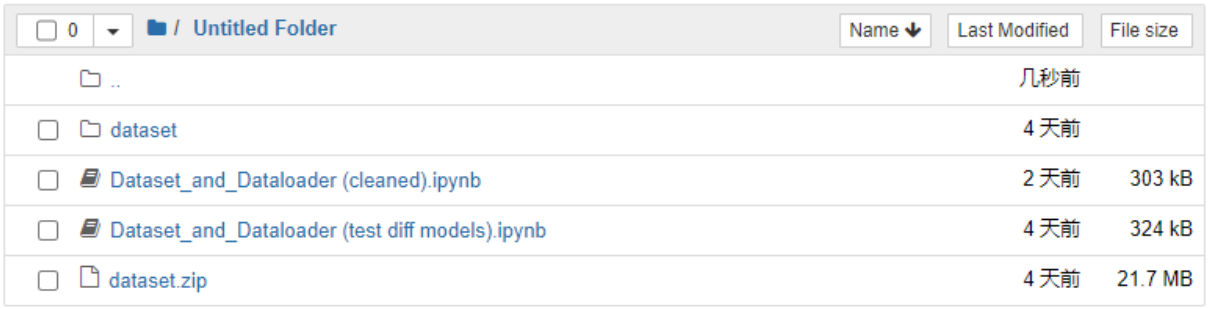
50.039 Deep Learning, Y2021

Small Project Report

Group 30: Chow Jia Yi (1003597) | Yuri Kim (1002334) | Guo Ziwei (1003702) | Dong Ke (1003713)

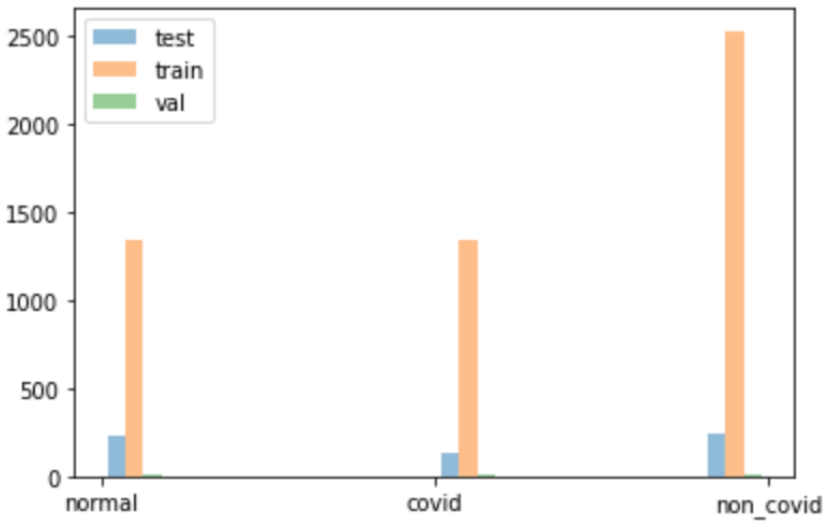
Instructions to run the code

To run code, you need to open the file Last_file.ipynb with Jupyter Notebook, and also place the dataset folder under the same folder. You can also load the saved_model.pth using the functions provided in the Jupyter file.
For example,



Additional libraries might need to be imported. The details could be found in the Readme.

Dataset Distribution Visualization



A balanced dataset is the one that contains an equal or almost equal number of samples from all the classes. If the samples from one of the classes outnumbers the other, the dataset is not

uniformly distributed, thus considered as imbalanced. As shown above in the graph, the dataset we used is not balanced between three classes. Number of samples in the non_covid dataset is almost twice of the number of samples in the other two classes. The imbalanced data may be skewed in favour of one of the classes. This may cause problems in training, because the majority class can dominate the minority class and the classifier may only learn one concept instead of two or three distinct concepts.

Architecture

Multiclass vs Binary Architecture

Each architecture has their pros and cons.

	Multiclass Architecture	Binary Architecture
Pros	- Easy to use - Ideal when there are many classes	- Converges fast - Ideal when there are a handful of classes
Cons	- Slower than binary classifiers during training - Takes a while to converge in high-dimensional problems	- Troublesome when there are too many classes - **Need to be careful when training to avoid class imbalances**

Based on the distribution of the given dataset, there is much lesser images of normal class compared to infected class, and the ratio is approximately 1:3. If we consider the binary classifier, the first binary classifier will have bias because there is imbalance between the two classes (normal and infected). Therefore, we decided to use the multiclass classifier.

