HIT Deep Learning – Final Project Report – Dementia Classification

Helit Bauberg, 027466002

Or Simhi, 201122892

This report is complementary to the inline report submitted within the project notebook; it details the thought and implementation process in of the various stages of the project, from the following aspects.

[A – Exploratory data analyses]
[B – Model Architecture, hyper parameters random search]
[C – impact of image manipulations on a baseline model]
[D – Benchmark model – pytorch implementation]
[E – feature extraction]
[F – final test runs]

[A – EDA]

The starting point was to download and look at the data.
Observations:

- Significant imbalance between 4 classes:

```
Train Samples - Class NonDemented: 49% (2560/5121)
Train Samples - Class VeryMildDemented: 34% (1792/5121)
Train Samples - Class MildDemented: 14% (717/5121)
Train Samples - Class ModerateDemented:  1% (52/5121)
```

  Action item 1: create more samples from underrepresented classes (how to create good samples, without distorting key spatial features of the sMRI images? With only 52 samples, how do we make sure we have enough variety for the training process?)

- The MRI scans seem to have already gone through some preprocessing.
  Action item 2: Research sMRI preprocessing tools and techniques:
  Yamanakkanavar, N.; Choi, J.Y.; Lee, B. MRI Segmentation and Classification of Human Brain Using Deep Learning for Diagnosis of Alzheimer's Disease: A Survey. Sensors 2020, 20, 3243 provides in-depth review of preprocessing and segmentation methods of AD MRI classification for ML applications. Based on the General pipeline for brain MRI analysis described in the article (figure 2), it seemed like our dataset includes (T1-W?) images that already went through a preprocessing pipeline: skull-stripping, Bias field correction, brain extraction, noise reduction and image registration.
  Action item 2.1: ***Research sMRI segmentation and apply transformations*** *(which?? What packages are available in python? Constraint: This must be optimized for implementation over googlecolab using pytorch) to our dataset prior to training: makes sense to go with Tissue segmentation: CNN trained on multiple patches and kernel sizes to extract information from each voxel.*

[A – implementation approach taken, environment considerations]
From browsing the web, most implementations of Alzheimer's classification ML models over the same dataset have used tensorflow with keras modules. Clear why - it is well documented and keras API is friendly. Moreover, many models rely on pre-trained models, and used size reshape image transformation (208, 176) -> (150,150) on the dataset. To our understanding, such resizing distorts the image and may compromise the model's accuracy - yet the reported results were excellent (see An MRI-based deep learning approach for accurate detection of Alzheimer's disease).
We then decided that this project will attempt to recreate a tensorflow CNN model that yielded good predictions in pytorch environment, while (1) creating our own pipelines and dataset as a mixture of reported data manipulation methods, and (2) do our own hyperparameter optimization.
This approach will provide a benchmark for the predicted results, as well as an interesting comparison between the performance of pytorch vs. tensorflow. *We should also test rescale/reshape image manipulation on the model.*
We define three transformations / pipelines:

- Random Resize Crop to (176,176) will be used for generating more samples for undersampled classes in the train set.

- p_preproc will be a preprocessing pipeline that train, validate and test datasets all will go through, that will include only center crop (176,176) and ToTensor() (that includes normalization – not clear if we need to do further normalization, tested on the notebook and seems ToTensor already sorted that out)
- p_manip_samples will be used for image manipulations on the expanded train and validation set, and will be used to create variance amongst the samples, for the model to not get fixated on irrelevant features during the training process. *Is there negative/inverse transform? Will this help training? We should check which transforms help/mess with training once we have baseline results.*

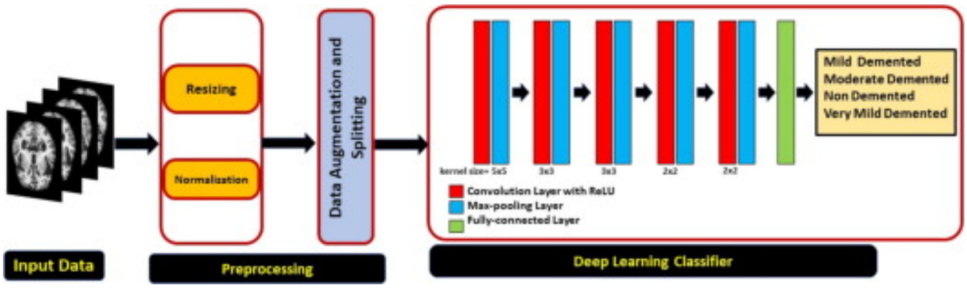
The data is loaded as greyscales, but Transform pipeline returns it as RGB. We then tested the various transformations before adding them to the pipelines ('test bed' cell on notebook). Contrast transformation was immediately ruled out as the output image won't contribute to the training (it wasn't used in neither of the MRI pipelines described in articles we read).

Augmenting more under sampled training data proved to be resource exhaustive and inefficient, probably due to bad implementation, as we lately realized. We ran our of RAM, sessions crushed, cuda complained. We resorted to creating the images offline – save the PIL image to the GoogleDrive class directory, so that once we create the dataset we can perform all other manipulations (pre-processing and random transformations) on the run, in the training loop, to the (now bigger and balanced) training set.

[B – Model Architecture]

Benchmark – tensorflow model:

Adapted from [An MRI-based deep learning approach for accurate detection of Alzheimer's disease]:



RELU(2D convolutional layer, kernel size 5 x 5, filters=64)
maxpool(2x2)
RELU(2D convolutional layer, kernel size 3 x 3, filters=128, stride=2)
maxpool(2x2)
Dropout(0.25)
RELU(2D convolutional layer, kernel size 3 x 3, filters=256, stride=2)
maxpool(2x2)
RELU(2D convolutional layer, kernel size 2 x 2, filters=512, stride=2)
maxpool(2x2)
RELU(2D convolutional layer, kernel size 2 x 2, filters=1,024, stride=2)
Dropout(0.3)
FC dense layer (fully connected, in=1,024, out=4)
Softmax

Epochs: 100
Learning rate: 0.001
Batch size: 40
Optimizer: Adam
Loss: categorical-cross-entropy

Reported Accuracy: 99.68%

* There are some inconsistencies in this article referring to their model architecture.

[B – hyperparameter selection]
We used random grid search to compare the same model's performance using various hyperparameters - two different optimizers (Adam/SGD), few learning rate, weight decay and momentum configurations.

