Image Project: GriGri Guys

Matias Paloranta Antti Pirttikoski

May 31, 2020

1 The data

The given image data set consists of 20000 images of size 128x128. Each image can belong to 14 classes: `baby bird car clouds dog female flower male night people portrait river sea tree`. In addition, each image can belong to multiple classes simultaneously, i.e. we have a multi-label classification problem. A large majority of the images are RGB, and a few are grayscale, so we decided to change the greyscale images to RGB during training. Almost half of the given images, 9824 more precisely, do not belong to any of the classes, so a network always predicting that an image does not belong to any of the classes would achieve $9824/20000 = 49.12\%$ accuracy. This can be used as a baseline to which we compare the performances of the networks.

2 Splitting the data

To establish our training pipeline, the first thing to do is to split the data into training and validation sets so that we can measure the performance of our neural network as training progresses. The class distribution of the data is pretty imbalanced, and we want both data sets to have a similar distribution. Otherwise we could end up with all examples of some class in the other data set, and none in the other. In multi-class classification we could do this by stratified sampling, but in our case the data is also multi-labeled, so the strata would not be exclusive. Luckily there is a method of stratified sampling for multi-label classification called *iterative stratification* (Sechidis, Tsoumakas, and Vlahavas (2011)), for which an implementation is provided in the Python library `skmultilearn`. We apply the commonly used 80/20 split, i.e. we split 80 % of the data into training and the rest into validation. Using iterative stratification, we end up with the following class distributions:

```
Training data relative class frequencies:
[0.0046875 0.018     0.0159375 0.05475   0.022375  0.161375  0.0380625
 0.1489375 0.029875  0.320125  0.1560625 0.006     0.008625  0.02625  ]

Validation data relative class frequencies:
[0.005   0.018   0.016   0.05475 0.0225  0.16125 0.038
0.149   0.03  0.32025 0.156   0.006   0.00875 0.02625]
```

The frequencies correspond to the classes `baby bird car clouds dog female flower male night people portrait river sea tree`, respectively. One can see that we have virtually identical class distributions in both data sets, as we had hoped.

3 Activation layer and loss function

We have three separate cases we wish to examine. First, we define simple CNN which we train from scratch and measure its performance on the given classification task. Second, we take a model pretrained on ImageNet and compare its performance vs the simple CNN. Third, we take the pretrained model and fine-tune it by training it on our own image data, and see if its performance improves.

In the given image classification task the images can belong to 14 different classes. In addition, each image can belong to multiple classes simultaneously. Thus we require that the output layer has 14 neurons corresponding to each of the 14 classes. Each neuron then spits out a probability that an image belongs to the class corresponding to that neuron, independently of the other neurons. Therefore we use sigmoid activation in the output layer.

Next we have to choose the loss function. Note that we can reduce the classification problem to 14 binary classification problems, where every neuron produces a probability that an image belongs to class 0 or 1. Thus we can use binary cross-entropy as a loss function for each of the individual neurons, and then take the average of these as our loss function. The other choise would be to use e.g. mean square loss. We tested the performance of both loss functions with SGD (learning rate=0.01, momentum=0.9) by training a pre-trained ResNet18 on our own image data for 20 epochs. We chose ResNet18 because it is relatively fast to train, while still achieving good performance. When measuring the performance of a network, we use micro-averaged F1-score as the performance metric, since it combines precision and recall into a single number, and is more suited to multi-label classification than e.g. accuracy, which doesn't take into account partially correct labels. A batch size of 32 is used throughout this report. This is because large batch sizes take a lot of GPU memory, and a regularizing effect is also provided by smaller batch sizes.

Using binary cross-entropy loss we achieved the following result:

```
Best val F1: 0.716980
Acc: 0.6402
```

The evolution of the loss, F1-score and accuracy is visualized in Fig. 1.

Using mean squared error loss we achieved the following result:

```
Best val F1: 0.659415
Acc: 0.5880
```

The evolution of the loss, F1-score and accuracy is visualized in Fig. 2.

