Image Classification of Pet animals

Project: Multimedia Retrieval Course

Upender Shana Gonda
Matrikelnummer: **87691**
shanag01@ads.uni-passau.de

Summer Semester 2021

Abstract

From a few years onwards neural networks are started using for various problems to get better performance. Image classification is one of the applications of it where Convolutional Neural Network is the model specially used for image classification problems. In, this project I am going to work on the image dataset of pet animals where the main objective is to know how accurate is my implemented CNN model to predict the true breed of a species from the testing data.

1 Introduction

A Convolutional Neural Network, mainly used in the Image Classification where an images are represented in the binary format. Every image are arranged with a series of pixels with respective pixel values. It is also known as ConvNet or CNN. CNN is one of the class of neural networks[1].
Convolutional Neural Network Architecture: A CNN mainly consists of 3 layers namely, a convolutional layer, a pooling layer, and a fully connected layer[1].

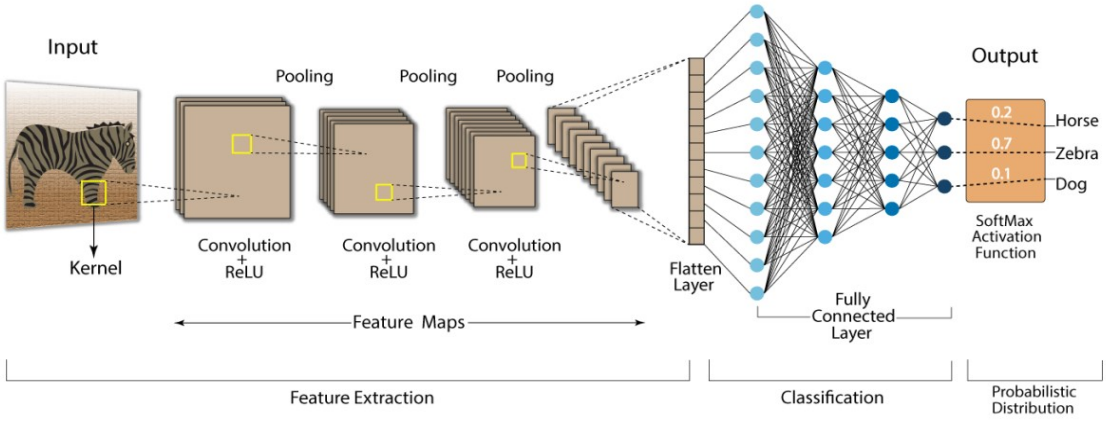

Figure 1: Architecture of Convolutional Neural Network[2]

Convolution Layer: The most important or core layer of CNN is convolution layer. The main portion of network's computational load was carried by this layer. The dot product of learnable parameters set matrix or kernel and receptive field matrix were performed by this layer. Where the kernel is in-depth irrespective of spatially smaller. The kernel slides through the width and height of the image to produce the respective region of an image and which implies a generating of a 2-D image known as activation map. The process of sliding the kernel in an image is called a stride[1].

Pooling Layer: In this layer, it allows the large images and compresses them to extract the important infor-mation from it. Mainly in this, maxpooling will be done which means it only stores the max-imum value from each and every window for better fitting of the features within the window[1, 3].

Fully Connected Layer: Finally, this layer will consider or takes the images with highly filtered and translates them into categories with labels[1, 3].

2 Data Acquisition and Preprocessing

2.1 Data Acquisition

For this project, the image dataset has been taken or downloaded from the Visual Geometry Group department, University of oxford[1]. This image dataset consists of 7390 images of pets both dogs and cats with respective species and breed names. Total this pet dataset with 37 categories with about

[1] https://www.robots.ox.ac.uk/~vgg/data/pets/

approximately 200 images per category or class. These images are captures with different size, pose, aspect ratio, etc. All the dataset images are placed in a single folder and every image is named with its respective class information.

2.2 Framework Used

The following open-source frameworks and libraries were used as part of the project implementation. The data analysis and manipulation were achieved by using the Pandas[2] and Numpy[3]. MatplotLib[4] functions were used for visualization of data. Sklearn/ scikit-learn[5] were also used for the data split and confusion matrix. The Convolution Neural Network model were implemented by using Tensorflow[6].

2.3 Data Preprocessing

Firstly, to create the target variable, from the image file names respective class names of all images were extracted. Thereafter, the 7390 images of the dataset were split into 3 datasets namely train, validation, and test datasets.

