INFOMCV PR4

Alexander Alexander, 6272932, a.p.apers@uu.nl
Robert Oost, 6922856, r.w.oost@students.uu.nl

March 2022

1 Summary

In this report we explore various properties of a standard Convolutional Neural Network (CNN) trained to classify images from the Fashion MNist dataset. We start off comparing a baseline model to four variant models during the training/validation phase, after which we take a closer look at two final models. Model performance was not the main priority, and instead we focus on the effects of key changes in model architectures.

2 Method

2.1 Dataset preparation

We started by splitting the training set into a smaller train (80%) and validation set (20%). However, because we relatively quickly switched to using K-fold validation we didn't end up using the original validation set. We normalised to greyscale pixel values so that all values are between 0 and 1 inclusive. We left the images as 28×28.

2.2 Baseline Model

This is the baseline model that we made is a model with 7 layers. The red layers indicate the layers that count towards 7.
(input) → Convolution→ ReLU → MaxPool → BatchNorm → Convolution → ReLU → MaxPool → BatchNorm → Flatten → Dense → ReLU → Dense (output)

We also have the Tensorflow summary print out.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 re_lu (ReLU)                (None, 28, 28, 32)        0

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 batch_normalization (BatchN  (None, 14, 14, 32)       128
 ormalization)

```

```
conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

re_lu_1 (ReLU)              (None, 14, 14, 64)        0

max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
2D)

batch_normalization_1 (Batc  (None, 7, 7, 64)         256
hNormalization)

