Test-3: Learning Mass of Dark Matter Halo

Image Shape - (150,150)

Output - Mass (single value)

Metric - MSE

```
import pandas as pd
import numpy as np
import glob
import matplotlib.pyplot as plt
import gc
from sklearn.preprocessing import MinMaxScaler,StandardScaler
```

```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
# !tar -xzvf /content/drive/MyDrive/lens_data_alt.tgz
```

Reading and preparing data

```
path_to_data =                              #"/content/lens_data"
npy_data = glob.glob(path_to_data + "/*.npy")
images = []
y = []
for file in npy_data:
   a = np.load(file,allow_pickle=True)
   images.append(a[0])
   y.append(a[1])

images = np.array(images).astype('float')
y = np.array(y).astype('float').reshape(-1, 1)

images = images.reshape(-1,150,150,1)
```

Normalising masses

Input values scaled between 0 and 1. Standardizing inputs had no significant

```
norm = MinMaxScaler()
norm.fit(y)
y = norm.transform(y)

# scaler = StandardScaler()
# scaler.fit(images)
# X = scaler.transform(images).reshape(-1,150,150,1)
X = images/255
```

```
print('y \nmean / std dev / max / min : %0.4f / %0.4f / %0.4f / %0.4f\n'%(y.mean(), y.s
```

```
y
mean / std dev / max / min : 0.3710 / 0.1396 / 1.0000 / 0.0000
```

Deleting temporary variables

```
del images
gc.collect()
```

```
419
```

Importing important Keras functions

```
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.initializers import TruncatedNormal
from keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D,BatchNorm
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras import backend as K
from keras.models import Model
from tensorflow.keras.losses import mean_squared_error
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.random import set_seed
set_seed(42)
```

Setting parameters for the model

```
lr_init     = 1.e-4     # Initial learning rate
batch_size  = 64       # Training batch size
epochs      = 25       # Number of epochs
doGPU       = True    # Use GPU
img_rows = 150
img_cols = 150


if doGPU:
    import tensorflow.compat.v1 as tf
    from tensorflow.compat.v1.keras.backend import set_session
    config = tf.ConfigProto()
    config.gpu_options.allow_growth=True
    set_session(tf.Session(config=config))


def rmse(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))
```

Augmentation layer to prevent overfitting

```
data_augmentation = tf.keras.Sequential([
  RandomFlip("horizontal_and_vertical"),    # Random flipping of the image
  RandomRotation((0,1)),                    # Rotation from 0 to 360 degrees
  RandomCrop(150,150)                       # random cropping
])
```

Main Model

ResNet50

Pre-trained weight

All layers trainable

Pre-trained weights helps in converging early, so does the making all the layers trainable (determined by experimentation).

To feed the image to ResNet, image had to be made of 3 channels. There were 2 options.

1. Add a Convolution layer
2. Make 3 channel image with same values in all the channels.

   1st option gave better results.

Loss for Regression - Root Mean Sqaured Error

Extra metric- Mean Absolute Percentage error (gives extremely large values for close to 0, used here just to observe the general trend.)

```
ResNet50_model = ResNet50(weights= 'imagenet', include_top=False, input_shape=(150,150,
for layers in ResNet50_model.layers:
    layers.trainable= True
opt = Adam(learning_rate=lr_init)


model = tf.keras.Sequential([
                             data_augmentation,
                             Conv2D(3,(3,3),padding='same'),
                             ResNet50_model,
                             Flatten(),
                             Dense(256,activation='relu'),
                             Dropout(0.5),
                             Dense(1,activation='sigmoid')      # After comparison, cho
                            ])

model.compile(loss = rmse, optimizer= opt, metrics=['mean_squared_error','mape'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

Fitting the model.

Used 2 callbacks to reduce learning rate and for early stopping, based on changes in validation loss.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1.e-8)
earlyStopping = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='min')
history=model.fit(X, y,\
        batch_size=batch_size,\
        epochs=epochs,\
        # validation_data=(x_val, y_val),\
        callbacks=[reduce_lr, earlyStopping ],\
        verbose=1, shuffle=True,
        validation_split=0.1)
```

```
Epoch 1/25
282/282 [==============================] - 237s 759ms/step - loss: 0.1816 - mean_s
Epoch 2/25
```

```
Epoch 2/25
282/282 [==============================] - 212s 752ms/step - loss: 0.1297 - mean_s
Epoch 3/25
282/282 [==============================] - 215s 762ms/step - loss: 0.1162 - mean_s
Epoch 4/25
282/282 [==============================] - 210s 746ms/step - loss: 0.1087 - mean_s
Epoch 5/25
282/282 [==============================] - 209s 742ms/step - loss: 0.1064 - mean_s
Epoch 6/25
282/282 [==============================] - 215s 762ms/step - loss: 0.1028 - mean_s
Epoch 7/25
282/282 [==============================] - 211s 748ms/step - loss: 0.1011 - mean_s
Epoch 8/25
282/282 [==============================] - 210s 744ms/step - loss: 0.0992 - mean_s
Epoch 9/25
282/282 [==============================] - 211s 749ms/step - loss: 0.0954 - mean_s
Epoch 10/25
282/282 [==============================] - 208s 739ms/step - loss: 0.0938 - mean_s
Epoch 11/25
282/282 [==============================] - 207s 734ms/step - loss: 0.0934 - mean_s
Epoch 12/25
282/282 [==============================] - 209s 741ms/step - loss: 0.0916 - mean_s
Epoch 13/25
282/282 [==============================] - 209s 742ms/step - loss: 0.0920 - mean_s
Epoch 14/25
282/282 [==============================] - 209s 740ms/step - loss: 0.0911 - mean_s
Epoch 15/25
282/282 [==============================] - 207s 735ms/step - loss: 0.0901 - mean_s
Epoch 16/25
282/282 [==============================] - 208s 736ms/step - loss: 0.0904 - mean_s
Epoch 17/25
282/282 [==============================] - 208s 739ms/step - loss: 0.0889 - mean_s
Epoch 18/25
282/282 [==============================] - 211s 749ms/step - loss: 0.0885 - mean_s
Epoch 19/25
282/282 [==============================] - 209s 740ms/step - loss: 0.0885 - mean_s
Epoch 20/25
282/282 [==============================] - 208s 736ms/step - loss: 0.0879 - mean_s
Epoch 21/25
282/282 [==============================] - 211s 748ms/step - loss: 0.0871 - mean_s
Epoch 22/25
282/282 [==============================] - 208s 737ms/step - loss: 0.0878 - mean_s
Epoch 23/25
282/282 [==============================] - 207s 733ms/step - loss: 0.0880 - mean_s
Epoch 24/25
282/282 [==============================] - 206s 731ms/step - loss: 0.0873 - mean_s
Epoch 25/25
282/282 [==============================] - 208s 736ms/step - loss: 0.0876 - mean_s
```

Validation MSE over training

```
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
```

```
plt.title('Mean Squared Error')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```

```
model.save_weights('model.h5')
from google.colab import files
files.download('model.h5')
```