From the total dataset, 10% of the images were used as a test dataset and another 10% of the dataset images were considered for validation and the remaining 80% of the dataset was used as a training dataset.

As I mentioned in the data acquisition, all the images in the dataset are with different aspect ratios and sizes. So, the padding is applied for the images to load with a fixed 256*256 resolution. Where the padding is helpful for the prevention of distortion mainly due to shrinking or stretching of images while changing the aspect ratios of images.

Next for the rotation, zoom or shift, etc. of the images during training, the image augmentation was done by using Keras ImageDataGenerator.

3 CNN Model Creation and Training

The model was created with a multiple of convolutional blocks including the max pool layer for each one of the blocks. Initially, the first layer will be the input layer for any CNNs which takes images and then passes to the next layers for feature extractions. After that, model also consists of a fully connected layer with 256 units.

The activation function used for this is the 'ReLU' function, as we know 'ReLU' stands for Rectified Linear Unit[4]. Where In CNN's this activation function is the most preferable one. The characteristic of the ReLU function is its behaves linearly for all positive values where as, for negative values, the function is zero. The main advantage of ReLU is to reduce the requirement of time for the model to train and it cannot be represented in any mathematical way[4]. Here, in this model at convolution blocks I've taken the Leaky ReLU[5] as activation function where there is an advantage of speeding up the training process faster and which fixes the problem of "dying ReLU" by having a small positive slope for negative values instead of zero slope[5]. Then the flatten layer[6] is responsible for converting the data into 1-D array and it flatten the convolutional layers output to create single feature vector and given as input for the next layer namely fully-connected layer or dense layer[6].

Then the sequential model was compiled with loss function of sparse categorical cross-entropy[7, 8] where each and every sample exactly belongs to only one class and metrics with sparse categorical accuracy[9] along with optimizer ADAM[10] with the respective learning rate of 0.0001. As a part of model implementation, a computational neural network was improvised by batch normalization and dropout. Where in detail:

[2] https://pandas.pydata.org/
[3] https://numpy.org/
[4] https://matplotlib.org/
[5] https://scikit-learn.org/stable/
[6] https://www.tensorflow.org/

Batch Normalization: To train a very deep neural network, it is a technique to stabilize or standardize the learning process of inputs to layers and it is also capable of reducing the number of the requirement of epochs for training the deep networks[11]. It is also helpful for lower the loss in quick time.
Dropout: It is a technique introduced in the network to minimize overfitting. Dropout forces the weights to take only the small values within the network, which implies regular distribution of the weight values in a network for overfitting reduction[12].

Here, in this model creation, I've applied the 0.2 dropouts to some layers and 0.1 as well for the other few to regularize the model. How dropout function work means, during training process from the applied layer it will drop out the some of the output units. For instance, as I considered in this implementation 0.1 and 0.2 as input fractional values for layers which means dropping out or killing the 10 percent or 20 percent of output units from respective layers randomly in every training epoch. The activation function softmax[13] were used for the probabilistic distribution at dense layer. Where dense layer generally tries to change the size or dimensions of the vectors.

I have also considered the Earlystop in the training phase of the model to avoid if any overfitting exists with a patience value = 10 and monitored with the validation loss.

Then, a model is fitted with the train and validation data and at the end of the training, We can observed that the model was ended with 113 epochs of training where 10 epochs are of early stop patience. So, I've considered the 103rd epoch value as my final result of train and validation data with respective accuracy and losses.

Here, Table **1** represents the relation or difference between the accuracy and loss of both training and validation data with a respective number of epochs, where we got these results at the time of training of the model. From the table**1** we can notice that as the number of epochs is increasing the respective accuracies of training and the validation also increasing where there is not much difference between both accuracies of validation and training with respective each epoch.

