General Classification of Dog Breeds

Jón Freysteinn Jónsson - jfjon@kth.se
Steinn Elliði Pétursson - petu@kth.se

KTH - DD2424
Deep Learning in Data Science

Abstract. In this project different image classification techniques will be applied to the Stanford dogs dataset [1]. The methods explored are classification using a feed-forward neural net, a convolutional neural net, and transfer learning using a pre-trained image classification neural network with added layers. We show that though our testing of the three different networks we manage to tune and train the transfer learning network to give us the best results. Additionally we experiment with tuning the network and find different ways to get better performance and scores out of our networks.

Keywords: Neural Networks, Classification, Image Processing, convolutional Networks, Inductive Transfer Learning, Dog Breeds

1 Introduction and Problem formulation

1.1 Competition

The idea for this project arose after looking thought the competitions that had been held on Kaggle[2]. Kaggle is a site that, among other things, hosts competitions in machine learning on open datasets. Looking though the available, and already completed, competitions the one on identifying dog breeds [3] was picked. This gives a classifying accuracy to aim for as well as a good data set to use and train on.

1.2 Goal

Simply put the goal was to achieve a score in the to 20% of the Kaggle leaderboard. The highest scoring teams in the competition managed to identify the dog breeds perfectly. We aimed for a multiclass loss of around 0.2 which would get us into the top 20% on the leaderboard.

2 Background

A lot of papers exist on the subject of image classification. Most of the papers with the papers utilize deep learning to produce their results. Following are some of the papers which were used for guidance discussed along with the approach taken in each of them.

The paper *ImageNet Classification with Deep Convolutional Neural Networks*[4] Got 89% accuracy on the CIFAR-10 dataset and got a winning top-5 test error rate in the ILSVRC-2012 competition of 15.3%, while the next entry had 26.2% error rate. The network used was comprised of 5 convolutional layers along with 3 fully connected layers, and they acknowledge that the network might produce even better results with more layers but that the gpu memory limitations were the limiting factor. This paper introduces the Rectified Linear Unit (ReLU) as an initialization function and demonstrate that it reaches a 25% training error rate on the CIFAR-10 dataset 6 times faster than a tanh activation function with a four layer convolutional neural network. Their network has two max pooling layers, with overlapping pooling with a pooling grid of size 3 x 3 and spacing of 2. To reduce overfitting the researchers use data augmentation where they do image translations and horizontal reflections. Another method they use is dropout where the probability of each hidden neuron in their network of being set to 0 in each propagation is 0.5.

The paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [5] discusses the dropout regularization technique in depth and proposes an alternative technique to what was used in the paper on ImageNet classification discussed in the last section. Whereas that paper proposed having a constant dropout rate of 0.5 between layers this paper proposes the dropout layers should not be constant but instead the dropout should increase in the later layers of the network, starting at 0.1 and going up to 0.5.

The papers mentioned so far all implement convolutional neural nets from scratch. This can be a long process and a lot of data is needed to train a good network from scratch. The paper *A Survey on Transfer Learning*[6] takes another approach to image classification, transfer learning. Transfer learning is based on the assumption that knowledge gathered by training neural network trained on one dataset with a certain feature space and distribution can be transferred onto another set of data with a different feature space and distribution. Different transfer learning techniques are discussed, but the technique relevant to the dog breeds project is the inductive transfer learning method. In inductive transfer learning the source and target domains are similar but related, and the same holds for the source and target tasks.

3 Approach

3.1 Image Pre-Processing

Since the dataset we worked with was fairly small (15394 images for training) we implemented some data augmentation on the training dataset. The data augmentation techniques used were as follows:

– Horizontal flip
– 40% rotation range
– 20% image shear range
– 10% width shift range
– 10% height shift range
– 20% zoom range

Each image was transformed randomly according to these augmentations, up to the percentage declared above and randomly flipped / not flipped. Additionally the pixel values of the image were all rescaled from 0-255 to a range between 0 and 1. The only data augmentation performed on the test set was rescaling.

