Convolutional Neural Network for Multiclass Image Classification

Department of Computer Science, University of Surrey, United Kingdom
Manuel Bradicic (mb01761)

Introduction

Classification is one of the most crucial tasks in Data Mining and has become very popular among the computer scientists and mathematicians, especially in the past decade. The idea is to assign items in a collection to target categories, called classes. The final goal is to accurately predict the target class for a given data.
The classification that we are going to deal with in this work is Supervised Image Classification. In supervised learning the classification model is trained on a set of images with a predefined target class, based on this process the model learns how to recognise and distinguish each class of dataset.

This work deals with Multiclass Image Classification of a butterfly dataset using Convolutional Neural Networks (ConvNets). The dataset consists of the training set of 10270 images of butterflies which were narrowed down into 23 classes – families of butterflies. Furthermore, the model's accuracy is tested on the test set of 15009 examples.

In the following columns, we will dive deep into analysis of the problems that I faced with during the development of this coursework. At the start of this project, I wasn't familiar with implementing other pretrained CNN models (Transfer learning) therefore the flow of this project will follow the progress of my understanding of the image classification with CNN. Furthermore, we will discuss different technical issues that I opposed and explain the importance of the research and understanding of the model's structure in order to achieve high accuracy. Nonetheless, we will talk through different techniques which helped to bring the model's accuracy up, how to prevent model's overfitting, balancing the dataset and explain hyperparameters that I used.

To solve the problem, I will be using Keras and TensorFlow frameworks. These frameworks, a collection of packages and libraries, are among the most popular frameworks when it comes to Deep Learning. [1]

Classification algorithms

Before going into the discussion about the solution to the problem, it is very crucial to understand what makes CNN the best classification algorithm for this problem. There are many different types of Classification Algorithms, such as: Logistic regression, K-nearest neighbour (KNN), Support vector machines (SVM) etc. However, not all of them are a potential solution to our problem.

In [5] the authors show the comparison of the performance of the best models against the other baseline techniques on the COCO dataset. The chart in [5] shown that Xception (a ConvNet), outperforms other classifiers. The non-linear classifiers such as SVM, Voting and MLP reveal a better performance than the linear ones and KNN.

However, it is important to understand why it is so. For instance, The SVM does not preform well when the number of features is greater than the number of samples. More work is needed in feature engineering for an SVM than that is needed for a ConvNet [6]. Moreover, another machine learning algorithm worth considering is KNN. It is one of the simplest machine learning algorithms. KNN calculates the Euclidean distance from every single data point in our training set and for a prediction it gives the output based on the majority 'k' nearest neighbours. It is not arguable that KNN can perform very well on image classification. Saying that, there are KNN algorithms which can classify handwritten digits and achieve accuracy of 95.32% [1]. At first glance, it seems like this ML algorithm would be a great point to start from, however, KNN works well with a small data sets and it struggles when the input data is large. Further, as the number of dimensions increases the volume of the input space increases at an exponential rate, and in simple 2D and 3D spaces breaks down. This might feel unintuitive at first, but this general problem is called the ***Curse of Dimensionality*** [7].

In conclusion, CNN utilize spatial information that for instance KNN algorithm doesn't which brings down the number of parameters and overall complexity while learning similar information. For small problems, this is unnecessary and can make CNN expensive to train. However, out train dataset consists of 10270 RGB images of 23 different classes which makes our dataset very complex and too compound for KNN algorithm to perform well. Further the model's performance is tested on the unseen dataset of 15009 images.

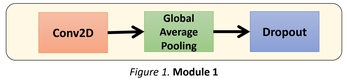

Figure 1. **Module 1**

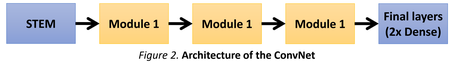

Figure 2. **Architecture of the ConvNet**

CNN Model

As you could conclude from the previous two sections, in order to solve this problem, I will be using Convolutional Neural Networks (CNN). CNN's architectures are designed to recognize visual patterns directly from pixel images. There are many different models which could be considered for our problem. Each model has been precisely designed by computer scientists and most of them have different topological depth of the network. These networks include layers which are basic building blocks of neural networks in Keras. In the following section we will consider a very simple CNN that I started with.