Our baseline model uses 3 conv layers, max filters 512.
We ran 5 epochs on random net params set, using 15% of training data for param benchmarking. Some of the outputs are (last two are loss and accuracy on test set):

```
Completed trial # XX ({'l_1': 64, 'l_2': 128, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.01,
'weight_decay': 0.0001, 'optim': 'SGD', 'momentum': 0.7, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.400233950394001, 50) ➔ no learning curve yet 50%
accuracy on test data
Completed trial # 3 ({'l_1': 64, 'l_2': 128, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.005,
'weight_decay': 0.0003, 'optim': 'SGD', 'momentum': 0.9, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.3848075411034366, 50)
Completed trial # XX ({'l_1': 64, 'l_2': 128, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.01,
'weight_decay': 0.0005, 'optim': 'SGD', 'momentum': 0.7, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.099855731352876, 12) ➔ nice learning curve yet 12%
accuracy on test data
Completed trial # 5 ({'l_1': 64, 'l_2': 256, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.005,
'weight_decay': 0.0005, 'optim': 'SGD', 'momentum': 0.9, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.3839444837975226, 50)
Completed trial # 4 ({'l_1': 64, 'l_2': 128, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.001,
'weight_decay': 0.0007, 'optim': 'SGD', 'momentum': 0.7, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.1801022217540666, 50)
Completed trial # 2 ({'l_1': 64, 'l_2': 128, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.005,
'weight_decay': 0.0003, 'optim': 'Adam', 'momentum': 0.5, 'epochs': 5, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.4099190023414876, 50) ➔ loss is all over the place
```

From the above basic benchmarking we decided to test the model using the following configuration:
Optimizer: SGD
Learning rate: 0.001
Weigh decay: 0.0007
Momentum: 0.7

We also created the train/val dataset loading method in such a way, that it can return either the entire dataset or a fraction of it and tested the model's performance with/without train/validate shuffling at each epoch.

[C – impact of image manipulations on a baseline model]

Our initial focus was to compare the impact of the various image transformation on the performance of the model, in order to establish the right set of transformations.
We used a model with 3 conv layers and 3 fully connected layers.
First experiments with the model proved that we cannot do batch_size=40 – cuda crushes, out of GPU memory. We went with batch_size=4

Initial run was for 20 epochs, 20% train data per epoch
Run time: 1:20 hours

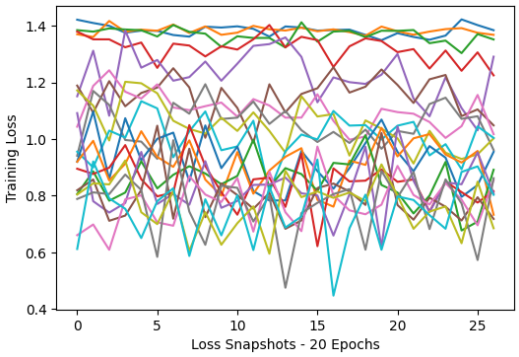
```
p_preproc = T.Compose([
    T.CenterCrop(176),
    T.ToTensor(),       # This normalizes all data to (0,1)
    ])
```

```
random_crop = T.RandomCrop(size=(176),antialias=None)
```

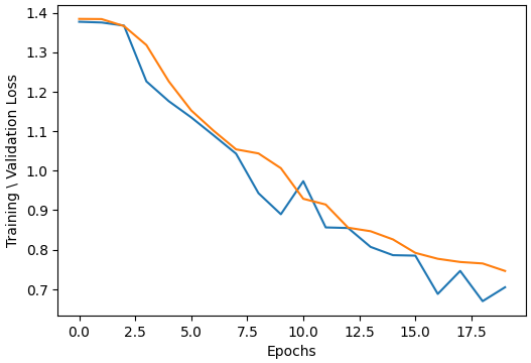
```
p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomHorizontalFlip(),
    T.RandomResizedCrop(size=(176),antialias=None),
    ]
```

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.7459351251554337
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.7046545226289722
TRAIN/VAL END RESULTS: Accuracy: 67% (234/345)
```

Train Loss behavior per epoch:



Train / Validation Loss




```
Completed trial # 0 Final Training loss:  0.7459351251554337  Final Validation loss:
0.7046545226289722
Test Accuracy of class MildDemented: 44% (80/179)
Test Accuracy of class ModerateDemented: 66% ( 8/12)
Test Accuracy of class NonDemented: 84% (538/640)
Test Accuracy of class VeryMildDemented:  2% (10/448)
Overall accuracy of the network on the 1279 test images: 49 %
```

➔ Observations:

- Need more epochs – loss was still improving.
- Best (84%) and worst (2%) results on classes we didn't augment -> Model is not learning the right features.

Next experiment – add more variety on trainset using various random transformations. Run for 30 epochs.

[C – EXP I – more random transformations, more epochs]

30 epochs, 20% train data per epoch

Run time: 2:10 hours

```
p_preproc = T.Compose([
    T.CenterCrop(176),
    T.ToTensor(),       # This normalizes all data to (0,1)
    ])
```

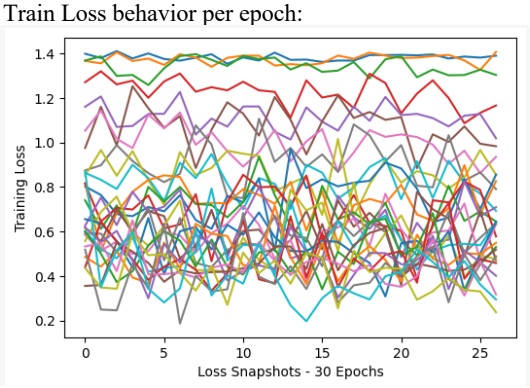
```
random_crop = T.RandomCrop(size=(176),antialias=None)
```

```
p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomHorizontalFlip(),
    T.RandomResizedCrop(size=(176),antialias=None),
    sharpness_transform,
   ]
```

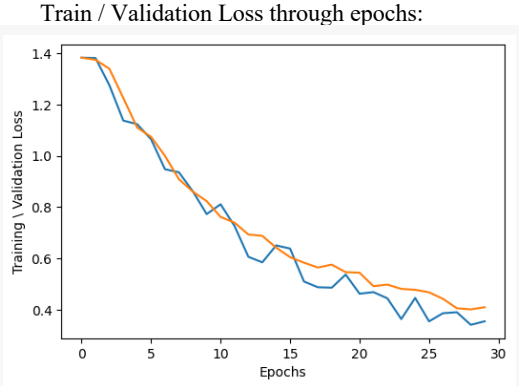
```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.4091880398099171
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.35419871405504866
TRAIN/VAL END RESULTS: Accuracy: 85% (295/345)
```

