

# Machine Learning Engineer Nanodegree Capstone Report

Patrick Albert Mikoaski de Souza

April 4th, 2021

## Detecting presence of fire in images (from Kaggle<sup>1</sup> datasets)

Field: Computer Vision

### 1. Project Overview

According to the wikipedia<sup>2</sup>, **computer vision** is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information in the forms of decisions.

As a person who lives in Brazil, which is a country that suffers a lot from forest fires annually, I believe it is possible to facilitate detection and combat this type of problem using computer vision as a tool.

From a local perspective, a greepeace<sup>3</sup> article stated that the fires in 2020 are the worst in the last decade.

Early detection systems using computer vision with AI could help firefighters combat blazes, aid in recovery, and prevent wildfires from starting to begin with. In addition, it is possible that environmental and political advocates create control and prevention systems with the mapped data obtained.

The dataset is provided by a kaggle dataset page, which was created by a group which modelled a similar problem in Nasa Space Apps Challenge in 2018.

Data<sup>1</sup> is divided in **fireimages** and **non-fireimages**. The fireimages contain images with fire and heavy smokes and the non-fireimages contain nature images in general.

The data is skewed, so some proper treatment or alternative metric need to be settled in the images analysis.

### 2. Problem Statement

The main problem is to construct a model that identifies if a image contain fire or not, so the problem is essentially a binary classification problem.

An image must be passed to the model, be processed and return if the image contains fire or not - returning a numeric representation as 0 or 1.

The plan is to use Convolutional Neural Networks and some similar alternatives (as transfer learning and data augmentation) to build the better model possible to detect fire presence into the images.

With cameras installed around the world, it would be possible to check critical points and susceptible to fire.

This model also can be implemented as a web app so the users can upload a image and verify if the image contain fire or not.

The expected solution is a model wich will recognize the majority of the imagens containing fire.

### 3. Metrics

The main metric is the accuracy, the number of the overall right answers of the model.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

The data is skewed and unbalanced, so we need to apply some alternative metrics to understand the results.

First we will analyze the **true positives** and **true negatives**, because of the type of the problem, it's better we have false non-fire photos than false fire photos. In the real life if we can't identify a fire this will generate a real problem, so it's better to misclassify a image as fire to than non-fire.

For that we will use **precision** and **recall**, and will observe the **confusion matrix** accordlly to the labels of each class.

$$\mathbf{Precision} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$\mathbf{Recall} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

In this project, we can say the labels are fire and non-fire, so we need to maximize the fire class recall, with the lesser FALSE NEGATIVES possibles.

The confusion matrix will help to visualize all the metrics in the same graphic and the trade-off between each model and metric.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

#### 4. Data Exploration

The dataset contains 755 fire images and 244 non fire images.

For the resolution of this problem i divided the dataset into train/val/test in the proportion: 0.7 / 0.2 / 0.1.

The division was stratified in the same proportion as the original dataset: Train has 528 fire files and 170 non fire files. Val has 151 fire files and 48 non fire files. Test has 76 fire files and 26 non fire files.

Train is used to train the model, val to verify the better epoch values in the training and test to verify the final performance of the model.

The images have different sizes but all the images passed to the neural network will be resized to a defined fixed size.

We can see the smallest, biggest and mean shapes of the images bellow:

```
Smallest image: (147, 220, 3)
Biggest image: (4608, 6144, 3)
Mean size of the images: (261, 357, 3)
```

The images are passed trough a generator that applies all the processing needed as normalizing and resize and some data augmentation techniques.