3.2 Multi class loss

The Kaggle competition determines the grading criteria in multi class log loss, also known as categorical cross-entropy loss. This metric looks at the probabilities a model outputs, that is instead of looking at the class with the highest accuracy as the result (binary classification) it measures how certain the model is of the result being the correct label. This way a model guessing correctly with a certainty of 0.5 gets a higher loss score than a model guessing correctly with a certainty of 0.9. The formula for multiclass log loss is as follows

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{1}$$

Where M is the number of class labels, y is a binary indicator of whether class label c is the correct classification for observation o, and p is the model's predicted probability that observation o is of class c [7].

3.3 Hyper Parameter Search

When constructing the networks we performed rudimentary searches for better parameters to use in the training of the three networks. We searched through the different activations, learning rates and optimizers so that we could select the best performing parameters for the network.

3.4 Feed Forward Network

For our simplest network we decided on four hidden layers, the node count in each layer was as follows, 1024:1024:512:256. This is not including the input layer and the final class layer. We never expected this network to reach any great accuracy but rather used it to get familiar with the tools and with the data. The network allowed us to test out different hyper parameters and see if we could get the network to learn from the data and classify better than a random guess would.

3.5 Convolutional Network

The final version of the convolutional network takes as input a 100x100 image. All of the convolutional layers have filters of size 3x3, with stride 1 and are 0 padded on the edges. The pooling layers have pooling size 2x2 with stride 2 and 0 padding as well. The dense layers are of size 1024. All layers except for the output layer use softplus [8]

$$softplus(x) = log(1 + e^x) \tag{2}$$

as an activation function. The output layer uses softmax as an activation. For regularization we used dropout, which has yielded some very impressive results in the past [5, 4] and are the industry standard today. Testing was done with a constant dropout rate with dropout percentage ranging from 0.1 to 0.8, which yielded mixed results. After trying the constant dropout approach on the CNN it was decided to switch to another dropout scheme with higher dropout percentage in the deeper layers of the network as described in the paper *Dropout: A simple way to prevent neural networks from overfitting* [5]. With dropout ranging from 0.1 to 0.5 throughout the network. This resulted in significant increase in network performance. We experimented with different optimizers, the two optimizers that resulted in best performance were the Adam optimizer [9] yielding an accuracy of about 15.5%. The optimizer which resulted in the best performance was the stochastic gradient descent optimizer with about 21% accuracy. The hyper parameters used in the final version had a learning rate of 0.01, decay $1e^{-5}$, momentum 0.95 with Nesterov momentum. We tried approaches with less momentum which yielded similar results. Decreasing the learning rate caused a severe underfit and the network was not able to get above the guessing rate of around 0.9 %.

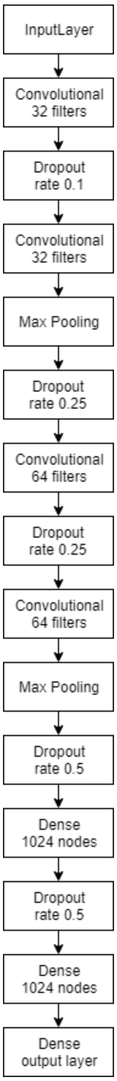

Fig. 1. Convolutional neural network setup

3.6 Inductive Transfer Learning

We set up a Transfer Learning network network starting with the pre-trained weights from the `Keras VGG19` model before adding two additional hidden layers of our own. The hidden layers were dense layers with 1024 nodes each. Lastly we set up an output layer with softmax activation. The `VGG19` is based on Simonyan and Zissermans [10] network from 2015 that is pre-trained on the ImageNet dataset. Our initial implementation froze the first five layers of the pre-trained weights, this was recommended in some of the tutorials we looked at.

Freezing a layer in a neural net implies that the layer will not be modified any further during training. Freezing the layers of the pre-trained weights offers some benefits when it comes to the wall-clock time it can take to train the network, with minimal loss in accuracy. However when looking into ways to improve the performance of the initial network we found evidence to the contrary, for the network type we are using, in a paper on progressively freezing layers during training [11]. They point out that VGG seems to be less well-suited for freezing. We ran training again on our network without freezing and noticed right away a change in the wall-clock computational time of each epoch. Furthermore we noticed a gain in accuracy, details on that can be seen in the result section.

