**README:**

The machine learning algorithm to detect explosions is still a work in progress. The first stage of the process was developed using a Convolutional Neural Network (CNN), designed using Keras and Tensorflow libraries for python. It is designed to train on/ test images to see if an explosion is occurring in the image.

In the CNN folder, there is a python code called Train\_model.py. This is the code that will train the algorithm to differentiate between the two classes (explosions vs non). To run this code, you need to have pillow (PIL), Keras, and matplotlib installed (also h5py is needed if not installed!).

The file will read in the train data set (in images\_explosion/train) to train the CNN. The easily changeable variable are ‘steps\_per\_epoch’, ‘epoch’, and ‘validation\_steps’. The validation data set is in images\_explosion/test\_set.

An Epoch is the number of loops you wish to employ to train your algorithm on, i.e. how many loos over the training data set. The larger the number, the better the algorithm, but there will be diminishing returns and the time will scale with the epoch number.

Steps per epoch is how many images it will use when doing a loop over the training set. You can declare this number but having ‘None’ sets the steps to include each image once training.

Validation steps is similar to the steps per epoch but for the validation set. Again, you can choose the number of images but setting this to ‘None’ will use every image one time.

After letting the program run, the program will save 3 objects. The first object the is trained algorithm itself. This is on line 54, classifier.save(). The other 2 objects are plots to show the Loss Curve (i.e. error) on the algorithm for the training and validation sets, and the Accuracy Curve of the two sets.

**Using the algorithm:**

To use the algorithm to test an image, you will use ‘Run\_model.py’. This function opens up the algorithm you trained (explosion\_trained.h5py) and ask for the image you wish to test. It will test the image using the algorithm and output whether the image is an Explosion or Non-Explosion.

**PHASE 2:**

The next stage is incomplete. I will list the logic here and planned future steps here.

There is a machine learning concept called Recurring Neural Network (RNN). At its core, it is designed to have a memory that allows to it to see what has changed from previous frames. This is ideal for what we are looking into, as an explosion is a large change in the frame.

First, we must build a data set that can be split into training and validation data. The larger the data set the better the algorithm can be. There are various data sets of videos that have been used for classification, but ‘explosion’ does not seem to be one of those classes. However, those videos will do well for the ‘non-explosion’ class. One very large data set is Youtube-8m.

After getting the images, we need to extract images from the video. There are various ways of doing this documented online.

After images are extracted, we can train our RNN and test it against our validation set, like we did with the CNN. This should perform better than just the CNN due to the memory of previous frames, but only testing will show if that is true.

**EXAMPLE:**

There was an example I have found that does something similar to what we want. The post is here:

<https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5>

I have downloaded the code and placed it into the RNN folder. It can be modified to perform what we wish (it even uses Keras already!). Make sure your requirements are upto date by looking into ‘Requirements.txt’.

The basic steps are as follows:

1. Get Data set
2. Extract Features (extract\_features.py)
3. Train model (train.py, change line 87 to ‘lrcn’ (RNN model described in model.py)
4. Validate model (validate\_rnn.py, change line 40 to ‘lrcn’ and to the saved model)

The example comes with instructions to download a dataset, but the classes it has is not does not include explosions. If you already have a dataset with the classes you want, the first step is finished. You will need to modify extract\_features to work with only 2 classes that are not the pre-defined classes in the example, but it is a starting point.