```
Completed trial # 0 Final Training loss:  0.4091880398099171  Final Validation loss:
0.35419871405504866
Test Accuracy of class MildDemented: 57% (103/179)
Test Accuracy of class ModerateDemented: 100% (12/12)
Test Accuracy of class NonDemented: 67% (431/640)
Test Accuracy of class VeryMildDemented: 19% (89/448)
Overall accuracy of the network on the 1279 test images: 49 %
```

Train Loss behavior per epoch:



Train / Validation Loss through epochs:



➔ Observations:

- Need more epochs – loss was still improving.
- Getting 100% accuracy (12/12) for ModerateDemented – most augmented class. Only 57% on augmented MildDemented. We created 40 RandomCrop samples per scan for ModerateDemented, and 2 RandomCrop samples per scan for MildDemented.
- Worst accuracy 19% on non-augmented class VeryMildDemented, lame 67% on non-augmented NonDemented.

➔ Can we get better results if we use RandomCrop instead of CenterCrop in the preprocessing pipeline?
Next experiment – use RandomCrop on p_preproc pipeline. Run for 40 epochs.

- We still keep RandomResizedCrop in our random image manipulation pipeline.

[C – EXP II - RandomCrop instead of CenterCrop]
40 epochs, 20% train data per epoch
Run time: 2:40 hours

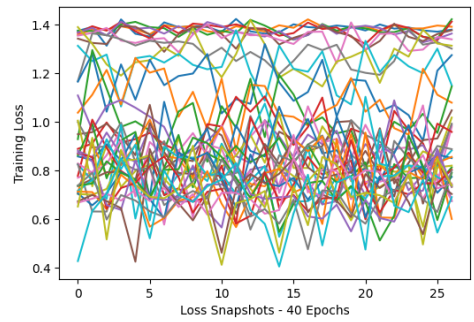
```
p_preproc = T.Compose([
    T.RandomCrop(size=(176))   # T.CenterCrop(176),
    T.ToTensor(),              # This normalizes all data to (0,1)
    ])
```

This network is learning much slower compared to the previous one – it was looping around 60%-65% accuracy on the validation set for ~20 epochs before it made a jump to 75% accuracy on epoch 30 and down to 68% on epoch 31.
Results were not impressive, and we will revert to CenterCrop moving forward.

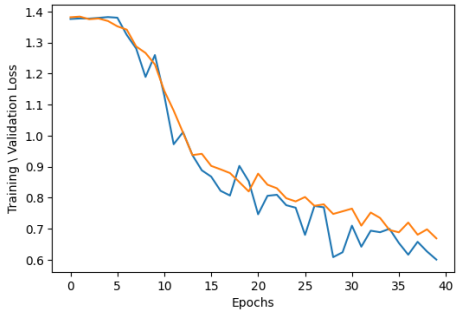
```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.6687005766881916
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.6002685105006308
TRAIN/VAL END RESULTS: Accuracy: 69% (241/345)
```

```
Completed trial # 0 Final Training loss:  0.6687005766881916  Final Validation loss:
0.6002685105006308
Test Accuracy of class MildDemented: 45% (82/179)
Test Accuracy of class ModerateDemented: 66% ( 8/12)
Test Accuracy of class NonDemented: 75% (486/640)
Test Accuracy of class VeryMildDemented: 25% (115/448)
Overall accuracy of the network on the 1279 test images: 54 %
```

Train Loss behavior per epoch:



Train / Validation Loss through epochs:

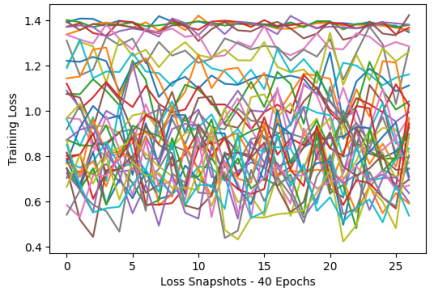


[C – EXP III – revert to CenterCrop on p-preproc, add more random transformations with p=1, add more epochs]
40 epochs, 20% train data per epoch
Run time: 2:40 hours

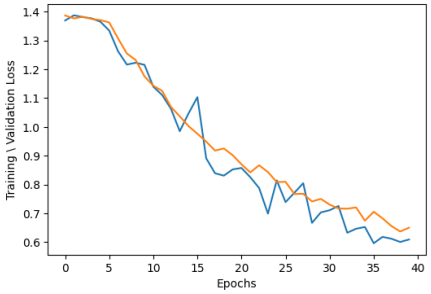
```
p_preproc = T.Compose([
    T.CenterCrop(176),
    T.ToTensor(),              # This normalizes all data to (0,1)
    ])

p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomHorizontalFlip(p=1),
    T.RandomResizedCrop(size=(176),antialias=None),
    T.RandomResizedCrop(size=(176),antialias=None),
    sharpness_transform,
    invert_transform,
    ]

TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.6502828578670314
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.6091721263558532
TRAIN/VAL END RESULTS: Accuracy: 72% (250/345)
```

```
Completed trial # 0 Final Training loss:  0.6502828578670314  Final Validation loss:
0.6091721263558532
Test Accuracy of class MildDemented: 20% (36/179)
Test Accuracy of class ModerateDemented: 100% (12/12)
Test Accuracy of class NonDemented: 75% (484/640)
Test Accuracy of class VeryMildDemented: 36% (163/448)

Overall accuracy of the network on the 1279 test images: 54 %
```

Train Loss behavior per epoch:



Train / Validation Loss through epochs:

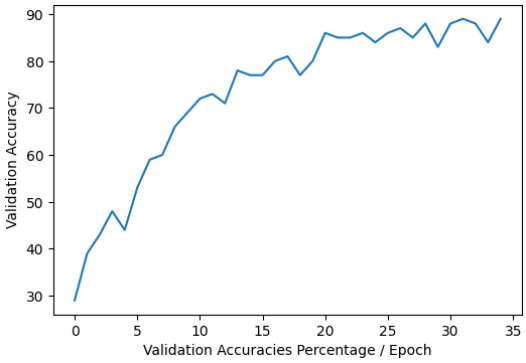


[C – EXP IV – more random/randomResize crops, include Gaussian transform:]
35 epochs, 20% train data per epoch
Run time: 2:20 hours

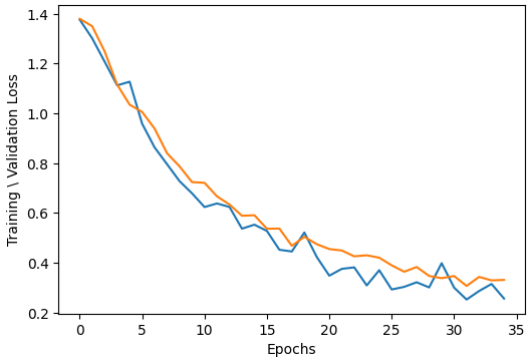
```
p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomHorizontalFlip(p=1),
         sharpness_transform,
    gaussian_transform,
    ]
```

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.3315591776261245
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.25669080549371337
TRAIN/VAL END RESULTS: Accuracy: 89% (310/345)
```

