Animal Image Classification Using CNN

In this project, image classification is done using CNNs on an animal image dataset from Kaggle (https://www.kaggle.com/min4tozaki/animal-classification). The dataset contains 5000 coloured images for 10 types of animals, one for each type. The animal type and its classification are:

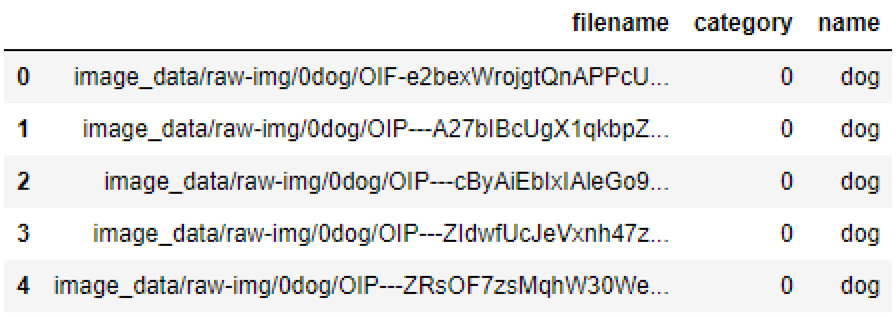
- Dog: 0
- Horse: 1
- Elephant: 2
- Butterfly: 3
- Chicken: 4
- Cat: 5
- Cow: 6
- Sheep: 7
- Spider: 8
- Squirrel: 9

Exploratory Data Analysis

The steps taken in EDA were as follows and are also described in detail after the list,

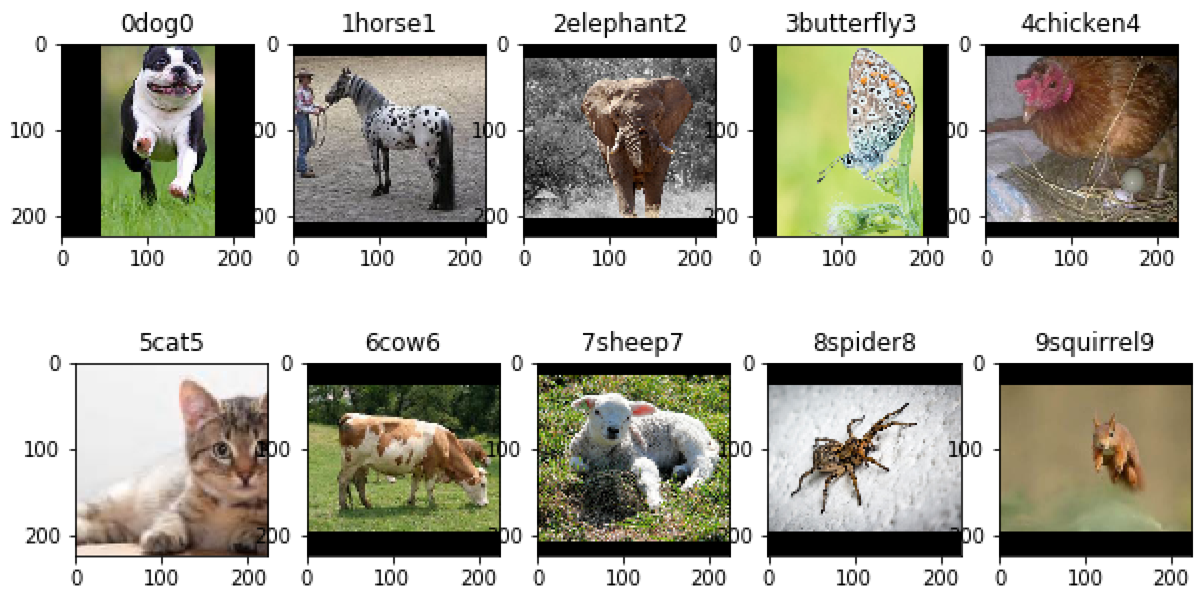
1. Map Class and Animal name to Image
2. Resize each image to the same dimensions
3. Scale image pixel data between 0 and 1
4. Shuffle and create train-test sets
5. Augment training set images to avoid overfitting

The dataset only contained the images, categorized into folders by animal name. A Dataframe was created with the columns as filename (with path), category and name (of animal).



	filename	category	name
0	image_data/raw-img/0dog/OIF-e2bexWrojgtQnAPPcU...	0	dog
1	image_data/raw-img/0dog/OIP---A27bIBcUgX1qkbpZ...	0	dog
2	image_data/raw-img/0dog/OIP---cByAiEbIxIAleGo9...	0	dog
3	image_data/raw-img/0dog/OIP---ZIdwfUcJeVxnh47z...	0	dog
4	image_data/raw-img/0dog/OIP---ZRsOF7zsMqhW30We...	0	dog

Since each image was of a different size, they were all resized to 225x225 by adding extra black space (0 vectors) to the appropriate dimension equally on both sides. The result was as follows,



The pixel data was then scaled down from 0-225 to between 0-1. Categorical values were created for the target variable. The data was shuffled and finally split for training and testing (4000:1000). The training set was then augmented. Images were rotated, horizontally flipped and shifted on the width and height axis.

