Identifying Implementation Bugs in Machine Learning based Image Classifiers using Metamorphic Testing: Appendices

Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao,
R.P. Jagadeesh Chandra Bose, Neville Dubash, Sanjay Podder
Accenture Technology Labs
Bangalore, India
{anurag.dwarakanath,manish.a.ahuja,s.sikand,raghotham.m.rao,jagadeesh.c.bose,neville.dubash,sanjay.podder}@accenture.com

ABSTRACT

This document provides further material to the ISSTA 2018 publication of the same title. Appendix A contains plots of validation loss values of original (non-buggy) code of different deep learning architectures on different data-sets. Appendix B contains plots of test loss for all the mutants for MR-1 & MR-2 of the ResNet application. Appendix C gives further experimental details on the execution of the Machine Learning applications with the MRs.

KEYWORDS

Testing Machine Learning based applications, Metamorphic Testing

ACM Reference Format:
Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao, R.P. Jagadeesh Chandra Bose, Neville Dubash, Sanjay Podder. 2018. Identifying Implementation Bugs in Machine Learning based Image Classifiers using Metamorphic Testing: Appendices. In *Proceedings of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3213846.3213858

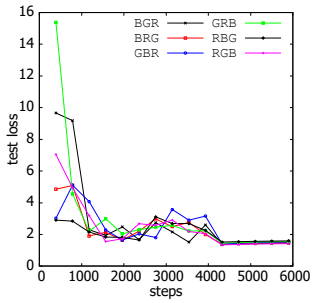
1 APPENDIX A

In this section, we will provide the results of the MR-1 (permuting the RGB order) and MR-2 (permuting the CONV Order on different data-sets and different architectures. These results are the output from original code (non-buggy). The combinations tried are shown in Table 1. The plot show that the expectation of having small variance in the validation (test) loss, due to the changes made in the input data as per MR-1 and MR-2, appears to hold. Particularly, the small variance (as measured by the maximum of the standard deviation of the loss) is not a specific property of ResNet architecture (because we see small variance across different architectures) and is not a specific property of the CIFAR-10 data-set (because we see small variance across different data-sets).

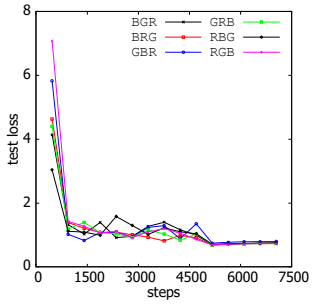
Deep Learning Architecture	Data-set	σ_{max} in validation loss due to MR-1 (permute RGB)	σ_{max} in validation loss due to MR-2 (permute CONV order)
ResNet	Cifar10	4.8	3.6
ResNet	SVHN [6]	1.4	3.3
ResNet	Kaggle Fruits [7]	0.8	2.1
ResNet	Kaggle digits [2]	1.1	0.9
AlexNet [4]	Cifar10	0.3	0.3
VGGNet [8]	Cifar10	0.1	0.1
NIN [5]	Cifar10	0.2	0.2

Table 1: Experiments conducted to validate MR-1 & MR-2

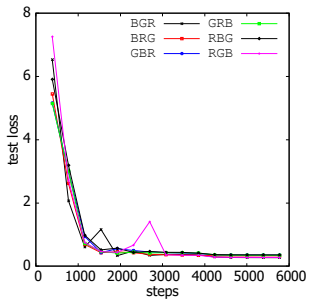
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISSTA'18, July 16–21, 2018, Amsterdam, Netherlands
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5699-2/18/07…$15.00
https://doi.org/10.1145/3213846.3213858



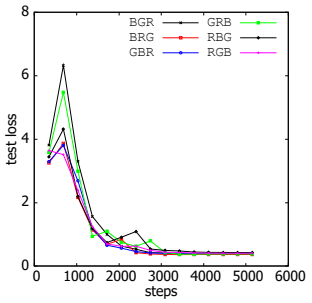
(a) MR-1 on ResNet code with CIFAR10 data