Validation Accuracy throughout epochs:



Train / Validation Loss throughout epochs:




```
Test Accuracy of class MildDemented: 36% (65/179)
Test Accuracy of class ModerateDemented: 66% ( 8/12)
Test Accuracy of class NonDemented: 58% (375/640)
Test Accuracy of class VeryMildDemented: 60% (273/448)
Overall accuracy of the network on the 1279 test images: 56 %
```

[C – EXP V - scaling and ratio augmentation on under-sampled classes:]
20 epochs, 20% train data per epoch
Run time: 1 hours

```
random_crop = T.RandomResizedCrop(size=(176), scale=(0.08, 1.0), ratio=(0.75, 1.25))
```

```
p_preproc = T.Compose([
    T.CenterCrop(176),
    T.ToTensor(),       # This normalizes all data to (0,1)
```

```
p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomVerticalFlip(),
    T.RandomResizedCrop(size=(176),antialias=None),
    sharpness_transform,
    gamma_transform,
    gaussian_transform,
    invert_transform,
    ]
```

```
END RESULTS: Achieved final Train Loss: 0.9583304969227107
END RESULTS: Achieved final Validation Loss: 1.0053230639129993
END RESULTS: Accuracy: 48% (126/259)
```

```
Completed trial # 0 ({'l_1': 64, 'l_2': 256, 'l_3': 256, 'l_4': 512, 'l_5': 1024, 'lr': 0.001,
'weight_decay': 0.0007, 'optim': 'SGD', 'momentum': 0.7, 'epochs': 20, 'batch_size': 1,
'input_size': (3, 176, 176), 'output_size': 4}, 1.0053230639129993, 50)
Best results: Loss:  1.0053230639129993 Accuracy:  50
```

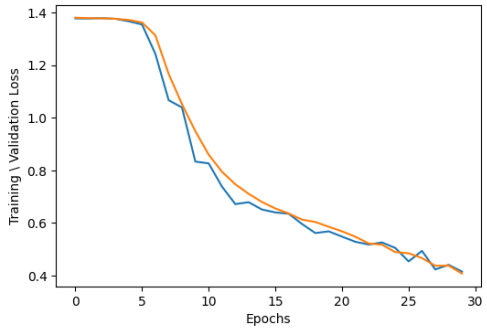
NETWORK IS NOT LEARNING ➔ we will not use any aspect ratio data manipulation on the training data.

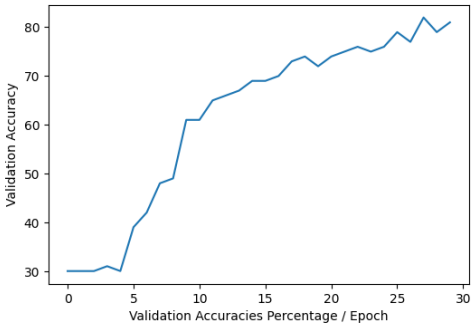
[D – Benchmark model – pytorch implementation]

We implemented the pytorch equivalent to the keras benchmark model described above. A second fully connected layer was added to our model, to provide an architecture that resembles LeNet 5
Here we were interested at testing the impacts of batch size and reshuffling of the trainset on the model performance. We tested a few scenarios with no reshuffling, and compared the results to matching scenarios, where new train / validation set loaders are randomly picked from the full dataset, every epoch:

[D - EXP I – single shuffle for train+val dataset]
30 Epochs. 45 min runtime

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.4079827176802327
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.4152900645377923
TRAIN/VAL END RESULTS: Accuracy: 81% (1407/1727)
```

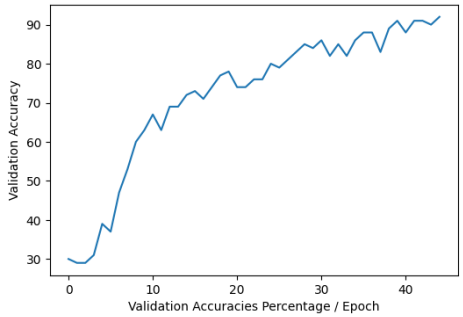


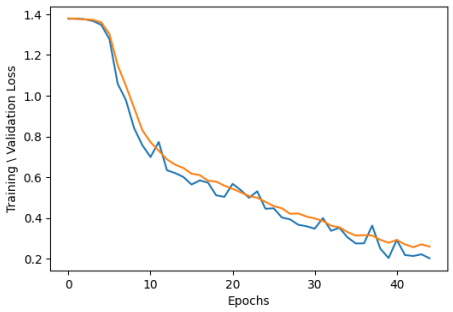



```
Test Accuracy of class MildDemented: 39% (71/179)
Test Accuracy of class ModerateDemented: 100% (12/12)
Test Accuracy of class NonDemented: 59% (381/640)
Test Accuracy of class VeryMildDemented: 57% (256/448)
Overall accuracy of the network on the 1279 test images: 56 %
```

[D - EXP II – new shuffle every epoch for same train+val dataset]
45 Epochs. 43 min runtime

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.25941040905007506
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.20156469089965026
TRAIN/VAL END RESULTS: Accuracy: 92% (1592/1727)
```

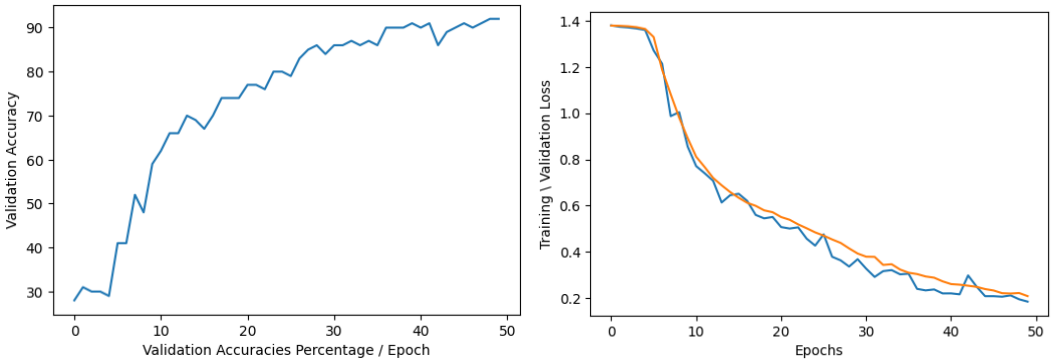





```
Test Accuracy of class MildDemented: 56% (101/179)
Test Accuracy of class ModerateDemented: 100% (12/12)
Test Accuracy of class NonDemented: 38% (244/640)
Test Accuracy of class VeryMildDemented: 77% (348/448)
Overall accuracy of the network on the 1279 test images: 55 %
```

