AML Report - HW2

Sapienza, Università di Roma, AY 20/21

November 24, 2021

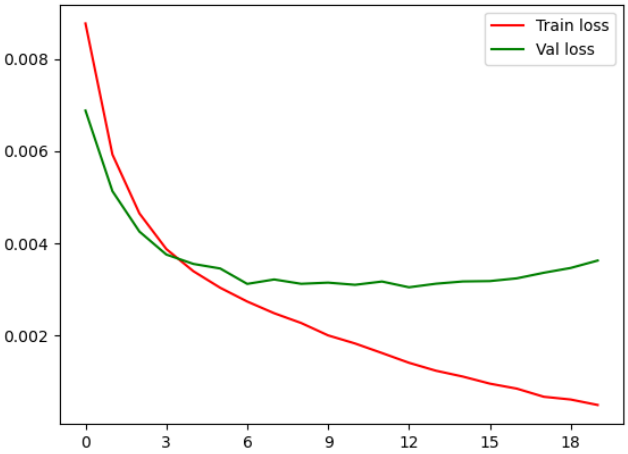
Mattia Capparella
capparella.1746513@studenti.uniroma1.it

Flavia Penta de Peppo
pentadepeppo.1758942@studenti.uniroma1.it

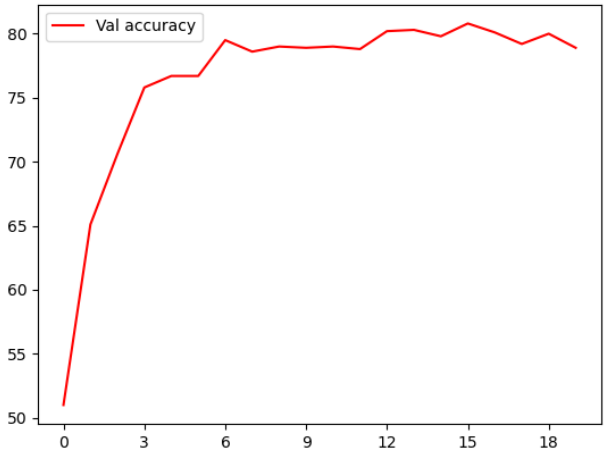
Antonio Zappia
zappia.1747573@studenti.uniroma1.it