Preliminary Research and Experiments

Before we built our own model, we conducted preliminary research on popular model architectures and experimented with our own dataset. This is to understand which model architecture would give us a higher accuracy and hence we can build our own model from scratch emulating the best architecture.

Additionally, we tried with both RGB channels and grayscale channels to see the output difference.

Here is a collection of our results. The details and plots could be seen in the appendix.

Results Comparison of Different Models

Model	Channels	Train Loss	Train Accuracy	Test Loss	Test Accuracy	Best Test Accuracy
ResNet18	3	0.238	0.905	1.238	0.289	0.752
	1	0.224	0.907	1.492	0.640	0.715
DenseNet121	3	0.430	0.816	0.821	0.751	0.805
	1	0.448	0.808	0.992	0.751	0.791
VGG11_BN	3	0.356	0.842	1.063	0.735	0.798
	1	0.348	0.849	0.980	0.783	0.832
SqueezeNet	3	0.332	0.854	1.185	0.718	0.781
	1	0.608	0.713	1.481	0.508	0.679
All the models are trained with 10 epochs, SGD optimizer, CrossEntropy loss function, 0.001 learning rate, image size of 150 by 150, no data augmentation and batch size of 4. Learning curves of the models are attached in the Appendix.						

Based on our preliminary studies and experiments, we found out that VGG has the best performance, especially the VGG with 1 channel input (grayscale). For our special dataset, VGG is the most suitable one.

Proposed model

Neural Network

Based on the experimental results above, VGG is the best model for this dataset. There are several reasons why VGG achieves high performance.

1. Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Since there are now 3 ReLU units instead of just one, the decision function is more discriminative. There are also fewer parameters (27 * number of channels) compared to AlexNet (49 * number of channels), preventing overfitting.
2. VGG incorporates 1x1 convolutional layers to make the decision function more nonlinear without changing the receptive fields.
3. The small-sized convolution filters allows VGG to have a large number of weight layers, which leads to improved performance.

Our proposed VGG network has the following architecture:

1. A 2D convolutional layer with 1 input channel, 64 output channels, kernel size 3 and stride 1.

2. A 2D convolutional layer with 64 input channels, 64 output channels, kernel size 3 and stride 1.
3. 2D MaxPooling with kernel size 2, stride 2 and padding 0.
4. A 2D convolutional layer with 64 input channels, 128 output channels, kernel size 3 and stride 1.
5. A 2D convolutional layer with 128 input channels, 128 output channels, kernel size 3 and stride 1.
6. 2D MaxPooling with kernel size 2, stride 2 and padding 0.
7. A 2D convolutional layer with 128 input channels, 256 output channels, kernel size 3 and stride 1.
8. Two 2D convolutional layers with 256 input channels, 256 output channels, kernel size 3 and stride 1.
9. 2D MaxPooling with kernel size 2, stride 2 and padding 0.
10. A 2D convolutional layer with 256 input channels, 512 output channels, kernel size 3 and stride 1.
11. Two 2D convolutional layers with 512 input channels, 512 output channels, kernel size 3 and stride 1.
12. 2D MaxPooling with kernel size 2, stride 2 and padding 0.
13. A 2D convolutional layer with 512 input channels, 512 output channels, kernel size 3 and stride 1.
14. 2D MaxPooling with kernel size 2, stride 2 and padding 0.
15. 2D Adaptive Average Pooling with a 7x7 output size.
16. A Fully Connected Layer with 4096 channels.
17. Dropout layer with 50% dropout
18. A Fully Connected Layer with 4096 channels.
19. Dropout layer with 50% dropout
20. A Fully Connected Layer with 3 channels, which is the number of classes (COVID, non-COVID, normal) in this project.

All the convolutional and fully connected layers use ReLU activation function.

Batch size

Batch size should be a multiple of 2 so that it will utilise the GPU.
Our final batch size is 4, due to the constraints of memory.

Loss function

We decided to use cross-entropy loss as our loss function, because it works well for multiclass classification which we are using in our proposed architecture.