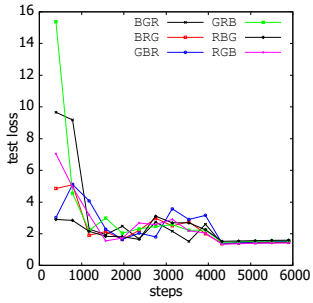
(b) MR-1 on ResNet code with SVHN data



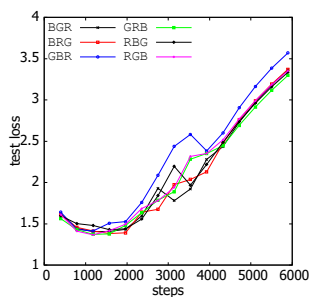
(c) MR-1 on ResNet code with Kaggle Fruit data



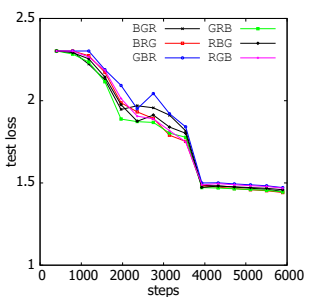
(d) MR-1 on ResNet code with Kaggle Digits data



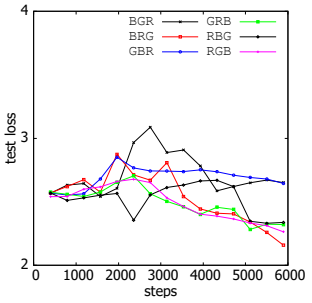
(e) MR-1 on ResNet code with CIFAR10 data