[D - EXP III – new shuffle every epoch for same train+val dataset + Gamma transform]
50 Epochs. 48 min runtime

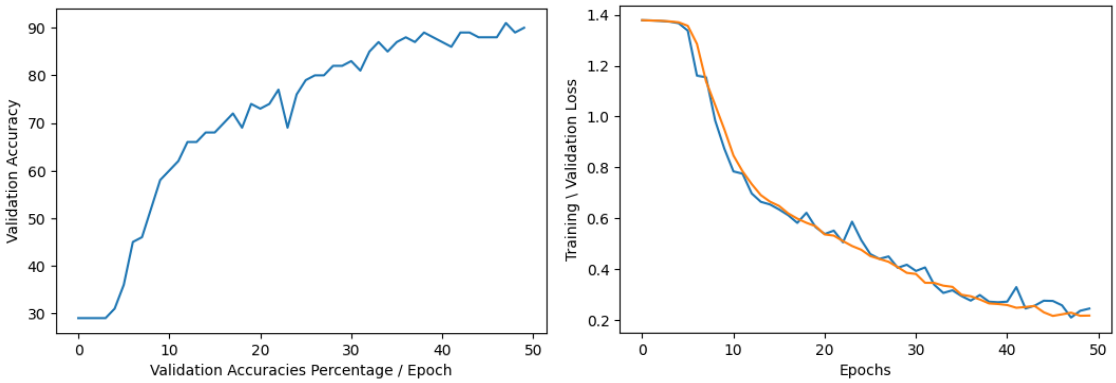
```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.20755004365189436
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.18364416726077154
TRAIN/VAL END RESULTS: Accuracy: 92% (1598/1727)
```




```
Test Accuracy of class MildDemented: 82% (147/179)
Test Accuracy of class ModerateDemented: 25% ( 3/12)
Test Accuracy of class NonDemented: 43% (276/640)
Test Accuracy of class VeryMildDemented: 66% (300/448)
Overall accuracy of the network on the 1279 test images: 56 %
```

[D - EXP IV – single shuffle for train+val dataset + Gamma Transform]
50 Epochs, 48 min runtime

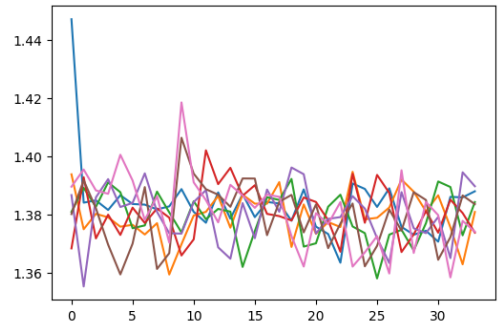
```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.21765590587051828
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.24516139697010642
TRAIN/VAL END RESULTS: Accuracy: 90% (1565/1727)
```




```
Test Accuracy of class MildDemented: 40% (73/179)
Test Accuracy of class ModerateDemented: 100% (12/12)
Test Accuracy of class NonDemented: 40% (256/640)
Test Accuracy of class VeryMildDemented: 84% (378/448)
Overall accuracy of the network on the 1279 test images: 56 %
```

[D - EXP V – try again to use ADAM instead of SGD. New shuffle every epoch for same train+val dataset]

Tried Learning rate = 0.01, 0.001 - NO learning curve, stopped both runs after 10 epochs.

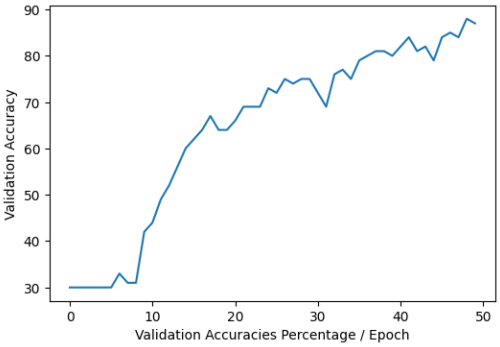


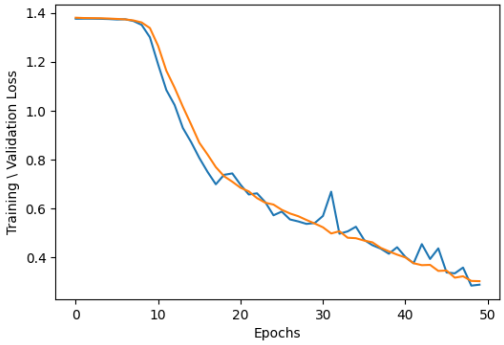
Loss per epoch

[D - EXP VI – single shuffle for train+val dataset, batch size = 8 (double what we've used so far)]

50 Epochs, 29 min runtime – significantly short

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.30215999425207785
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.28786914384949747
TRAIN/VAL END RESULTS: Accuracy: 87% (1518/1727)
```



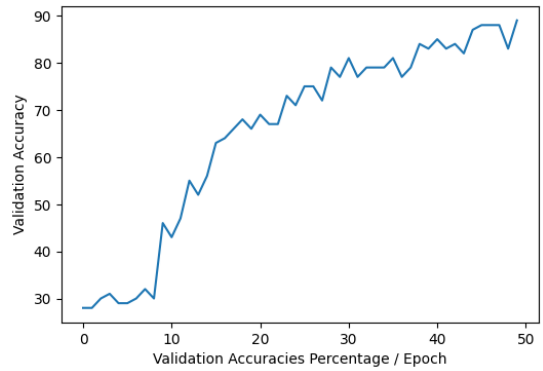


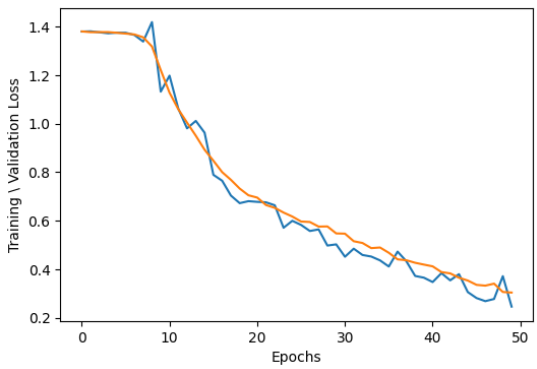

