Multinomial Regression using Convolutional Neural Networks

Kristoffer Gjerde, Alon Tron & Jon Ryfetten

CSE154 Deep Learning

Gary Cotrell

November 15, 2019

Abstract

Face recognition has over the years shown to have increasing relevance and importance in the industry. Convolutional Neural Networks (CNN) have gained huge popularity because of its ability to classify images. Using a dataset of 200 identities in total, this report will present one possible solution to build a classifier using CNNs. The dataset we will be using is imbalanced. To handle some of the challenges this brings, we will be presenting our results with a range of different metrics to gain better insight into how our models perform.

Our results show that using transfer learning, we get a slightly better result, than training a network from scratch.

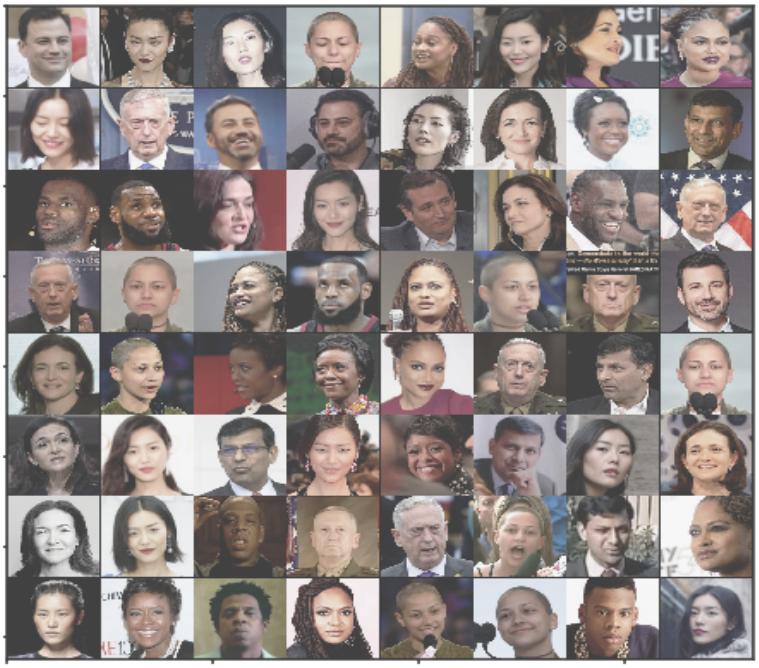

Figure 1: 64 sample images from the Split_Original_Face dataset.

1 Introduction

In this assignment, we will explore a classification task for face detection. The dataset consists of 200 different celebrity' identities, which each contains an imbalanced number of images. This means that some classes represent more frequent than others. The input images are of size 224x224, which makes it hard to only do fully connected layers. We are therefore going to build a deep convolutional neural network architecture. Networks like this are shown to do really good on problems where the input is a set of images.

In total, we are testing out three different architectures. The first model presented is the baseline model we were provided. This model is only used as a guideline for what our other models are at least supposed to perform.

The second model is an extension of what we were provided in the baseline. We made it deeper, in hope of it learning more advanced features.

Third, we are using transfer learning to initialize a ResNet18 model. Here we are downloading a pre-trained model while switching out the fully connected layers to fit our problem of 200 different identities.

We have learned in class the importance of right weights initialization and how too small or too big initial weights might cause radical loss gradients and the network will take longer to reach a good accuracy. Xavier weight initialization helps us find the right balance of the sizes of the weights, and keep the variance of the signal along with many layers in a stable in order avoid it from exploding to high value or vanishing to zero source[2].

So if our goal is the to keep variance stable across layers we need to preserve:

$$var(y) = var(w_1x_1 + w_2x_2 + ... + w_Nx_N + b) \tag{1}$$

The fact that we initialize the mean of a given weight and input to be 0 along with the samples identically distribution we get:

$$var(y) = N \cdot var(w_1) \cdot var(x_1) \tag{2}$$

where N is the number or neuron inputs.

Hence if we want to maintain the variance of y we need

$$N \cdot var(w_i) = 1 \tag{3}$$

so we need,

$$var(w_i) = \frac{1}{N} \tag{4}$$

And we conclude the variance of each weight should be initialized to

$$\frac{1}{\frac{Ni+No}{2}} \tag{5}$$

Where N_i is the number of inputs to the neuron and N_o is the number of outputs. The reason Xavier made the the average number between the input and the output is to also preserved the variance of the back propagation signal.

