Deep Learning Project - Sisay Menji

Classifying animal images using CNN and transfer learning

12-Sep-2021

In [1]:
```
# Import necessary libraries
import numpy as np, pandas as pd, matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator

%matplotlib inline
```

1. Data Description

The data for this project is taken from [Kaggle data repository (https://www.kaggle.com/alessiocorrado99/animals10)](https://www.kaggle.com/alessiocorrado99/animals10). The dataset contains 28K animal images belonging to 10 categories (dog,cat,horse, butterfly, chicken, sheep, cow, squirrel and elephant). The data is downloaded from Kaggle and has separate folders containing the images of the different animals.

2. Objectives of the Analysis

In this study I will try to use different deep learning algorithsm based on CNN to predict the images of the animals. In order to improve the accuracy of the predictions I will conduct plain neural netrowks model and four variants of the CNN approach. The models to be built and tested are:

1. Plain neural network model without convolutional layers
2. Basic CNN with only convolutional layers
3. Basic CNN plus max pooling between each layers
4. Basic CNN plus dropout
5. Finally, I will use transfer learning by implementing VGG16 model

3. Data Preparation

In this section Keras is used to prepare the data in the images folder. After the data is prepared it will be split into "full training" and test sets. The "full training" set is further split into training and validation sets. I will use 50% of the 28K data for test and the 50% for training and validation. I used 50% for vaidation because the data was large on my computer and difficult to fit 70% of the data. The following code implements the data preparation steps

Tensorflows Kera's ImageDataGenerator is used to generate the image data from the directory containing the images. After the split into training (which includes the validation set also) and test, the training data contains ~13K images while the test data contains ~13K images.

Target size of the image is reduced from 256x256 to 32x32. This is inorder to reduce the number of parameters required to estimate the CNN as this well better learning. With 256x256, the CNN has 1,040,526,602 parameters to compute (which is nearly impossible to train on my personal laptop) but with 32x32 it has ~1.5M trainable parameters.

In [2]:
```
# settings
results = {}
bsize = 64 #  batch size
ep=5 # number of epochs limited to 5 because of the large dataset
ishape = (32,32,3) # image size adjusted from 256x256x3
np.random.seed(100)
tf.random.set_seed(100)
tsize = (32,32)
img_dir = "archive/raw-img/"
```

In [3]:
```
import os
total_pics = 0
for c in os.listdir(img_dir):
    path = img_dir + c + "/"
    total_pics += len(os.listdir(path))
```

In [4]:
```
img_gen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
```

In [5]:
```
train_df = img_gen.flow_from_directory(img_dir, subset="training",
                                      shuffle=True,batch_size=bsize, class_mode="categorical", target_size=tsi
ze)
test_df = img_gen.flow_from_directory(img_dir, subset="validation",
                                     shuffle=True,batch_size=bsize, target_size=tsize, class_mode="categorica
l")
```

```
Found 18331 images belonging to 10 classes.
Found 7848 images belonging to 10 classes.
```

In [6]:
```
train_pics, test_pics = train_df.n, test_df.n
train_pics, test_pics
```

Out[6]:
```
(18331, 7848)
```

In [7]:
```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
```

4. Plain Neural Networks Model

In [47]:
```
nn_model = keras.models.Sequential([
    keras.layers.Input(shape=ishape),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
nn_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
nn_model.fit_generator(train_df, steps_per_epoch=train_pics//bsize, epochs=ep, verbose=1)
```

```
Epoch 1/5
286/286 [==============================] - 31s 107ms/step - loss: 2.1616 - accuracy: 0.2161
Epoch 2/5
286/286 [==============================] - 31s 107ms/step - loss: 2.0389 - accuracy: 0.2623
Epoch 3/5
286/286 [==============================] - 29s 102ms/step - loss: 1.9951 - accuracy: 0.2831
Epoch 4/5
286/286 [==============================] - 30s 104ms/step - loss: 1.9651 - accuracy: 0.2971
Epoch 5/5
286/286 [==============================] - 30s 104ms/step - loss: 1.9315 - accuracy: 0.3185
```

Out[47]:
```
<tensorflow.python.keras.callbacks.History at 0x20ab7b0a4c8>
```

In [48]:
```
nloss, nacc = nn_model.evaluate(test_df,verbose=0)
nloss, nacc
```

Out[48]:
```
(1.9622446298599243, 0.3002038598060608)
```

Our neural network model above with 3 dense layers (2 hidden and 1 output) provided a poor accuracy both on the training and test sets that are close to 30%. Next we will run different CNN models to improve accuracy.