```
Test Accuracy of class MildDemented: 21% (39/179)
Test Accuracy of class ModerateDemented: 75% ( 9/12)
Test Accuracy of class NonDemented: 20% (134/640)
Test Accuracy of class VeryMildDemented: 91% (412/448)
Overall accuracy of the network on the 1279 test images: 46 %
```

[D - EXP VI – New shuffle every epoch for same train+val dataset, batch size=8]

50 Epochs, 29 min runtime

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.30382222839074935
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.24579443655279143
TRAIN/VAL END RESULTS: Accuracy: 89% (1553/1727)
```



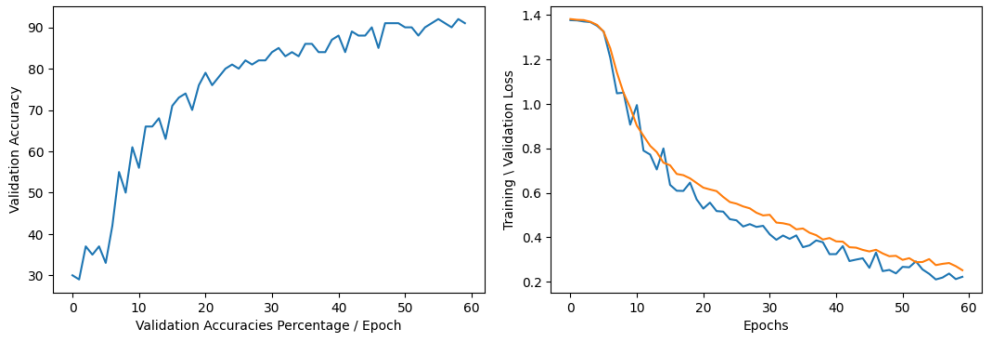


```
Test Accuracy of class MildDemented: 53% (95/179)
Test Accuracy of class ModerateDemented: 75% ( 9/12)
Test Accuracy of class NonDemented: 58% (377/640)
Test Accuracy of class VeryMildDemented: 59% (266/448)
Overall accuracy of the network on the 1279 test images: 58 %
```

➔ This is the most balanced classification results so far. We will continue with batch size = 8 and Random test/validation shuffle per each epoch. Also, more epochs are required.

[D - EXP VII – Remove second fully connected layer]
Instead of 1024 -> 50 -> 4 we move from 1024 -> 4. Dropout 0.25 after maxpool conv3, dropout 0.3 before fully connected: 60 epochs, 31 min runtime

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.2523704699502829
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.22281956811288925
TRAIN/VAL END RESULTS: Accuracy: 91% (1582/1727)
```

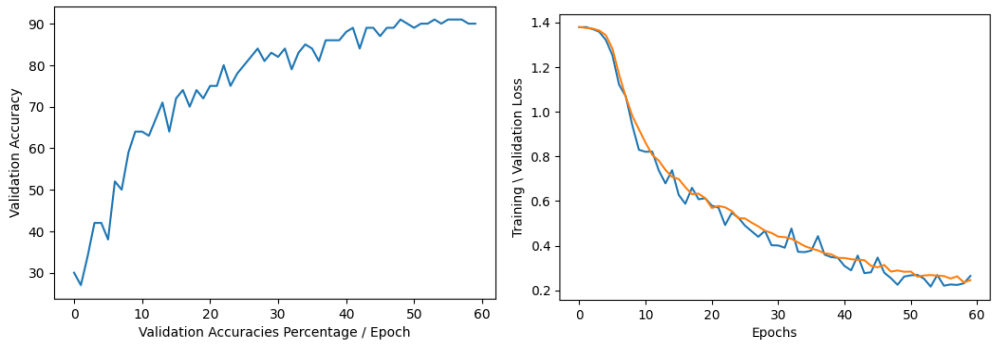



```
Test Accuracy of class MildDemented: 56% (102/179)
Test Accuracy of class ModerateDemented: 58% ( 7/12)
Test Accuracy of class NonDemented: 75% (482/640)
Test Accuracy of class VeryMildDemented: 21% (98/448)
Overall accuracy of the network on the 1279 test images: 53 %
```

➔ model learns much faster, but final classification is less accurate.

[D - EXP VIII – Remove second Dropout]

```
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.2454507477073262
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.2644051708420092
TRAIN/VAL END RESULTS: Accuracy: 90% (1560/1727)
```