The cross-entropy loss is defined as $$J = -\frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{y_i} \cdot \log(\hat{\mathbf{y}}_\mathbf{i})\right)$$.

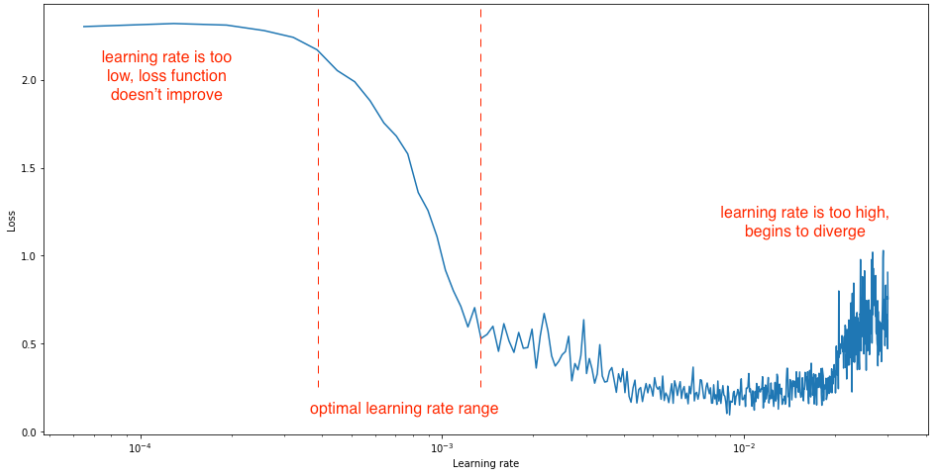
It rewards/penalises probabilities of correct classes only, and the value is independent of how the remaining probability is split between incorrect classes.

To use the loss function, we also converted the one-hot vector to class labelling in the data loader, as one-hot labelling is not compatible with the loss function we chose.

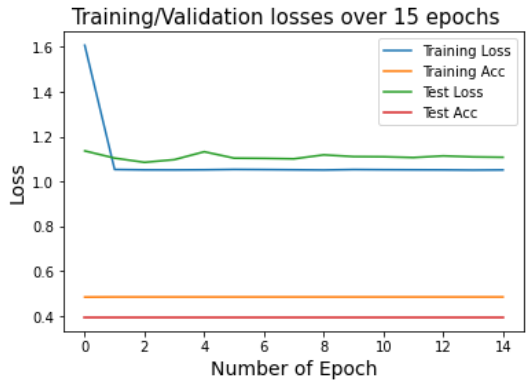
Optimizer

We decided to use SGD optimizer with momentum 0.9. We did not choose Adam optimizer. Although **Adam** is great and supposedly it's much faster than **SGD**, for our case, **Adam** has convergence problems and often **SGD** + momentum can converge **better** though with longer training time.

We chose 0.001 as the learning rate because it allows the neural network to converge without slowing down the convergence. We also added momentum to make our model more robust and make it converge to global minimum.

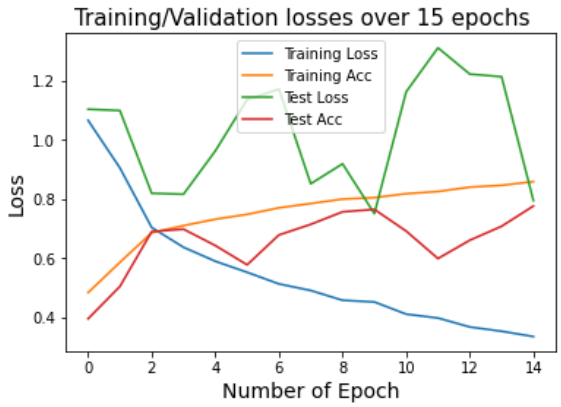



```
Epoch: 15/15 -  Training Loss: 1.051 -  Training Accuracy: 0.485 -  Test Loss: 1.108 -  Test Accuracy: 0.394
Training complete in 11m 37s
Best val Acc: 0.394137
```



The model used Adam optimizer, it fails to converge after epoch two.

```
Epoch: 15/15 -  Training Loss: 0.334 -  Training Accuracy: 0.858 -  Test Loss: 0.794 -  Test Accuracy: 0.775
Training complete in 8m 46s
Best val Acc: 0.775244
```



However, with the SGD optimizer, it is able to converge.