Q1 Implementing a CNN with PyTorch

a) Training, Validation Losses and Accuracies of CNN_5



(a) Train/Val Losses



(b) Validation Accuracy

Figure 1: CNN_5 performances

b) CNN_5 Size

```
Model size:  7678474
```

(a) Model Size

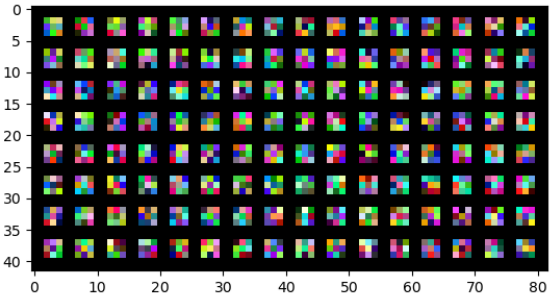
```
Model Size:  7682826
```

(b) Model Size with BN

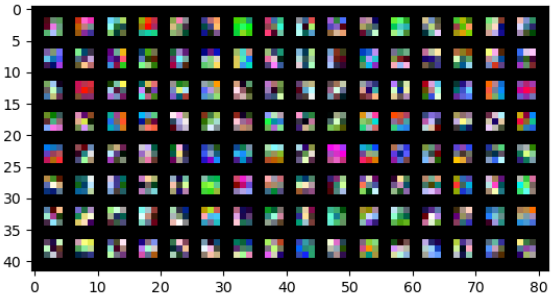
Figure 2: CNN_5 Sizes

c) Filters Visualization

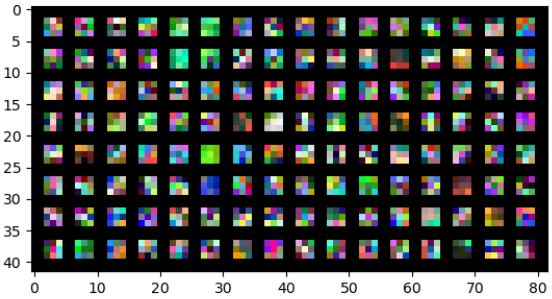
As expected, the filters before training appear *"non-sense"* colorful little squares filled at random. After different experiment instead, most of them start to acquire a sort of *"logic"*, *e.g.* we can note that some of them look for patches of shades of a specific color (*e.g.* green shades), while others seem to highlight vertical edges, and some other filter for gradients between different colors (cyan to orange or red to purple).



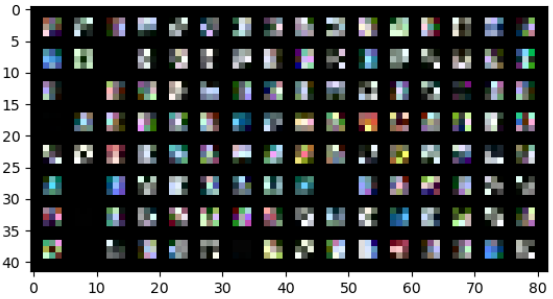
(a) Filters before Train



(b) Filters after Train



(c) Filters after Train with BN



(d) Filters after Train with Augmentation

Figure 4: Comparing the filters before and after Train

Q2 Improving training of CNN

a) Batch Normalization (**BN**)

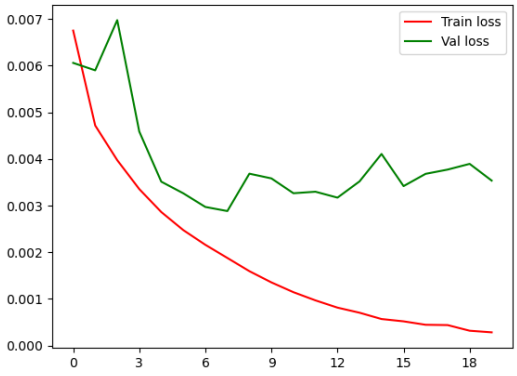
To understand the behavior and the differences between a CNN with and without BN, it is useful to compare both the Train/Val Loss plots and the Accuracy ones.

From 1a and 5a we can observe how the Train Loss descents behave smoothly and start converging around the 12^{th} epoch, while the Validation Losses behaves in different manner:

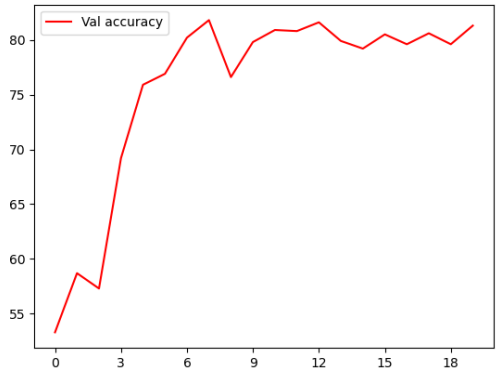
1. without BN the descent is smooth, but stabilizes after few (5) epochs and starts increasing after the 12^{th} epoch;

2. with BN instead, the Val Loss behaves more jiggly as noted also in Training Deep Neural Networks Without Batch Normalization, in which the authors claim that a meaningful Loss descent requires at least 70 epochs.

However, since in (2) the gap between Train and Val losses remains constant for more epochs, we can observe from 5b a better performance in generalization (less overfit).