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

valid_datagen = ImageDataGenerator(rescale=1. / 255)
```

The descriptive statistics of each class can be seen in the table below:

	Mean	Median	Min	Max	Std
<b>Fire</b>	80.023530	61.684768	0.369536	254.981457	62.981823
<b>Non-fire</b>	81.651768	70.270492	0.196721	251.856557	56.423184

So the mean/median values of the non-fire are higher, we can assume the average of the non-fire images are brighter than the fire images. The fire images have more deviation in pixels values, probably because the fire images contain a lot of similar information as the non-fire with addition of the fire (or smoke) effect in the images, increasing the possibility of greater variation in the pixels of the images.

Some characteristics of the dataset is the fact that the fire images can be images with fire, with smoke or with both. There are NO mislabeled images because the dataset is well collected.

The images can be blurry because of the smoke in the images and some of the images can be out of focus or with lesser quality (because of the variability in shapes).

Let's see some few examples:

Example of tiny image:



Example of blurred image:



Example of image with good resolution:



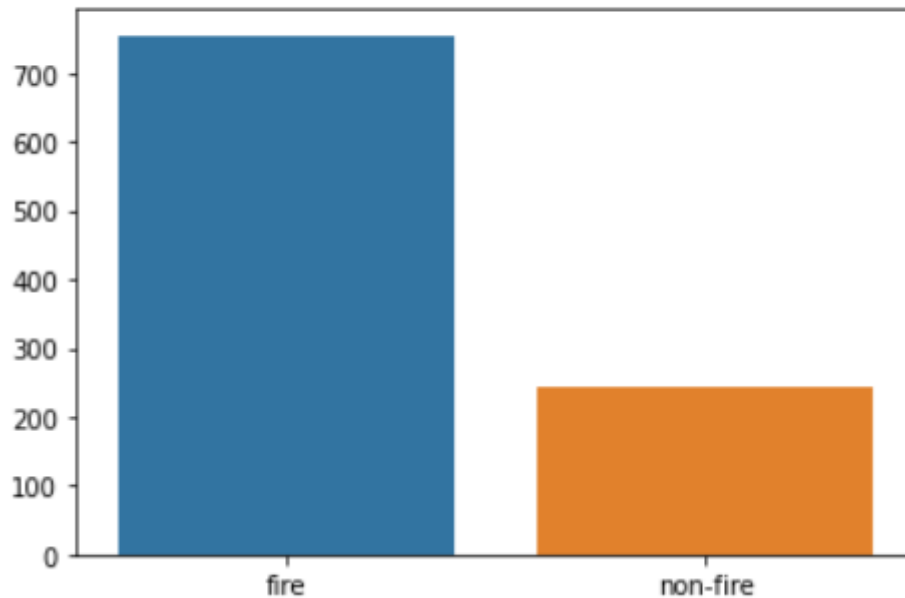
Example of image with bad resolution:



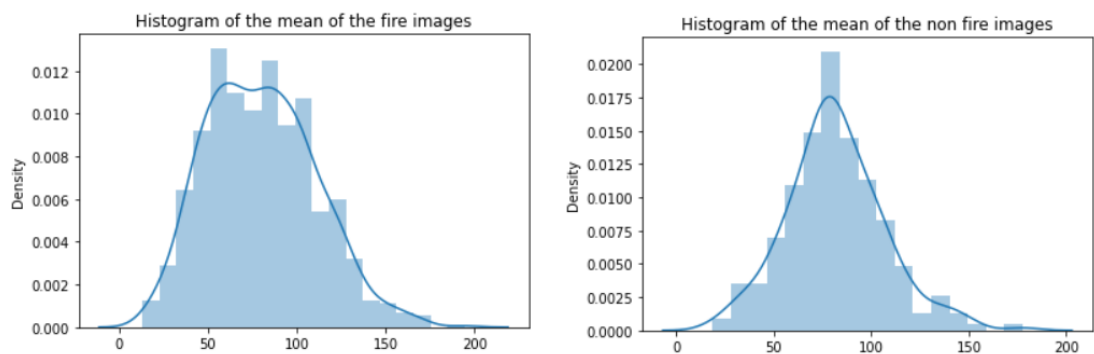