Table 1: **Accuracy and Loss of Training and Validation data for respective Epoch**

Sl.No.	**Epoch Number**	**Training Accuracy**	**Validation Accuracy**	**Training Loss**	**Validation Loss**
1	10	0.2074	0.2222	2.8654	2.8137
2	20	0.3549	0.3348	2.2529	2.2670
3	30	0.4571	0.4159	1.8260	1.9426
4	40	0.5422	0.4354	1.5140	1.8816
5	50	0.6069	0.5300	1.2956	1.5367
6	60	0.6536	0.5240	1.1345	1.5293
7	70	0.6939	0.5826	0.9968	1.3699
8	80	0.7186	0.6036	0.8994	1.2682
9	90	0.7442	0.6021	0.8122	1.3084
10	100	0.7659	0.6291	0.7460	1.2770
11	103	**0.7619**	**0.6637**	**0.7314**	**1.1199**

Similarly, if we see at the losses of training and validation we can observe the same scenario where there is not much difference in the losses as the number of epochs increasing. The same observations can be seen in the visualization figure **2** and **3** .

From the figure **2** we can see that the number of epochs are considered on X-axis and respective accuracy was taken on the Y-axis.

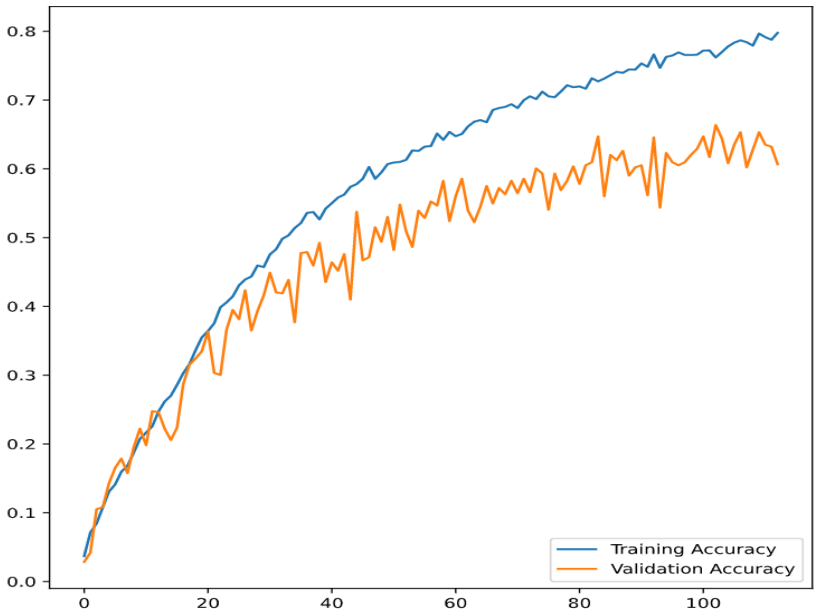

Figure 2: Visualization of Training and Validation Accuracy

From the figure **2**, we can observe that as the number of epochs increasing the respective accuracies for both training and validation data also increasing. We can also observe that after around 103 epoch the accuracies and loss of both data training and validation started fluctuating which we noticed during model training(Epochs).

Similarly, if we look at the figure **3** the training loss and validation loss, both started declining from the starting point but with not much difference between them.

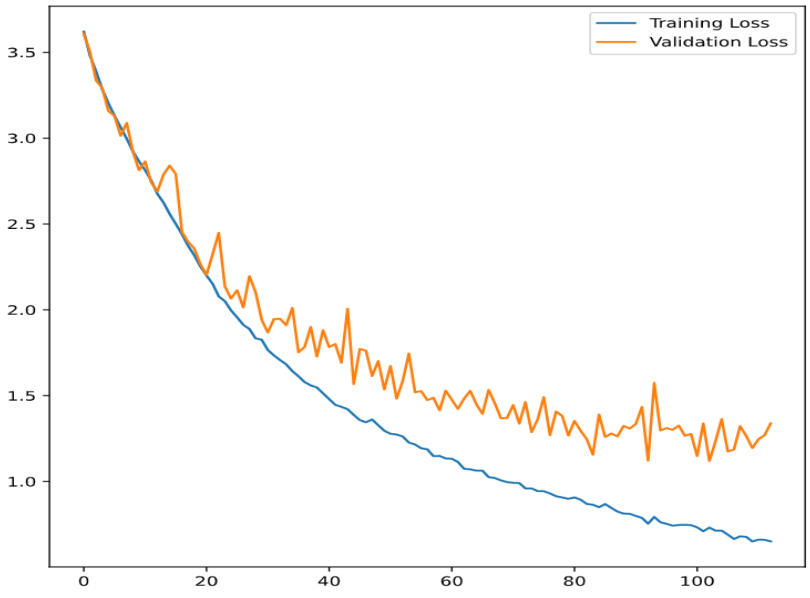

Figure 3: Visualization of Training and Validation Loss

4 Evaluation of model on Testing data

This is the important part, where the implemented or trained model will get tested with the unknown testing data. After testing the model on testing data the results are:

- Loss on Testing data = 1.0731
- Accuracy on Testing data = **0.6657** or **66.57**%.