(f) MR-1 on AlexNet code with CIFAR10 data

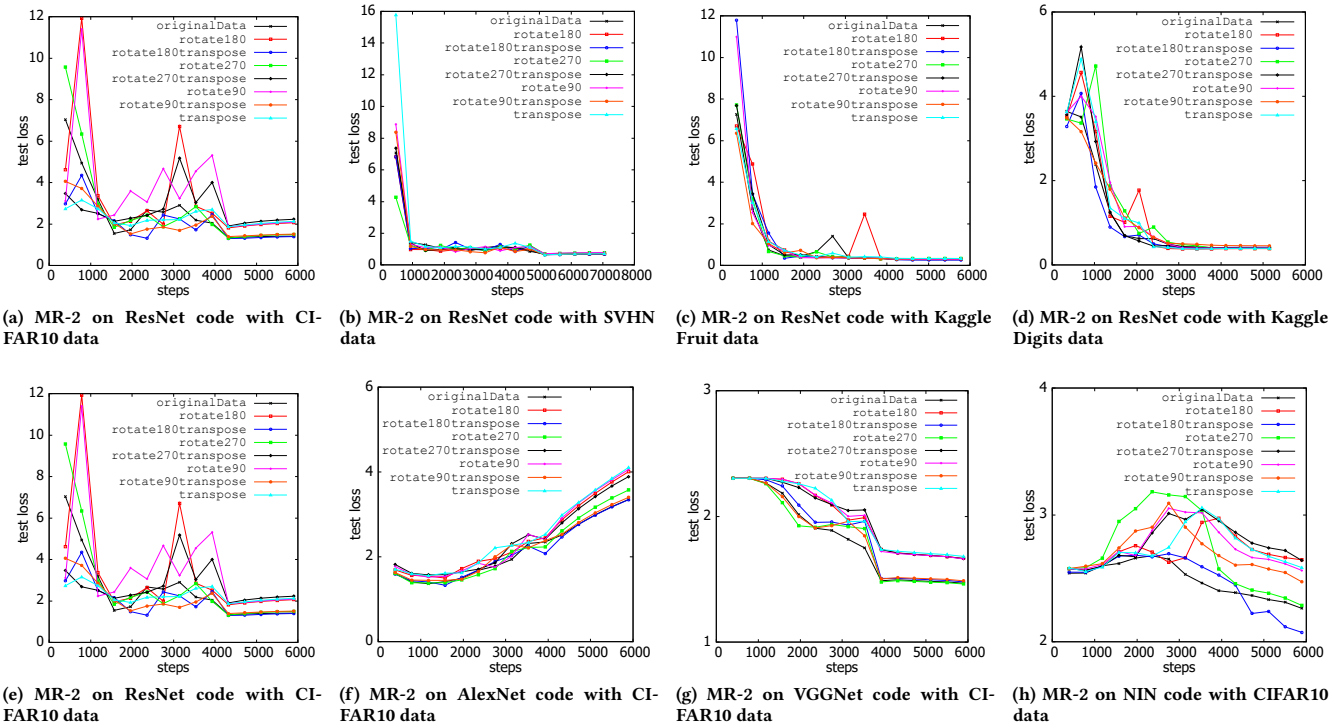


(g) MR-1 on VGGNet code with CIFAR10 data



(h) MR-1 on NIN code with CIFAR10 data

Figure 1: Validation loss from original code (non-buggy) due to changes MR-1: Permutation of RGB channels. Top four graphs for different data-sets on ResNet. Bottom four graphs for different architectures on CIFAR-10.



(a) MR-2 on ResNet code with CIFAR10 data

(b) MR-2 on ResNet code with SVHN data

(c) MR-2 on ResNet code with Kaggle Fruit data

(d) MR-2 on ResNet code with Kaggle Digits data

(e) MR-2 on ResNet code with CIFAR10 data

(f) MR-2 on AlexNet code with CIFAR10 data

(g) MR-2 on VGGNet code with CIFAR10 data

(h) MR-2 on NIN code with CIFAR10 data

Figure 2: Validation loss from original code (non-buggy) due to changes of MR-2: Permutation of CONV order. Top four graphs for different data-sets on ResNet. Bottom four graphs for different architectures on CIFAR-10.

2 APPENDIX B

In this section, we will provide the plots of the validation loss for all of the mutants when executed against MR-1 & MR-2. In some of the plots, we can see clear outliers and these outliers form the basis for catching the mutant. Quantitatively, we calculate the maximum of the standard deviation of the validation loss among the different variants of the MR. For example, in MR-1, at each step, 6 data-points of the test loss are available (each data-point corresponding to BGR, BRG, GBR, GRB, RBG, RGB). We calculate the standard deviation (σ) among these six variants and use the maximum of the standard deviation across all steps as the metric to measure the results of MR-1 against a mutant. If this maximum is greater than a threshold, we say the mutant is killed. The threshold used for MR-1 & MR-2 is 9, however, we found that the threshold is fairly robust in the sense it makes little difference in the number of mutants captured.

Mutant	MR-1 (permute RGB)	MR-2 (permute CONV order)	MR-3 (normalize data)	MR-4 (scale data)
c9	(2.30158)	(3.11631)		
c29		✓(18.1)		
c30	(3.44491)	(3.65127)		
c31	(5.24647)	✓(9.1)		
c32	(5.24647)	✓(9.1)		
c43	✓(9.6)	✓(11.5)		
c44	✓(27.4)	✓(9.2)		
c45	(4.7805)	(3.63453)		
c49	✓(23.3)	✓(23.8)		✓
c50	(4.19537)	(5.16786)	✓	✓
c116	(1.59469)	(2.63158)		
c221	✓(∞)	(0.918598)		
r6	(2.13284)	(2.08726)		
r48	(2.71402)	(2.58457)		
r49	(2.04281)	(3.02269)		
r67	(3.41174)	(3.91142)		

Table 2: MRs applied on mutants of ResNet. ✓ implies the mutant was killed. Values in brackets report σ_{max}. In total 8 out of 16 (50%) of bugs were caught.

2.1 Discussion on the Results

From Tables 2, 3 and the plots Figures 3, 4 we see that certain classes of mutants are caught and few cases are missed. Here we will detail some of the thinking on why the mutants were caught and why some were missed.

