Name: Marcela Pantoja

Class: CS 4375.004

Image Classification with DL

```
#Adding a bunch of necessary packages needed for seperating the data set.
#Already have come from the beginning
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import tensorflow as tf
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
num_images = 0

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
        num_images = num_images +1
```

```
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/e830bd6011902b40.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/4352b19e155bd4ed.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/10737ee53398f08a.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/eab5a653e7bc2c95.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/adfdaaeb0e86ac94.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/84fea0403c89c288.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0054521c7ff56e05.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/f7bef9767d3d1596.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/691ec9943cac126b.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/00659b5ea7f16836.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0a8857458ae8e01e.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/06c269f657184270.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/1b755120ca2f6cba.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/00fac52924506248.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/91c7305af51a297c.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/002c2d9952ea0b75.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/1e65eb17460379e9.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/06370304ca3a4de9.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/003d62f84395408f.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/00ad4439a6573080.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/430fd1bf11e8576c.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0005dcb871947109.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/70116560e280ac2c.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/13e2acb33f29b384.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/9e914b54d40d8800.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0568d569fd07c96c.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/006a69cecb56c6b4.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/18322c4182d99007.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/526af4a635970b26.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/05dcb92ec07ac597.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/003ff467d787984e.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/509b768aa44dd931.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/68bf461dd7b96ade.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/c4aeb4f3632823f0.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/eb226814a3585764.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/388a91732c5f45f0.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/2224360ced682d66.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/93a37bb83b4994f3.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/046a01240ac36aa2.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/017aca08f66253f5.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/3bc3f1339b345d32.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/9a8ba5c4717c153a.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/00076a1ba8912d87.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/87c3f099efb2eff9.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/d35eb436350a6c04.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/aa3c7c46a34dadd5.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/082dc957e408ba17.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0920d7dbdc425032.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/b2915ebd9ce1dbf9.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/b1c0565f938bd9a7.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/c248190804433af6.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/eb551b1c169fe188.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/e8c4adb3a073997c.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/0017218b3f99aef7.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/c60eec6c2fd32c18.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/000b4ad7e2dbed61.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/8459035ec5737355.jpg
/kaggle/input/lions-or-cheetahs-image-classification/images/Lions/00082f7d340d2883.jpg
```

```
print("Number of images in the data set: ", num_images)
```

```
Number of images in the data set:  200
```

Divide the Dataset into train/test split

```
#We're going to be splitting the data set into a 80/20 split, so with 200 images it should be 160/40.
#Theres only two classes, cheeta and lion

train_img = tf.keras.utils.image_dataset_from_directory('/kaggle/input/lions-or-cheetahs-image-classification/images', validation_split=0.2, subset="trainin

val_img = tf.keras.utils.image_dataset_from_directory('/kaggle/input/lions-or-cheetahs-image-classification/images', validation_split=0.2, subset="validatio

val_batches = tf.data.experimental.cardinality(val_img)
test_seg = val_img.take(val_batches // 5)
val_img = val_img.skip(val_batches // 5)
```

```
Found 200 files belonging to 2 classes.
Using 160 files for training.
Found 200 files belonging to 2 classes.
Using 40 files for validation.
```

Graph showing the distribution of the target classes

```
plt.figure(figsize=(6,5))
plt.xticks(np.arange(2))
plt.title("Distribution of Target Classes")
plt.xlabel("Target Classes")
plt.ylabel("Num of Images")
plt.bar(['Cheetas', 'Lions'], 100)
plt.show()
```

The data set I have chosen contains and even number of images of cheetas and lions. The model I am creating should be able to give an accurate prediction of whether or not the image shown is a cheeta of a lion. There are images of cheetas and lions in the data set in a variety of different positions and areas in an effort to train the model better.

Note: I had to make the image sizes very large in order to get the pictures to show up as legible, I experimented for a while with the image sizes. There are also a number of images that seem to be netiher a lion or a cheeta, these have a tendency to be in the lion class though.

Create a sequential model and evaluate on the test data

```
#Ok, so here is the model building part

model = tf.keras.models.Sequential([
  tf.keras.layers.Rescaling(1./499),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(500, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(500, activation='relu'),
  tf.keras.layers.Dropout(0.2),
```

```
  tf.keras.layers.Dense(2, activation='softmax')
])



#Here is the fixing the build and making sure its good
model.build(input_shape=(None, 125, 125, 3))

#Outputting the summary of the model befor compiling
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_7 (Rescaling)     (None, 125, 125, 3)       0

 flatten_7 (Flatten)         (None, 46875)             0

 dense_21 (Dense)            (None, 500)               23438000

 dropout_14 (Dropout)        (None, 500)               0

 dense_22 (Dense)            (None, 500)               250500

 dropout_15 (Dropout)        (None, 500)               0

 dense_23 (Dense)            (None, 2)                 1002

=================================================================
Total params: 23,689,502
Trainable params: 23,689,502
Non-trainable params: 0
_________________________________________________________________
```

```
#Model compilation
#Using optimizer Adamax at first loss and metric=accuracy of course
model.compile(optimizer='Adamax', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

#8 epochs to start, maybe change if bad accuracy
history = model.fit(train_img, validation_data=val_img, epochs=8)
```

```
Epoch 1/8
7/7 [==============================] - 4s 340ms/step - loss: 10.6832 - accuracy: 0.5562 - val_loss: 9.3781 - val_accuracy: 0.4500
Epoch 2/8
7/7 [==============================] - 3s 318ms/step - loss: 5.8222 - accuracy: 0.4875 - val_loss: 5.1514 - val_accuracy: 0.4500
Epoch 3/8
7/7 [==============================] - 3s 315ms/step - loss: 3.2184 - accuracy: 0.5250 - val_loss: 1.9272 - val_accuracy: 0.5500
Epoch 4/8
7/7 [==============================] - 3s 318ms/step - loss: 2.4875 - accuracy: 0.5125 - val_loss: 0.8778 - val_accuracy: 0.5500
Epoch 5/8
7/7 [==============================] - 3s 316ms/step - loss: 1.7194 - accuracy: 0.5500 - val_loss: 0.9717 - val_accuracy: 0.4500
Epoch 6/8
7/7 [==============================] - 3s 315ms/step - loss: 1.7105 - accuracy: 0.5375 - val_loss: 1.2379 - val_accuracy: 0.4250
Epoch 7/8
7/7 [==============================] - 3s 317ms/step - loss: 1.3054 - accuracy: 0.6250 - val_loss: 0.7920 - val_accuracy: 0.4750
Epoch 8/8
7/7 [==============================] - 3s 317ms/step - loss: 1.2452 - accuracy: 0.5688 - val_loss: 0.7280 - val_accuracy: 0.6250
```