Another important concept we have been using in our model is the Batch Normalization Layer, it has been long known that faster convergence is reachable by normalization of the the input values.[9] For big amount of the data, there are computational difficulties in calculating the entire data normalized form, so instead of trying to do so, we normalize batches of the input with its own. We describe more about the mathematical motivation in the Methods section.

2 Related work

The motivation for dealing with convolutional neural networks in this assignment is building on research done by Yann LeCun, which we consider the founder of this kind of network. His work on LeNet [8] has inspired many researchers through the years and is probably the reason why we are learning about this kind of network today. In his work, he is showing results on a dataset consisting of written digits, where he was classifying numbers from zero to nine. This was how he managed to demonstrate how efficient convolutional layers were to classify images. As this could be seen as a similar problem as the one we are dealing with, CNNs would be the natural choice for a task like this.

We also got motivation from what we have read on the ResNet [5]. This is just one example of a CNN that scored very good in the contest on classifying the ImageNet dataset. Back in 2015, this was the state of the art model that won this competition.

3 Methods

3.1 image pre-processing

In order to help the neural network to work on the data and to achieve invariant properties in our final model we made the following preprocessing to the pictures. First we normalized all the pictures to have zero mean and one variance. We did it by applping the following formula for each channel:

$$x = \frac{x - E(x)}{STD(x)} \tag{6}$$

where x is the original image.

To get the mean and standard deviation of each channel, we run over the train set and found the average value of each pixel in each channel with its standard deviation.

Secondly, to train our model to be able to handle rotated images, we applied a transformer that rotates the input randomly in a range of angles. When letting the model train in many different orientations we decrease the possibility that the model learns to handle one specific orientation, which is not the case in the real world but maybe it is the case of the data set.

3.2 Baseline model

The baseline model consists of four convolutional layers, all interchanged by both ReLU and Batch-Normalization layers. The full model can be seen in Table 1. Since this is a classification task, the appropriate activation on the last layer would be softmax. The best loss function to use in combination with the softmax is the cross-entropy loss. This can be proved from the maximum likelihood. We also chose to use Adam optimizer for this and all the other models presented in the report.

Layers	Description
Conv2d	In: 3, Out: 21 Kernel: 3x3, Stride: 2 Activation: ReLU
Conv2d	In: 21, Out: 20 Kernel: 3x3, Stride: 2
BatchNorm2d	Number of features: 20 Activation: ReLU
Conv2d	In: 20, Out: 15 Kernel: 3x3, Stride: 2 , Padding: 1
BatchNorm2d	Number of features: 15 Activation: ReLU
Conv2d	In: 15, Out: 7 Kernel: 5x5, Stride: 2 , Padding: 1
BatchNorm2d	Number of features: 7 Activation: ReLU
Linear	In: 1183, Out: 300 Activation: ReLU
Linear	In: 300, Out: 201 Activation: ReLU
Softmax	In: 201, Out 201

Table 1: Baseline Architecture

3.3 TronNet

A dilemma in machine learning is that we want to maximize the amount of training and test data. To help us here, we can use a method called K-fold cross-validation [3]. K-fold cross-validation splits the dataset into K partitions. Then we run K different experiments, where each experiment selects their own partition used as a validation set. K results are then averaged to a single estimation. The advantage of this method is that all observations are used for training and validation. Each of the observations has been used for validation only one time. We then select the best of the K models, which we then use to calculate the metrics on the test data.

The TronNet architecture consists of 7 convolutional layers, 2 max-pooling layer and several batch normalization layers between them. The full architecture can be seen in Table 2. The initial thought behind the network was that deeper was better. We started with the baseline model and kept adding layers from here. The weight initialization that showed the best results, was the Xavier as described in the introduction. This helps us find the right balance for the size of the weights, and keep the variance of the signal along with many layers stable. For the gradient descent optimizer, we used Adam optimizer, as this was the one we got recommended. Here we used a learning rate of 0.0001 and added a weight decay of 0.005. The weight decay of 0.005 was our way of doing regularization.

Since the dataset was highly imbalanced, we needed to find a way to deal with this when building our model. We handled this by enforcing the network to learn to categorize the infrequently seen classes. The weighted loss was therefore calculated for the dataset and passed on to the cross-entropy loss function which took care of the rest. The code for calculating these weights can be found in the *utils.py* file, inside the *Loader* class. The way we choose to weigh each class was by taking 1

divided by the number of images of that class. This way we are giving higher weights to the classes which has a low amount of images.