Although with MSE loss the training and validation curves for all metrics seem to follow each other more closely, with BCE loss we achieve higher performance metric values

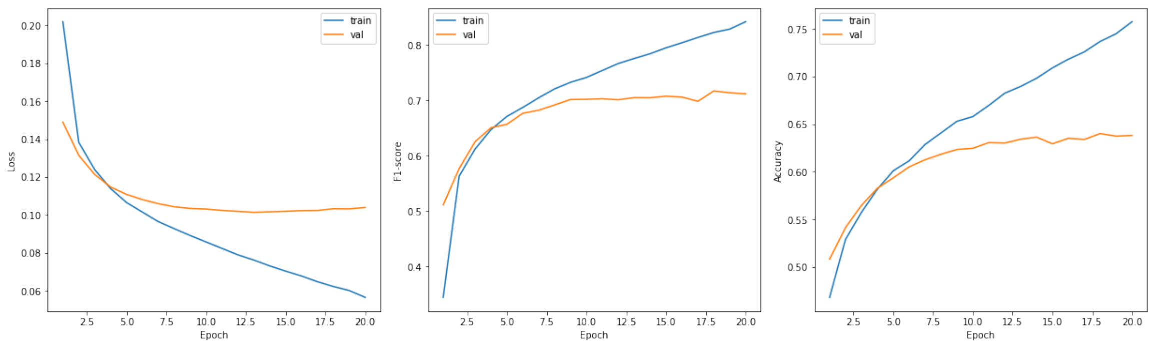

Figure 1: Loss, F1-score and accuracy of ResNet18 trained for 20 epochs using binary cross-entropy loss and SGD with learning rate of 0.01 and momentum of 0.9.

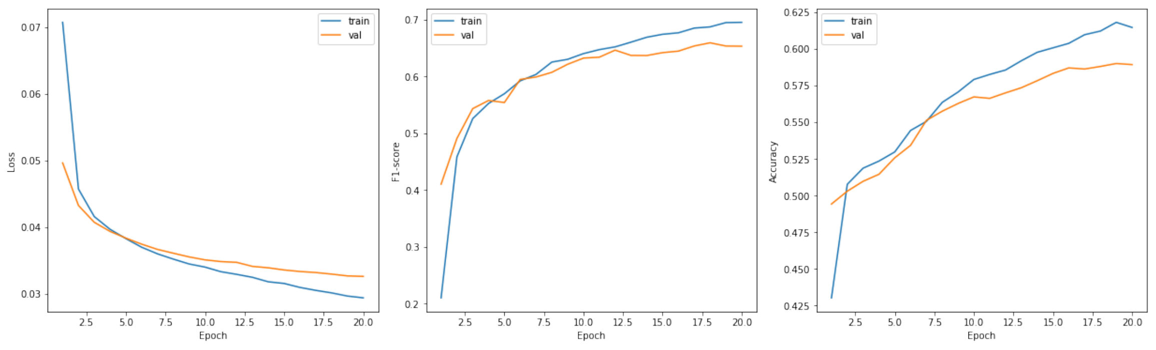

Figure 2: Loss, F1-score and accuracy of ResNet18 trained for 20 epochs using mean squared error loss and SGD with learning rate of 0.01 and momentum of 0.9.

quicker, despite the obvious overfitting. This is why we decided to stick with BCE as the loss function, and instead focused more on regularization to prevent overfitting.

4 Optimizers

In order to make the learning more efficient, we thought about using other optimizers than SGD. We compared the performance of SGD with momentum and learning rate scheduling to Adam, which was chosen because it is robust to hyperparameter values and usually good. The values chosen were learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. We trained the same model (ResNet18) as in previous section for 20 epochs. The loss function is binary cross-entropy. The results for Adam are shown in Fig. 3. As we can see, using Adam the network overfits greatly. To reduce this overfitting, we tried weight decay (i.e. L2-norm) values of up to 0.001, but the shapes of the curves remained very similar. Going to higher weight decay values caused the network to pretty much stop learning altogether. RMSprop and AMSgrad variant of Adam were also tested, but they performed similarly or even worse than Adam. This prompted us stick with SGD.