Ok, so after running the sequential model using the Adamax optimizer and 8 epochs, the accuracy isn't good, but also isnt completely terrible at 0.6250. This means that we didnt overfit the data (as could be seen with a really high accuracy score of like 99.

Try a different architectures like CNN, etc and evaluate on the test data

```
#First Im going to try CNN

#Model is the only thing different; build, cummary, compile and history all the same
#Kinda going to have the values match the sequential model where they can
CNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(25, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(25, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(25, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D((2,2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(500, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(2, activation='softmax')
])
```

```
#Here is the fixing the build and making sure its good
CNN.build(input_shape=(None, 125, 125, 3))

#Outputting the summary of the model befor compiling
CNN.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 123, 123, 25)      700

 max_pooling2d_9 (MaxPooling  (None, 61, 61, 25)       0
 2D)

 conv2d_10 (Conv2D)          (None, 59, 59, 25)        5650

 max_pooling2d_10 (MaxPoolin  (None, 29, 29, 25)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 27, 27, 25)        5650

 max_pooling2d_11 (MaxPoolin  (None, 13, 13, 25)       0
 g2D)

 flatten_11 (Flatten)        (None, 4225)              0

 dense_30 (Dense)            (None, 500)               2113000

 dropout_19 (Dropout)        (None, 500)               0

 dense_31 (Dense)            (None, 2)                 1002

=================================================================
Total params: 2,126,002
Trainable params: 2,126,002
Non-trainable params: 0
_________________________________________________________________
```

```
#Model compilation
#Using optimizer Adamax at first loss and metric=accuracy of course
CNN.compile(optimizer='Adamax', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

#8 epochs to start, maybe change if bad accuracy
history = CNN.fit(train_img, validation_data=val_img, epochs=8)
```

```
Epoch 1/8
7/7 [==============================] - 4s 303ms/step - loss: 0.1322 - accuracy: 0.9563 - val_loss: 2.7683 - val_accuracy: 0.4750
Epoch 2/8
7/7 [==============================] - 3s 287ms/step - loss: 0.0617 - accuracy: 0.9750 - val_loss: 4.1373 - val_accuracy: 0.4750
Epoch 3/8
7/7 [==============================] - 3s 284ms/step - loss: 0.0455 - accuracy: 0.9750 - val_loss: 5.0786 - val_accuracy: 0.4500
Epoch 4/8
7/7 [==============================] - 3s 284ms/step - loss: 0.0511 - accuracy: 0.9812 - val_loss: 4.7388 - val_accuracy: 0.5000
Epoch 5/8
7/7 [==============================] - 3s 322ms/step - loss: 0.0171 - accuracy: 0.9937 - val_loss: 4.9991 - val_accuracy: 0.5000
Epoch 6/8
7/7 [==============================] - 3s 288ms/step - loss: 0.0345 - accuracy: 0.9937 - val_loss: 5.4822 - val_accuracy: 0.5000
Epoch 7/8
7/7 [==============================] - 3s 288ms/step - loss: 0.0328 - accuracy: 0.9875 - val_loss: 5.9400 - val_accuracy: 0.4750
Epoch 8/8
7/7 [==============================] - 3s 287ms/step - loss: 0.0487 - accuracy: 0.9812 - val_loss: 5.2210 - val_accuracy: 0.5500
```

The CCN model's accuracy was 0.5050 which is not good at all. I used the same optimizer and epochs as I did in the previous (sequential model) model, but the result was much worse.

Analysis of the performance of various approaches

Neither the Sequential nor CNN model performed very well, but the seuqnetial model permormed quite a bit better than the CNN model despite them both having the same number of epochs and optimizer. However, I could not seem to get either of my models up to a resonably good accuracy (around 80) by matching them.

I did try out the CNN model with a different optimizer (Adam instead of AdamMax) and the accuracy actually ended up dropping by around 0.1 so that did not improve accuracy. I think this low accuracy may be a result of there being images inside the dataset the included nethier a lion or a cheetah (dog, gorilla, statues, ect.) as it may have messed up the training set for the models.