Another pre-trained network we tried was the `Xception` network [12] which is inspired by the older Inception networks. The `Keras Xception` application is 162 layers, and each epoch took up to 10 minutes to complete. To squeeze out better performance from the resources we decided to freeze the first 10 layers of the Xception net. The added layers of this network were the same as with the pre-trained `VGG19` net, two hidden layers with 1024 weights each and a softmax output layer. The Xception net was pre-trained on `ImageNet`. Both the inductive learning networks used SGD as its optimizer since it was giving us the best results during testing. Initial settings for the parameters were a learning learning rate of 0.01, decay $1e^{-5}$, momentum 0.95 with Nesterov momentum.

4 Experiments

Feed Forward Network (FFN) We experimented with five different hyper parameters for the FFN; optimizer, activation function, batch size, learning rate, momentum and dropout. For these activations we searched through a predefined range of values for the best parameter.

The initial parameters for the network were a small batch size of 4 and running only for 10 epoch and 0.1 dropout, we used that as a starting basis for search for other parameters.

Optimizer	Accuracy
Adagrad	0.00771
Adadelta	0.00829
Adam	0.00732
SGD	0.03740
Adamax	0.00771
Nadam	0.00887

Table 1. Validation Accuracy for different optimizers after 10 epochs

There were two different optimizers we looked at when it came to deciding on a batch size. We noticed in our initial trials that the Adam Optimizer did not work well with our initial small batch sizes of 4 as seen in Table 1, so we tested larger batch sizes for Adam and found that increasing the batch size for the default SGD did not help but did improve the performance of the Adam optimizer. The tests ran 10 epochs with default settings for the optimizers.

Adam Batch Size	Accuracy
256	0.00906
128	0.02680
64	0.01060

SGD Batch Size	Accuracy
4	0.03239
8	0.03143
16	0.03008
24	0.02969
32	0.02506

Table 2. Validation Accuracy for SGD and Adam after 10 epochs

Using the SGD optimizer we ran a grid search on momentum and learning rate to see if we could find better values than the default ones given by Keras. Table 3 shows the results for the grid search, leading us to better parameters for SGD optimizer. We then ran the FFN until the model over-fit to see how far we could get with it. Figure 2 shows the training and validation accuracy and loss over the 200 epochs it ran for. 5% is quite low compared to the other networks, better than random guessing, so not very useful.

Mom \ LR	0.001	0.01	0.1	0.2
0.0	0.0913	0.0913	0.0913	0.0913
0.2	0.1324	0.0913	0.0913	0.0913
0.4	0.0913	0.0913	0.0913	0.0913
0.6	0.0913	0.0890	0.0913	0.0913
0.8	0.0913	0.0913	0.0913	0.0913
0.9	0.0913	0.0913	0.0913	

Table 3. Validation Accuracy for different optimizers after 10 epochs

Convolutional neural network (CNN) A lot of experimentation went into setting up a functional CNN, with little to no results initially. Due to a misunderstanding with the general setup of CNNs we started with the filter size decreasing as we got deeper in the network, starting with 64 filters going down to 16.

The results from running the dataset on that CNN did not get above guessing level. After correcting those errors as described in section 3.5 we were ready to experiment on different hyperparameters.

Table 4 shows results of training the net with different batch sizes. The results

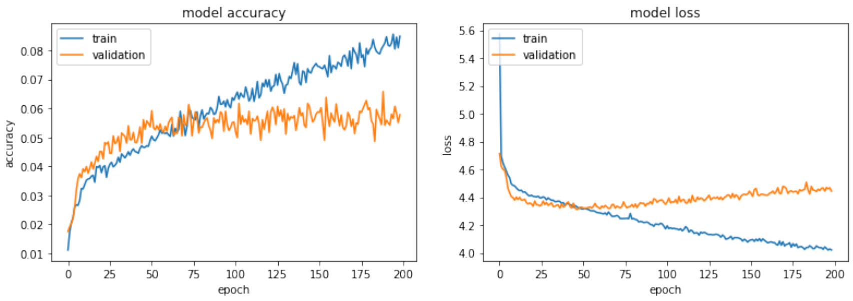

Fig. 2. Simple Feed Forward Neural Network. Achieving around 5% on the validation set

Batch Size	Accuracy
4	0.01214
10	0.01214
24	0.01214
48	0.02005
76	0.01061
100	0.03721

Table 4. Accuracy after running CNN for 5 epochs with different batch sizes

show that larger batch sizes are preferable to smaller ones. We would have liked to test even larger batch sizes but we did not have the resources for it.