Choice of initialization

We chose kaiming initialisation.

We do not want to initialise the weights to 0 because the learning rate would be the same for all the weights of the neural network. However, random initialisation can cause exploding or vanishing gradients.

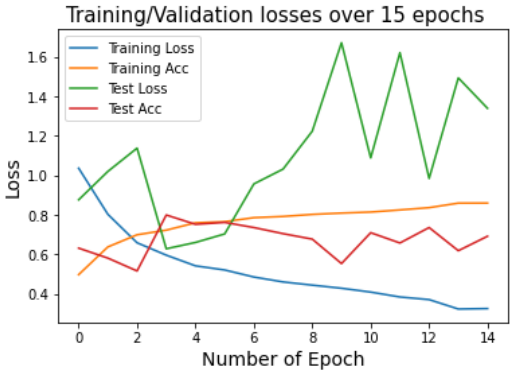
Kaiming initialisation solves this problem by multiplying random initialisation with a factor $\sqrt{\frac{2}{size^{[l-1]} + size^{[l]}}}$. This helps to avoid slow convergence, and ensure that we don't keep oscillating off the minima.

Kaiming initialisation is also more accurate than Xavier initialisation, especially if the activation function does not have a derivative of 1 at 0, like the ReLU function which we are using in our project.

Results visualization and discussion

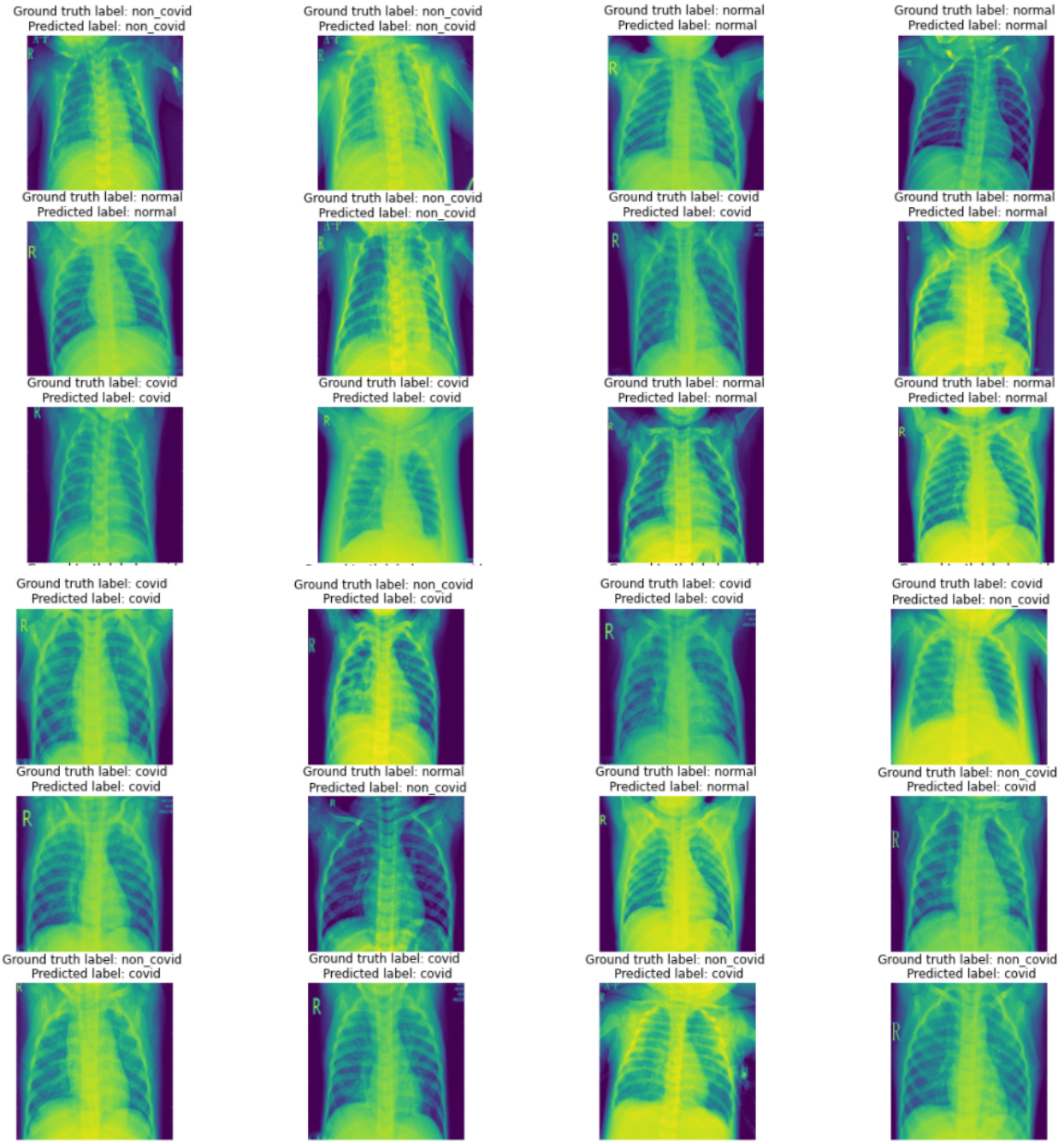
Result obtained with our proposed VGG network training for 15 epochs, SGD optimizer, CrossEntropy loss function, 0.001 learning rate, image size of 150 by 150, no data augmentation and batch size of 4:

```
Epoch: 15/15 -  Training Loss: 0.326 -  Training Accuracy: 0.860 -  Test Loss: 1.340 -  Test Accuracy: 0.692
Training complete in 7m 46s
Best val Acc: 0.799674
```




```
Accuracy of normal : 87 %
Accuracy of covid : 87 %
Accuracy of non_covid : 37 %
-----------------------------
Accuracy of the model: 70.8%
Number of false negative of covid: 1/8
```