## 5. Visualization

The frequency of each class can be seen at the bar plot:

Fire files : 755  
Non fire files: 244



We can see the histogram of the intensity for the mean pixels of each class:

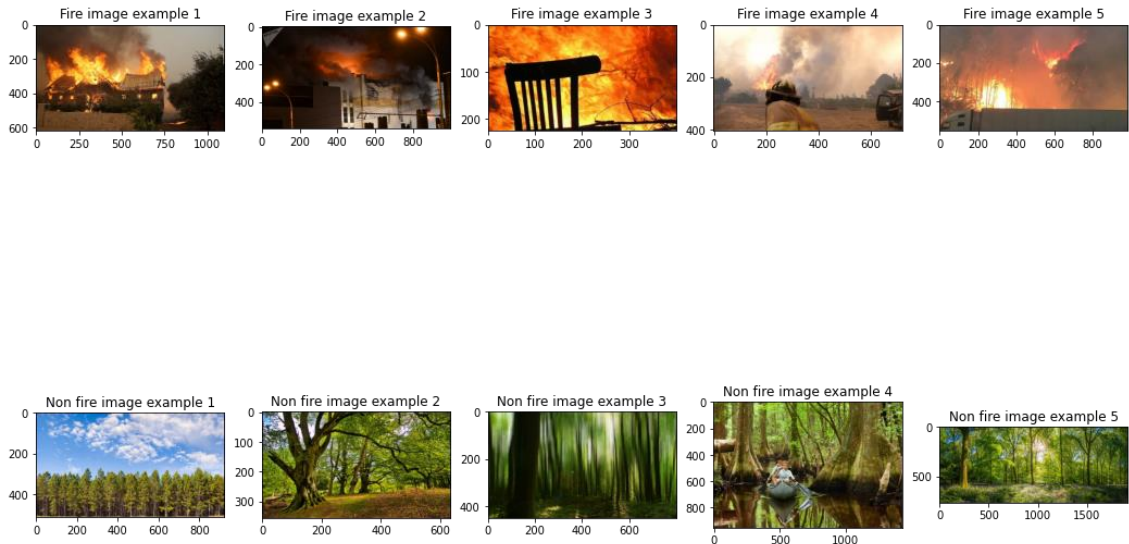


The shape of the histogram was similar but the distribution of non fire images was concentrated in the central points. We can assume that the non fire images are brighter in general, with a higher mean value and the fire images have more variability in the pixel values.

We can assume the majority of non-fire images have a similar distribution of mean pixels cause the histogram for non fire is much thinner.

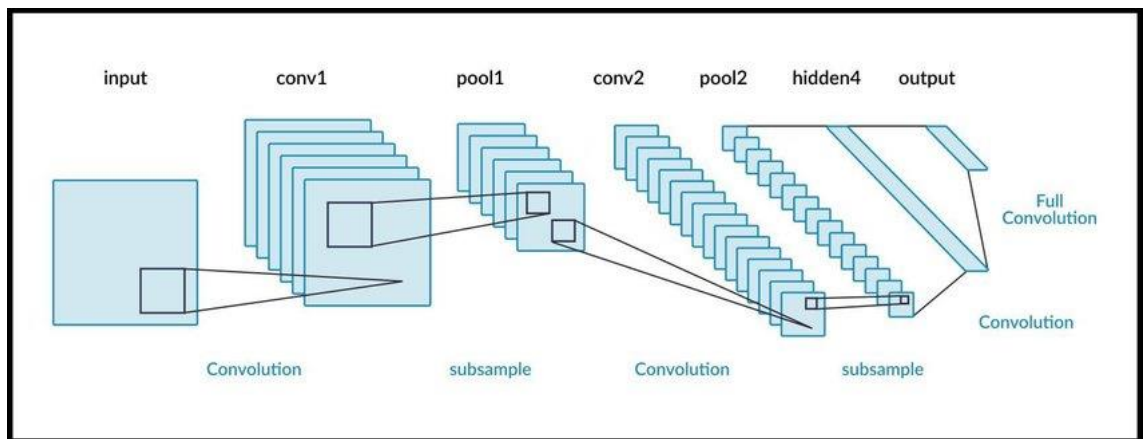
To analyze the data i plotted the visualization of 10 random images from train dataset:





## 6. Algorithm

The classification model was constructed using Convolutional Neural Network structure, with the objective to extract the features from the images and generate features maps to more complex analysis than traditional classification.



The CNN structure used in the first model is a classic combination of Convolutional Blocks plus Classification Block:

### 1. Conv Blocks:

- Conv 2D
- MaxPooling2D
- Conv 2D
- MaxPooling2D
- Conv 2D

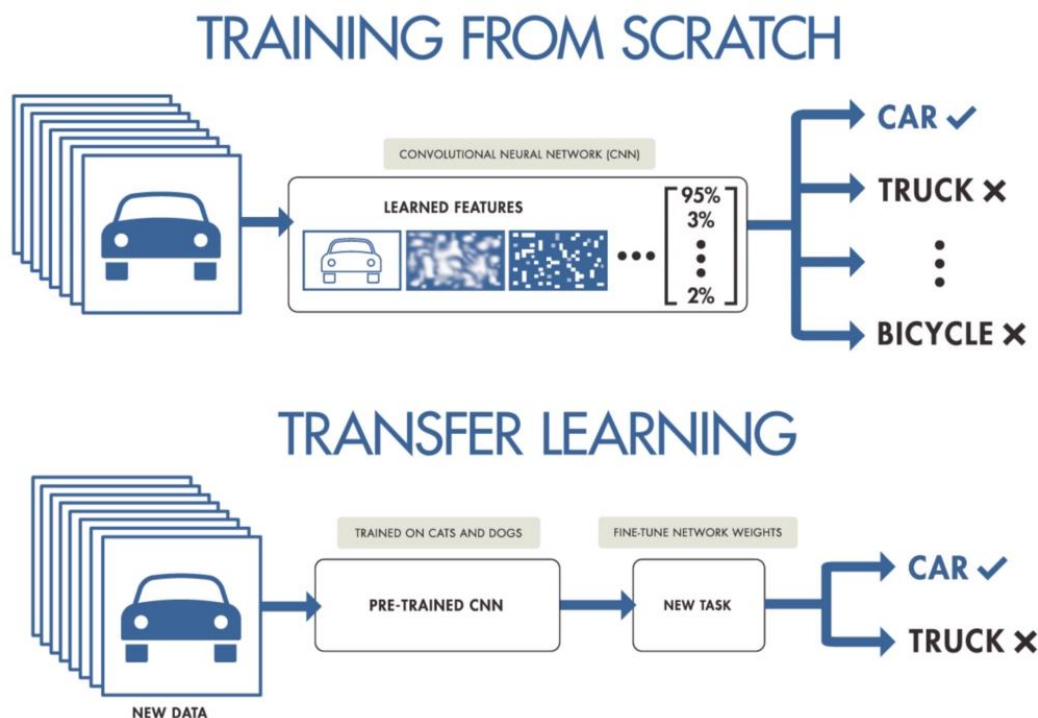
- MaxPooling2D

## 2. Classification Block

- Flatten
- Dense (with a lot of neurons, to increase complexity, with relu activation)
- Dense (with the number of the classes of the target and a softmax or sigmoid activation)

Another technique used to improve the accuracy is use pre-trained network to inherit complex features from bigger datasets.

We use the pre-trained conv blocks from another neural network and combine with a newer classification block (accordly to our problem).



## 7. Benchmark Model

As stated in the proposal document, i analyzed others results from [Kaggle](#). One of the best results was 96% accuracy using this data and a traditional cnn architecture. So as a benchmark model i used a traditional cnn architecture and aim to at least 95% accuracy.

## 8. Data Processing

The traditional data processing steps for image tratament is rescale the images multiplying all the pixels to 1/255, normalizing all the pixels to values between 0 and 1.



We need to resize all the images to the same size for input in the neural network. In our example i used the values 160x160.

Another step possible to maximize the results is applying data augmentation, increasing the variability of the dataset applying images manipulations as brightness, rotation, zoom and shear before the image loading.

This was used directly on the generator of the dataset, inside of the training script, and will be applied to the train data, augmentating the original dataset.