```
Test Accuracy of class MildDemented: 94% (170/179)
Test Accuracy of class ModerateDemented: 75% ( 9/12)
Test Accuracy of class NonDemented: 38% (244/640)
Test Accuracy of class VeryMildDemented: 39% (179/448)
Overall accuracy of the network on the 1279 test images: 47 %
```

➔ good accuracy on augmented classes, bad on original samples. Fastest learning curve.
Next – try to get better understanding what this model is learning.

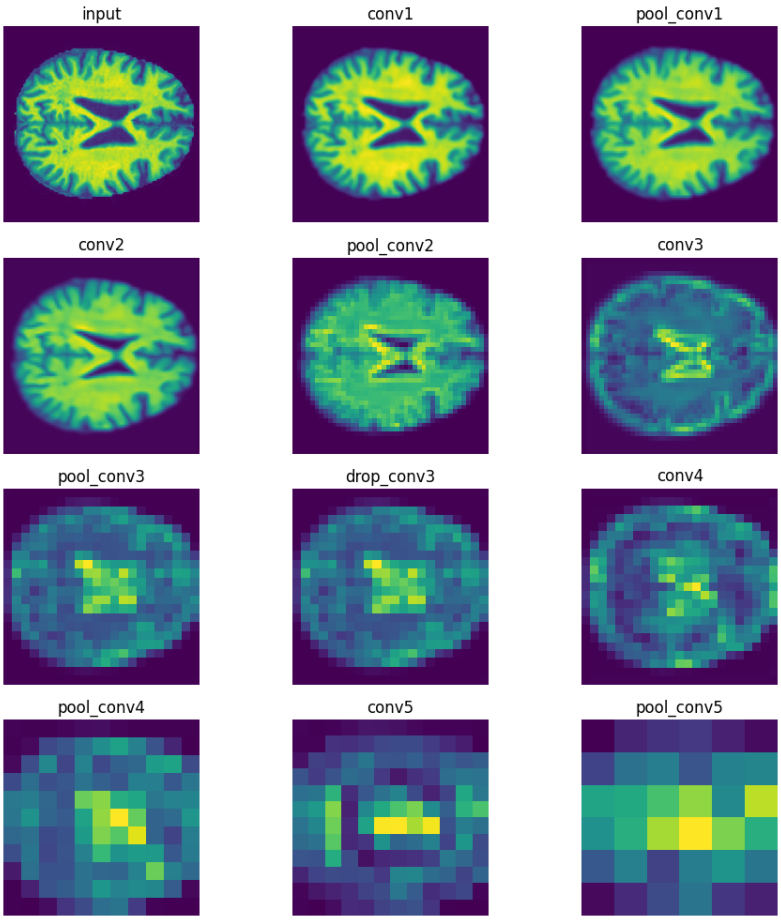
[E – feature extraction]

Frustratingly, due to lack of time, we didn't get to play with this part of the project as much as we wanted to.

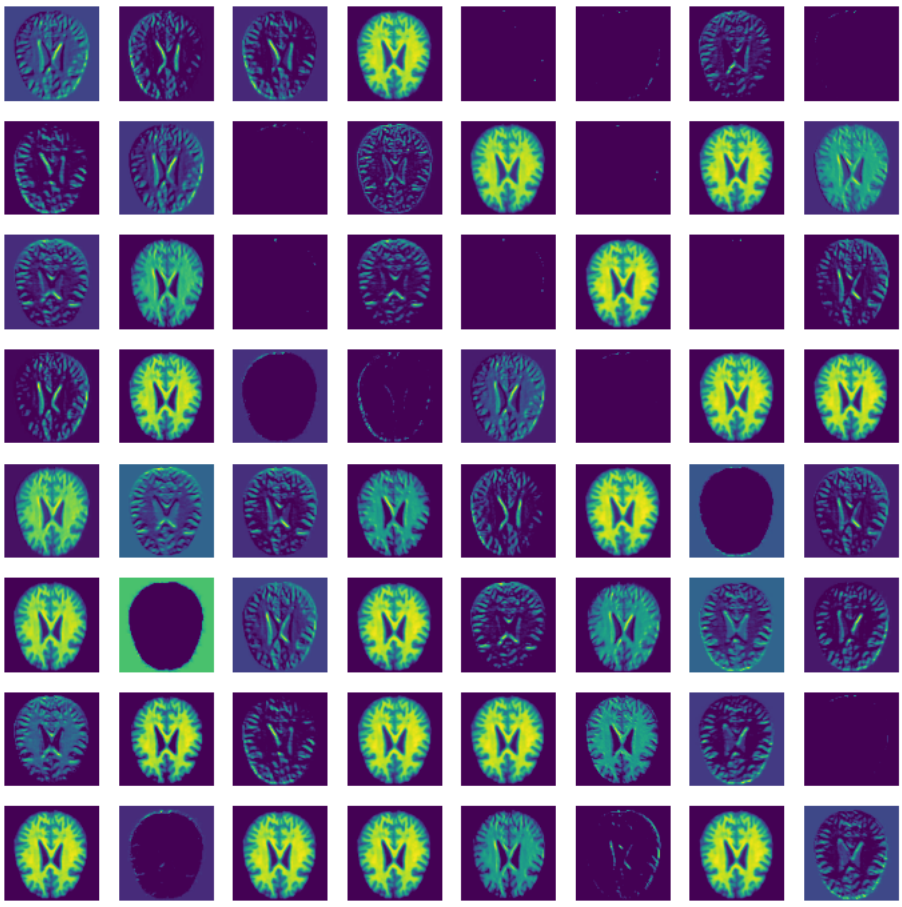
[feature extraction – model D-VIII]
100 epochs, runtime 54 min