Layers	Description
Conv2d	In: 3, Out: 21 Kernel: 3x3, Stride: 1 Activation: ReLU
Conv2d	In: 21, Out: 30 Kernel: 3x3, Stride: 1, Padding: 1 Activation: ReLU
BatchNorm2d	Number of features: 30 Activation: ReLU
MaxPool2d	Kernel: 3x3
Conv2d	In: 30, Out: 35 Kernel:3x3, Stride: 1, Padding: 1
BatchNorm2d	Number of features: 35 Activation: ReLU
Conv2d	In: 35, Out: 25 Kernel: 3x3, Stride: 1, Padding: 1
BatchNorm2d	Number of features: 25 Activation: ReLU
MaxPool2d	Kernel: 3x3
Conv2d	In: 25, Out: 20 Kernel: 3x3, Stride: 1, Padding: 1
BatchNorm2d	Number of features: 20 Activation: ReLU
Conv2d	In: 20, Out: 15 Kernel: 3x3, Stride: 1, Padding: 1
BatchNorm2d	Number of features: 15 Activation: ReLU
Conv2d	In: 15, Out: 10 Kernel: 5x5, Stride: 1, Padding: 1
BatchNorm2d	Number of features: 10 Activation: ReLU
Flatten	
Linear	In: 1210, Out: 300 Activation: ReLU
Linear	In: 300, Out: 201
Softmax	In: 201, Out 201

Table 2: TronNet Architecture

3.4 Transfer Learning - ResNet 18

Transfer Learning is the idea of using a knowledge of a pre-trained neural network for solving our own or refit it to our data. One benefit of using this is an significant speed up in training time, as we only train the fully connected layers in the CNN. The model chosen is the ResNet-18[5]. ResNets builds on the constructs known from pyramidal cells. It does this by adding short cuts, or skip connections over some layers.

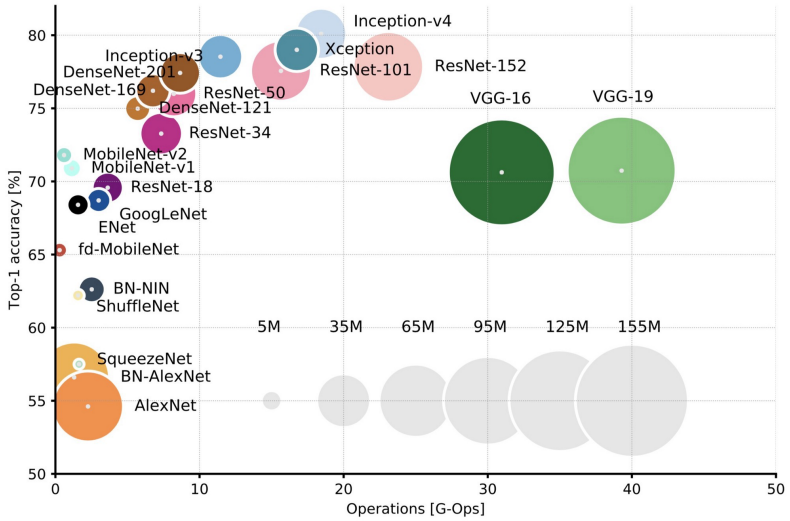

Figure 2: Top-1 one-crop accuracy versus amount of operations required for a single forward pass. Illustration by Neural Network Architectures [4]

We picked the model based on figure 2. The model offer high accuracy, with relative low operations. In our initial test, we tried using VGG-16, but had trouble with the performance of the model.