Validation set picture, with predicted and ground truth label.
Average performance 17/24 = 70.8%
Class-wise perfomance: normal: 87.5% | covid: 87.5% | non_covid: 37.5%



You might find it more difficult to differentiate between non-covid and covid x-rays, rather than between normal x-rays and infected (both covid and non-covid) people x-rays. Was that something to be expected? Discuss.

Yes, because both COVID and non-COVID respiratory illnesses cause pneumonia on the lungs. Ibrahim et al mentioned in their paper that the CXR scan images of different types of viral pneumonia are similar, making it hard for a radiologist to distinguish between COVID-19 with other viral pneumonia. However, a healthy pair of lungs has no symptoms and thus looks much different from infected lungs. [1]

Would it be better to have a model with high overall accuracy or low true negatives/false positives rates on certain classes? Discuss.

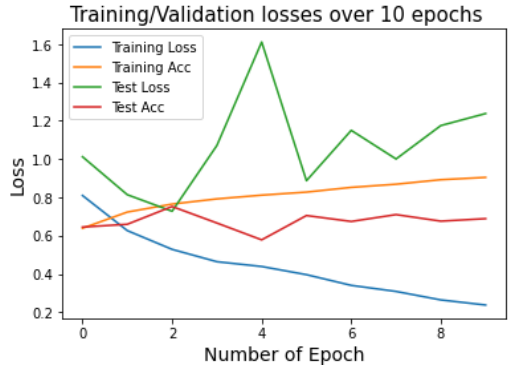
It is better to have low true negatives/false positives, especially in distinguishing between COVID-19 cases compared to other types of viral pneumonia. Otherwise, it could lead to people being wrongly diagnosed with COVID, or missing out people who have COVID, the latter of which has much worse consequences.

[1] https://link.springer.com/article/10.1007/s12559-020-09787-5

Appendix
Models loaded from pytorch and train from scratch:
All the models are trained with 10 epochs, SGD optimizer, CrossEntropy loss function, 0.001 learning rate, image size of 150 by 150, no data augmentation and batch size of 4.