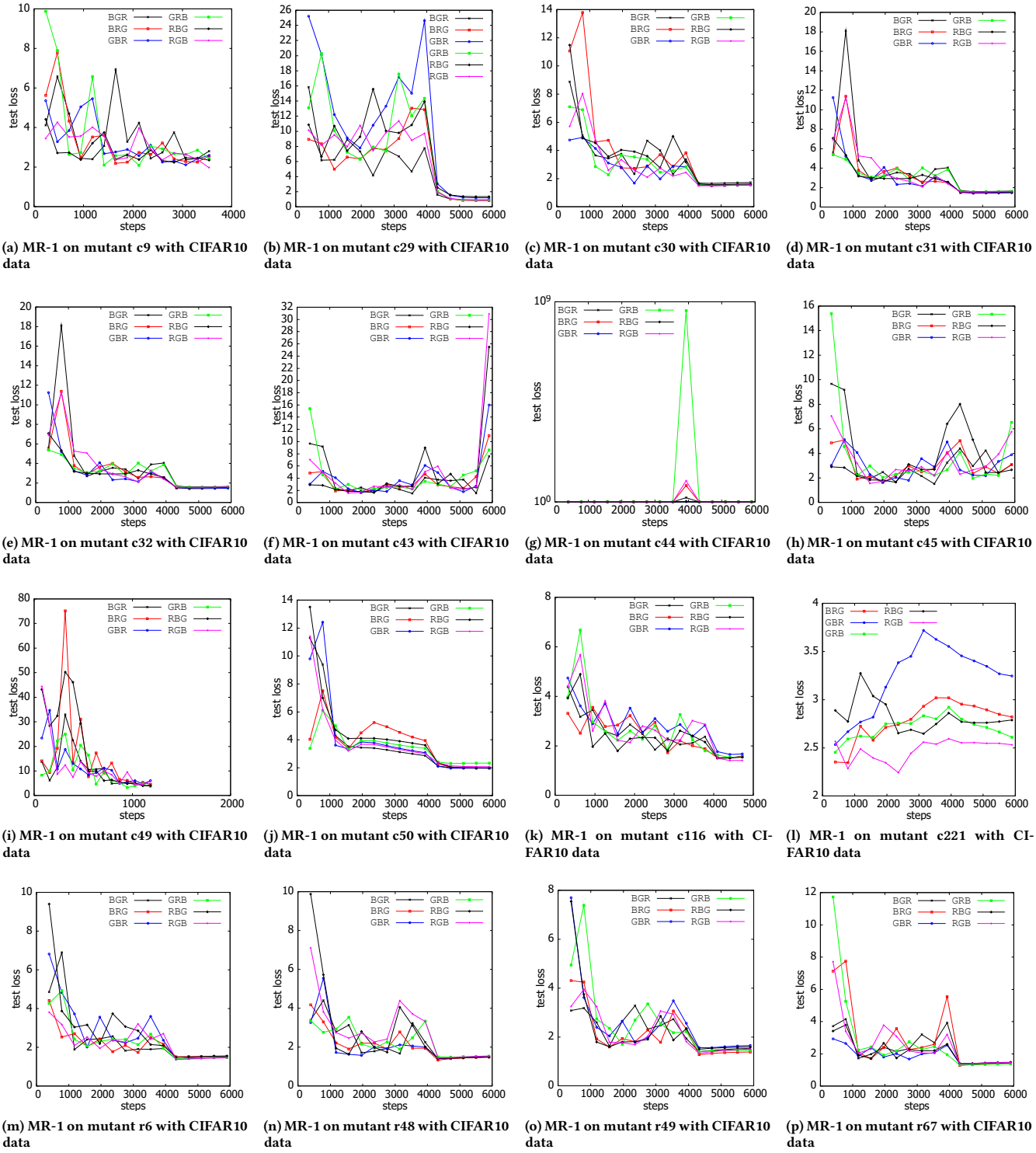
Mutant Type	Num. of Mutants	Num of Mutants Caught by the MRs
Reduce the training data files	3 (c9, c49, c116)	2 (c29, c49): 66%
Change the loss function	4 (c29, c30, c31, c32)	3 (c29, c31, c32): 75%
Changes to the learning rate decay	3 (c43, c44, c45)	2 (c43, c44): 66%
Interchange training and testing	2 (c50, c221)	1 (c221) : 50%
Change the architecture of ResNet	2 (r6, r67)	0
Pad the wrong channels	2 (r48, r49)	0
Total	16	8

Table 3: Types of Mutants created for ResNet application & % caught.