CNN Models

3 CNN models were built and trained. Each model was trained using '*categorical_crossentropy*' loss and '*RMSprop*' optimizer for 10 epochs. The batch size was 50. A different number of filters with varying sizes were used for each layer in each model.

1. Model 1: '
   a. Convolution layer 1
   b. Pooling layer 1
   c. Convolution layer 2
   d. Pooling layer 2
   e. Dense layer 1
   f. Dense layer 2
   g. Dense layer 3
2. Model 2:
   a. Convolution layer 1
   b. Pooling layer 1
   c. Convolution layer 2
   d. Pooling layer 2
   e. Convolution layer 3
   f. Pooling layer 3
   g. Dense layer 1
   h. Dense layer 2
   i. Dense layer 3

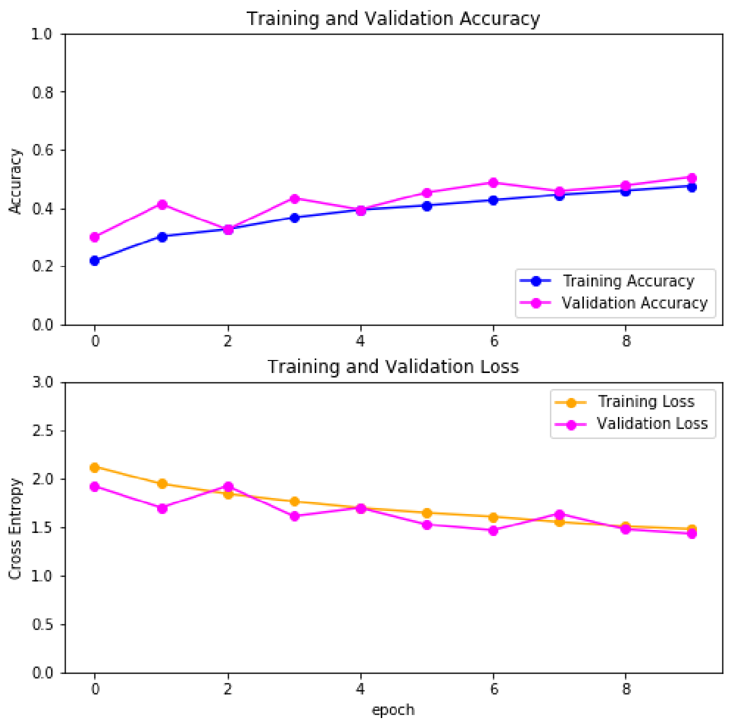
3. Model 3:
    a. Convolution layer 1
    b. Convolution layer 2
    c. Pooling layer 1
    d. Convolution layer 3
    e. Convolution layer 4
    f. Pooling layer 2
    g. Dense layer 1
    h. Dense layer 2
    i. Dense layer 3