However, for SGD we used fine-tuning to achieve a comparable capacity using weight decay of 0.001 and learning rate scheduling. We used cyclic learning rate scheduling as presented in Smith and Topin (2017), which cycles the learning rate between some high

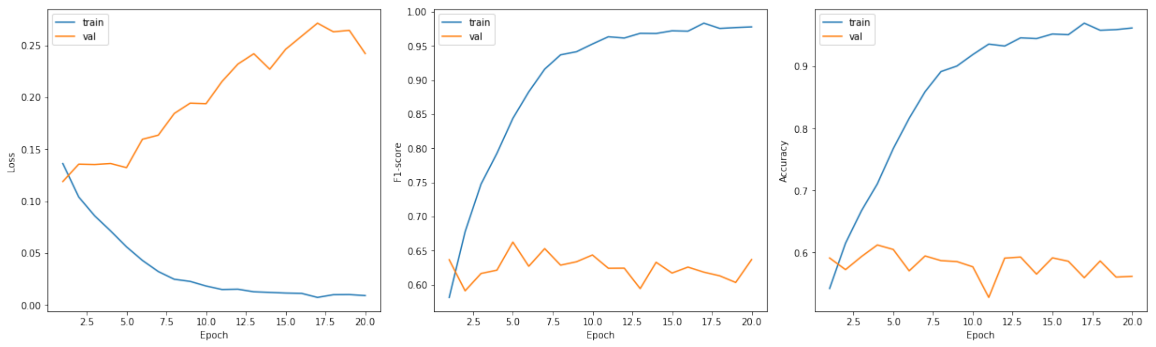

Figure 3: Loss, F1-score and accuracy of ResNet18 trained for 20 epochs using binary cross-entropy loss and Adam with learning rate of 0.001 and $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

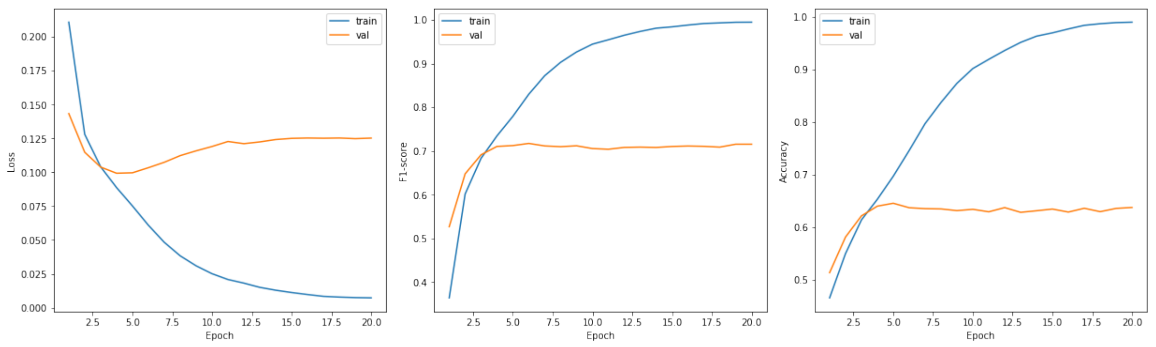

Figure 4: Loss, F1-score and accuracy of ResNet18 trained for 20 epochs using binary cross-entropy loss and SGD with weight decay of 0.001 and cyclic learning rate scheduling, where the learning rate is cycled between 0.01 and 10^{-7} and momentum between 0.85 and 0.95.

value (here 0.01) and some low value (here 10^{-7}). However, the authors suggested using a high maximum value for the learning rate (e.g. 0.1 or even 1), but we found that a modest choice of 0.01 performed better. The momentum is also cycled between 0.85 and 0.95. This way we achieved a training accuracy and F1-score of nearly 100 % (see Fig. 4), while the validation metrics remained much smoother than with Adam.

5 Data augmentation

To reduce overfitting and improve generalization performance, we applied data augmentation to the training images. The applied transforms were random resized cropping, random horizontal flipping and random rotation. The resized crop creates crop of random size (between 8 % and 100 %) of the original size and also a random aspect ratio (3/4 to 4/3) of the original aspect ratio is made. Then this crop is finally resized to the given value, which here was the original size of 128x128 pixels. Random horizontal flip has a 50 % chance to flip image horizontally, while random rotation rotates the image randomly -25 to 25 degrees. Again, we trained the same ResNet18 model for 20 epochs with the SGD defined in the previous section. The results are below (and visualized in Fig. 5):