```
train_datagen = ImageDataGenerator(  
    rescale=1. / 255,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True  
)  
  
valid_datagen = ImageDataGenerator(rescale=1. / 255)
```

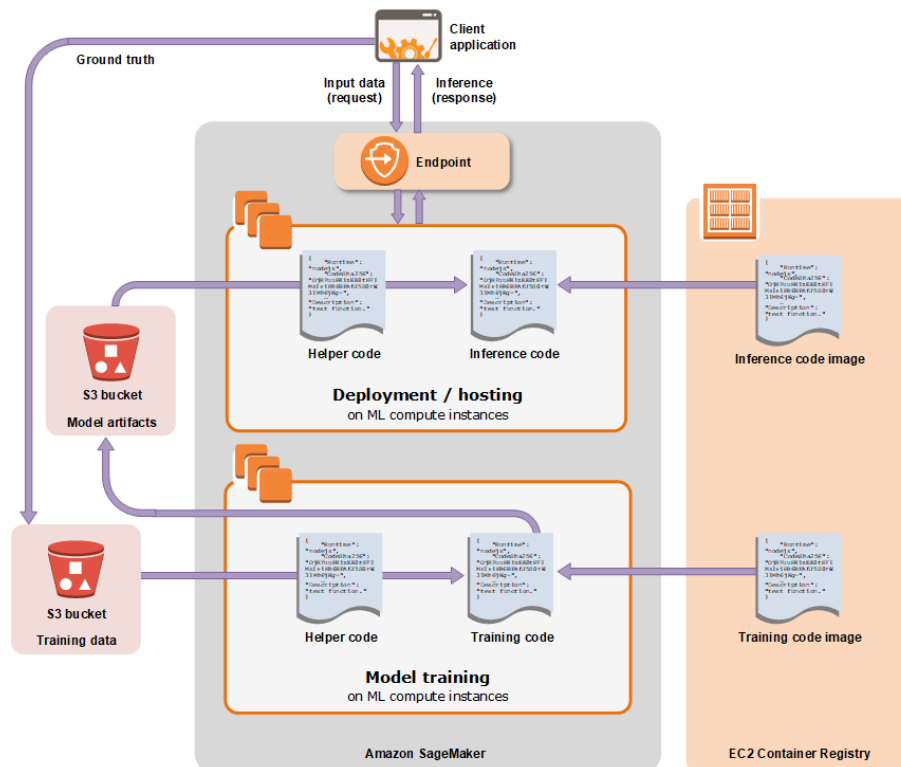
For the transfer learning, we will need to pass the pre-processing function used in the original pretrained neural network (which is probably something similar to the manual rescale we used).

Trying to give better results i passed another data augmentation technique in one of the examples, using more augmentations functions for ImageDataGenerator.

```
train_datagen = ImageDataGenerator(  
    shear_range=0.2,  
    zoom_range=0.2,  
    rotation_range=30,  
    height_shift_range=0.2,  
    width_shift_range=0.2,  
    horizontal_flip=True,  
    brightness_range=[0.8,1.2],  
    preprocessing_function=preprocess_input  
)  
  
valid_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

## 9. Code Implementation, Refinement, Model Evaluation and Validation

The solution is implemented using the steps learned during the Machine Learning Engineering Nanodegree course, applying the AWS Sagemaker<sup>5</sup> service to build, train and deploy this model.