After comparing these results with both training and validation data. We can conclude that the accuracy of Testing and the validation is almost the same just above 66% of accuracy. Whereas the accuracy of training data is 76%. The following are might be the reasons for less model accuracy:

Reason: One of the most important reasons for getting less accuracy is that in the whole dataset some of the images of cats and dogs are unclear.

Reasons for unclear images are:

- Some of the images are taken from very long distances. So, it's very hard for a model to train itself and then to identify those images.
- some of the images are with more lightening or focused. This kind of image also creates a problem for the model to get trained and then to predict.
- Some of the images are with almost similar features in looks like hairstyle, face structure with a similar pose, etc. example: breed Havanese and wheaten terrier.
- Also, Some of the dog or cat images are with hiding faces as well.

Figure 4: Unclear image which got predicted Wrongly

Here is one of the example image in figure **4** of those, where our model predicted wrongly breed of a species from the testing data. Where there is a high possibility of one of the above-mentioned reason

for predicting the wrong one. In the image figure **4**, we can observe that the image of the species is almost unclear and it's difficult for a model to predict which breed it belongs to.

One of the other most important reasons for wrong prediction of the breed of a species might be that:

- Due due to the huge amount of images in the dataset with different classes while splitting, the training dataset might not get all breeds(including different colors) of species to train the model(because of random splitting) which creates difficulty for a model to predict later on with testing data.

Figure 5: wrongly predicted image due to lack of model train on that particular class or due to similar features

From the figure **5** one more time we can observe that, the model is unable to predict correctly because the reason might be the above one. Strongly for this wrong prediction, as human observation, we can say that either model not got trained with breed Havanese class or else the Havanese and the wheaten terrier looks almost similar in features.

4.1 Truely predicted images

Now, the figure **6** is an effort to show the some of the correctly predicted images by the CNN model. Where by observing images, we can notice that both the predicted images looks very clear with different pose.

Figure 6: Example Images of correctly predicted by our model

4.2 Final Metric Results

Table **2** shows the final results of training, validation and testing data accuracy and losses. Where the final accuracy of the implemented model is 66.57%

Table 2: **Metrics**

	Training Data	**Validation Data**	**Testing Data**
Dataset Split	5,985	666	739
Loss	0.7324	1.1199	1.0731
Accuracy	76.19%	66.37%	**66.57**%

4.3 Confusion Matrix

From the figure **7** we can see the relation between true label vs predicted label of images. Where all the diagonal blocks are truly predicted images by our model where we can also notice that all diagonal predicted images are high in number compare to neighboring blocks. This concludes, most of the time our model predicted the correct breed of a species.