5. Basic CNN with only convultuional layers

In this section I will implement a basic CNN with only convolutional layers and no pooling. I will use two CCN layers and 2 dense layers for this purpose. I didn't mange to use three CNN layers as my laptop was busy and not responding.

In [8]:
```
base_model = keras.models.Sequential([
    keras.layers.Conv2D(64,3 ,input_shape=ishape, activation="relu"),
    keras.layers.Conv2D(64,3,activation="relu"),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
base_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
base_model.fit_generator(train_df, steps_per_epoch=train_pics//bsize, epochs=ep, verbose=1)
```

```
WARNING:tensorflow:From <ipython-input-8-31521cd4b0fa>:9: Model.fit_generator (from tensorflow.python.keras.e
ngine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/5
286/286 [==============================] - 45s 157ms/step - loss: 2.0076 - accuracy: 0.2924
Epoch 2/5
286/286 [==============================] - 43s 151ms/step - loss: 1.5833 - accuracy: 0.4568
Epoch 3/5
286/286 [==============================] - 43s 151ms/step - loss: 1.3201 - accuracy: 0.5548
Epoch 4/5
286/286 [==============================] - 45s 156ms/step - loss: 1.0578 - accuracy: 0.6475
Epoch 5/5
286/286 [==============================] - 44s 155ms/step - loss: 0.7659 - accuracy: 0.7482
```

Out[8]:
```
<tensorflow.python.keras.callbacks.History at 0x20a95b17cc8>
```

In [9]:
```
base_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 64)        1792      
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        36928     
_________________________________________________________________
flatten (Flatten)            (None, 50176)             0         
_________________________________________________________________
dense (Dense)                (None, 32)                1605664   
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330       
=================================================================
Total params: 1,644,714
Trainable params: 1,644,714
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
bloss, bacc = base_model.evaluate(test_df,verbose=0)
bloss, bacc
```

Out[10]:
```
(1.5404306650161743, 0.5263761281967163)
```

As can be seen from the above results, the CNN model has an accuracy of ~75% on the training set but only ~53% on the test set which is clearly an indication of overfitting.In the next sections I will use various methods including max pooling, drop out and data augmentation to try to reduce the overfitting.In addition I will use trained models (transfer learning) to improve the accuracy.

6. CNN model with MaxPooling

In the following section, a pooling CVV is run with two pooling layers added next to the convolutional layers.Adding the pooling layer did not increase either the test or training scores as shown below. But at least with pooling we don't have overfitting problem and train and test results are close. This implies that pooling helps with overfitting problem but not improve the accuracy.

In [11]:
```
pooling_model = keras.models.Sequential([
    keras.layers.Conv2D(64,3 ,input_shape=ishape, activation="relu"),
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(64,3,activation="relu"),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
pooling_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
pooling_model.fit_generator(train_df, steps_per_epoch=train_pics//bsize, epochs=ep, verbose=1)
```

```
Epoch 1/5
286/286 [==============================] - 34s 117ms/step - loss: 2.1360 - accuracy: 0.2333A: 15s - loss: 2.1
918 - accuracy: 0.201 - ETA: 14s - loss: 2.1915 - accuracy: 0. - ETA: 14s - los
Epoch 2/5
286/286 [==============================] - 34s 119ms/step - loss: 1.9214 - accuracy: 0.3252
Epoch 3/5
286/286 [==============================] - 34s 118ms/step - loss: 1.7393 - accuracy: 0.3839
Epoch 4/5
286/286 [==============================] - 34s 119ms/step - loss: 1.6178 - accuracy: 0.4282
Epoch 5/5
286/286 [==============================] - 36s 124ms/step - loss: 1.5594 - accuracy: 0.4511 27s - loss: 1.549
1 - accuracy: 0. - ETA: 27s - loss: 1.5625 - ac - ETA: 6s - loss: 1.5 - ETA: 5s - loss: 1.5566 - ac - ETA: 4s
-
```

Out[11]: <tensorflow.python.keras.callbacks.History at 0x20a96771d88>

In [12]:
```
ploss, pacc = pooling_model.evaluate(test_df,verbose=0)
ploss, pacc
```

Out[12]: (1.5432500839233398, 0.45654943585395813)