My Very First ConvNet
Firstly, we will consider the first model that I have made. This model doesn't use any of the pre-trained models or already pre-assigned weights. This model consists of multiple layers (Module 1 - *Figure 1*). It uses Conv2D, global average pooling and a dropout layer. Convolutional layers are used to produce a tensor of outputs and emphasise the features of the input data. Afterwards, the number of the parameters are reduced by using global average pooling in order to minimize overfitting and to reduce the noise of the data. Finally, dropout layer which 'turns off' a certain percentage of the hidden neurons to minimize overfitting as well. This module is repeated 3 times. Followed by flattening the data, and two dense layers (Module 2 - *Figure 2*). The images are imported in the resolution of 64x64, final dense layer uses '*softmax*' activator which normalizes the output into a probability distribution and the gradient descent optimization algorithm Adam with learning rate of 1e-4.

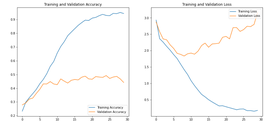

Figure 3. **Model's training and validation accuracy.** As show the model achieved the accuracy of 48% on the test dataset. It is evident that the model's topology (Figure 2) is not complex enough in order to achieve high accuracy before it starts to overfit itself. In other words, model's structure is too weak to learn most of the patterns in the images. Hence, it becomes very tight and inflexible

There are a couple of techniques that could've been used to bring the accuracy slightly up (such as balancing the dataset), however, that wouldn't increase the accuracy to the needed level. Therefore, we should seek for some alternatives and the one that we are going to look up is transfer learning.

Transfer Learning

Transfer learning is the reuse of a pre-trained model on a new set of data, and it has got several benefits, but the main advantages are saving training time and better performance of Neural Networks [2]. In fairness, transfer learning is a process of modifying an existent CNN model by fine-tuning previously trained CNN. So, instead of training the neural network from the scratch (such as in *My Very First ConvNet*), we use pre-defined weights and finetune them for our own purpose

In the following section we will explore the performance and the architecture of some of the most popular ConvNets:

Model	Parameters	Depth	Validation Acc.	Test acc.
Xception	22,910,480	126	95.29%	93.50%
InceptionV3	23,851,784	159	95.10%	92.60%
VGG16	138,357,544	23	87.50%	86.23%
EfficientNetB5	**30,562,527**	**-**	**96.56%**	**94.81%**

Table 1. ***Famous*** **ConvNets, Their Structure and Performance Results on the Butterfly dataset.**

All of these models from the *Table 1* have got very complex and compound architectures. They were built by the world leading scientists and researchers such as VGG16 which was designed by the scientists from the University of Oxford. Furthermore, these models were trained for ImageNet competition – ' Large Visual Recognition Challenge' on a database of 1000 classes distributed across 14 millions images across multiple GPUs for 2-3 weeks.

As visible from the table *(Table 1)*, I did a comparison of these 4 models on the butterfly database. It is important to notice that I didn't trained these models from scratch but fine-tuned them. Which simply means that I used the pre-defined weights, as the initial weights, from the ImageNet classification and modified them for my purpose. This way we save some time on optimizing the weights according to the dataset.
It is evident from the table that the models achieve significant test accuracy with the highest test accuracy of **94.81%.** This was achieved using EfficientNetB5 model. All the models were trained on the balanced dataset (more in the Handling Imbalanced Dataset).
In the rest of the project, we will considerate all the hyperparameters and many different techniques which rose the accuracy by a noteworthy percentage.

Model Scaling and EfficientNet family

ConvNets are mostly designed at a fixed resource budget. Although achieving higher accuracy requires greater resources, we have already touched the hardware memory limit. Therefore, scientists have to turn towards developing more efficient models.