Before changing to variable dropout from 0.1 to 0.5 as the network progressed we began experimenting on the effects of constant dropout on the network. We started with a dropout rate of 0.1 and intended to test rates up to 0.8. After

Dropout rate	Accuracy
0.1	0.05418
0.2	0.04203
0.4	0.02526

Table 5. Accuracy after running CNN for 10 epochs with different constant dropout rates

running three rates seen in 5 we noticed a decrease in accuracy as the dropout increased. We decided to stop early and read more about dropout rates before continuing. That lead us to the paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* described in section 2. After that we decided to call of the experiment and switch to varying dropout through the network starting with 0.1 up to 0.5 as is the standard in the industry.

With the final setup we got the accuracy of the model up to 20%, with a multi class loss of 3.28. The result of that can be seen in figure 3. There we compare the performance of the network where the final dense layers of the smaller network had 512 nodes, and an accuracy of 10%, to the performance of the final approach taken where the network had 1024 nodes.

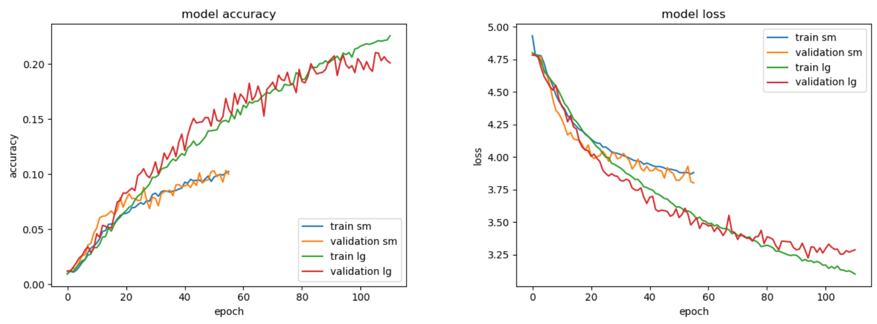

Fig. 3. Comparing different sized images for the Xception network

Transfer Learning (TLN) We experimented on two pre-trained networks, `VGG19` and `Xception`. It caught our attention that the `Xception` network was much quicker in adapting to the new objective of classifying dog breeds. `Xception` could be trained up to 62% accuracy and a log loss of 1.37 within an hour. However it would quickly plateau around that point. For the transfer learning networks we mainly focused on tuning the learning rate as there were not many layers that needed to be regulated as was the case with the convolutional network. The main drawback with using the pre-trained networks was training time and limitations in batch sizes. With the `Xception` and `VGG19` networks we could only train the network on a batch size of 50 and 100 respectively on images of size 100, and batch sizes of 20 and 50 on larger images. We had stuck to an image size of 100x100 with the convolutional and feed-forward neural networks we trained, but for the TLN part we decided to try increasing the image size. With `Xception` we managed to train the network on images sized 150x150, and 192x192 on the `VGG19` network. Increasing the image size resulted in a large increase in network accuracy and drop in multiclass loss. The results from the experiment on the Xception network can be seen in figure 4. Increasing the image size resulted in a boost in performance of around 50%, accuracy went up from 40% to 62% and multiclass log loss went down from 2.5 to 1.36.

After finding great increases in the accuracy of our TLN `Xception` by increasing the image size we decided to try the same thing for the `VGG19` network. We increased the image size from 100x100 to 192x192 pixels. Training the network with this increased size took around five times as long on wall-clock time. there fore we do not have as many epochs of execution as we do for the smaller

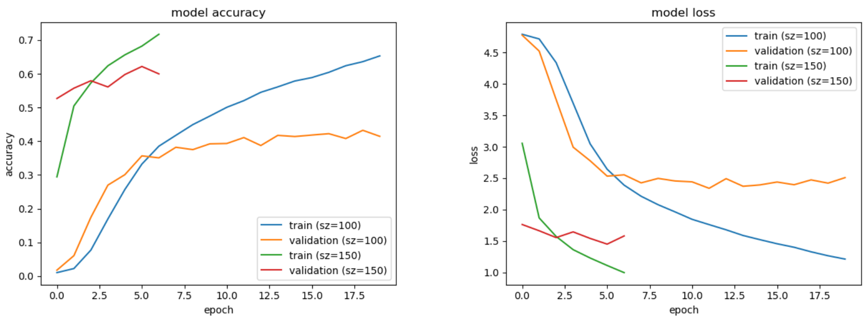

Fig. 4. Comparing different sized images for the Xception network

image size. However the results are quite clear and give us a 18% increase in the accuracy of the model. Figure 5 shows how the network not only reaches a higher accuracy but trains at a much faster rate, this tells us that the network manages to extract features much better when the image is larger.