flatten (Flatten)           (None, 3136)              0

dense (Dense)               (None, 128)               401536

re_lu_2 (ReLU)              (None, 128)               0

dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 422,026
Trainable params: 421,834
Non-trainable params: 192
_________________________________________________________________
None
```

We used a very standard ordering of layers for a CNN. It starts with applying the convolution operation on the input which is a linear operation to extract features (lower-level features for the first layer). The result of this operation is a stack of feature maps The number of feature maps in this stack will be equal to the number of filters the use in the layer. In the first convolutional layer we use 32 filters. We used a rather small filter size of 3×3 applied with a stride of 1 both horizontally and vertically. The padding parameter was set to *same*, which means that padding is applied so that the output height and width will be equal to the input height and width.

To introduce non-linearity into the model we use the commonly used rectified linear unit (ReLU) activation function. This will set all units in the feature maps that have a value smaller than 0 to 0 and leave the other values as is.

Then we apply maxpooling to reduce the dimensions of the tensors. This has three main advantages. One is to extract the most important features from the feature maps. This also improves computational load. The final advantage is to introduce some level of invariancy to the network. Using maxpooling can improve the network's ability to detect objects independent of object translation, rotation and scale. We set the pool size to 2×2 and left the stride parameter to the default which is equal to the pool size.

After this we applied batch normalisation. After so many operations it's important to apply some sort of normalisation to make sure the values stay within a certain range. Batch normalisation can speed up the point of convergence and can even improve the performance. We left the parameters for this at the default values.

Then we did another round of convolution, this time with 64 filters to produce more feature maps but with the same filter sizes, strides and padding. Then ReLU again and maxpooling with the same settings as before followed by batch normalisation.

Finally, we assume that the convolutional layers have extracted the most relevant features. Now the input is flattened and passed to a fully connected layer with 128 units. A final ReLU is applied before passing the data to the output layer with 10 nodes corresponding to the 10 classes. We didn't apply a softmax layer to the output. This is done automatically by our loss function.