For the architecture of ResNet18, with our 201 ouput classes, see figure 3

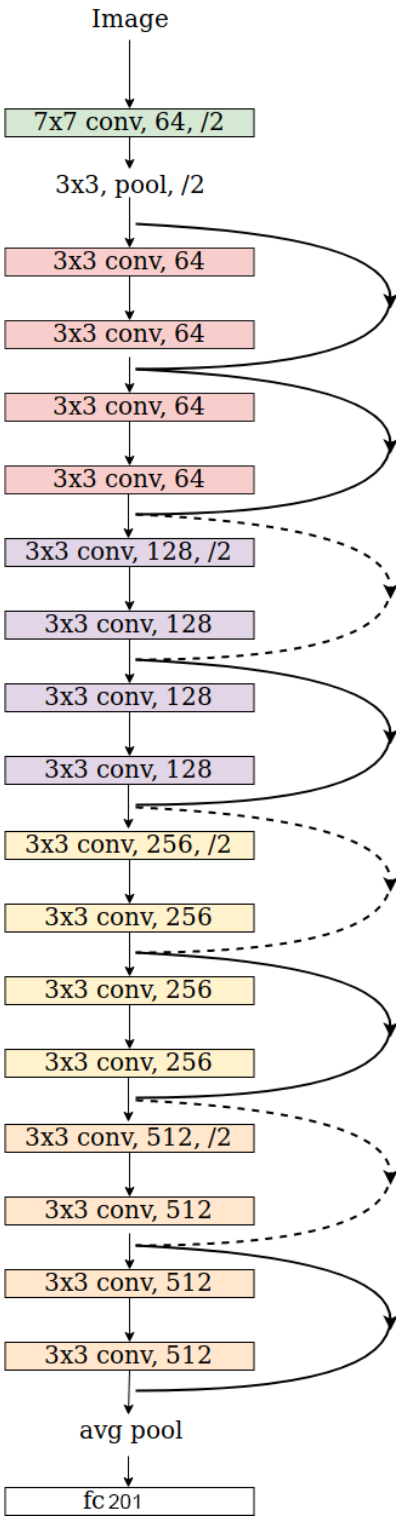

Figure 3: The ResNet18 architecture, with 201 output classes. The lines on right side represent the skip connections. Illustration by thatsnotbyname71[6] and Jon Ryfetten

We started with a trained model provided by the pytorch library, and changed the fully-connected layer to help the model fit to our data. The parameters stayed the same as we used in TronNet. A learning rate of 0.0001 and decay of 0.005. The results are presented in the next section.

4 Results

4.1 Performance metrics

The top 5 performance metrics can be found in Table 3, the worst 5 in 4 while the overall averages for each of the metrics can be found in Table 5.

Top 5	Nnet	TronNet	TransferNet
1	Sheik hHasina (186) Acc=1.0, Precision=0.625, Recall=1.0, BCR=0.812, Agg=0.859	Alessandro Michele (199) Acc=1.0, Precision=1.0, Recall=1.0, BCR=1.0, Agg=1.0	Sheikh Hasina (186) Acc=0.994, Precision=1, Recall=0.95, BCR=0.975, Agg=0.98
2	Alessandro Michele (199) Acc=1.0, Precision=1.0, Recall=1.0, BCR=1.0, Agg=1.0	George Church (183) Acc=1.0, Precision=0.857, Recall=1.0, BCR=0.929, Agg=0.946	Virgil Abloh (139) Acc=0.994, Precision=0.909, Recall=1.0, BCR=0.955, Agg=0.964
3	Rodrigo Duterte (58) Acc=1.0, Precision=0.706, Recall=1.0, BCR=0.853, Agg=0.890	Karl Lagerfeld (47) Acc=1.0, Precision=0.833, Recall=1.0, BCR=0.917, Agg=0.937	Pope Francis (134) Acc=0.994, Precision=0.933, Recall=1.0, BCR=0.967, Agg=0.973
4	Samantha Bee (153) Acc=0.958, Precision=0.676, Recall=0.958, BCR=0.817, Agg=0.853	Angela Merkel (118) Acc=0.957, Precision=0.917, Recall=0.957, BCR=0.937, Agg=0.942	Shinzo Abe (81) Acc=0.994, Precision=0.947, Recall=1.0, BCR=0.974, Agg=0.979
5	Kumail Nanjiani (157) Acc=0.933, Precision=0.700, Recall=0.933, BCR=0.817, Agg=0.846	Sheikh Hasina (186) Acc=0.95, Precision=0.613, Recall=0.95, BCR=0.781, Agg=0.824	Karl Lagerfeld (47) Acc=0.994, Precision=0.909, Recall=1, BCR=0.955, Agg=0.964

