Dog Breed Classifier Project Documentation

Introduction

This project consists in building a dog breed classifier that identifies 4 different breeds: French Bulldog, Golden Retriever, Poodle and German Shepherd. It also has to identify when the photo is from a dog that is not any of those breeds and label it as “others”.

Due to a very limited dataset provided some techniques and adjustments were necessary to reach an acceptable accuracy. In this documentation I will explain as much as I can how the model works and how was the process to get this solution.

Architecture and Process

At first I started with a CNN with convolutional layers, max pooling layers and fully connected layers. As It’s known, this architecture is the most common and simple while talking about image models but I had several problems dealing with the accuracy, to be more specific the model does not label any image as “others”.

When I was dealing with this problem I tried several things, some as changing learning rates but what helped me to have a better accuracy was adding some layers. To be more specific I added an extra Conv2D layer with ‘relu’ as the activation, I also added a Dropout layer to prevent overfitting. The other thing that I changed was the optimizer from RMSprop to an Adam optimizer.

All those changes drove me into a more effective and more accurate model, but there was a problem with the predictions, the “other” category was working when I changed the threshold to predict the images but there It started to give me problems with the other 4 categories.

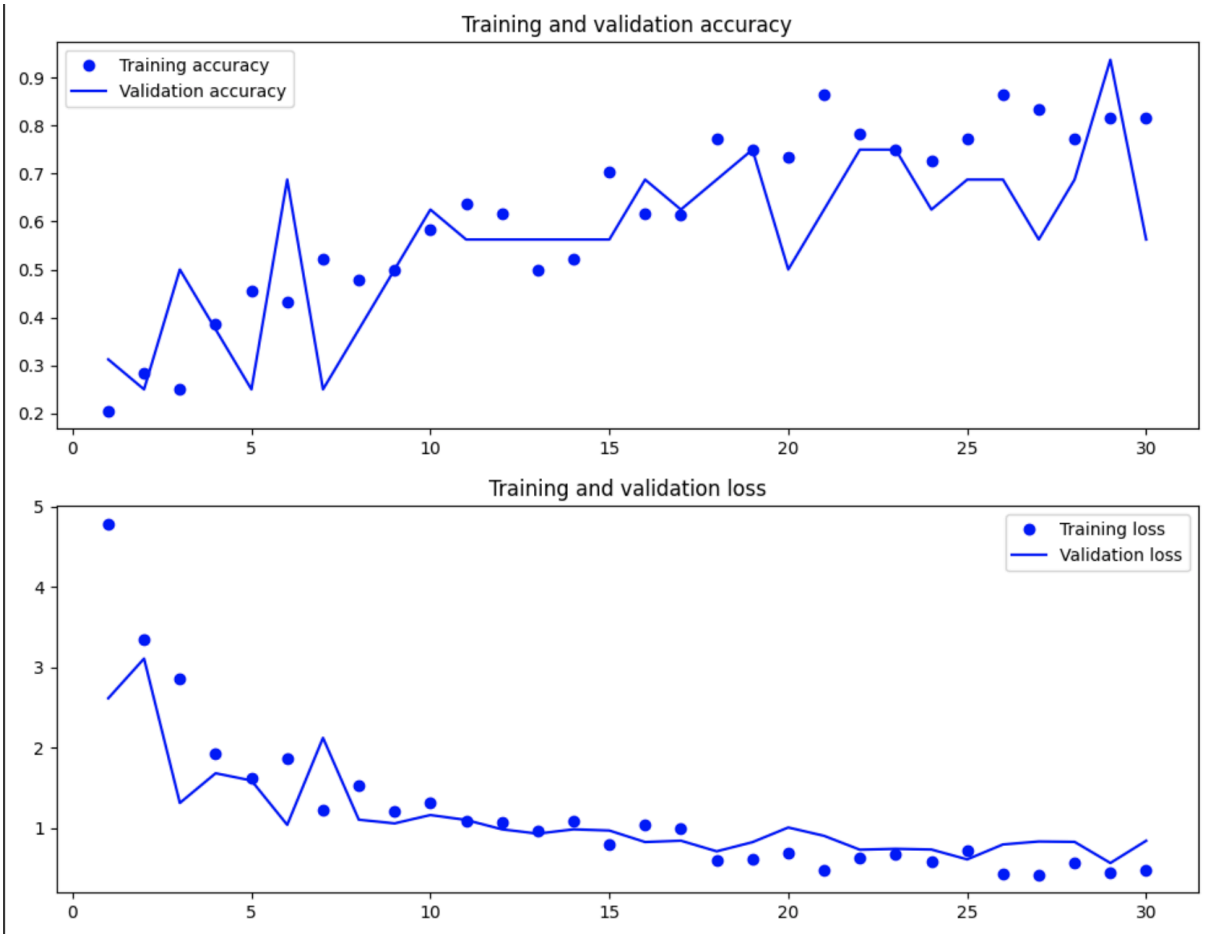
Reaching that point, through some research I came to the idea to use a pre-trained model to make it even more accurate. Doing that and changing the learning rate I finally got to the point that the accuracy was good enough to make good predictions.

Architecture:

```python
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

conv_base.trainable = False

model = models.Sequential([
    conv_base,
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4, activation='softmax')
])

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    metrics=['accuracy']
)

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size if train_generator.samples // train_generator.batch_size > 0 else 1,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size if validation_generator.samples // validation_generator.batch_size > 0 else 1
)
```

Python

Accuracy:



Explanation of Layers

- **VGG16 Base**: Pre-trained on ImageNet, it provides a robust feature extraction mechanism.
- **Flatten Layer**: Converts the 3D output of the convolutional base to 1D.
- **Dense Layer (512 units)**: Adds a fully connected layer with ReLU activation.
- **Dropout Layer (0.5)**: Prevents overfitting by randomly setting 50% of the input units to 0.
- **Output Layer (4 units)**: Produces a probability distribution over the four classes.

How It works

Firstly I use data argumentation to generate more training samples, this decision was made to address the limited dataset size. Then comes the architecture of the neural network that I already explained in this documentation, so later I trained the model with the augmented data and monitored its performance on a validation set.
The “other” category was created in the post processed step of the model, this is caused by the lack of possibility to change the dataset and add that category to It.

Possible next steps

Due to some research, I reached the conclusion that to make It even more accurate without changing the dataset, I would have to fine-tune the pre-trained model, so I can make it even more accurate to my specific use case. I can also add a batch normalization layer to stabilize training and the most difficult one is to combine predictions from many different models so I can improve the accuracy.

Conclusion

By using data augmentation techniques, pre-training models, and making necessary modifications based on the choice of architecture we chose to work with a dog breed classifier that could deal with a very few data samples. Consequently, We were able to improve our outcome by adjusting the learning rate, leveraging a pre-trained model as well as using post-processing methods to detect Out of Distribution (OOD) images more effectively.