(a) Train/Val Losses with BN



(b) Validation Accuracy with BN

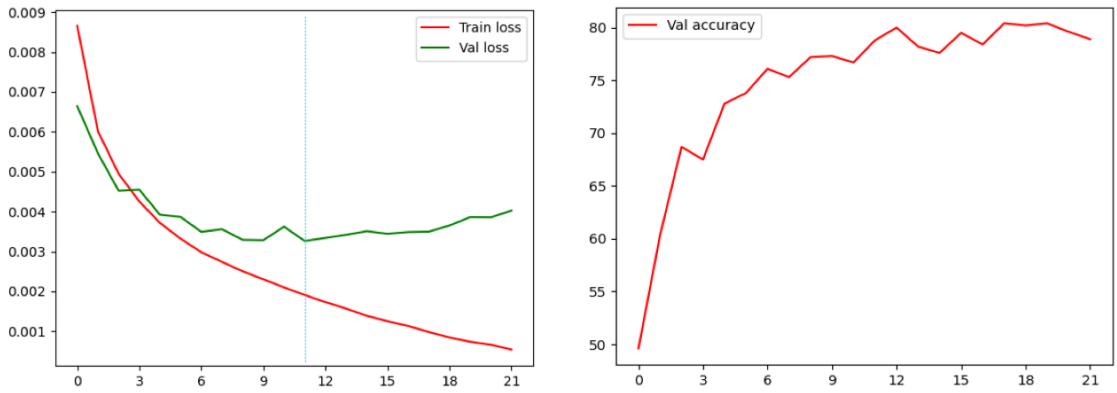
Figure 5: CNN_5 performances with BN

b) Early Stopping (**ES**)

In this point we used ES, a form of regularization useful for avoiding overfitting, since it allows to specify an arbitrary large (in this case, 50) number of training epochs and stop training once the model performance stops improving on the validation dataset. For this purpose we monitored both the *Validation Loss* and *Validation Accuracy*: since we are dealing with a ***balanced classification problem*** we choose the *best model* according to the metric by which it is evaluated in the domain, *i.e.:* the Accuracy. To implement the ES, we have set a *patience* = 10 and a *tolerance* = 1e-5, that is:

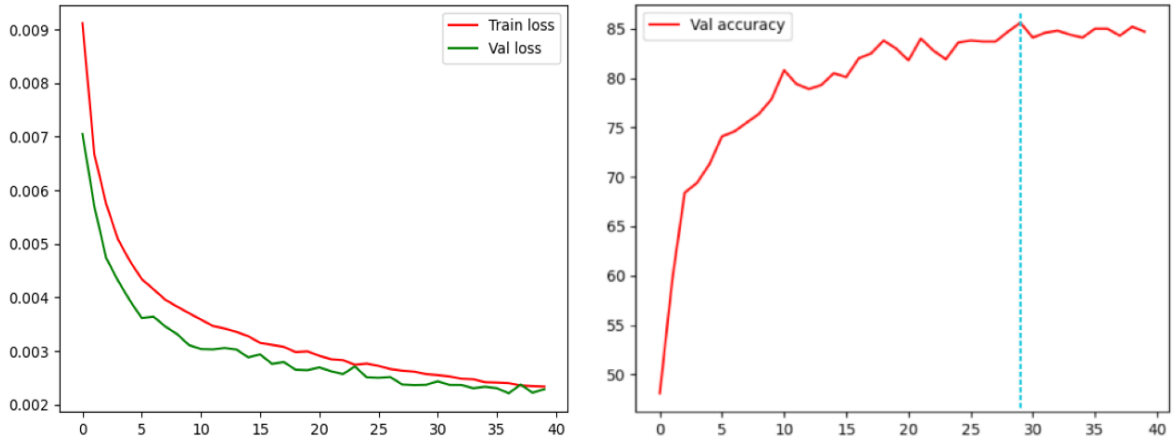
if the ***validation loss/accuracy*** increases, remains equal or decreases only by a small value ϵ = 1e-5 within 10 epochs, then stop training!

Due to the randomness, we trained the models multiple times. In 6 we can see the outputs of one of the experiments; the *dotted lines* represent the epoch in which we obtained the best model: with BN we both obtained the *best model* and stop the *training phase* earlier w.r.t. the model without BN.