2.3 Model Variants

We made four variants based on the baseline model. All of these models are identical to the baseline model except for where specified.

- **Learning rate variant**
  This model lowered the learning rate to $1 \cdot 10^{-4}$. Theoretically, this means the network would update its weights less at each timestep which means that it converges more slowly. However, we expect that it converges in a better local optimum because it has more time to explore the loss landscape.

- **Dropout variant** This model add dropout after the two maxpooling layers and after the fully connected layer. Dropout is a regularisation technique that turns off a portion of the neurons during training to prevent overfitting. After the pooling layers, dropout was applied with a rate of 0.25 and after the dense layer with a rate of 0.5. In the original dropout paper, [1], dropout was applied to the fully connected layers, which is why we applied most of the dropout there. However, more recent research, [2], shows that adding dropout after a convolutional layer can also help. We expect that using dropout will mitigate some of the overfitting problems.

- **Softpool variant** This model replaces all the maxpool layers in the model with softpool layers as described in [3]. We expect that the usage of softpool may improve the model by preserving information that would otherwise be lost during pooling.

- **Leaky ReLU variant** To counter the problem of dead neurons caused by standard ReLU, we tried using Leaky ReLU. It is defined as

$$\text{Leaky ReLU}(x) = \begin{cases} -\alpha x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

  where α is 0.1 in our case. This prevents neurons from fully dying and being set to 0 as they will always still have a small negative value which allows to pass some information about x. We expect performance to be slightly better.

2.4 Training

All models were trained for 10 epochs with the Adam optimiser and a learning rate of $1 \cdot 10^{-3}$. The loss function that we used is categorical crossentropy which is suitable for a classification problem like this. By setting the parameter *from_logits* to *true* the loss function automatically applies a softmax to the output to transform it into a probability distribution. We trained with a batch size of 32. For the first choice task, We implemented K-fold cross validation with $K = 10$. The plots will show the mean metrics along with the standard deviation over the 10 runs.

3 Results