Table 3: Top 5 Results

Worst 5	Nnet	TronNet	TransferNet
1	Miley Cyrus (96) Acc=0, Precision=0, Recall=0, BCR=0, Agg=0	Miley Cyrus (96) Acc=0, Precision=0, Recall=0, BCR=0, Agg=0	Kaia Gerber (93) Acc=0.671, Precision=0.222, Recall=0.345, BCR=0.284, Agg=0.38
2	Jeanette Vizguerra (26) Acc=0, Precision=0, Recall=0, BCR=0, Agg=0	Jeanette Vizguerra (26) Acc=0, Precision=0, Recall=0, BCR=0, Agg=0	EmmaStone (67) Acc=0.695, Precision=0.344, Recall=0.733, BCR=0.539, Agg=0.578
3	Theo Epstein (61) Acc=0.182, Precision=1.0, Recall=0.182, BCR=0.591, Agg=0.489	Jaclyn Corin (165) Acc=0, Precision=0, Recall=0, BCR=0, Agg=0	Deepika Padukone (191) Acc=0.72, Precision=0.097, Recall=0.143, BCR=0.12, Agg=0.27
4	JayZ (10) Acc=0.182, Precision=0.222, Recall=0.182, BCR=0.202, Agg=0.197	Brian Chesky (182) Acc=0.182, Precision=0.4, Recall=0.182, BCR=0.291, Agg=0.264	Theo Epstein (61) Acc=0.726, Precision=0.146, Recall=0.636, BCR=0.391, Agg=0.475
5	Aragaki Yui (79) Acc=0.200, Precision=0.400, Recall=0.200, BCR=0.300, Agg=0.275	Theo Epstein (61) Acc=0.200, Precision=0.333, Recall=0.200, BCR=0.267, Agg=0.25	Beyonce (158) Acc=0.756, Precision=0.158, Recall=0.429, BCR=0.429, Agg=0.409

Table 4: Worst 5 Results

Metric	Nnet	TronNet	TransferNet
Accuracy	0.62	0.63	0.93
Precision	0.70	0.71	0.67
Recall	0.62	0.62	0.62
BCR	0.66	0.67	0.65
Aggregated	0.65	0.68	0.71

Table 5: Average metrics overall classes

4.2 Loss Visualization

The best runs for each of the models are graphed in Figure 4 for the baseline model, Figure 5 for the TronNet and Figure 6 for the ResNet model.

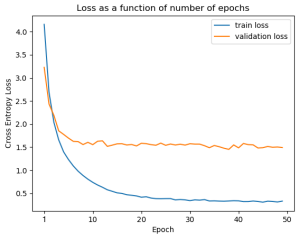

Figure 4: Baseline Model Learning Loss

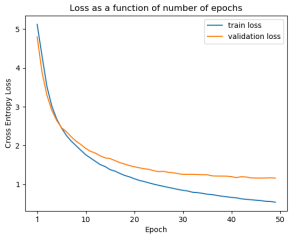

Figure 5: TronNet Learning Loss

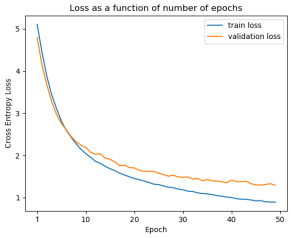

Figure 6: ResNet Model Learning Loss

4.3 Weighted Loss

As described in the Methods section, we used the weighted loss approach to handle the class imbalance problem. The results is shown in Table 6 and 7

Top 5	Weighted Loss TronNet
1	Alessandro Michele (199), Acc=1, Precision=0.462, Recall=1, BCR=0.731, Agg=0.798
2	George Church (183), Acc=1, Precision=0.667, Recall=1, BCR=0.833, Agg=0.875
3	Pope Francis (134), Acc=1, Precision=0.7, Recall=1, BCR=0.85, Agg=0.887
4	Sinta Nuriyah (133), Acc=1, Precision=1, Recall=1, BCR=1, Agg=1
5	Moon Jae-in (121), Acc=1, Precision=0.875, Recall=1, BCR=0.937, Agg=0.953

Table 6: Top 5 Results for weighted loss

Worst 5	Weighted Loss TronNet
1	Tom Brady (70), Acc=0.111, Precision=0.2, Recall=0.111, BCR=0.156, Agg=0.144
2	Margot Robbie (21), Acc=0.194, Precision=0.545, Recall=0.194, BCR=0.37, Agg=0.326
3	Jeanette Vizguerra (26), Acc=0.2, Precision=0.167, Recall=0.2, BCR=0.183, Agg=0.187
4	Aragaki Yui (79), Acc=0.2, Precision=0.4, Recall=0.2, BCR=0.3, Agg=0.275
5	Jacinda Ardern (72), Acc=0.211, Precision=0.4, Recall=0.211, BCR=0.305, Agg=0.282