```
Node names: ['x', 'float', 'conv1', 'relu', 'pool', 'conv2', 'relu_1', 'pool_1', 'conv3',
'relu_2', 'pool_2', 'dropout1', 'conv4', 'relu_3', 'pool_3', 'conv5', 'relu_4', 'pool_4',
'dropout2', 'flatten', 'fc1']
```

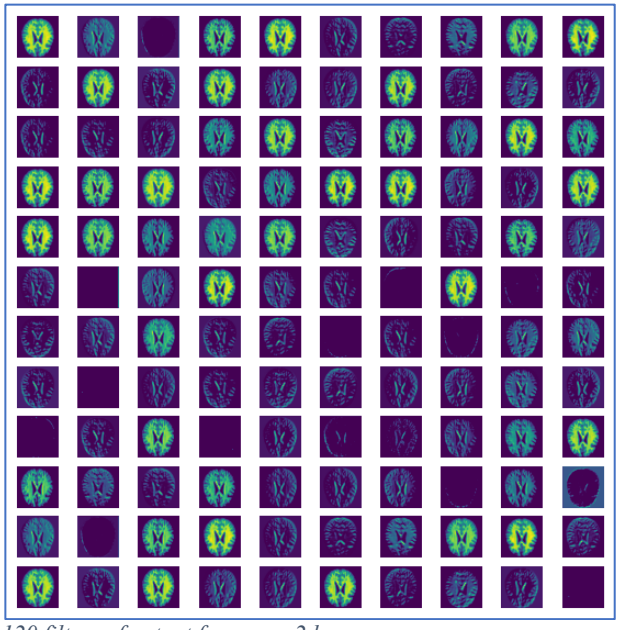
```
# convolutional layers
self.conv1 = nn.Conv2d(in_channels=3, out_channels=l_1, kernel_size=5, padding=2)
self.conv2 = nn.Conv2d(in_channels=l_1, out_channels=(l_1*2), kernel_size=3, stride=1, padding=1)
self.conv3 = nn.Conv2d(in_channels=(l_1*2), out_channels=(l_1*4), kernel_size=3, stride=1,
padding=1)
self.conv4 = nn.Conv2d(in_channels=(l_1*4), out_channels=(l_1*8), kernel_size=2, stride=1,
padding=1)
self.conv5 = nn.Conv2d(in_channels=(l_1*8), out_channels=(l_1*16), kernel_size=2, stride=1,
padding=1)
```



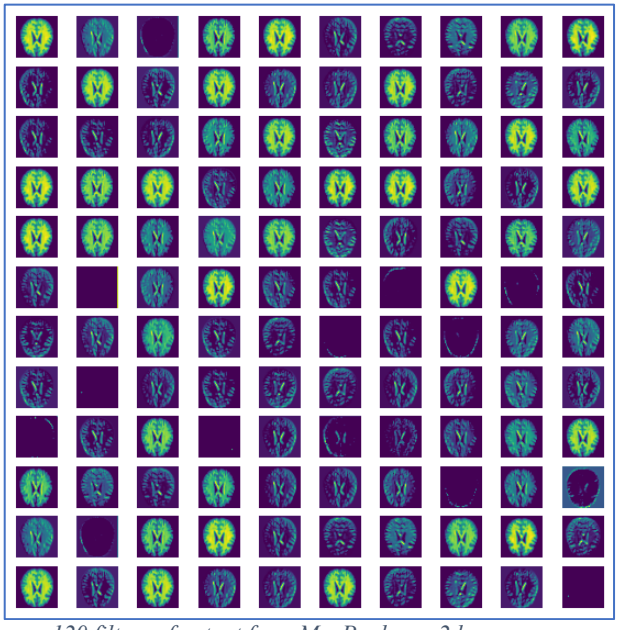
feature extraction after each layer



All 64 filters of output from MaxPool conv1 layer



120 filters of output from conv2 layer



120 filters of output from MaxPool conv2 layer

[F – Final test runs]

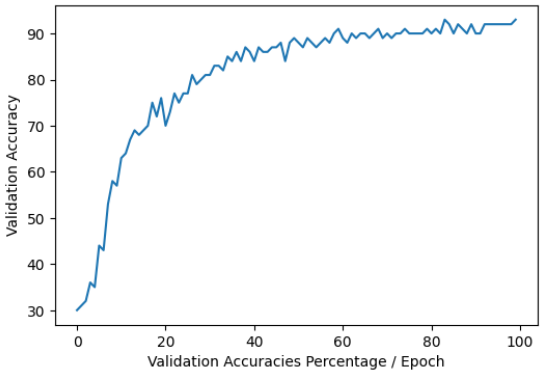
Common environment settings: 100 epochs, no dataset shuffle, batch size=8

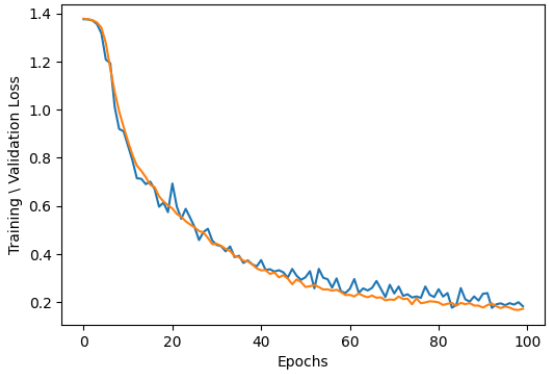
```
p_manip_samples = [
    T.Lambda(lambda img: img),
    T.RandomHorizontalFlip(p=1),
    T.RandomResizedCrop(size=(176),antialias=None),
    T.RandomCrop(size=(176)),
```

[F - EXP I – initial number of features for conv1 – 64]

*Same layer architecture as the benchmark keras model.
Runtime 55 min

```
[100,   200] Validation loss: 0.152 Accuracy: 93% (1493/1600)
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.1725515825527071
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.18283609839434373
TRAIN/VAL END RESULTS: Accuracy: 93% (1607/1727)
```



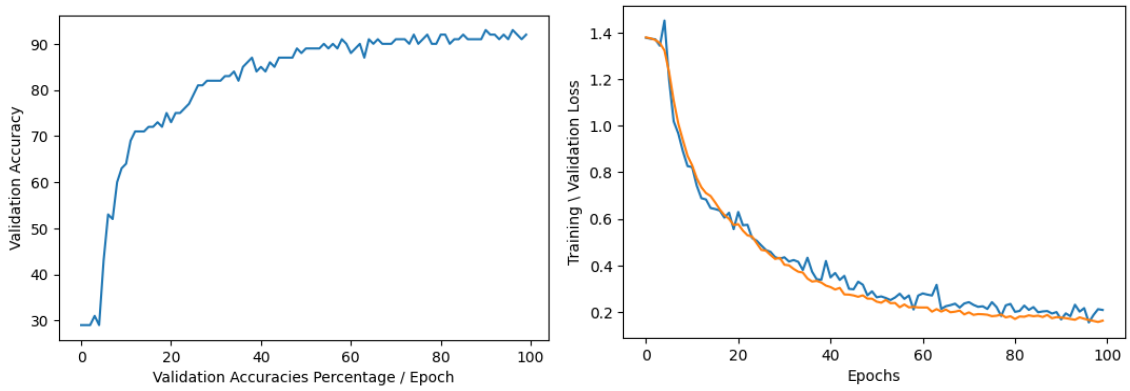



```
Completed trial # 11 Final Training loss:  0.1725515825527071  Final Validation loss:
0.18283609839434373
Test Accuracy of class MildDemented: 79% (142/179)
Test Accuracy of class ModerateDemented: 83% (10/12)
Test Accuracy of class NonDemented: 62% (401/640)
Test Accuracy of class VeryMildDemented: 51% (230/448)
Overall accuracy of the network on the 1279 test images: 61 %
```

[F - EXP II – initial number of features for conv1 – 128]
Looking at the feature maps, it seems that after maxpooling conv3 layer, we have already reduced the resolution and granularity of the samples significantly. We may want to actually want to start from a bigger feature count. We doubled the input number of features from 64 to 128.
Runtime 2:24 hrs

```
[100,   200] Validation loss: 0.188 Accuracy: 92% (1476/1600)
TRAIN/VAL END RESULTS: Achieved final Train Loss: 0.16258255692540574
TRAIN/VAL END RESULTS: Achieved final Validation Loss: 0.20850415428598085
TRAIN/VAL END RESULTS: Accuracy: 92% (1593/1727)
```

This network has learned much faster:



Yet final accuracies on the test set have actually decreased:

```
Completed trial # 11 Final Training loss:  0.16258255692540574  Final Validation loss:
0.20850415428598085
Test Accuracy of class MildDemented: 92% (166/179)
Test Accuracy of class ModerateDemented: 75% ( 9/12)
Test Accuracy of class NonDemented: 47% (305/640)
Test Accuracy of class VeryMildDemented: 43% (193/448)
Overall accuracy of the network on the 1279 test images: 52 %
```

Again we are seeing better classification on the augmented classes, and poor classification on the original samples.

➔ Future work would be to augment samples for all classes to create the train/val set before feeding it to the model.