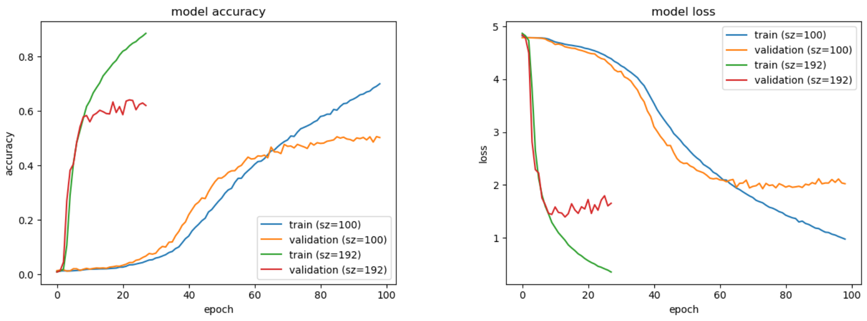

Fig. 5. Comparing different sized images for the VGG19 network

Changing the learning rate of the classifier did not seem to have much effect on the network performance, decreasing the learning rate caused the network to take significantly longer to reach top accuracy of about 43% for the Xception network with image size 100, seen in figure 6.

We got to the same conclusion with training the Xception network on the larger images, where the accuracy got up to around 62%, the comparison can be seen in figure 7. The difference with the larger images is that the multiclass log loss got down to 1.37 with the slower learning rate of 0.0001, while it only got down to 1.58 with the faster learning rate of 0.001. So even though the network was not getting more accurate, it was getting more and more certain with its class picks.

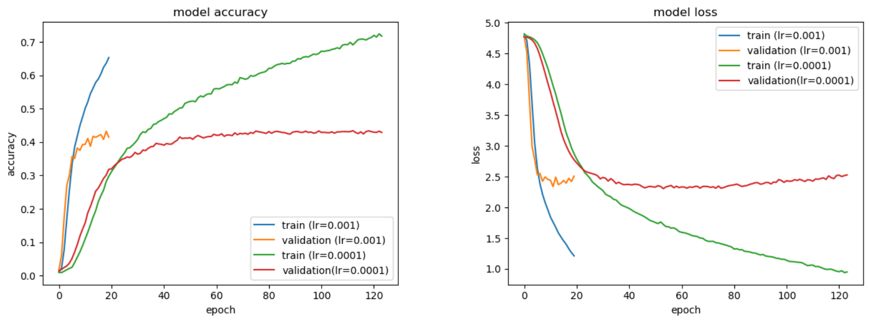

Fig. 6. Comparing different learning rates for the Xception network, image size 100

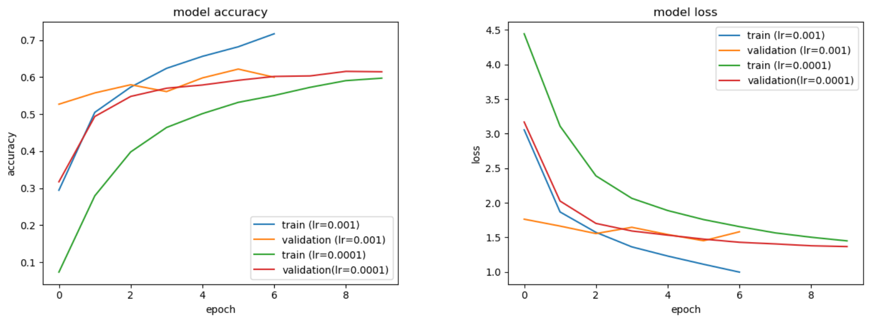

Fig. 7. Comparing different learning rates for the Xception network, image size 150

Increasing the learning rate on the VGG network drastically decreased the time it took the network to reach a plateau in the accuracy that could be learned. Figure 8 shows the difference in the accuracy and loss at the epochs.