Table 7: Worst 5 Results for weighted loss

And the averages results are presented in 8

Metric	Nnet
Accuracy	0.637
Precision	0.652
Recall	0.637
BCR	0.645
Aggregated	0.643

Table 8: Average metrics overall classes

The loss plot for TronNet with Weighted Loss is presented in Figure 7

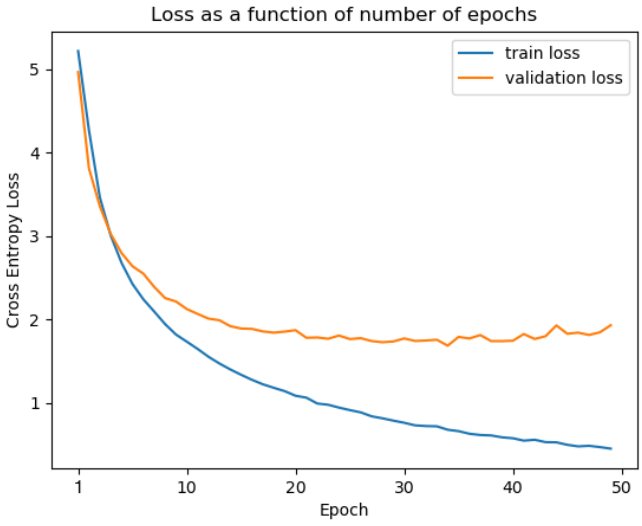

Figure 7: TronNet Model using Weighted Loss Learning Loss

4.4 Weights Visualization

After we trained out models, we created a visual presentation of the kernels of each model. These are the representations of the kernals, the intensity of the weight is presented in RGBA scale. Each figure we see corresponds to one convolution layer and each matrix corresponds the convolution function applied on one input channel.

Baseline Model
The architecture provided in the assignment.

Figure 8: Kernels for baseline model - layer 1

Figure 9: Kernels for baseline model - layer 3

Figure 10: Kernels for baseline model - layer 6

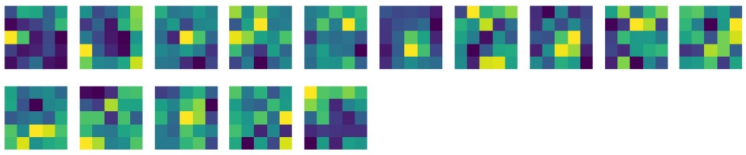

Figure 11: Kernels for baseline model - layer 9

TronNet Model
The new architecture we implemented.

Figure 12: Kernels for TronNet model - layer 1

Figure 13: Kernels for TronNet model - layer 3

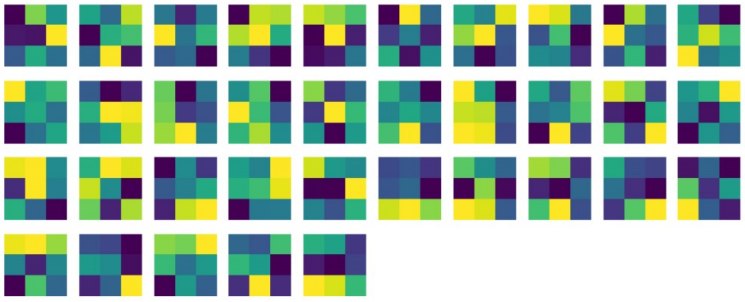

Figure 14: Kernels for TronNet model - layer 10

Figure 15: Kernels for TronNet model - layer 14

Figure 16: Kernels for TronNet model - layer 17

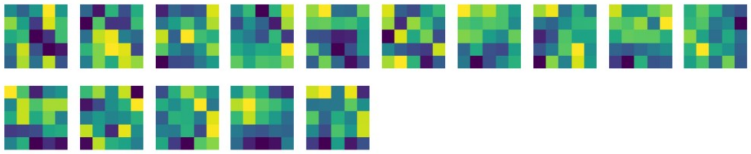

Figure 17: Kernels for TronNet model - layer 20

ResNet Model
The ResNet architecture we used for transfer learning.