2.1.1 Change in Loss Function. These mutants are of the kind shown in Figure 5. These mutants make the model overfit on the data by reducing the influence of regularization. For example, from Figure 5, the optimization method will find that by increasing the size of the weights, the overall loss is reduced (because of '-' instead of '+').

From this, we can reason that when there are small changes in the input, there are wild swings in the output because of the overfitting. Thus, most of such bugs (75%) were caught.

Mutant 30, although of this kind was not caught. This is because this mutant did not reduce the regularisation effect as drastically as other mutants. Mutant 30 is shown in Figure 6. Note the change is from '*' to '/'.



(a) MR-1 on mutant c9 with CIFAR10 data

(b) MR-1 on mutant c29 with CIFAR10 data

(c) MR-1 on mutant c30 with CIFAR10 data

(d) MR-1 on mutant c31 with CIFAR10 data

(e) MR-1 on mutant c32 with CIFAR10 data

(f) MR-1 on mutant c43 with CIFAR10 data

(g) MR-1 on mutant c44 with CIFAR10 data

(h) MR-1 on mutant c45 with CIFAR10 data

(i) MR-1 on mutant c49 with CIFAR10 data

(j) MR-1 on mutant c50 with CIFAR10 data

(k) MR-1 on mutant c116 with CIFAR10 data

(l) MR-1 on mutant c221 with CIFAR10 data

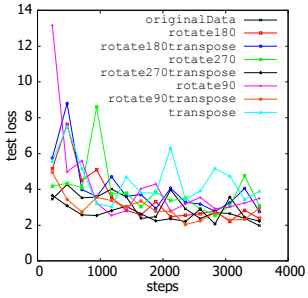
(m) MR-1 on mutant r6 with CIFAR10 data

(n) MR-1 on mutant r48 with CIFAR10 data

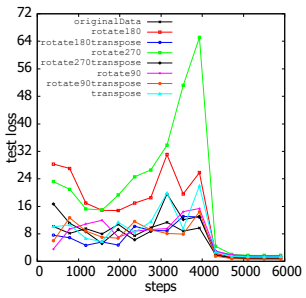
(o) MR-1 on mutant r49 with CIFAR10 data

(p) MR-1 on mutant r67 with CIFAR10 data

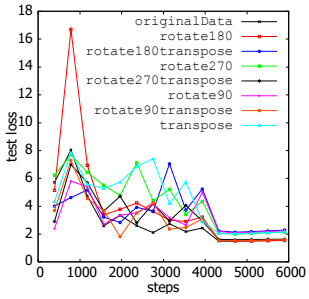
Figure 3: Validation loss from MR-1: Permutation of RGB order for the mutants on ResNet Application.



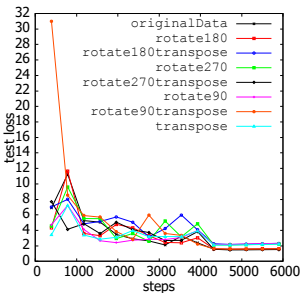
(a) MR-2 on mutant c9 with CIFAR10 data



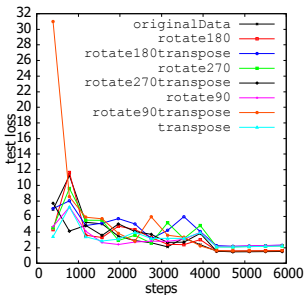
(b) MR-2 on mutant c29 with CIFAR10 data



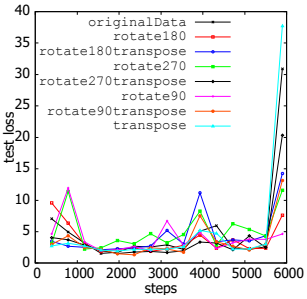
(c) MR-2 on mutant c30 with CIFAR10 data



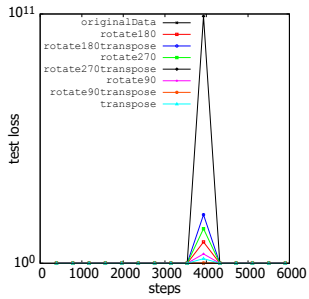
(d) MR-2 on mutant c31 with CIFAR10 data



