Dog Breed Identification using Deep Learning (Image Classification)

Muhammed Zahid Bozkuş
Department of Computer Science
Turkish-German University
e170503104@stud.tau.edu.tr

1 Introduction

Today, deep learning is used in many different fields to perform different operations. One of the most frequently encountered areas is image classification. In general terms, image classification can be defined as a process that allows images to be separated into different classes according to their visual content. The main purpose of this project, in which image classification will be used, is to create a model to distinguish different dog breeds. For the development phase, the use of Convolutional Neural Networks is the obvious choice.[1]

In the initial phase of the project, we will build a model from scratch, so there will be no transfer learning. This model will try to learn from zero, without having any previous trained weights. In the second stage, transfer learning will be used. Here we apply different neural network models to our problem using the same database. The aim is to see how much of a difference transfer learning with more complex architectures and already trained weights actually makes in comparison to the model we build from scratch.

2 Dataset and Features

The dataset to be used for this project is the "Stanford Dogs Dataset". This dataset contains 20.580 images of dogs, which come from 120 different breeds. Since there are a lot of classes, it is difficult to show the distribution of breeds, but according to the creators of the dataset, there are 150 images per breed on average.[2] Here are some examples from the dataset:

Figure 1: A walker hound

Figure 2: A japanese spaniel

Figure 3: A redbone

As it can be expected, the total number of images may not be sufficient for proper learning. For this reason, methods like data augmentation will be used.[3] Since the dataset is relatively small, we need an appropriate split for the train, validation and test sets. For this specific work, a split of 60/20/20 (train/development/test) has been used. [4]

3 Preprocessing

The images in the dataset have different sizes. It is necessary to pre-process them to fit the network architecture that will be used. Hence, they are resized to 128x128 pixels. The pictures are normalized, so they are in similar ranges, which makes the learning process much faster. Like it was mentioned before, we also need to do data augmentation, so we have sufficient data to work with. For this purpose modified versions (rotated, shifted) of the pictures are also being used.

4 Related Work

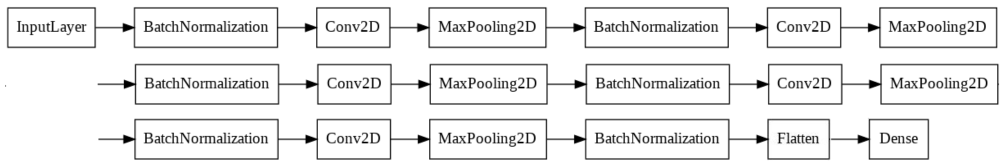
There are many similar projects that have been completed using the Stanford Dog Dataset. Mostly what makes a difference are the methods and models that have been used. Students from the Stanford University compared 7 different models, where they also had models of different complexity. [5] They started off with some baseline models and went on with transfer learning. David Hsu, again from the Stanford University, tested different CNNs based on LeNet and GoogleLeNet architectures. [6] Ayanzadeh and Vahidnia compared four different architectures in their work based on ImageNet, where they achieved very high test accuracies. [7] More details about the results of these works can be found in the section Evaluation and Results.

5 Methods and Approach

In the beginning phase of the project, we are going to build a CNN-Model from scratch to train. Like in many similar Convolutional Neural Networks, we are going to have multiple convolutional layers, pooling layers and dense layers.

In the beginning we have an 128x128 RGB image as the input for our model. The input will be first processed in the hidden layers. The base model we build is quite simple and follows a certain pattern. After the input layer, we have a Conv2D layer, which is followed by a pooling layer and batch normalization. This seqeunce of a convolutional layer, a pooling layer and batch normalization is repeated for five times. The only changing variable is the size of the filter for the convolutional layer, which increases as we go deeper in the model. For the kernel initializer we have the he normal initializer and ReLU is used as the activation function. Before the last layer, flattening is applied. The final layer is a dense layer with the softmax activation, which gives us the probability that an image belongs to a certain category for every single input.