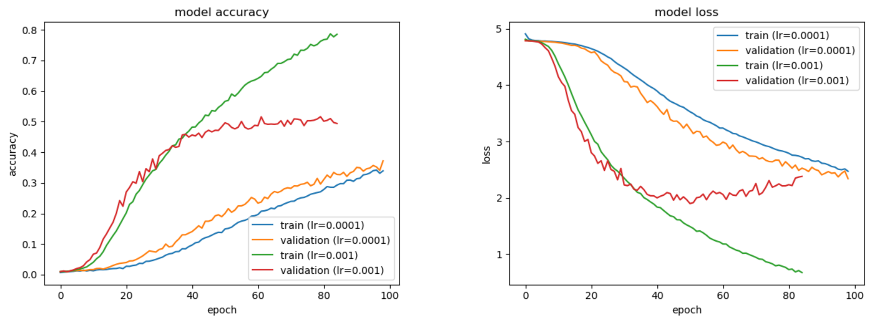

Fig. 8. Comparing different learning rates for the VGG network, image size 100

As discussed earlier in section 3.6 freezing layers in the pre-trained `VGG` network resulted in worse accuracy. Figure 9 shows the increase in accuracy that is gained from allowing the whole network to train without freezing. Execution time of the training did increase by around 15% when the layers were unfrozen.

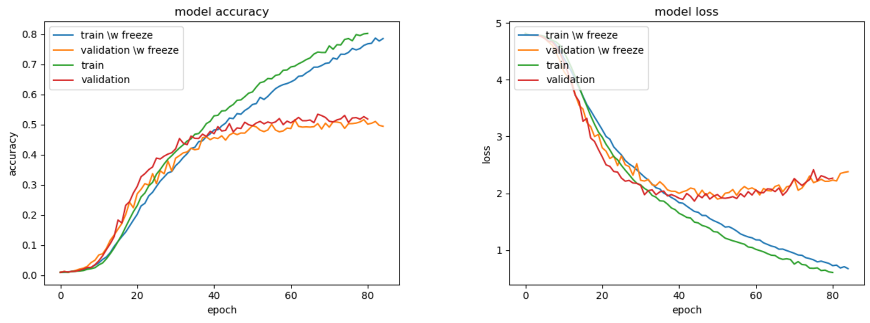

Fig. 9. Transfer Learning Network, comparison between freezing and not freezing layers in the pre-trained `VGG19` network.

5 Conclusion

The goal we set was to get within the top 20% of submissions in the Kaggle dog breeds competition. We did not reach that goal, instead we reached the top 50%

with a multiclass log loss of 1.36. There are several factors which we think might have contributed to that. They are outlined below as well as general conclusions about the network. The major factor regarding the competition performance is in our opinion computational power, having to scale down the images gives the neural nets less information to work with and makes feature extraction harder. Here are some general conclusions we have reached after working on this project.

– larger images as input allows for better feature selection
– freezing layers in VGG was beneficial in decreasing execution time but did decrease accuracy quite a lot
– Training with more powerful computers capable of handling larger images and neural nets would produce better results and was probably the biggest factor for us not getting better results.

-

6 Link to project repository

Here is the link to the github repository https://github.com/steinnp/DeepLearning. In the project we worked with Keras and Tensorflow with gpu support as a backend. The GPUs used for training were NVIDIA Quadro M2000M, and NVIDIA GTX780.

References

1. Aditya Khosla, Nityananda Jayadevaprakash, B.Y.L.F.F.: Stanford dogs dataset (http://vision.stanford.edu/aditya86/ImageNetDogs/)
2. Kaggle Inc.: Kaggle homepage (https://www.kaggle.com)
3. Kaggle Inc.: Dog breed identification (https://www.kaggle.com/c/dog-breed-identification)
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
5. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1) (2014) 1929–1958
6. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering **22**(10) (2010) 1345–1359
7. Multi class log loss formulation: (http://wiki.fast.ai/index.php/Log_Loss#Multi-class_Classification)
8. Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., Garcia, R.: Incorporating second-order functional knowledge for better option pricing. In: Advances in neural information processing systems. (2001) 472–478
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
11. Brock, A., Lim, T., Ritchie, J., Weston, N.: Freezeout: Accelerate training by progressively freezing layers. arXiv preprint arXiv:1706.04983 (2017)
12. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. arXiv preprint (2016)