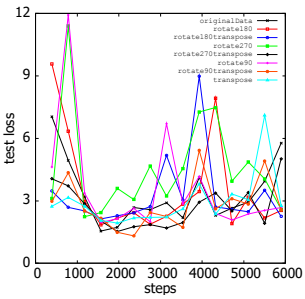
(e) MR-2 on mutant c32 with CIFAR10 data



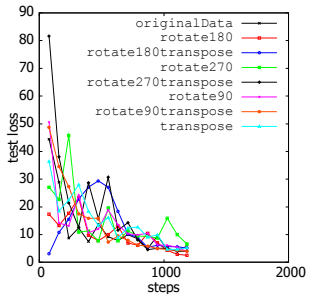
(f) MR-2 on mutant c43 with CIFAR10 data



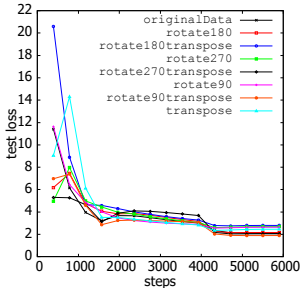
(g) MR-2 on mutant c44 with CIFAR10 data



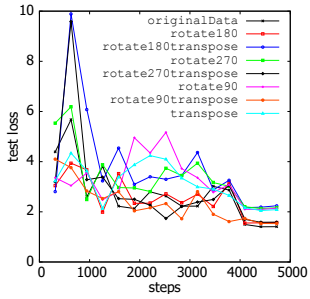
(h) MR-2 on mutant c45 with CIFAR10 data



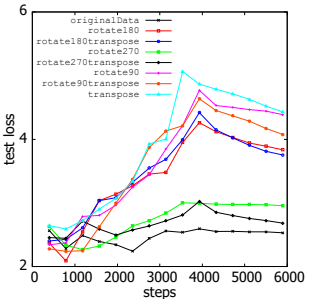
(i) MR-2 on mutant c49 with CIFAR10 data



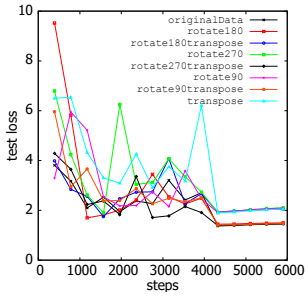
(j) MR-2 on mutant c50 with CIFAR10 data



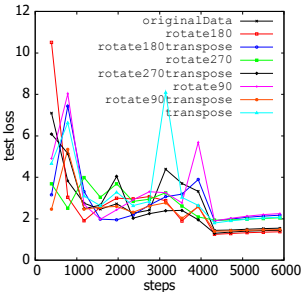
(k) MR-2 on mutant c116 with CIFAR10 data



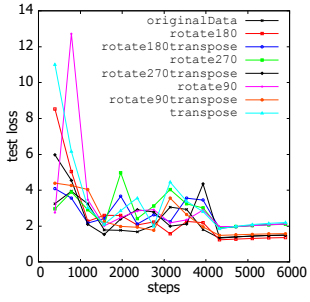
(l) MR-2 on mutant c221 with CIFAR10 data



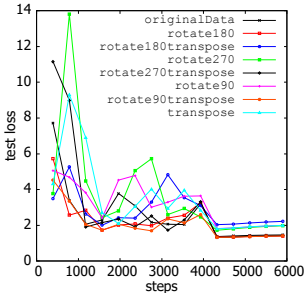
(m) MR-2 on mutant r6 with CIFAR10 data



(n) MR-2 on mutant r48 with CIFAR10 data



(o) MR-2 on mutant r49 with CIFAR10 data



(p) MR-2 on mutant r67 with CIFAR10 data

Figure 4: Validation loss from from MR-2: Permutation of CONV order for the mutants of ResNet application.

```
loss = cross_entropy +
              _WEIGHT_DECAY * tf.add_n([tf.nn.l2_loss(v)
                   for v in tf.trainable_variables()])

loss = cross_entropy -
              _WEIGHT_DECAY * tf.add_n([tf.nn.l2_loss(v)
                   for v in tf.trainable_variables()])
```