3.1 Cross Validation Variants

The baseline model clearly shows signs of overfitting as the training loss continues to drop while the validation loss starts to go up after epoch 2. After epoch 2 it would be good to do early stopping. The validation accuracy is 91.5-91.8% (depending on epoch) which is relatively high.

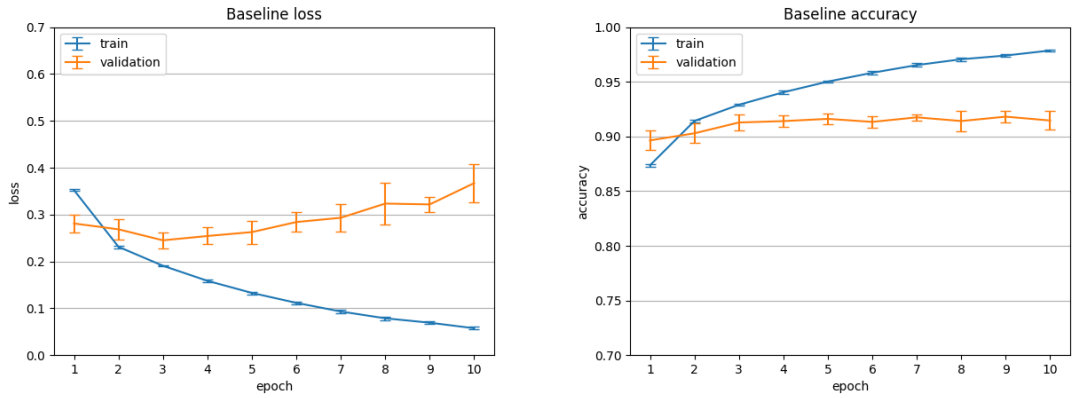

Figure 1: Baseline Model

It appears that changing the learning rate didn't really have a big effect on training.

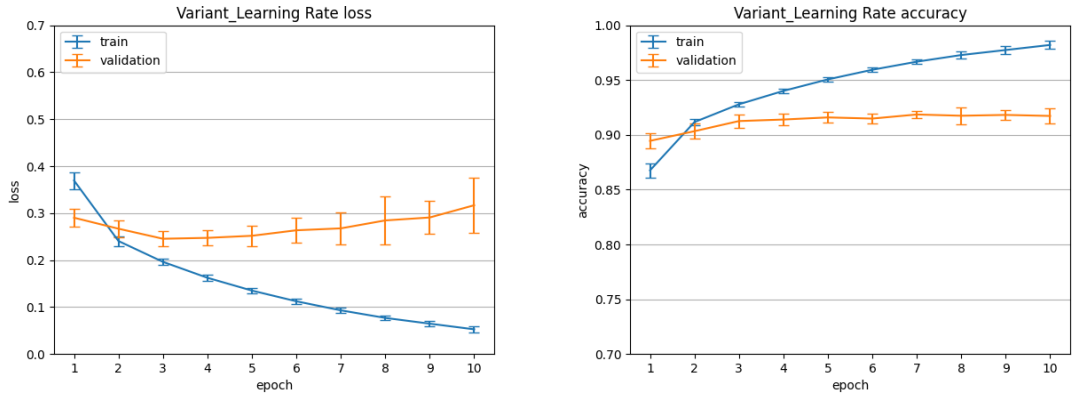

Figure 2: Learning Rate Variant

Dropout significantly increases the standard deviation of the performance during training. This is because we are purposefully turning off neurons randomly which can have different effects. We can also notice that where the baseline is massively overfitting at the final epochs indicated by the validation loss going up, the dropout variant validation loss goes up more slowly.
The softpool plots look almost identical to the baseline model.
The same for the Leaky ReLU. It doesn't significantly impact performance and looks very similar to the baseline model.

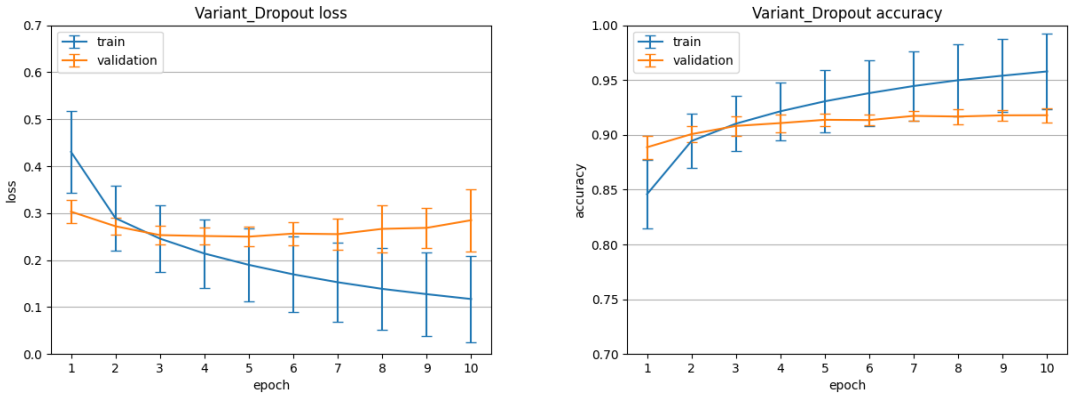

Figure 3: Dropout Variant

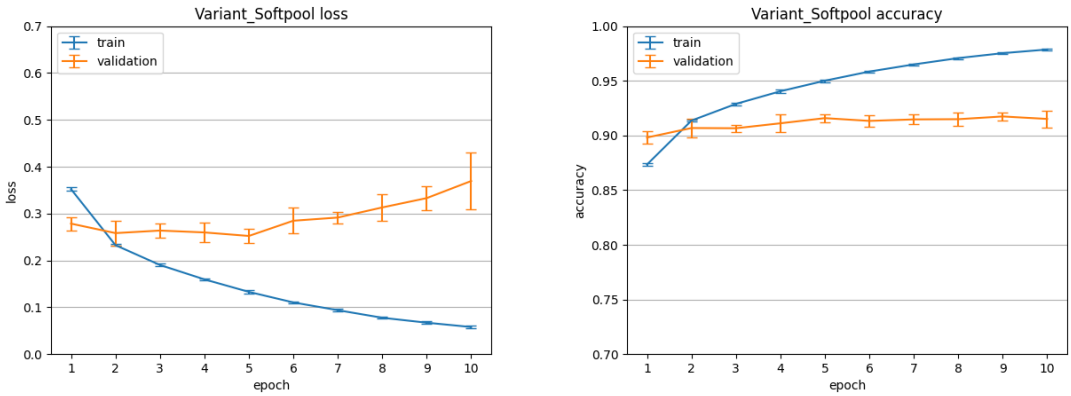

Figure 4: Softpool Variant

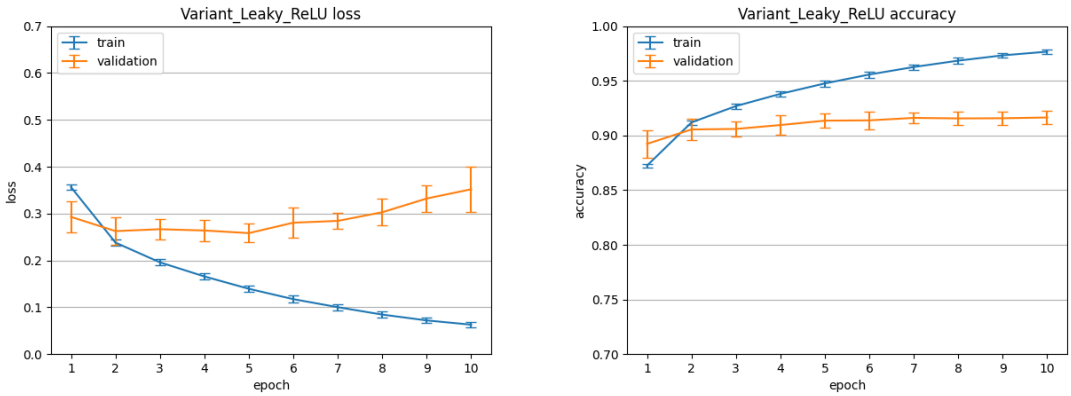

Figure 5: Leaky ReLU Variant

	Mean training accuracy	Mean validation accuracy	Epoch
Baseline	0.974	0.918	9
Variant Learning Rate	0.967	0.919	7
Variant Dropout	0.958	0.918	10
Variant Softpool	0.975	0.917	9
Variant Leaky ReLU	0.977	0.916	9

Table 1: Maximum accuracies over 10 epochs, and the epochs at which they were achieved.

	Mean training accuracy	Mean validation accuracy
Baseline	0.979	0.915
Variant Learning Rate	0.982	0.917
Variant Dropout	0.958	0.918
Variant Softpool	0.979	0.915
Variant Leaky ReLU	0.977	0.916

Table 2: Final accuracies after 10 epochs.

We recorded both the accuracies after 10 epochs and the maximum accuracies. In addition to this, we recorded the epoch at which the model reached its maximum accuracy score. Comparing these values gives a good indication for the amount of epochs that should be used in further training for each of these models. As expected the loss is lower and the accuracy is higher on the training set but this doesn't give an indication of the performance of the model on unseen data. When looking at the maximum validation accuracy values, we see that they are relatively close together. The Learning Rate Variant has the highest performance with 91.9% followed by the baseline and dropout with 91.8%, softpool with 91.7% and finally Leaky ReLU with 91.6%. However, keep in mind that these values are only based on 10 folds and might have some uncertainty. After 10 epochs the dropout variant performs the best with 91.8% most likely because it overfits the least of all the models after training for 10 epochs.