(a) Train/Val Losses without BN (b) Val_Acc without BN

Figure 6: ES monitoring Val_Loss

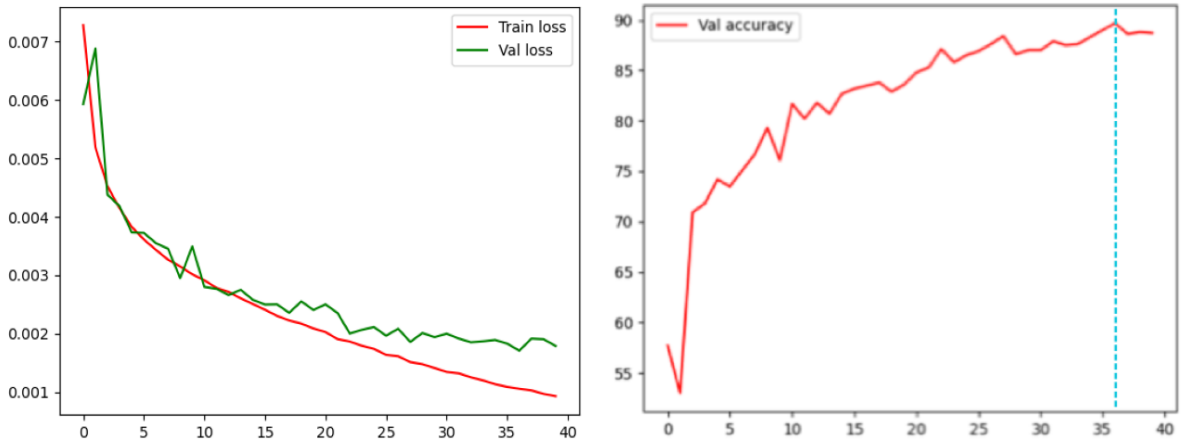


(a) Train/Val Losses without BN (b) Val_Acc without BN

Figure 7: ES monitoring Val_Acc

We found out that for this particular case, in which the classes are balanced, by monitoring the Val_Acc we could max out the limit of "20ish epochs", zeroing the gap between the losses and then avoid overfitting: by doing so, we scored 85% Val_Acc.

Eventually, we trained our model applying the BN scoring higher accuracy and, as expected, in less epochs.



(a) Train/Val Losses with BN

(b) Val_Acc with BN

Figure 8: ES+BN monitoring Val_Acc

Q3 Implementing the feedforward model

a) Data Augmentation (**DA**)

As pointed out in Deep Learners Benefit More from Out-of-Distribution Examples, deep architectures benefit much more from DA than shallow architectures: we have experienced similar results by applying some standard *geometric and color transformations* from torchvision library, such as:

- RandomEqualize(p=1)
- ColorJitter()
  - brightness(0.2, 0.8)
  - contrast(0.2, 0.5)
  - saturation(0.1, 0.3)
  - hue(-0.2, 0.2)
- Grayscale(0.2)
- RandomAdjustSharpness(scale=2, p=0.7)
- RandomPerspective(scale=0.2, p=0.6)

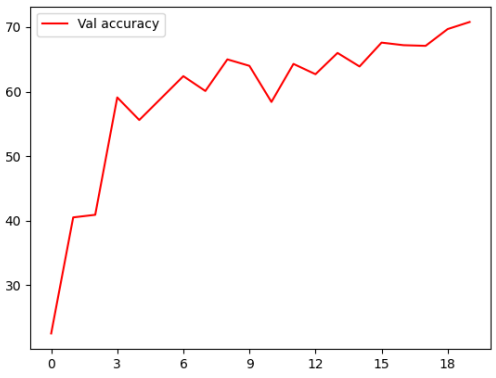
- RandomEqualize(p=1)
- ColorJitter()
  - brightness(0.3, 0.8)
  - contrast(0.3, 0.7)
  - saturation(0.1, 0.4)
  - hue(-0.3, 0.3)
- Grayscale(p=0.4)
- RandomAdjustSharpness(scale=1, p=0.5)
- RandomPerspective(scale=0.7, p=0.5)