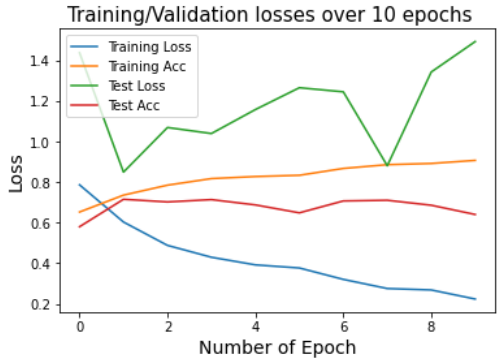
ResNet18:
3 channels:

```
Epoch: 10/10 -  Training Loss: 0.238 -  Training Accuracy: 0.905 -  Test Loss: 1.238 -  Test Accuracy: 0.689
Training complete in 3m 40s
Best val Acc: 0.752443
```



1 channel:

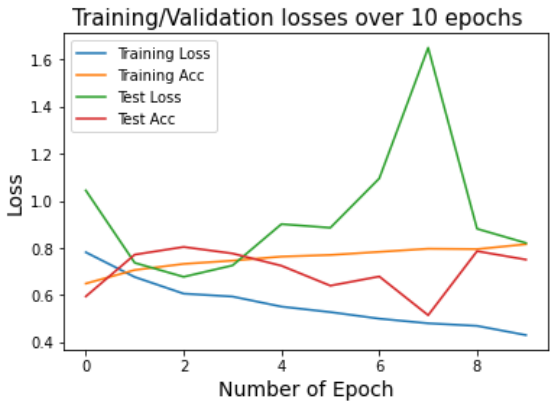
```
Epoch: 10/10 -  Training Loss: 0.224 -  Training Accuracy: 0.907 -  Test Loss: 1.492 -  Test Accuracy: 0.640
Training complete in 4m 6s
Best val Acc: 0.714984
```



DenseNet121:

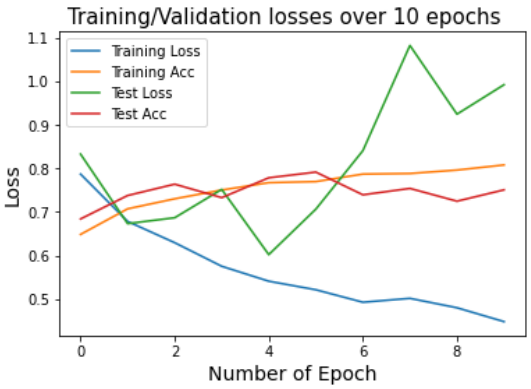
3 channels:

```
Epoch: 10/10 -  Training Loss: 0.430 -  Training Accuracy: 0.816 -  Test Loss: 0.821 -  Test Accuracy: 0.751
Training complete in 18m 0s
Best val Acc: 0.804560
```



1 channel:

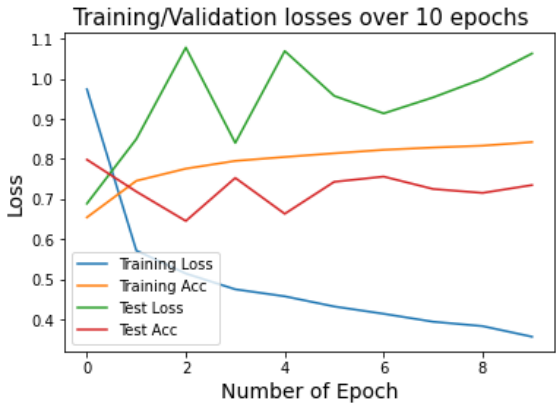
```
Epoch: 10/10 -  Training Loss: 0.448 -  Training Accuracy: 0.808 -  Test Loss: 0.992 -  Test Accuracy: 0.751
Training complete in 18m 9s
Best val Acc: 0.791531
```



VGG11_BN:

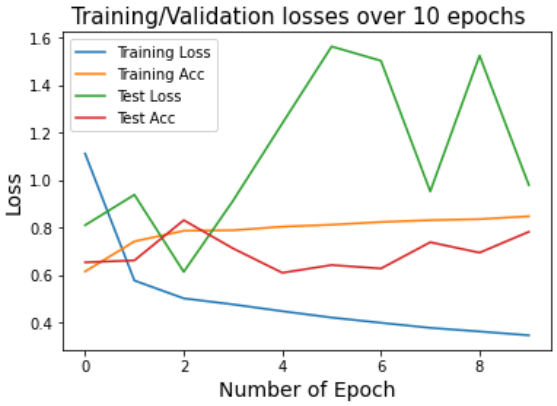
3 channels:

```
Epoch: 10/10 -  Training Loss: 0.356 -  Training Accuracy: 0.842 -  Test Loss: 1.063 -  Test Accuracy: 0.735
Training complete in 5m 17s
Best val Acc: 0.798046
```