3.2 Test Results

The final two models we chose with the highest validation performance are the baseline model and the learning rate variant. As we can see the models are still overfitting and there is some more jitter in the test loss as it goes up and down. A good point to stop would be around epoch 3 to prevent overfitting. For the second choice task, we implemented a learning rate scheduler which halves the learning rate at epoch 5. Using this scheduler, we trained a third model using the baseline.

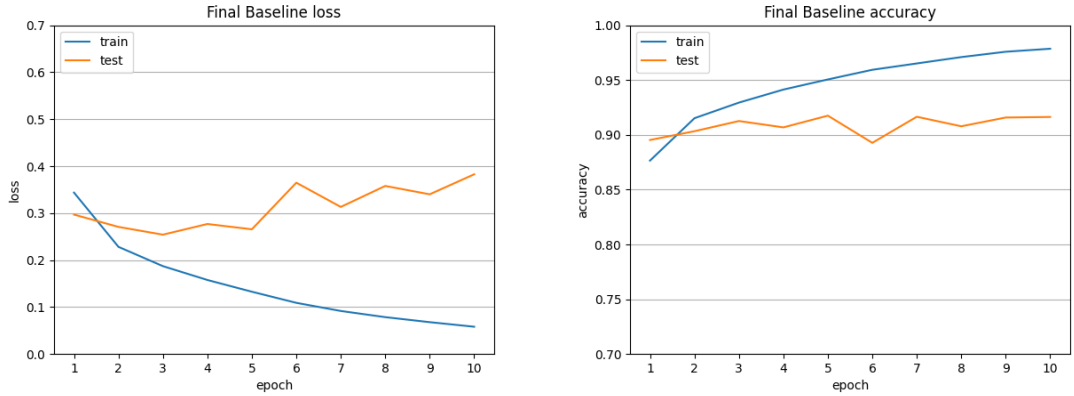

Figure 6: Baseline Testing Performance

The metrics look very similar for the learning rate variant.

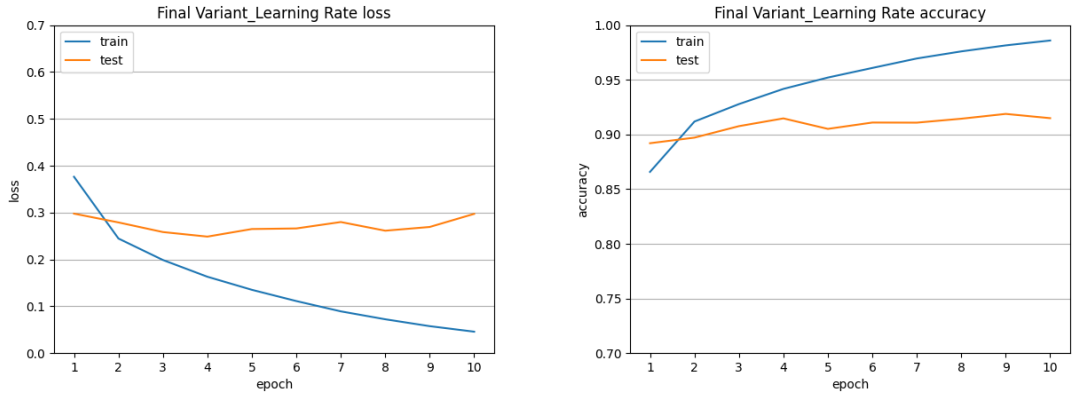

Figure 7: Learning Rate Variant Testing Performance

The final test accuracies of the baseline, Learning Rate Variant and baseline with halved learning rate are 91.6%, 91.5% and 92.1% respectively. It appears that the baseline with halved learning rate performs slightly better compared to the other two models. However, it's hard to tell whether this is due to random fluctuations in the test set or an actual effect since this is only based on a single run. Visually, it looks like the test accuracy of the baseline with halved learning rate trends slightly higher compared to baseline after epoch 5, which makes it our best performing model.

For the halved learning rate model, you can notice a small decrease in loss and increase in accuracy at epoch 5 compared to the regular baseline model. This effect also translated to the test set, however this improvement is very small.

For the third choice task, we made a confusion matrix of the predictions of the baseline model on the test set. The main diagonal is clearly visible which indicates a good performance. The confusion matrix also shows common mistakes that the model makes. For example, most of the mistakes it

	Accuracy	Loss
Baseline	0.916	0.383
Variant Learning Rate	0.915	0.297
Baseline Halved LR	0.921	0.379

Table 3: Results of the final model tests after 10 epochs

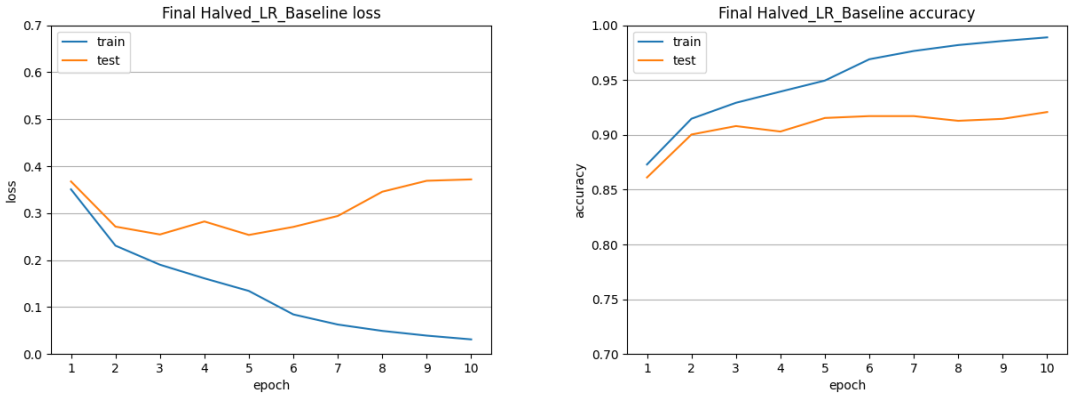

Figure 8: Halved Learning Rate Baseline Testing Performance

makes are distinguishing *Shirt* and *T-shirt/top*. In general *Shirts* are most frequently misclassified with an accuracy of only 79%. The model also predicts images to be shirts incorrectly relatively often indicated by off-diagonal values in the *Shirts* row and column. Another common mistake is classifying *Pullovers* as *Coats*. Interestingly *Coats* are not classified as *Pullovers* as often. The model performs the best on *Sandals* with a accuracy of 99.1% which is very high.