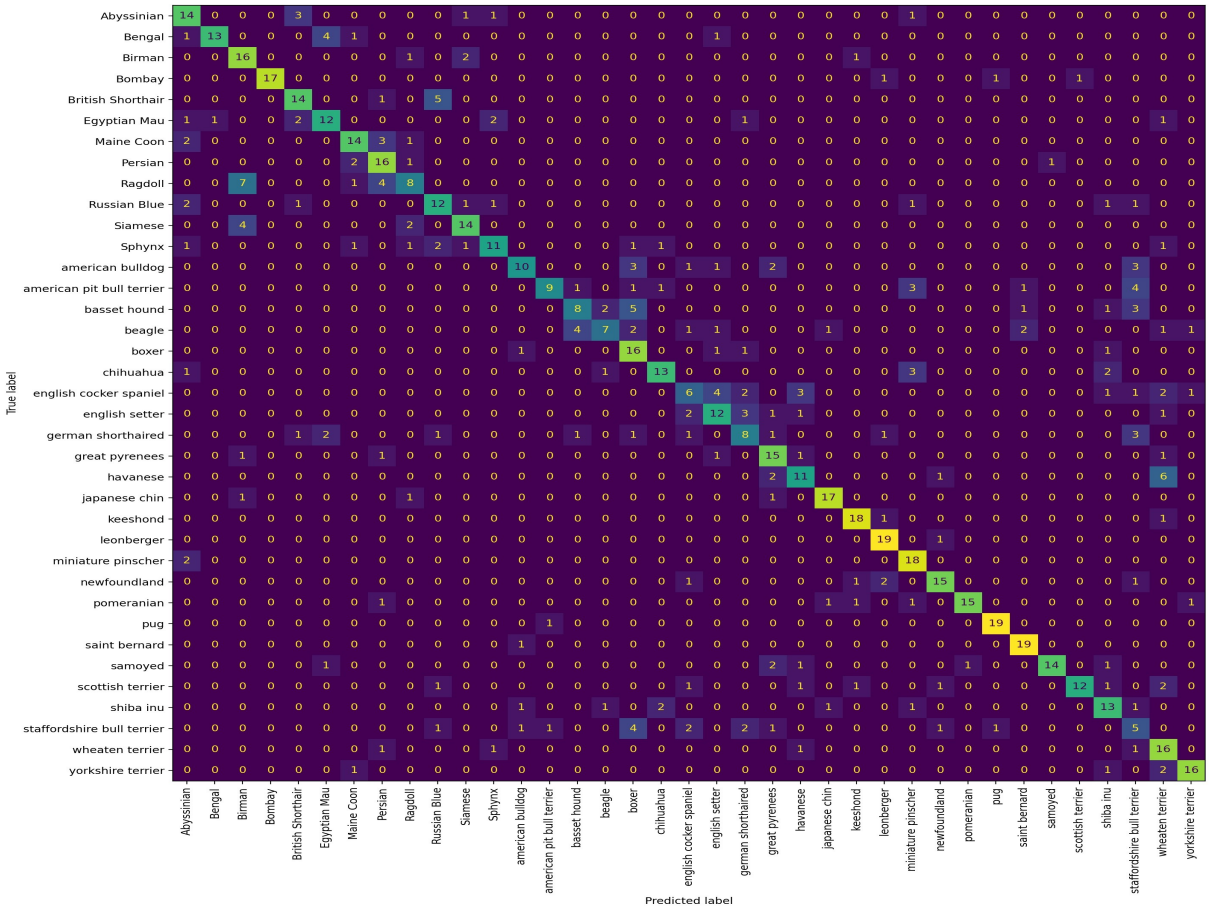

Figure 7: Confusion Matrix: True vs Predicted

5 Conclusion

The implemented CNN model on an image dataset can predict the breed of a species with an accuracy of above 66%. Despite the huge dataset and the unclear images present in the dataset, the implemented model has arisen with good and moderate accuracy.

References

[1] Convolutional neural networks, explained — by mayank mishra — towards data science. `https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939`. (Accessed on 07/28/2021).

[2] Convolutional neural networks — understand the basics of cnn. `https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-understand-the-basics/`. (Accessed on 07/28/2021).

[3] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia computer science*, 132:377–384, 2018.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[5] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[6] flatten layer in cnn - google search. https://www.google.com/search?q=flatten+layer+in+cnn&rlz=1C1CHBF_deDE944DE944&sxsrf=ALeKk023kxpOhfLMmYunXNtQPmBw7yNaBg%3A1627763992282&ei=GLUFYanQEIPolwTospzoDg&oq=flatten+layer+in+cnn&gs_lcp=Cgdnd3Mtd2l6EAxKBAhBGABQAFgAYK1WaABwAHgAgAFfiAFfkgEBMZgBAMABAQ&sclient=gws-wiz&ved=0ahUKEwipvKa0lY7yAhUD9IUKHWgZB-0Q4dUDCA4. (Accessed on 07/31/2021).

[7] Sparse categorical crossentropy loss with tf 2 and keras – machinecurve. https://www.machinecurve.com/index.php/2019/10/06/how-to-use-sparse-categorical-crossentropy-in-keras/. (Accessed on 07/31/2021).

[8] neural network - sparse_categorical_crossentropy vs categorical_crossentropy (keras, accuracy) - data science stack exchange. https://datascience.stackexchange.com/questions/41921/sparse-categorical-crossentropy-vs-categorical-crossentropy-keras-accuracy. (Accessed on 07/31/2021).

[9] Keras - difference between categorical_accuracy and sparse_categorical_accuracy - intellipaat community. https://intellipaat.com/community/2619/keras-difference-between-categoricalaccuracy-and-sparsecategoricalaccuracy. (Accessed on 07/31/2021).

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[13] Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.