Figure 5: Original Code (top) and Mutant c29 (below). This mutant will reverse the effect of regularization & will attempt to overfit on the data.

```
loss = cross_entropy +
              _WEIGHT_DECAY * tf.add_n([tf.nn.l2_loss(v)
                   for v in tf.trainable_variables()])

loss = cross_entropy +
              _WEIGHT_DECAY / tf.add_n([tf.nn.l2_loss(v)
                   for v in tf.trainable_variables()])
```

Figure 6: Original Code (top) and Mutant c30 (below). This mutant will reverse the effect of regularization & will attempt to overfit on the data, but not as drastically as c29.

```
values = [initial_learning_rate *
          decay for decay in [1, 0.1, 0.01, 0.001]]

values = [initial_learning_rate /
          decay for decay in [1, 0.1, 0.01, 0.001]]
```

Figure 7: Original Code (top) and Mutant c43 (below). This mutant will start increasing the learning rate after 100 epochs instead of decreasing the learning rate. The network will suddenly start diverging instead of converging.

2.1.2 Change to Learning Rate decay. Mutants of this kind made the bug as shown in Figure 7. Note the change is from a '*' to a '/'.

To show the effect of this mutant, we calculated the `values` variable. In the orignal code (non-buggy), the variables holds a value of `[0.1, 0.010000000000000002, 0.001, 0.0001]`. In the mutant, the variable holds a value of `[0.1, 1.0, 10.0, 100.0]`. Thus, as the network progresses training, small changes in the input direction of the gradient (as is the case done by the different inputs of our MRs), there would be large deviations in the output. Thus, we see large variations in the outputs for the different variants of the MRs.

2.1.3 Change in Architecture of ResNet. These mutants changed the way the ResNet architecture is built. The 2 mutants in this category are distinct in effect and we explain both below (why we think they were not caught).

Mutant r6 is shown in Figure 8. This mutant has altered the computation in the skip connection where the original code added tensors. However, in the mutant, the tensors are subtracted. Conceptually, it is difficult to understand how this change impact the overall functioning of the ResNet architecture. Further, in the experimental results, these changes brought no (visible) change in output.

```
      return inputs + shortcut

      return inputs - shortcut
```

Figure 8: Original Code (top) and Mutant r6 (below). This mutant will subtract tensors (through the skip connections) instead of adding tensors. It's not clear how this change conceptually alters the network.

```
      for _ in range(1, blocks):

      for _ in range(2, blocks):
```

Figure 9: Original Code (top) and Mutant r67 (below). This mutant will reduce the number of blocks in the resnet architecture. Reducing the number of blocks reduces the capacity of the network to learn (by reducing the number of tunable parameters).

```
padded_inputs = tf.pad(inputs, [[0, 0], [pad_beg, pad_end], [
        pad_beg, pad_end], [0, 0]])

padded_inputs = tf.pad(inputs, [[0, 0], [pad_beg, pad_end], [
        pad_beg, pad_end], [1, 0]])
```

Figure 10: Original Code (top) and Mutant r48 (below). This mutant will pad the depth channel in addition to padding the spatial dimensions. Extra padding will increase the capacity of the network by increasing the number of trainable parameters.

Mutant r67 is shown in Figure 9. This mutant reducing the number of 'blocks' in the ResNet architecture. A block is a group of a Conv Layer, an activation layer and a batch-norm layer (See figure 3 of the main paper). By reducing the number of blocks, simply the capacity of the network to learn has been reduced (since the number of trainable parameters is reduced). Therefore, it might be difficult any of the MRs to catch this since the difference between variants might be negligible. Although this mutant can be thought of as a change in the hyper-parameter (for example reducing the number of layers in the network will have a similar effect), we kept this as valid mutant since the range of the variable in the loop is a obvious error in implementation.

2.1.4 Pad the wrong channel. The two mutants of this type were of the kind shown in Figure 10. Note the change from '0' to '1'.