The ImageNet competition takes place every year since 2010 and as we hit the memory limits, further accuracy gain requires well planned and systematic structure of models. Scaling up ConvNets is widely used to achieve better accuracy. For instance, GPipe achieved 84.3% ImageNet top-1 accuracy by scaling up a baseline model four time larger. However, it is well known that ConvNets have never been very well understood, hence there currently many ways of doing it.

One of the scaling methods proposed by Google AI [4], so called: 'Compound Scaling method', proposed that fining the balance between all dimensions of network width/depth/resolution is very crucial for bringing the accuracy up. It is generally assumed that the models' structures are too wide, deep or with a very high resolution.

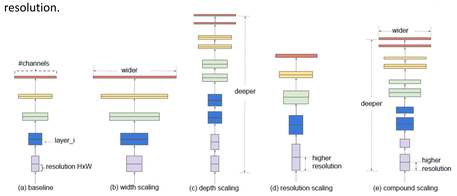

Figure 4. **Model Scaling.** (a) is a baseline of a ConvNet; (b)-(d) demonstrate scaling by increasing only one dimension width, depth, resolution. (e) represents compound scaling method that scales all three dimensions with a fix ration[4]

The paper proposes that deeper structure doesn't always bring the accuracy up. The family of ConvNets called EfficientNet was designed using the proposed scaling method. It was proved that EfficientNet family with considerably fewer numbers of parameters is efficient and provides better results.

It was proposed that if we want to use use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image size by γ^N where α, β, γ are constant coefficients. EfficientNet uses a compound coefficient Φ to uniformly scale networks dimensions. In particular, the best proposed values for EfficientNetB0, to scale up the architecture, are $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.5$, considering that $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.

The same study has shown that EfficientNetB7 achieves the same accuracy as Gpipe even though its structure contains 8.4x fewer params and 6.1x faster.

Challenges

There are couple of challenges that I faced with during the development of this project, mainly because of the hardware limits.

One of the biggest challenges was RAM over usage. There are many different ways of delivering images to the model during the train process. If one tries to load all the images into a NumPy array, at some point he will reach RAM memory limit, moreover, using a high-resolution images won't be an option as it requires even more memory. Hence, developers must design cost-effective way of delivering input data.
In order to prevent RAM over usage I implemented Data Generators. Essentially, data generator loads dataset in real time of batch size 32 and feed it right away to the model. This allows us to import images in high resolution and work with large datasets.

Further, when the model size becomes very large due to the size of input images, it is not possible to keep the entire model in the GPU. In my case I had to set up input data resolution to 224x224 otherwise model size exceeds the GPU memory of 15.9GB. Many papers have shown that higher resolution brings the accuracy up by the significant percentage. Moreover, I did the same experiment, and it proved the same. We can conclude that higher accuracy increases the size of hidden layers which allows the model to consider more patterns in the images.

Finally, after observing the training dataset I noticed that the dataset is very unbalanced. (check *Figure 5)*

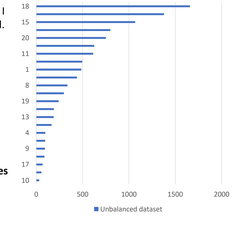

Figure 5. **Unequal distribution of the images across the 23 classes.**

EfficientNet-B5

EfficientNet-B5 is one of the eight 'members' of EfficientNet family. Using this pre-trained model on ImageNet dataset, I achieve validation accuracy of 96.56% and test accuracy of 94.81%.

In order to achieve higher accuracy, I had to fix imbalanced dataset (more in *Handling Imbalanced Dataset*), which rose the accuracy of the model from 89.98% to 94.81% (check Figure 13). The total count of images used for training and validation the model was 18312, where the train and validation set split in ratio of 75:25.

Furthermore, the model was trained using following settings: The resolution of the input data was 224x224 (see *Challenges)*. In the final layer I included further data augmentation on the input dataset as well as pre balanced dataset. Moreover, the final layer consists of a GlobalAveragePooling2D layer, a dense layer of size 512, and a dropout layer of 55%. The model was trained on a scale of 20 epoch, and the data was delivered in batches of size 32. The optimizer was RMS with the learning rate of 10e-4.