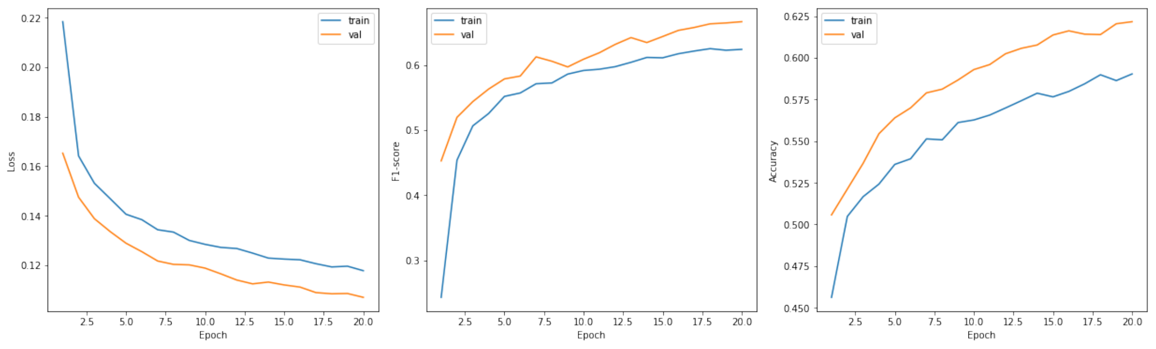

Figure 5: The same ResNet18 model trained for 20 epochs using data augmentation and SGD with cyclic learning rate scheduling as defined in section 4.

```
Best val F1: 0.666839
Acc: 0.6218
```

Using data augmentation, we reduced overfitting significantly, although the network now takes longer to train.

6 Model selection

What we have learned so far?

- Use BCE as the loss function for faster learning.
- Use SGD with momentum and cyclic learning rate scheduling as optimizer.
- Use data augmentation and weight decay for regularization and improving generalization performance.

With all these in mind, we want to choose a network with good performance on ImageNet, while making sure that our network, alongside the data, can fit inside a GPU with 4 GB of memory. In addition, the pre-trained models provided by `torchvision` assume input images with height and width at least 224 pixels. However, this would again increase the memory required, which further limits our choices. After a series of experiments, we end up at two different choices for our model: ResNet34 is the best performing model we can train on resized images of size 224x224 pixels, and Wide ResNet-50-2 is the best performing model we can train on images of the original size of 128x128 pixels, while adhering to the memory requirements. We trained both networks for 20 epochs using BCE loss and SGD as defined in section 4 with weight decay value of 10^{-4}. In addition, we used data augmentation as eplained in section 5. The result for ResNet34 with image size 224x224 pixels is below:

```
Best val F1: 0.770906
Acc: 0.6913
```

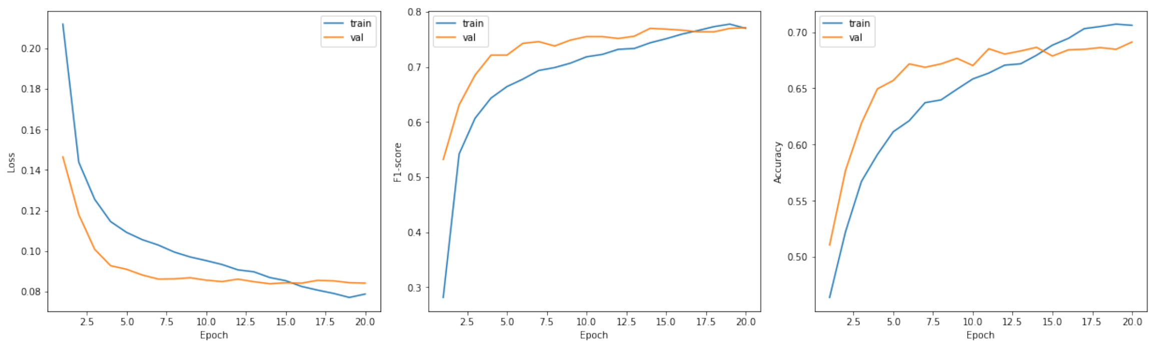

Figure 6: Loss, F1-score and accuracy of ResNet34 trained for 20 epochs on data-augmented 224x224 images using binary cross-entropy loss and SGD with weight decay and cyclic learning rate scheduling.