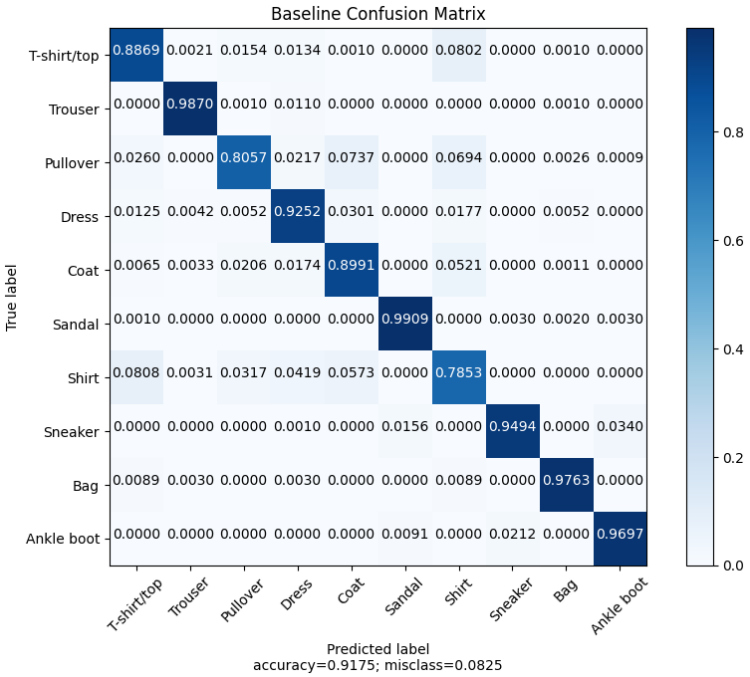

Figure 9: Halved Learning Rate Baseline Testing Performance

4 Discussion

All variants didn't appear to significantly change the performance of the models. All the best validation accuracies remain around 91.6-91.9% which is relatively high. We think the reason for this is that we're not significantly altering the complexity of the network from the baseline. Most of the variants only have small tweaks which don't really impact the network's ability to learn the desired relationship between input and output. Bigger changes such as removing a layer, adding a layer, changing the number of filters are more likely to result in performance changes. We learned that in our case, our small tweaks to the baseline model didn't really have much impact.

An improvement could be to run each model more times (more folds) to get more accurate accuracies, since the accuracies are very close together. More random initialisation will give more accurate scores. The model weights that we uploaded are currently taken after the 10th epoch but this is likely a very overfitted model and the validation loss has possibly already increased again. Ideally, we would have saved the model weights when the validation loss was minimal, which is usually around epoch 3 for most models. The same also holds for measuring the validation accuracy. This should ideally be measured when the loss is minimal.

5 Weights

All model weights are available here.

6 Choice Tasks

- We implemented k-fold cross-validation and used the mean accuracy and loss as a more accurate estimate of model performance.
- After every 5 epochs, the learning rate for each model's optimizer is halved using the learning rate scheduler callback.
- Confusion matrix, we plotted the confusion matrix of the baseline model on the test set.

References

[1] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. DOI: 10.48550/ARXIV.1207.0580. URL: https://arxiv.org/abs/1207.0580.

[2] Sungheon Park and Nojun Kwak. "Analysis on the Dropout Effect in Convolutional Neural Networks". In: *Computer Vision – ACCV 2016*. Ed. by Shang-Hong Lai et al. Cham: Springer International Publishing, 2017, pp. 189–204. ISBN: 978-3-319-54184-6.

[3] Alexandros Stergiou, Ronald Poppe, and Grigorios Kalliatakis. *Refining activation downsampling with SoftPool*. 2021. DOI: 10.48550/ARXIV.2101.00440. URL: https://arxiv.org/abs/2101.00440.