Observation: as visible in the *Figure 6.* The model achieved the highest accuracy using RMS optimizer. Initial learning rate (lr) was 10e-4 however, *ReduceLROnPlateau (see Useful techniques)* would decrease the lr by 0.5 if the val_loss doesn't increase 3 times in a row. This way optimizer finds the lowest point in the minimum that is located as the steps would get smaller and smaller. Furthermore, I performed several trainings and ration 3:1 between training and validation set proved to be the best.

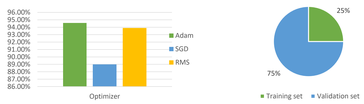

Figure 6. **Optimizers vs Accuracy.** The model was trained, and validation was tested using several optimizers: Adam, SGD, RMS.
Figure 7. ***Train and validation split.*** The dataset was split between train and validation set. Ratio: 75:25

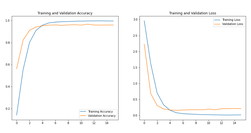

Figure 8. **Training and Validation Accuracy/Loss.** It is visible that the model's accuracy peaks on the 11th epoch. Using EarlyStop and ModelCheckPoint I managed to save the model on the 11th epoch and prevent overfitting and stopping the training process early enough. Further, model's loss in the lowest on the 11th epoch

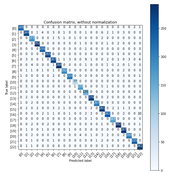

Figure 9. **Confusion Matrix of imbalanced dataset.** This Figure represents the confusion matrix of the validation set.

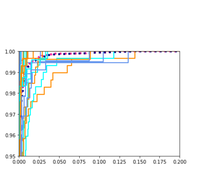

Figure 10. **Final Model ROC Curve**

Evidenced from the Figure 10, on average the model is almost having the AUC of 1 proving that it has got a good measure of separability. Whereas a poor model has AUC near to the 0. Furthermore, it is visible from the Figure 11, that the average precision and average recall are very similar one to each other. This means that the ration of FP and FN is very low.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	229
1	0.95	0.92	0.94	291
2	0.93	0.95	0.94	197
3	0.99	0.99	0.99	296
4	0.96	1.00	0.98	256
5	0.99	1.00	0.99	266
6	0.94	0.98	0.96	296
7	0.97	0.97	0.97	298
8	0.98	0.97	0.97	198
9	0.97	0.97	0.97	222
10	0.97	1.00	0.99	77
11	0.97	0.94	0.95	185
12	0.91	1.00	0.95	135
13	0.98	0.98	0.98	281
14	1.00	0.99	0.99	210
15	0.94	0.94	0.94	300
16	0.95	0.92	0.94	263
17	0.98	0.99	0.99	164
18	0.98	0.98	0.98	298
19	0.96	0.99	0.97	200
20	0.97	0.92	0.94	226
21	0.99	0.98	0.99	240
22	0.94	0.99	0.96	284

Average Precision: 0.9656572741403338
Average Recall: 0.9654405263157895
Average F1: 0.9653478788371641

Figure 11. ***Precision, Recall, F1***

Useful techniques

Training model in an iterative process with many parameters and methods to adjust. It is crucial to be able to train model in a normal amount of time with high accuracy. Hence it is of great importance to consider many different approaches to improve training process.

Early Stop – it is always challenging to decide for how long to train a model. This callback monitors accuracy of model and if no improvement is a certain number of epochs, the model stops training. Further it prevents overfitting (Figure 6 epoch 11)
I was monitoring 'val_loss' and patience was 6.
Model checkpoint – saves weights of a model with highest accuracy.(Figure 6 – epoch 11) This technique is performed to reduce the number of unnecessary compiles.
ReduceLROnPlateau - This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. In my case, the lr would get reduced by a factor of 0.5 and the patience equals to 2.

Handling Imbalanced Dataset

One of the most important steps of building up a successful solution, is analysing and exploring the dataset. One of the most common issues found in datasets that are used for classification is **imbalanced classes** issue [3]. Balancing the dataset is very important because we want our model to spend approximately the same amount of time on all the classes, otherwise the model gets too 'addicted' to the classes with the majority of samples (such as class 7 and 18), and unable to predict the classes with low number of samples (such as class 0,3,4). Moreover, it makes the model more suitable to overfitting.