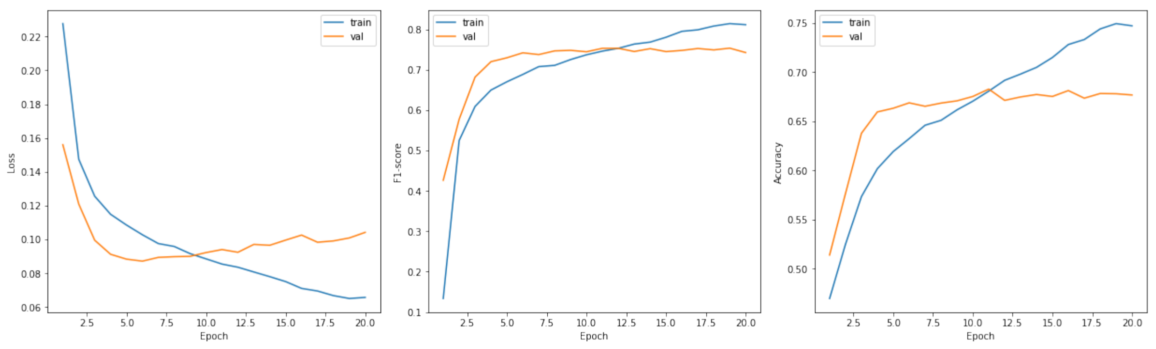

Figure 7: Loss, F1-score and accuracy of Wide ResNet-50-2 trained for 20 epochs on data-augmented 224x224 images using binary cross-entropy loss and SGD with weight decay and cyclic learning rate scheduling.

The results are visualized in Fig. 6.

The result for Wide ResNet-50-2 with image size 128x128 pixels is below:

```
Best val F1: 0.753539
Acc: 0.6780
```

Again, the results are visualized in Fig. 7.

Looking at the figures, Wide ResNet-50-2 overfits more while its accuracy and F1-score on the validation remain lower than that of ResNet34. Thus we choose ResNet34 as our final model. It still overfits a little, but not too much.

Now that we have chosen our network, we can make a sanity check by passing our input through the *untrained* network to see that training really does improve its classification ability. Passing our whole validation set through the network results in:

```
Acc: 0.0000 F1: 0.1256
```

So the network actually does learn (a lot) by training.

Another check we can do is if we only train the final layer by freezing all layers except the final one, and seeing if this is an improvement over fine-tuning the whole network. So, again we train for 20 epochs with the exact same parameter settings as previously. The results were as follows:

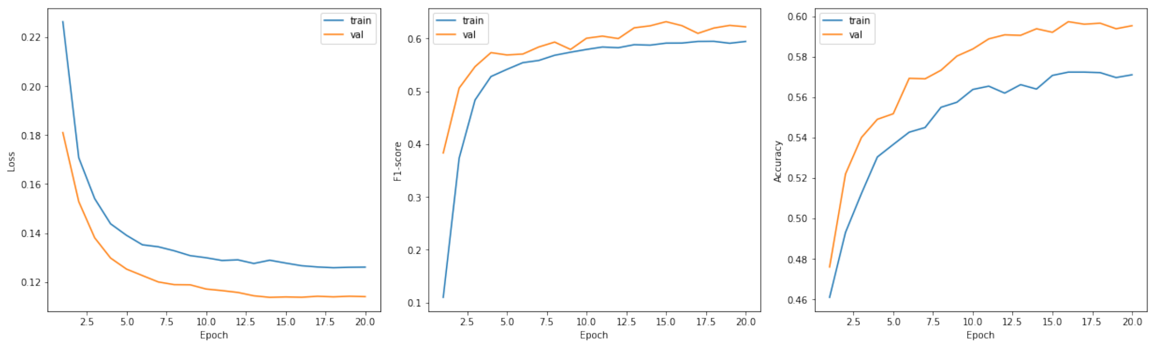

Figure 8: Loss, F1-score and accuracy of ResNet34 trained for 20 epochs on data-augmented 224x224 images using binary cross-entropy loss and SGD with weight decay and cyclic learning rate scheduling. The weights of all other layers than the output layer are frozen.

```
Best val F1: 0.631523
Acc: 0.5920
```

This is visualized in Fig. 8. The network seems to have pretty much converged.

We observe that fine-tuning the whole network yields an improvement over training only the final layer.

Finally, we can check if changing the probability threshold for selecting a label improves the performance of the model. We can do this by using an optimization algorithm to minimize the *negative* F1-score w.r.t. the threshold value. So far we have used a general threshold of 0.5, but this might not be optimal since we have an imbalanced data set. After running the algorithm we end up with a threshold of around 0.375. This results in an accuracy of 0.7015 and F1-score of 0.7870 on the validation set. A small improvement. If we instead use the same algorithm to select the threshold for each class individually, we end up with all thresholds close to 0.5, and accuracy of 0.7120 and F1-score of 0.7834 on the validation set. So regardless of the choice of a global threshold or individual thresholds, we end up with virtually the same result. Here we choose to go with the global threshold of 0.375.