## Step 1: Data Processing

1. Create a notebook instance in Sagemaker
2. Download and prepare the data in the notebook
3. Verify the data characteristics inside of the notebook
4. Pass all the data to S3 service

## Step 2: Data Modeling

5. Build the model  
(In this case, a keras/tf model using train.py)
6. Implement estimator and train the model

## Step 3: Evaluate and Inference

7. Evaluate the performance comparing different models (one cnn model x transfer learning model x data agumentation model)
8. Deploy the best model

### 1.1 Create Notebook Instance in Sagemaker

I created a notebook instance named **project** and created a Jupyter Notebook using **conda tensorflow p36** environment.

## 1.2 Download and prepare the data in the notebook

To download the data from kaggle we need to download a token named **kaggle.json** from the user settings to access the data trough API.

I uploaded that .json to the notebook and installed the needed libraries.

After that i used some commands to download the data.

```
!pip install -q kaggle
!pip install -q kaggle-cli
!mkdir -p ~/.kaggle
!cp "kaggle.json" ~/.kaggle/
!cat ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json
```

```
{"username": "pkamsz2", "key":
```

```
# Download the dataset
```

```
!kaggle datasets download -d "phylake1337/fire-dataset"
```

```
Downloading fire-dataset.zip to /home/ec2-user/SageMaker
```

[illegible]

And unzipped the files:

```
# Unzip files
```

```
!unzip 'fire-dataset.zip'
```

Archive: fire-dataset.zip

### 1.3 Verify the data characteristics inside of the notebook

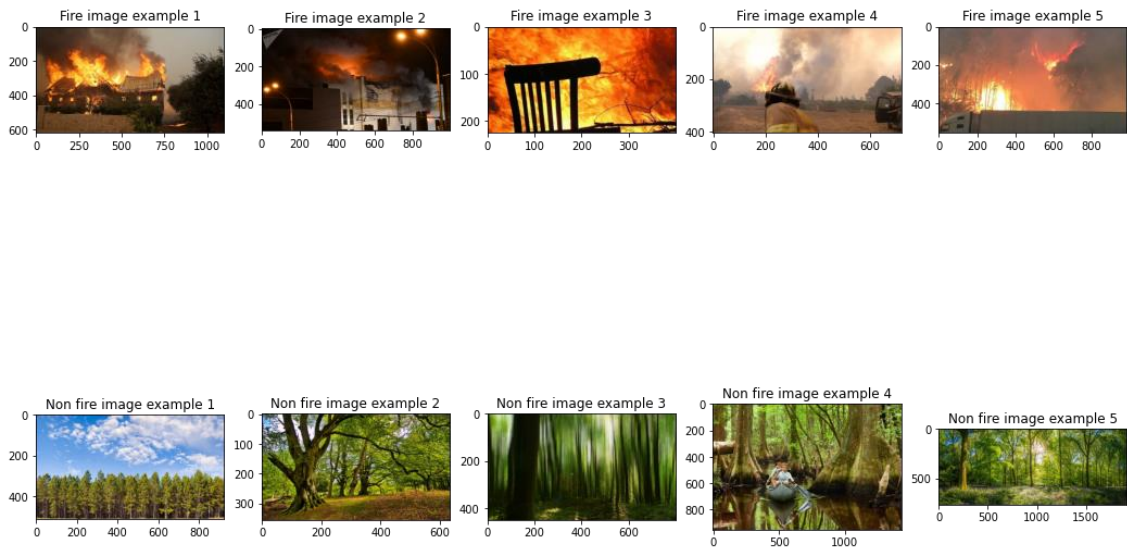
Verified the files resulted from the unzip:

```
fire_files = os.listdir('fire_dataset/fire_images')
nfire_files = os.listdir('fire_dataset/non_fire_images')

print('We have {} fire images and {} non fire images.'.format(len(fire_files), len(nfire_files)))
```

We have 755 fire images and 244 non fire images.