Figure 18: Kernels for ResNet model - layer 1

Figure 19: Kernels for ResNet model - layer 3

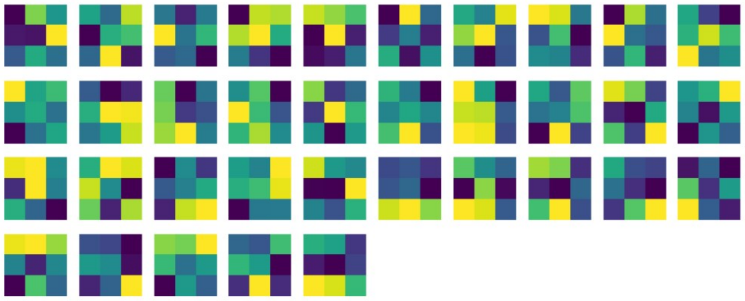

Figure 20: Kernels for ResNet model - layer 10

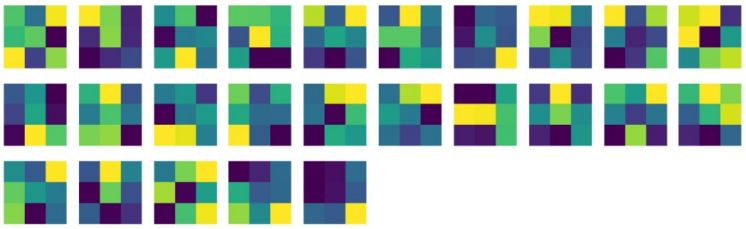

Figure 21: Kernels for ResNet model - layer 14

Figure 22: Kernels for ResNet model - layer 17

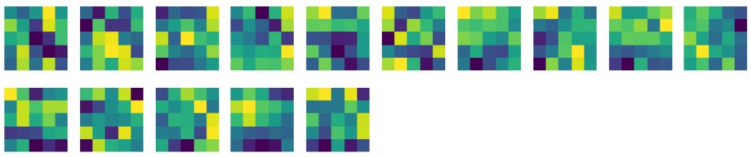

Figure 23: Kernels for ResNet model - layer 20

5 Discussion

5.1 Performance Analysis

By comparing the three models we can firstly pay attention to the loss curves that have been formed during the learning. By observing the validation loss each model reached, we can tell that ResNet, the model we used for transfer learning, achieves better Cross-Entropy Loss of 1.25 among all the three models. The second-best model is the TronNet, with a final validation loss of 1.32 and the third is the baseline model with a loss of 1.55. The fact that the ResNet performs better than the others doesn't surprise us since is a well-known architecture that was presented good performance and has been trained with similar data. Since the TronNet is an upgraded model of the baseline with more convolutions layers, max-pooling layers, better weights initialization, and more comprehensive transformers it doesn't surprise us that it performs better than the baseline.

Even though it is hard to conclude from the metrics table general conclusions we can see that all models learned well, with an aggregated score of greater than 0.82, at least 5 classes. we can see that the ResNet performed especially well as it learned 20 classes with an aggregated score greater than 0.9.

As reported in Table 5, an average aggregated score of the ResNet model is 0.72 while the median is 0.747, these are the best result we got. The average aggregated score of the TronNet model is 0.68 while the median is 0.686, bad result compares to the baseline model that has an average of 0.65 and a median of 0.667. These numbers correspond with the argument we stated at the beginning of the section.

5.2 Common Confusions

The Baseline and the TronNet both performed poorly on Miley Cyrus (96), it probably occurred due to the fact the Miley use to change haircuts, makeup and accessories during her life and she actually

has very different features in every single image. The fact that she has also transformed from a child to an adult in her career could also make it more difficult to classify the images of her. All three also struggled to identify Theo Epstein, it is hard to elaborate on why because most of his images in the training and test set look similar without radical changes.

5.3 Weighted Loss

By comparing the TronNet with and without the Weighted Loss feature, we can observe a few changes. The first one is that the overall loss on the validation set increased, it probably happened due to the different calculation of the loss made in the new model that gives significant weight to special classes and might damage the loss of other most popular classes. Also the overall metrics seemed to be damaged a bit, we can assume that the method of the weighting the classes made the overall performance of the model a little bit worse due to the new target function but still because of the low number of times we ran the comparisons it is still very possible that these small changes caused just from worst learning sessions.

Another interesting thing we can see from the Top and Worst 5 table is the fact that the weighted loss has softened the top and worst classes in a manner that the metrics are less radical now in both ends. We assume that the weighted loss feature identified these classes and treat them as described in the Methods section.