After we build our model, we need to determine some hyperparameters to train our model with. For the optimization we use the Adam optimization algorithm. For the loss function we will use the categorical cross-entropy, as we have a multiclass classification problem. We get the following model in the end:



6 A CNN Model Built From Scratch

The first model has been built by using the methods mentioned in the Methods and Approach section. For the initial phase, a pre-trained model has not been used. In other words, the model started to learn from zero, as the accuracy was nearly zero. With more and more epochs, the loss started to get lower and the accuracy to get higher. The first results after running 10 epochs can be seen in the following images:

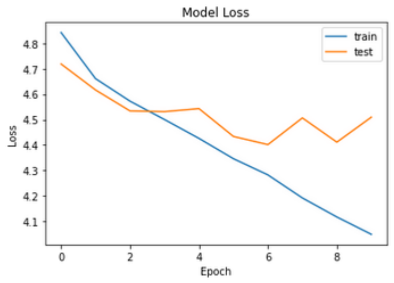


Figure 4: Training and validation set loss

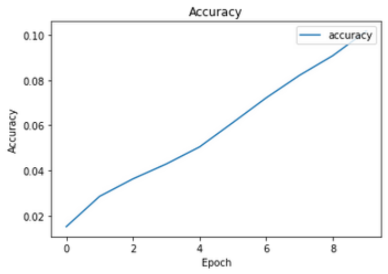


Figure 5: Accuracy of the model

As we can see the loss for both the training and validation set is quite high in the beginning. The loss for the training set goes down from 4.9 to 4.0 in these 10 epochs. At the same time we can observe that the accuracy increases. This can be interpreted as a good sign, but it is obvious that there is potential for improvement.The process could probably be much faster with different parameters or a better architecture. The loss for the test set is also an issue to be worked on. As the loss gets lower for the training set but not for the test set, we could interpret that the model is overfitting to the training set and should do some changes, like simplifying our model, adding some dropout layers or similar methods to prevent it.

After an analysis, the realization is that the initial model is far too complex. There are a lot of layers added, which made it difficult to understand where the problem exactly was. Therefore the model was simplified and included less layers as it progressed. The results for the current version of the model can be seen in the following images:

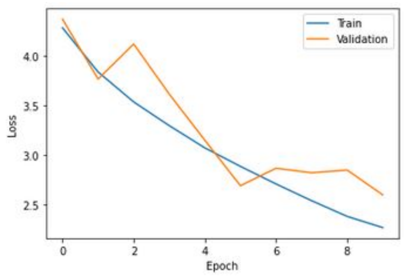


Figure 6: Training and validation set loss

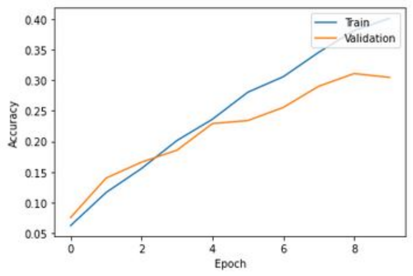


Figure 7: Accuracy of the model

The updated model run again for 10 epochs. This time the accuracy for both sets increased quickly, as we had about 40% for the train set and about 30% for the validation set. The loss value for the train set decreased as expected. The loss for the validation set showed variations, as it went up and down, but the trend was headed in the right direction. The weights were saved and used for the next epochs. After 10 more epochs it was seen that the accuracy for the validation set wasn't changing anymore. In the end there was an accuracy of about 35%. Modifying the different parameters (learning rate, batch size) didn't improve the results visibly. For the evaluation, the model was applied to the test set, where we had an accuracy of about 33%. As it isn't hard to guess, 33% accuracy is not acceptable for a classification problem, which is why we use transfer learning in the next step.