The mutant r48 will pad the depth dimension. For example, if the input image is of dimension 32 * 32 * 3, this mutant will convert the image to 32 * 32 * 4. By increasing a new dimension, the overall number of weights increase as well. For example, in the first CONV layer the number of weights in the original code was 2 * 2 * 3. Now, with the mutant, the number of weights would be 2 * 2 * 4. Thus, the number of trainable parameters increases and would make the

capacity of the network to increase. We might expect the network to overfit, but this was not seen in the experiments. In fact, there was very little variance in the results of the mutants from the original.

Overall, both the mutants of type `Change in Architecture of ResNet` and `Pad the wrong channel` impact the number of parameters of the network. As such, we are not sure whether specific MRs to catch this can be made. It would be extremely interesting to develop MRs that may catch such cases.

3 APPENDIX C

In this section, we will provide some additional details pertaining to the execution of the mutants with the MRs.

3.0.1 Non-deterministic nature. One of the challenges working with the ResNet application (and possibly deep neural networks) is the stochastic nature of the output -i.e. for the same set of inputs, the application gives different results for two runs (see Table 4. 'Original' denotes the code as available in the application). Such differing outputs is a challenge for Metamorphic Testing, as all the MRs are based on relation between the outputs of subsequent runs. We investigated the reason for stochasticity and found it to be due to the random seeds that are used in TensorFlow. We updated the code to use a fixed seed for each run (this version is termed as as 'deterministic' in Table 4). Fixing the seed made the application deterministic when run on the CPU, unfortunately, the application was still stochastic when executed on the GPU. We could not determine the exact cause for the non-determinism on the GPU, but it appears to be an issue with the NVidia CUDA libraries [3] [1].

Code	Hardware	Num of Epochs	Run Num	Test accuracy	Test Loss	Loss μ	Loss σ
Original	CPU	5	1	0.23861386	5.9598055	5.2752532	1.898073321
Original	CPU	5	2	0.14554456	3.1301816		
Original	CPU	5	3	0.1079208	5.5404329		
Original	CPU	5	4	0.18613862	7.9412613		
Original	CPU	5	5	0.12871288	3.8045847		
Deterministic	CPU	5	1	0.13564357	5.4860325	5.4860325	0
Deterministic	CPU	5	2	0.13564357	5.4860325		
Deterministic	CPU	5	3	0.13564357	5.4860325		
Deterministic	CPU	5	4	0.13564357	5.4860325		
Deterministic	CPU	5	5	0.13564357	5.4860325		
Original	GPU	5	1	0.21089108	2.48737	7.30143576	3.728118329
Original	GPU	5	2	0.089108914	9.8327751		
Original	GPU	5	3	0.18613862	4.8291798		
Original	GPU	5	4	0.13663366	11.752038		
Original	GPU	5	5	0.10990099	7.6058159		
Deterministic	GPU	5	1	0.11881188	4.6922503	8.32788488	6.32573703
Deterministic	GPU	5	2	0.15643564	3.6995111		
Deterministic	GPU	5	3	0.1069307	18.292465		
Deterministic	GPU	5	4	0.14257425	10.990524		
Deterministic	GPU	5	5	0.12772277	3.964674		

Table 4: Experimental results on the original code. We modified the code by introducing fixed seeds for each run. This made the application deterministic on the CPU, however, the GPU was still non-deterministic.

REFERENCES

[1] antares1987. 2017. About Deterministic Behaviour of GPU implementation of tensorflow. https://github.com/tensorflow/tensorflow/issues/12871. (2017). [Online; accessed 15-Jan-2018].

[2] Olga Belitskaya. 2018. Handwritten Letters 2. https://www.kaggle.com/olgabelitskaya/handwritten-letters-2. (2018). [Online; accessed 12-Jan-2018].

[3] jkschin. 2017. Non-determinism Docs. https://github.com/tensorflow/tensorflow/pull/10636. (2017). [Online; accessed 15-Jan-2018].

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[5] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[6] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, Vol. 2011. 5.

[7] Mihai Oltean. 2018. Fruits 360 Dataset. https://www.kaggle.com/moltean/fruits/data. (2018). [Online; accessed 12-Jan-2018].

[8] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).