6. CNN with drop out

In [22]:
```
dout_model = keras.models.Sequential([
    keras.layers.Conv2D(64,3 ,input_shape=ishape, activation="relu"),
    keras.layers.Conv2D(64,3,activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
dout_model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])
dout_model.fit_generator(train_df, steps_per_epoch=train_pics//bsize, epochs=ep, verbose=1)
```

```
Epoch 1/5
286/286 [==============================] - 49s 170ms/step - loss: 0.3149 - accuracy: 0.2053
Epoch 2/5
286/286 [==============================] - 50s 173ms/step - loss: 0.2973 - accuracy: 0.2657
Epoch 3/5
286/286 [==============================] - 49s 173ms/step - loss: 0.2860 - accuracy: 0.3120
Epoch 4/5
286/286 [==============================] - 49s 172ms/step - loss: 0.2739 - accuracy: 0.3528
Epoch 5/5
286/286 [==============================] - 49s 173ms/step - loss: 0.2591 - accuracy: 0.3962
```

Out[22]: <tensorflow.python.keras.callbacks.History at 0x20a97f466c8>

In [23]:
```
dloss, dacc = dout_model.evaluate(test_df,verbose=0)
dloss, dacc
```

Out[23]: (0.25625014305114746, 0.4085117280483246)

As can be seen from the above results adding drop out and max pooling separately on the base CNN model doesn't improve the fit. This might be because I have only 2 CNN layers and adding multiple layers might help. But since it is taking long to run each of the models, I have not tested it with additional layers.

7. Transfer learning - Apply VGG and ResNet model

In [24]:
```
from keras.applications import VGG16
```

In [25]:
```
from keras.applications import ResNet50
```

In [26]:
```
vgg = VGG16(input_shape=ishape, weights='imagenet', include_top=False)
```

In [27]:
```
rss = ResNet50(input_shape=ishape, weights='imagenet', include_top=False)
```

In [38]:
```
pre_out = keras.layers.Flatten()
pre_out2 = keras.layers.Dense(32, activation="relu")
out_layer = keras.layers.Dense(10, activation="softmax")
vgg_model = keras.Sequential([vgg,pre_out,pre_out2, out_layer])
vgg_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
vgg_model.fit_generator(train_df, epochs=1, steps_per_epoch=train_pics//bsize,verbose=1)
```

```
286/286 [==============================] - 128s 448ms/step - loss: 1.9982 - accuracy: 0.2760
```

Out[38]:
```
<tensorflow.python.keras.callbacks.History at 0x20aac5870c8>
```

In [39]:
```
vloss, vacc = vgg_model.evaluate(test_df,verbose=0)
vloss, vacc
```

Out[39]:
```
(1.9545968770980835, 0.29791030287742615)
```

In [35]:
```
# making some of the layers trainable train only the last 6 layers from the VGG16
for layer in vgg.layers[:10]:
    layer.trainable = False
pre_out = keras.layers.Flatten()
pre_out2 = keras.layers.Dense(32, activation="relu")
out_layer = keras.layers.Dense(10, activation="softmax")
vgg_model = keras.Sequential([vgg,pre_out,pre_out2, out_layer])
vgg_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
vgg_model.fit_generator(train_df, epochs=ep, steps_per_epoch=train_pics//bsize,verbose=1)
```

```
Epoch 1/5
286/286 [==============================] - 135s 474ms/step - loss: 1.9978 - accuracy: 0.2737
Epoch 2/5
286/286 [==============================] - 135s 472ms/step - loss: 1.9737 - accuracy: 0.2807
Epoch 3/5
286/286 [==============================] - 137s 480ms/step - loss: 1.9708 - accuracy: 0.2844
Epoch 4/5
286/286 [==============================] - 166s 579ms/step - loss: 1.9626 - accuracy: 0.2865
Epoch 5/5
286/286 [==============================] - 134s 470ms/step - loss: 1.9628 - accuracy: 0.2854
```

Out[35]:
```
<tensorflow.python.keras.callbacks.History at 0x20aac44edc8>
```

In [41]:
```
# making some of the layers trainable train only the last 2 layers from the VGG16 and adding 2 hidden layers
for layer in vgg.layers[:14]:
    layer.trainable = False
pre_out = keras.layers.Flatten()
pre_out2 = keras.layers.Dense(64, activation="relu")
pre_out3 = keras.layers.Dense(32, activation="relu")
out_layer = keras.layers.Dense(10, activation="softmax")
vgg_model = keras.Sequential([vgg,pre_out,pre_out2,pre_out3, out_layer])
vgg_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
vgg_model.fit_generator(train_df, epochs=1, steps_per_epoch=train_pics//bsize,verbose=1)
```

```
286/286 [==============================] - 127s 443ms/step - loss: 1.9932 - accuracy: 0.2743
```