5.4 Metrics Analysis

All our reports of metrics are calculated on a per-class basis, including accuracy.

Accuracy for a given class is the fraction of correctly predicted samples for the given class, dived by the number of samples. In a highly imbalanced dataset, an issue that could occur is the classifier always or often predict negative or positive for a class to maximize the amount of correctly predicted classes. This will make the classifier useless. To avoid and discover this pattern, we can bring in new metrics, like precision, recall, and BCR.

Precision is telling us how many of the predicted values that were predicted to be a current class and actually was labeled that class as well. Recall on the other hand is telling us how many of the images of one class the model is managing to predict as that class. We use both precision and recall to calculate the balanced classification rate, which then gives us a measurement of how both of them scored overall. Lastly, the aggregated score gives us an overall score of all the different metrics that are calculated.

From the results presented in Table 3, 4 and 5 we can see that our TransferNet using ResNet18 is has the highest accuracy, but not necessary the best. Precision is a good measure to determine when the cost of False Positive is high, while recall actually calculating how many of the actual pictures of one class our model capture through labeling it as positive.

Recall would be very important when there is a high cost associated with a false negative. By looking at the results we can see that even though we get high accuracy in some classes, the precision and recall might not give as good results.

One example of such a case is the transfer learning we made, even though the accuracy metric reports good results, in cases when we want to minimize the type I or II mistakes we may prefer the TronNet or the baseline model. In table 5 we see that all three models similar in their performance except to the Transfer net which present far best accuracy, that's the model we'll choose if we would like to maximaize the how close the predicted average is to the true value of the quantity being measured.

5.5 Weights Visualization

First, we discuss the kernels trained in the Baseline and in the TronNet. In Figure 3 & 7, we can see the first kernels applied on the input, we have 3 of them because each corresponds to a different color channel in the input image. The first good sign we see in the kernels is that they are not noisy and not correlated with each other, we see it by observing the small patterns the model trained [1]. Moreover, as deep as the kernels go we're seeing more significant patterns and shapes in the filters.

In the last kernel of each model, we can even observe lines and squares that may correspond to different features in the human face.

We also used the ResNet Model for transfer learning. The ResNet model starts with a 7-size kernel that corresponds to the three color channels. then the number of channels gets bigger as the model gets deeper.

We can see that the deeper the kernel gets, we get bolder and easier to see structural patterns and shapes that might be formed during over time with the big dataset the model's trainer had (figure 17).

6 Authors' Contributions and References

Kristoffer Gjerde
Wrote the code for the model evaluation in the *utils.py* file. He initializes and contributed on building the *cnn.py* script which is the general file for running and training the models, as well as contributing

Alon Tron
Implemented parts of the alternative weight initialization, image prepossessing and normalization and started the transfer learning model. Alon also contributed in running the different models on the servers and writing the report.

Jon Ryfetten
Implemented transfer learning and k-fold cross-validation. Figured out how to get the code up and running on the GPUs. Contributed as well on building the general code base and writing the report.

References

[1] Cottrell G.: (2019), Convolutional Networks, Part II Lecture Slides.
https://piazza.com/class_profile/get_resource/k12h2h0zo6w27f/k2xn0xqz7lh5mp

[2] Glorot, Xavier & Bengio, Bengio. (2010). Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track. 9. 249-256.
http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

[3] Hastie, T.; Tibshirani, R. & Friedman, J. (2009), The elements of statistical learning: data mining, inference and prediction , Springer. Chapter 7.

[4] Eugenio Culurciello, Neural Network Architectures
https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778. arXiv 1512.03385.
https://arxiv.org/abs/1512.03385

[6] thatsnotmyname71, ResNet architecture
https://ai.stackexchange.com/questions/13842/what-do-the-numbers-in-this-cnn-archi

[7] Krizhevsky, Alex & Sutskever, Ilya & E.Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-

[8] Lecun, Yann & Bottou, Léon, Bengio, Yoshua, Haffner, Patrick. (1998). Gradient-Based Learning Applied to Document Recognition
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[9] LeCun et al. (2011). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift 657-665.
https://piazza.com/class_profile/get_resource/k12h2h0zo6w27f/k2xn0xqz7lh5mp

[10] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.
https://arxiv.org/pdf/1409.1556.pdf