- RandomEqualize(p=1)
- RandomCrop(32, padding=4)
- RandomErasing(p=0.5, scale=(0.39, 0.40), ratio=(1.,1.))
- RandomAffine(degrees=(-20, +20))

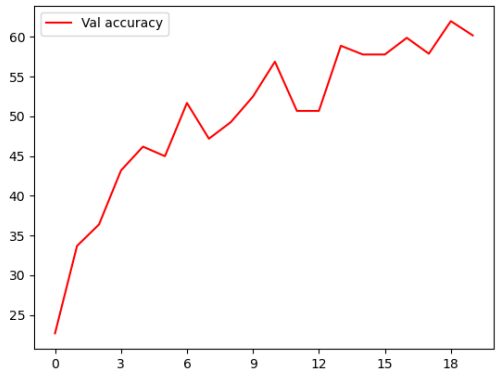
We tried out different versions (using ***different hyperparameters***) of these configurations and we observed that the first two were probably a bit "too aggressive" and the input images were altered in a fashion such that incapacitated our shallow model. A good compromise was achieved by following the *augmentation pipeline* proposed in Improved Regularization of Convolutional Neural Networks with Cutout that merged in our third proposal.

With these configuration and 20 epochs, we managed to achieve between 50% $\sim$ 70% accuracy on both Validation and Test Set.

Note: although RandomHFlip() seems to be an important and valid DA, we preferred to omit it in this stage since it was already present in the *ex3_pretrained* code.



(a) Validation Accuracy 1^{st} Configuration



(b) Validation Accuracy 3^{rd} Configuration

Figure 9: CNN_5 performances with DA

b) Dropout (**D**)

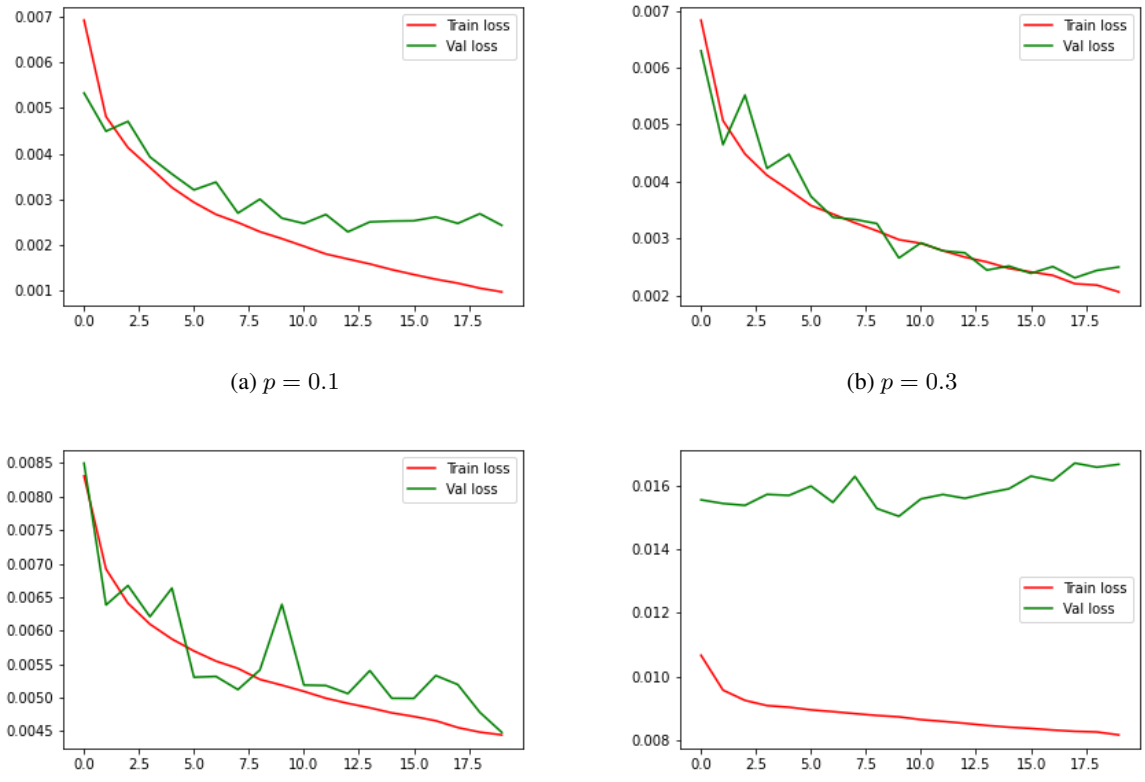
While D was found to be very effective at regularizing fully-connected layers, it appears to be less powerful when used with convolutional layers. The reasons behind that are:

1. the convolutional layers already have much fewer parameters than FC-layers, and require less regularization

2. neighbouring pixels in images share much of the same information. If any of them are dropped out, the information they contain will likely still be passed on from the neighbouring pixels that are still active.

For these reasons, D in convolutional layers simply acts to increase robustness to noisy inputs, rather than having the same model averaging effect that is observed in FC-layers.

We achieved the best result, ***i.e.*** small losses and low overfit, with *p=0.3*, the same value also used in Improved Regularization of Convolutional Neural Networks with Cutout



(a) $p = 0.1$

(b) $p = 0.3$

(c) $p = 0.7$

(d) $p = 0.9$

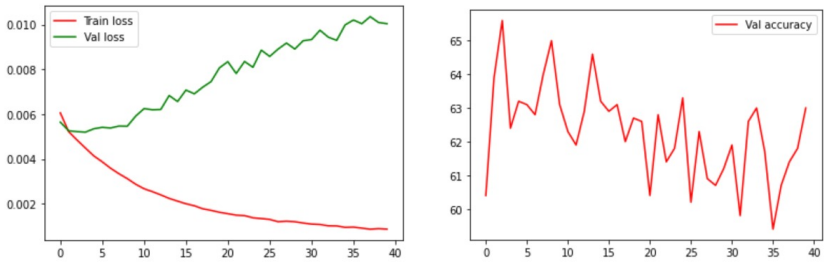
Figure 10: Train/Val Losses with D

Q4 Pretraining and Transfer Learning with CNN

a) Partial Finetuning Pretrained VGG_11_bn

The only DA we applied to the *train dataset* fed to the models was the one already inserted in the code, *i.e. RandomHorizontalFlip(p=0.5)*. This was due to the fact that when we trained with the DA we used in Section Q3.a we achieved much lower scores. However, for completeness these scores are inserted at the end of the report.

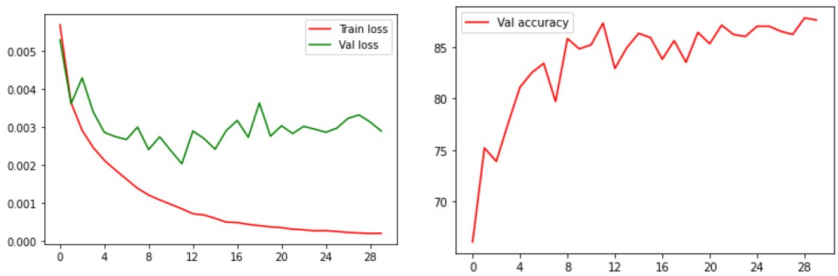
With *fine tuning*, we achieved the expected Accuracy scores, ranging between 60% and 65%, and as can be seen from 11a we are also overfitting the data after very few epochs.