Comparison of Results

The validation accuracy of the respective models was

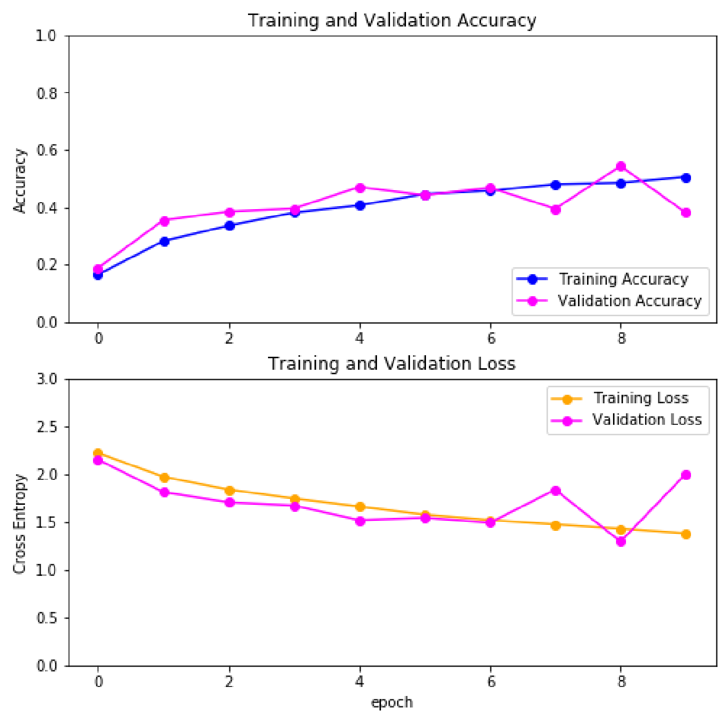
1. 50.60%
2. 38.00%
3. 45.20%

Model 1



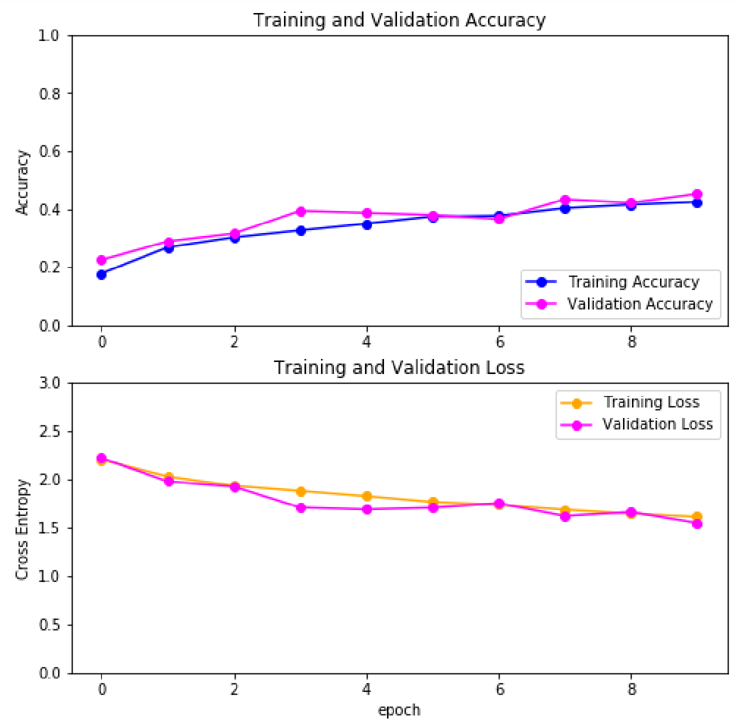
The model reached upto an accuracy of 50% by the end of 10 epochs. Looking at the two graphs, we can see the accuracy slowly increasing and the loss slowly decreasing. The model might require more epochs to reach an acceptable accuracy level with perhaps a larger learning rate.

Model 2



This model faced the same issues as the previous model. In the final epoch, the validation accuracy suddenly dipped down to 38% from 54% and subsequently the validation loss also increased. Similar measures in hyperparameter tuning could be taken to improve this model too.

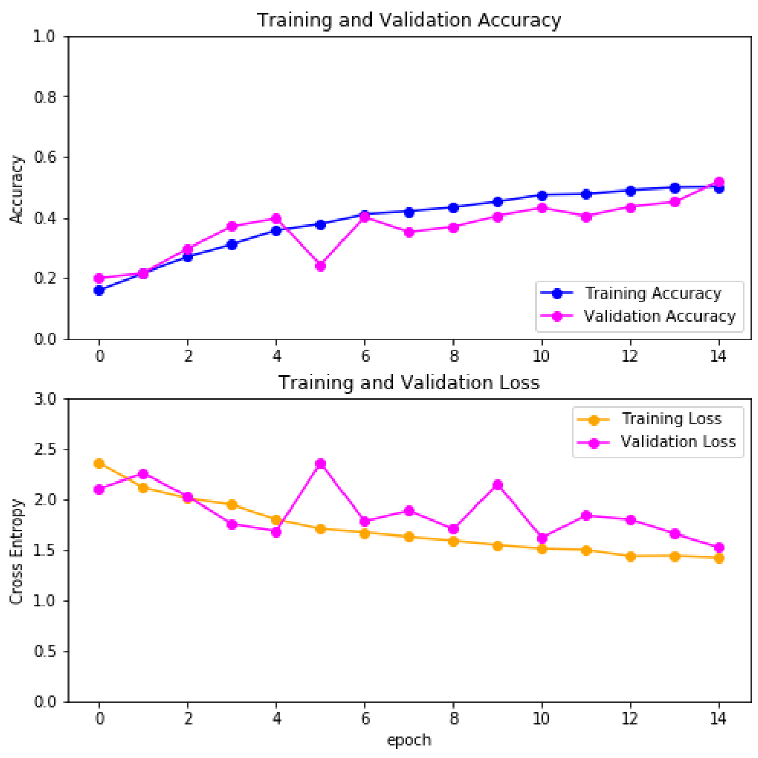
Model 3



This model faced the same issue, indicating the maybe the model complexity or depth may not necessarily be the cause of low accuracy.

To test this hypothesis, model 1 is re-run for 15 epochs and with a higher learning rate (x10). The results are given in the following section.

Model 1.1



Increasing the learning rate and the number of epochs did not improve the results significantly. The validation accuracy increased to just 51.9% (earlier 50%). The rate of increase in accuracy is still low. In conclusion, all the models were underfitting.

Conclusion

In conclusion, none of the models gave satisfactory results for the chosen set of hyperparameters and CNN architecture. Some possible reasons for the underfitting could be,

- Less number of epochs
- Model not complex/deep enough
- Batch size too large
- Learning rate too small

Very deep or complex models could not be analysed due to computational constraints. VGG16 model was tried but could not be trained due to computational constraints.

The results from CNN could also be compared with other classifiers such as SVM. Dimensionality reduction techniques could also be used before classifying.