Out[41]:
```
<tensorflow.python.keras.callbacks.History at 0x20aa98c4488>
```

As shown above VGG model with additional layers did not provide significant gains in accuracy. This implies that transfer learning is useful case by case and on how the trained models resemble the existin problem.Even adding some of the original vgg layers (six in this case) trainable didn't help to improve the fit significantly. As the # of epochs is not providing significant gains in accuracy in this case for the ResNet I will use only 1 epoch to train fast.ResNet resutls below show that VGG has better performance compared to ResNet. Again with better fine tuning (which I couldn't do because fine-tunning requires significant computing power), the resutls will improve.

In [42]:

```
pre_out = keras.layers.Flatten()
pre_out2 = keras.layers.Dense(32, activation="relu")
out_layer = keras.layers.Dense(10, activation="softmax")
rss_model = keras.Sequential([rss,pre_out,pre_out2,out_layer])
rss_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
rss_model.fit_generator(train_df, epochs=1, steps_per_epoch=train_pics//bsize,verbose=1)
```

```
286/286 [==============================] - 463s 2s/step - loss: 2.3444 - accuracy: 0.2065
```

Out[42]:

```
<tensorflow.python.keras.callbacks.History at 0x20aae00a6c8>
```

In [43]:

```
pre_out = keras.layers.Flatten()
pre_out2 = keras.layers.Dense(64, activation="relu")
out_layer = keras.layers.Dense(10, activation="softmax")
rss_model = keras.Sequential([rss,pre_out,pre_out2, out_layer])
rss_model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
rss_model.fit_generator(train_df, epochs=1, steps_per_epoch=train_pics//bsize,verbose=1)
```

```
286/286 [==============================] - 469s 2s/step - loss: 2.2936 - accuracy: 0.2191
```

Out[43]:

```
<tensorflow.python.keras.callbacks.History at 0x20ab4b8f0c8>
```

In [44]:

```
rloss, racc = rss_model.evaluate(test_df,verbose=0)
rloss, racc
```

Out[44]:

```
(2.1653473377227783, 0.24554026126861572)
```

In [50]:

```
model = ["Plain Neural Networks","Base CNN", "Base CNN+pooling", "Base CNN+droput","VGG16","ResNet50"]
loss = [nloss,bloss, ploss,dloss,vloss,rloss]
acc = [nacc,bacc,pacc,dacc,vacc,racc]
df = pd.DataFrame()
df["Model"] = model
df["Loss"] = loss
df["Accuracy"] = acc
print("Test loss and accuracy of the different models")
df
```

```
Test loss and accuracy of the different models
```

Out[50]:

	Model	Loss	Accuracy
0	Plain Neural Networks	1.962245	0.300204
1	Base CNN	1.540431	0.526376
2	Base CNN+pooling	1.543250	0.456549
3	Base CNN+droput	0.256250	0.408512
4	VGG16	1.954597	0.297910
5	ResNet50	2.165347	0.245540

As the summary results above show the CNN models gave much higher resutls than transfer learning models (VGG and ResNet). This implies that transfer learning is effective only when the features are similar. The results with CNN also show that there are improvement areas which can happen by fine tuning the different parameters to improve the fit. Importantly enjoying the power of CNNs requires strong computing powers inorder to fine tune the different parameters and layers.

8. Summary and next steps

In this project, an effort has been made to classify a 10 class image data using CNN. The results show that though CNN's are strong, **they require a lot of fine tuning and computing power to give satisfactory results**. The prediciton accuracy found on this project was low and show that further work is required in improving the work including:

- Running the CNN model with multiple layers. I haven't done this as even with only 2 CNNs my personal laptop was slow to run the model, but in the future with better capacity computers increasing layers can improve performance
- Conducting Hyperparameter tunning. Configuring the various hyperparameters and testing different options can give better insights to understand how performance varies and in the end help tune the performance. But again this aso requires better computing performance
- Using different activation functions and seeing which performs better
- Training the data with other transfer learning algorims. I used here only VGG16 and ResNet but testing with other options is good. In addition I applied the models using epoch=1 only because it was taking long but with better computing power adding epochs can also improve performance. Transfer learning resutls also show that like using trained CNN models applying learned models (transfer learning) also requires significant fine tuning of hyperparameters and computing power.

================================== Thank you! ================================