7 Simple ConvNet

A simple convolution network was implemented to compare the performance with the pre-trained models. This network consists of convolution layers which perform 3x3 convolutions with stride of 1 and padding of 1. Between the convolution layers there are pooling layers, where 2x2 max pooling is performed without padding and with stride of 1. The last three layers are fully connected layers. In all the layers RELU is used as activation function and the binary cross entropy was chosen as the loss function. The exact architecture of the model is shown below.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
```

```
===============================================================
           Conv2d-1         [-1, 32, 128, 128]             896
           Conv2d-2         [-1, 32, 128, 128]           9,248
        MaxPool2d-3           [-1, 32, 64, 64]               0
           Conv2d-4           [-1, 64, 64, 64]          18,496
           Conv2d-5           [-1, 64, 64, 64]          36,928
        MaxPool2d-6           [-1, 64, 32, 32]               0
           Conv2d-7          [-1, 128, 32, 32]          73,856
           Conv2d-8          [-1, 128, 32, 32]         147,584
           Conv2d-9          [-1, 128, 32, 32]         147,584
       MaxPool2d-10          [-1, 128, 16, 16]               0
          Conv2d-11          [-1, 256, 16, 16]         295,168
          Conv2d-12          [-1, 256, 16, 16]         590,080
          Conv2d-13          [-1, 256, 16, 16]         590,080
       MaxPool2d-14            [-1, 256, 8, 8]               0
          Conv2d-15            [-1, 512, 8, 8]       1,180,160
          Conv2d-16            [-1, 512, 8, 8]       2,359,808
          Conv2d-17            [-1, 512, 8, 8]       2,359,808
          Linear-18                  [-1, 100]       3,276,900
         Dropout-19                  [-1, 100]               0
          Linear-20                  [-1, 100]          10,100
         Dropout-21                  [-1, 100]               0
          Linear-22                   [-1, 14]           1,414
===============================================================
Total params: 11,098,110
```

The network was trained 20 epochs. During the training the best accuracy achieved on the validation set was 49.26 %. This is practically the same accuracy as one would get by predicting always the most probable outcome i.e. that the picture does not belong to any of the given 14 categories. The F1-score was 0 since the model did not predict any positive outcomes. As this simple model performs so poorly (or takes forever to learn), we decided not to spend too much time fine-tuning and moved on to the pre-trained models, since they perform better anyway.

8 Summary and discussion

Our benchmark to beat was 49.12 % accuracy, which would be achieved by always choosing an image to not belong in any of the classes. However, this would yield an F1-score of 0. By training a simple CNN from scratch (albeit not very optimized) we basically achieved this result.

On the other hand, using transfer learning and fine-tuning a network pre-trained on ImageNet resulted in an accuracy of over 70 % and F1-score of almost 0.79 on the validation set, which is a massive improvement.

How to improve this result? One could spend a small eternity fixing the hyperparameters of the optimizers and learning rate scheduling, but we didn't do this very much since the training was pretty slow. One could thus see improving here. Another and likely much more massive improvement would probably come from just collecting way more data. Some classes in the training data were very underrepresented (e.g. 0.5 % of the pictures had the `baby` label, while 32 % had `people`), which makes it hard for the network to learn features corresponding to these classes. This could also be somewhat remedied by using some advanced sampling technique to oversample the minority classes and/or undersample the majority classes. However, the multi-labeled nature of this classification problem makes this complicated. Furthermore, almost half of the images didn't belong to any of these classes, which is again asking for more data. Having more powerful GPUs with larger memory would also be helpful, allowing to experiment more with state of the art architectures and larger images.

References

Sechidis, K., Tsoumakas, G., & Vlahavas, I. (2011). On the stratification of multi-label data. In D. Gunopulos, T. Hofmann, D. Malerba, & M. Vazirgiannis (Eds.), *Machine learning and knowledge discovery in databases* (pp. 145–158). Berlin, Heidelberg: Springer Berlin Heidelberg.

Smith, L. N., & Topin, N. (2017). *Super-convergence: Very fast training of neural networks using large learning rates.*