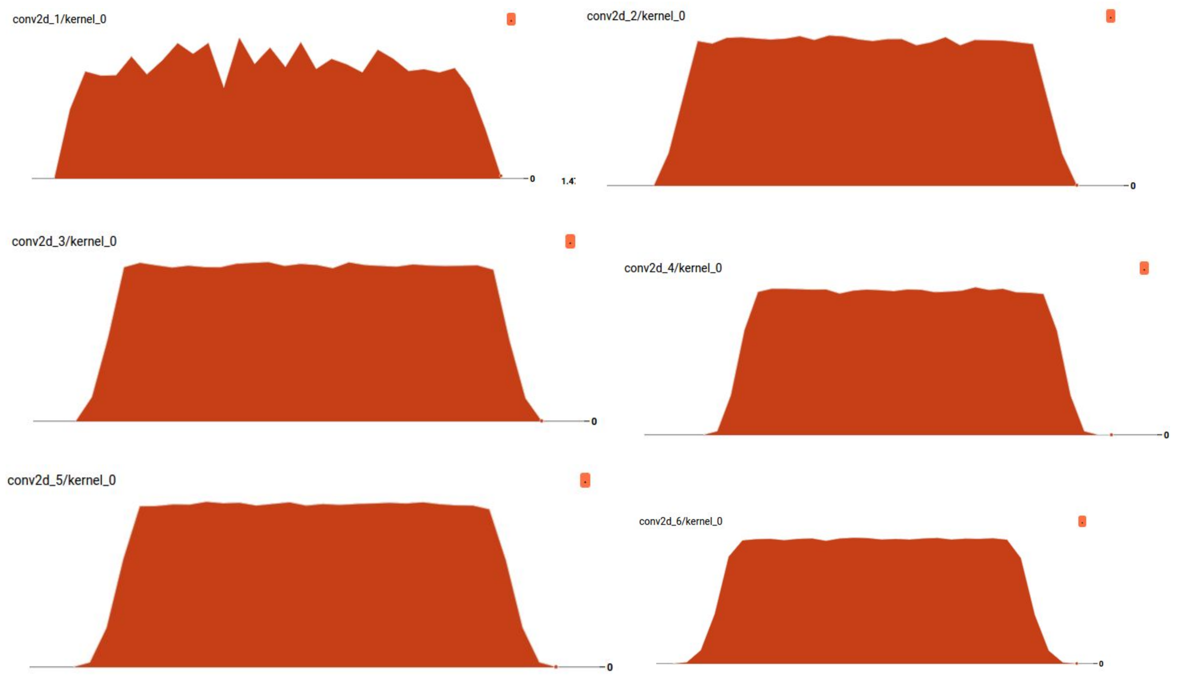
A reasoning to the case of *L2* regularization may be given for this case.

1.3.3. As per my knowledge, the following regularization technique has not been used + published in any well known conference (did not show up on searching on the Internet). The technique is data-based. For every data point, one of the 3 channels are randomly selected and set to the mean value (0.5) for that channel.





Intuition behind method: Since this process is random, and we know that data across channels is correlated to an extend, this method should help the model learn to classify even when not all channels are available, thus enforcing it to extract meaningful representations from any of the given channels. For this configuration, this layer is applied drop-channel rate of 0.6 (randomly drop one in three channels).
Histograms of the learnt kernel weights:



The filter weight histograms are like plateaus, having almost a constant distribution; this may be interpreted as it having almost similar kernel weights across channels.

Images which were classified correctly by the regularized model and not the vanilla one are almost random; there is no specific distribution. This is more proof to the fact that this kind of a regularization technique does not work well, since the images here are too simple and have features similar across all samples of that class. Some example images are:

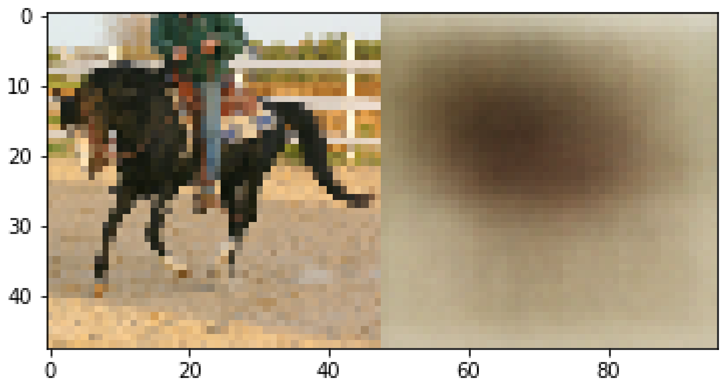
As far as images correctly classified by the vanilla model and not the regularized one are concerned, almost all classes have samples in this set. Some examples are:

2. The paper critique is include in another file, named *Critique.pdf*

3. (**Bonus**) Since the given image size was too big for creating a conventional autoencoder with the given architecture, all images were *2x* downsampled for this part. Also. all of the training

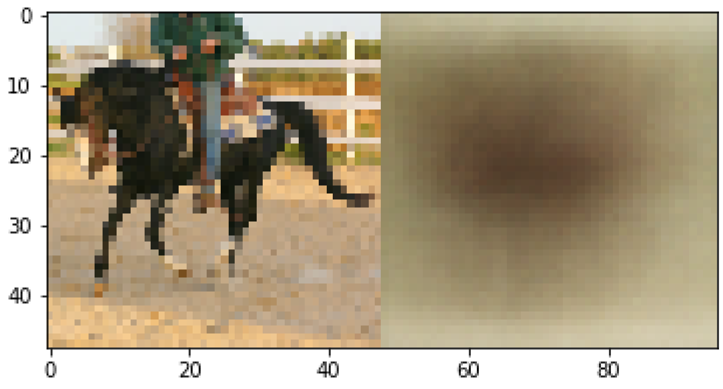
data was used for training the autoencoder as well as the MLP attached with the autoencoder.

3.1. The training process is very slow. It keeps converging on the default learning rate, but the convergence goes on even for 500+ epochs. To cap this (given the time constraint) and a fair comparison with the second model, it was trained for 200 epochs. In the end, *MSE* of 0.029 was obtained on the test set, and 0.0297 on the training set, indicating that the model has not learnt to simply learn the given images. A sample reconstruction image is shown here.



On using an MLP on the encoded representation of these images and training it for 100 epochs, a testing accuracy of 41 % was achieved, which is close to the vanilla CNN model. However, there is lesser overfitting: this is expected, as the encoding learnt is supposed to be meaningful, and thus any network trained on that encoding should perform better than a model that just looks at all of the data together.

3.2. For the stacked setting, both iterations were trained using 100 epochs each to ensure fair comparison, as the total number of epochs is same for both the settings. At the end of training, a sample reconstruction looks like the images shown here. It is not significantly better than the previous case. In fact, the final reconstruction error (after all stacks) si 0.035, higher than the previous case. This may be because each layer got fewer epochs to refine itself.



On using an MLP on the encoded representation of these images and training it for 100 epochs, a testing accuracy of 38% was achieved, which is significantly lower than all the previous models. This may be explained by the fact that the learnt representations are not very powerful as indicated by the autoencoder loss, thus having a lower classification accuracy.