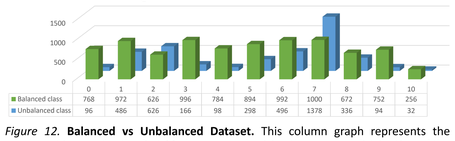

Figure 12. **Balanced vs Unbalanced Dataset.** This column graph represents the number of images for the first 11 classes, emphasizing the unequal distribution of the dataset. It is crystal clear that our dataset is not balanced. For instance, class 10 contains 32 images, whilst on the other hand class 7 has got 1378 images. This leaves us with the 43:1 ratio between those two classes.

Therefore, we must resample the dataset in order to solve the class imbalance problem. **Undersampling** is the process when we randomly delete some of the samples of the classes, on the other hand, **Oversampling** (aka Data Augmentation) is a technique to increase the amount of data by adding slightly modified copies of already exiting data.

Evidenced in Figure 8, Undersampling was performed on the class 7 which contains 1378 images, however during the training process, the class was unsampled to 1000 images. Furthermore, class 9 was oversampled, and its dataset was increased from 94 to 752 images. The initial dataset contains 10270 images, however performing data augmentation the model was trained on 18312 images.

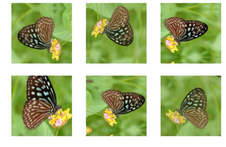

Image 1. **Data Augmentation.** Oversampling process for classes with low number of samples. Random contract, rotation, Flip (horizontal and vertical), translation and zoom were used in order to achieve these images (Keras Library)

.

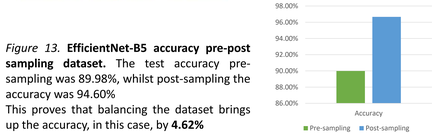

Figure 13. **EfficientNet-B5 accuracy pre-post sampling dataset.** The test accuracy pre-sampling was 89.98%, whilst post-sampling the accuracy was 94.60%
This proves that balancing the dataset brings up the accuracy, in this case, by **4.62%**

What is the Next Step?

Up to the present, EfficientNet-B7 has achieved the highest accuracy on ImageNet dataset. However, my GPU memory resources are not large enough to run EfficientNet-B7 with high image resolution, nor EfficientNet-B5 with resolution 299x299. Therefore, the next step would be trying some of the new EfficientNet models and increasing the resolution – as suggested in the paper [5].

Reference List

[1] **Data Recognition** (March 2017, **Rahul Bhalley**) Towards Data Science, Link: https://towardsdatascience.com/mnist-with-k-nearest-neighbors-8fbe7003fab7, Access date: 05.04.2021
[2] **Transfer Learning** (Sept. 2020, **Niklas Donges**) Built In, Link: https://builtin.com/data-science/transfer-learning, Access date: 13.04.2021
[3] **Having an imbalanced dataset** (Feb., 2019, **Will Badr),** Towards Data Science, Link: Having an Imbalanced Dataset? Here Is How You Can Fix It. | by Will Badr | Towards Data Science, Access date: 18.04.2021
[4] **EfficientNet: Rethinking Model Scaling for CNN** (May, 2019, **Mingxing Tan, Quoc V. Le**), Cornell University, Link: https://arxiv.org/abs/1905.11946, Access date: 15.04.2021
[5] **Image Classification using ML and Deep Learning** (May, 2020, **Islam Hasabo**) Medium, Link: https://medium.com/swlh/image-classification-using-machine-learning-and-deep-learning-2b180e46691f, Access date: 17.04.2021
[6] **Advantages of AI NN over SVM** (Unknown) Pico, Link: https://www.pico.net/kb/advantages-of-artificial-neural-networks-over-support-vector-machines , Access date: 05.05.2021
[7] **Why cannot we use KNN for large dataset** (Uknown) i2tutorials Link: https://www.i2tutorials.com/why-cannot-we-use-knn-for-large-datasets, Access date: 08.05.2021