7 Transfer Learning

Since the results we have from our own model are suboptimal, we have to use other methods to improve them. The most obvious choice is to use transfer learning. For this part, we use and compare 3 different CNN architectures, VGG16 [8], MobileNetV2 [9] and InceptionV3 [10], all with pre-trained weights on ImageNet. There are various reasons why these three specific architectures are used. One of them is that these models are very widely used in image classification problems. Apart from the obvious fact that they have very different architectures and layer constructions, they have different depths and contain different numbers of parameters, which make them suitable for a comparison. Because we want to see essentially how the models differ from each other for our dataset, we keep every variable that has a more general purpose the same. The preprocessing of the data is always identical. Other than that we add some layers for the output of the model and replace the final dense layer with one that has 120 classes, since we have 120 different dog breeds. Just like the preprocessing step, the adding of the output layers is also identical in each model. It is a common application to set some of the last input layers to be trainable, because these are the ones which contain the smaller details. Hence they can improve the model drastically. In this work, all of the input layers are set to be non-trainable in opposition to the output layers. The main reason for that is that the models have different number of layers and therefore it wouldn't have the same effect if we just went on to make the last 50 or 100 layers trainable. The focus in this comparison is how they differ in their "rawest" form.

7.1 VGG16

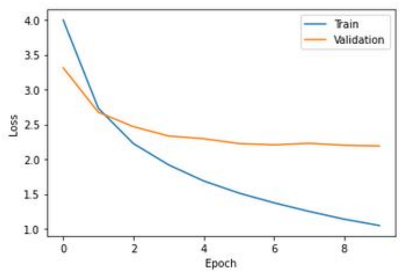


Figure 8: Loss for VGG16

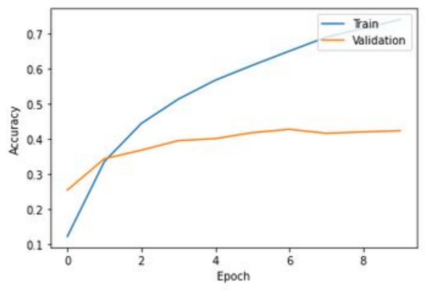


Figure 9: Accuracy for VGG16

7.2 MobileNetV2

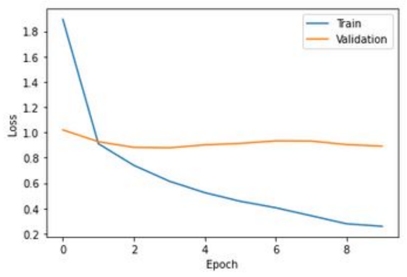


Figure 10: Loss for MobileNetV2

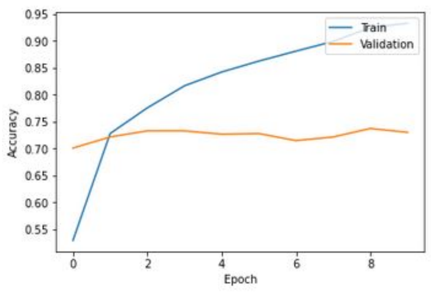


Figure 11: Accuracy for MobileNetV2

7.3 InceptionV3

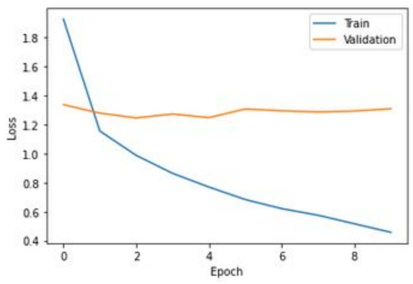


Figure 12: Loss for InceptionV3

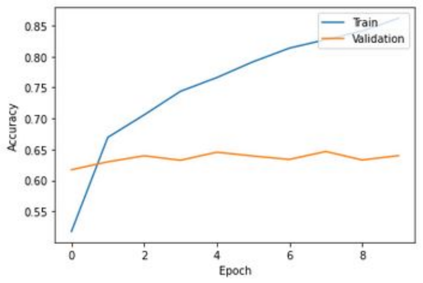
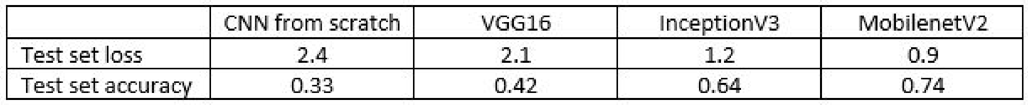