(a) VGG_11_bn Train/Val Losses (b) VGG_11_bn Val_Acc

Figure 11: VGG_11_bn Partial Finetuning

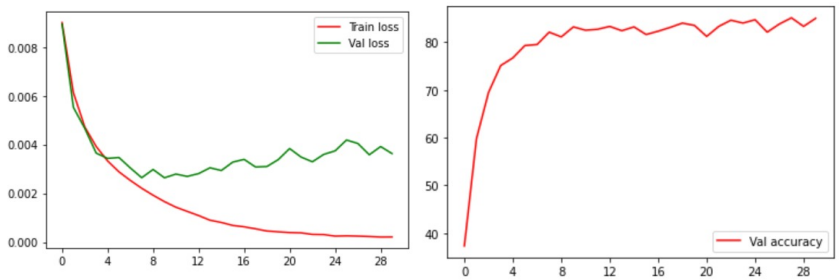
b1) Pretrained VGG_11_bn *finetuned*



(a) VGG_11_bn Train/Val Losses (b) VGG_11_bn Val_Acc

Figure 12: VGG_11_bn Total Finetuning

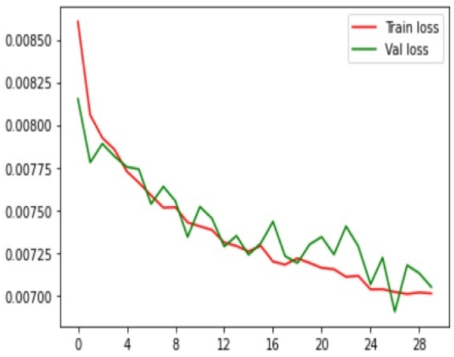
b2) Pretrained VGG_11_bn *trained from scratch*



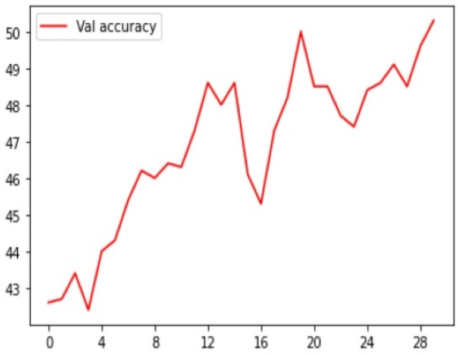
(a) VGG_11_bn Train/Val Losses (b) VGG_11_bn Val_Acc

Figure 13: VGG_11_bn Training From Scratch

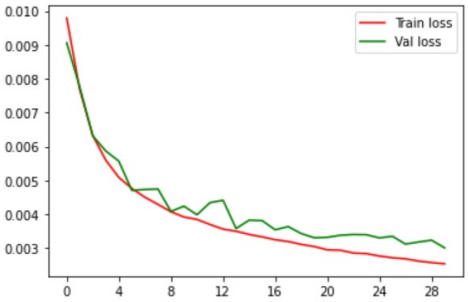
The results obtained with Q3.a DA are:



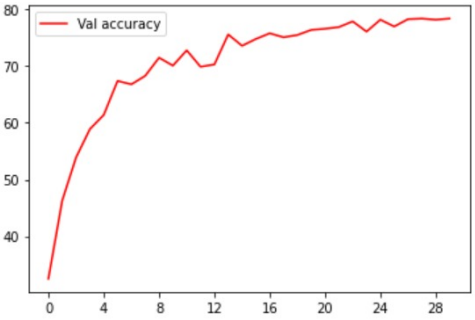
(a) Losses with Partial Finetuning



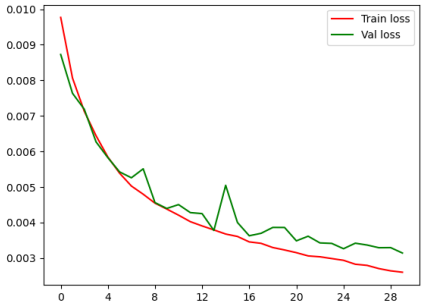
(b) Accuracy with Partial Finetuning



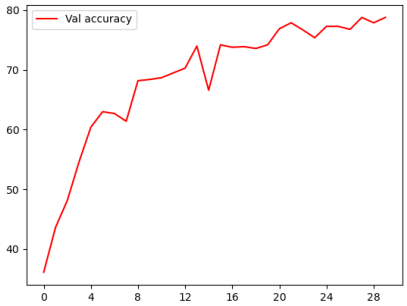
(c) Losses with Total Finetuning



(d) Accuracy with Total Finetuning



(e) Losses with Training from Scratch



(f) Accuracy with Training from Scratch

Figure 14: Transfer Learning with DA