1 channel:

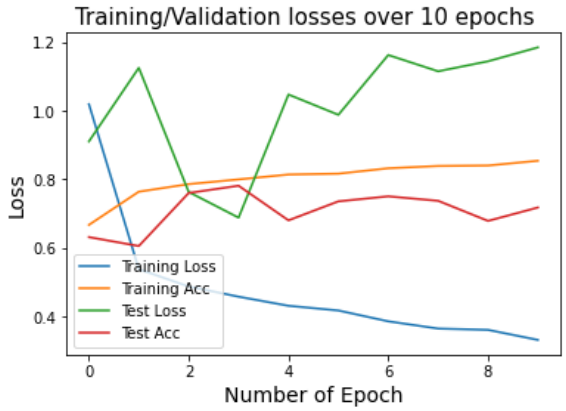
```
Epoch: 10/10 -  Training Loss: 0.348 -  Training Accuracy: 0.849 -  Test Loss: 0.980 -  Test Accuracy: 0.783
Training complete in 4m 59s
Best val Acc: 0.832248
```



SqueezeNet:

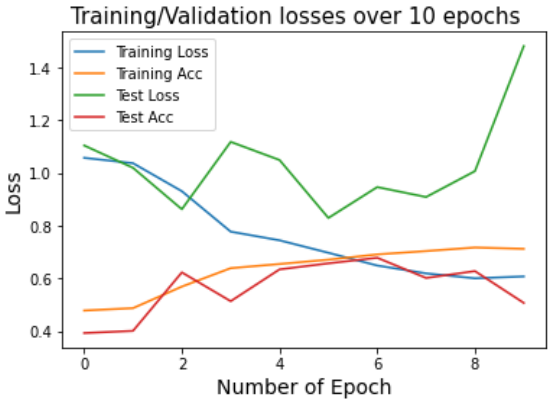
3 channels:

```
Epoch: 10/10 -  Training Loss: 0.332 -  Training Accuracy: 0.854 -  Test Loss: 1.185 -  Test Accuracy: 0.718
Training complete in 5m 16s
Best val Acc: 0.781759
```



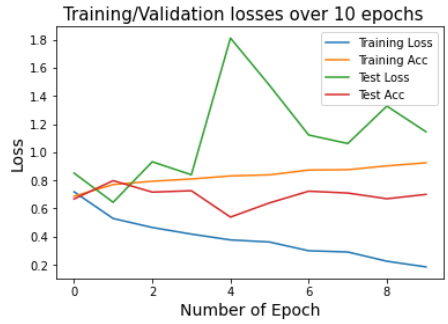
1 channel:

```
Epoch: 10/10 -  Training Loss: 0.608 -  Training Accuracy: 0.713 -  Test Loss: 1.481 -  Test Accuracy: 0.508
Training complete in 3m 3s
Best val Acc: 0.679153
```



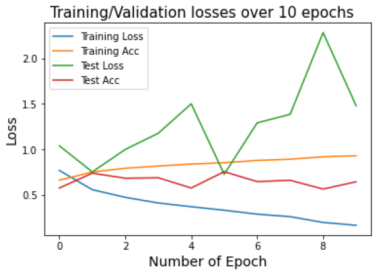
Code ResNet from scratch with random initialisation:

```
Epoch: 10/10 -  Training Loss: 0.185 -  Training Accuracy: 0.925 -  Test Loss: 1.145 -  Test Accuracy: 0.700
Training complete in 3m 37s
Best val Acc: 0.798046
```



Code ResNet from scratch with Kaiming initialisation:

```
Epoch: 10/10 -  Training Loss: 0.168 -  Training Accuracy: 0.930 -  Test Loss: 1.480 -  Test Accuracy: 0.645
Training complete in 3m 35s
Best val Acc: 0.755700
```



Code ResNet from scratch with Xavier initialisation:

```
Epoch: 10/10 -  Training Loss: 0.170 -  Training Accuracy: 0.930 -  Test Loss: 1.157 -  Test Accuracy: 0.712
Training complete in 3m 34s
Best val Acc: 0.749186
```