Figure 13: Accuracy for InceptionV3

8 Evaluation and Results

After we get our results for the training and validation sets, we need to evaluate our models to see how good they are actually working. For this purpose we compare them based on the loss value and the accuracy on the test set. The test set is basically new to the model, as it wasn't included in the training. As we feed our test set to the evaluate function, we get following results for the models:



	CNN from scratch	VGG16	InceptionV3	MobilenetV2
Test set loss	2.4	2.1	1.2	0.9
Test set accuracy	0.33	0.42	0.64	0.74

As we can see, the model we built from scratch has the worst performance, which is comprehensible because of its simplicity and depth. After that we have the VGG16 architecture with 42% test accuracy, which is also quite low. With 64% test accuracy, the InceptionV3 architecture places second. We have the best result with MobileNetV2, which has an accuracy of 74%. At first glance, 74% also doesn't seem like a very good result for an image classification problem, but we have to consider the fact, that this dataset has 120 classes.

Similar results can also be seen in other projects that were mentioned in the section Related Works. The students from Stanford University have a 22% accuracy for their base model, whereas the accuracy values for MobileNetV2 and ResNet50 are 79% and 83% respectively. Ayanzadeh and Vahidnia have some of the highest values for the test accuracy with this dataset. With four different architectures, ResNet-50, DenseNet-121, DenseNet-169 and GoogleNet, they achieved 89.66% , 85.37%, 84.01% and 82.08% test accuracy respectively. This shows us that the architectures they used may be more suitable for this dataset.

9 Conclusion and Future Work

Even though this is a rather simple project in comparison to more complex deep learning projects, there are a lot of takeaway points. First of all, the dataset was insufficient in terms of number of images, what was also expected, considering we have a classification problem with 120 classes. Therefore data augmentation and fine-tuning is always a good idea for datasets like this one. It was interesting to observe, that even with some of the pre-trained models from Keras we had bad results. That showed us that even very big architectures with a deep network can fail depending on other reasons. An idea for the future work here could be to analyze why there are big differences in the results of these different models. Other than that it turned out to be quite difficult to build a model from scratch. The interesting part was that a simpler model showed better results in comparison to the initial model that was built, which included a lot of layers. Another next step for this project could be to examine how the results change depending on the layer structure and depth of a model. The use of different evaluation methods like a confusion matrix or AUROC score might give better ideas about the models, which also is an idea for further work.

Code Repository

https://github.com/chevamikado/ImageClassificationCNN

References

[1] Bonner, A. (2019, February 2). The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks and Image Classification.
Towardsdatascience. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[2] Khosla, Aditya, Jayadevaprakash, Nityananda, Yao, Bangpeng and Fei-Fei, Li. Novel dataset for Fine-Grained Image Categorization. *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[3] Perez, Luis and Wang, Jason. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv preprint arXiv: 1712.04621*, 2017.

[4] Kumar, S. (2020, May 1). Data splitting technique to fit any Machine Learning Model.
Towardsdatascience. https://towardsdatascience.com/data-splitting-technique-to-fit-any-machine-learning-model-c0d7f3f1c790

[5] Chen, Daniel, Lahera, Estefania, James, Hailey and Wang, Winnie. CS109 Final Project: Dog Superbreed Classification.
GitHub. https://hljames.github.io/dog-breed-classification/

[6] Hsu, David. (2018). Using Convolutional Neural Networks to Classify Dog Breeds.
Stanford University. http://cs231n.stanford.edu/reports/2015/pdfs/fcdh_FinalReport.pdf

[7] Ayanzadeh, Aydin and Vahidnia, Sahand. (2018). Modified Deep Neural Networks for Dog Breeds Identification.

[8] Simonyan, Karen and Zisserman, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv: :1409.1556*, 2014.

[9] Sandler, Mark, Howard, Andrew, Zhu, Menglong, Zhmoginov, Andrey and Chen, Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv preprint arXiv: :1801.04381*, 2018.

[10] Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jonathon and Wojna Zbigniew. Rethinking the Inception Architecture for Computer Vision. *arXiv preprint arXiv: :1512.00567*, 2015.