And we can show some of them:



Splitting the data in the proportion 0.7/0.2/0.1 using Split-folders library:

```
# Installing a lib to split the dataset into train and val folders  
!pip install split-folders
```

```
Collecting split-folders  
  Downloading split_folders-0.4.3-py3-none-any.whl (7.4 kB)  
Installing collected packages: split-folders  
Successfully installed split-folders-0.4.3
```

```
import splitfolders  
splitfolders.ratio("fire_dataset", output="fire_dataset_split", seed=42, ratio=(.7, .2, 0.1))
```

Resulting in:

```
Content of fire_dataset_split folder: ['train', 'test', 'val']  
Train has 528 fire files and 170 non fire files.  
Val has 151 fire files and 48 non fire files.  
Test has 76 fire files and 26 non fire files.
```

#### 1.4 Pass all the data to S3 service

Passed all the data to S3 using the `upload_data` module from `sagemaker.Session()`:

## Upload data to S3

```
: role = get_execution_role()
bucket = sagemaker.Session().default_bucket()

def upload_to_s3(bucket, channel, path):
    s3_path_to_data = sagemaker.Session().upload_data(bucket=bucket,
                                                        path=path,
                                                        key_prefix=channel)

: # Data copy to s3
s3_train_key = "fire_dataset_split/train"
s3_validation_key = "fire_dataset_split/val"
s3_test_key = "fire_dataset_split/test"

# upload all data to s3
upload_to_s3(bucket, s3_train_key, s3_train_key)
upload_to_s3(bucket, s3_validation_key, s3_validation_key)
upload_to_s3(bucket, s3_test_key, s3_test_key)
```

## 2.5 / 2.6 Build the model and implement estimator and train the model

I tested three models, all of them using the same call method using TensorFlow estimator of the Sagemaker module.

Estimator:


```
train_input_path = "s3://{}/{}".format(bucket, s3_train_key)
validation_input_path = "s3://{}/{}".format(bucket, s3_validation_key)

tf_version = tf.__version__
tf_version

estimator = TensorFlow(
    entry_point="train_cnn.py",    # My entry script
    role=role,
    train_instance_count=1, # "The number of GPUs instances to use"
    train_instance_type='ml.p2.xlarge',
    framework_version=tf_version,
    py_version="py3",
    script_mode = True,
    hyperparameters={'epochs': 30}
)

print("Training ...")
estimator.fit({'train': train_input_path, 'eval': validation_input_path})
```

.py files:

☐  train\_cnn.py

☐  train\_tf.py

☐  train\_tf\_augmentation.py

I trained the models using TensorflowEstimator class of the Sagemaker service. I wrote three .py scripts, each one using one of the structures cited in the proposal: traditional cnn model and two transfer learning methods (with and without data augmentation).

#### Traditional CNN Model:

```
# === Model ===

model = Sequential()

model.add(Conv2D(512, kernel_size=(3, 3), input_shape=(HEIGHT, WIDTH, DEPTH), activation="relu", name="inputs",
padding="same"))
model.add(MaxPooling2D(3,3))
model.add(Conv2D(224,(3,3),activation='relu'))
model.add(MaxPooling2D(3,3))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(3,3))

model.add(Flatten())

model.add(Dense(256, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.25))
model.add(Dense(NUM_CLASSES, activation="sigmoid"))
```

#### Transfer Learning (VGG16 structure, used with and without data augmentation):

```
# === Model ===

# ResNet structure without classification layer
model = Sequential()
model.add(VGG16(
include_top=False,
pooling='avg',
weights='imagenet'
))

# Output Layer
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.layers[0].trainable = False
```

### 3.7 Evaluate models

The generator applied to the data give the **label 0** to **FIRE** images and **1** to **NON FIRE** images.

The first model, the traditional CNN network, returned the results:



Test accuracy: 0.9803921568627451

Confusion Matrix:

```
[[76  0]
 [ 2 24]]
```

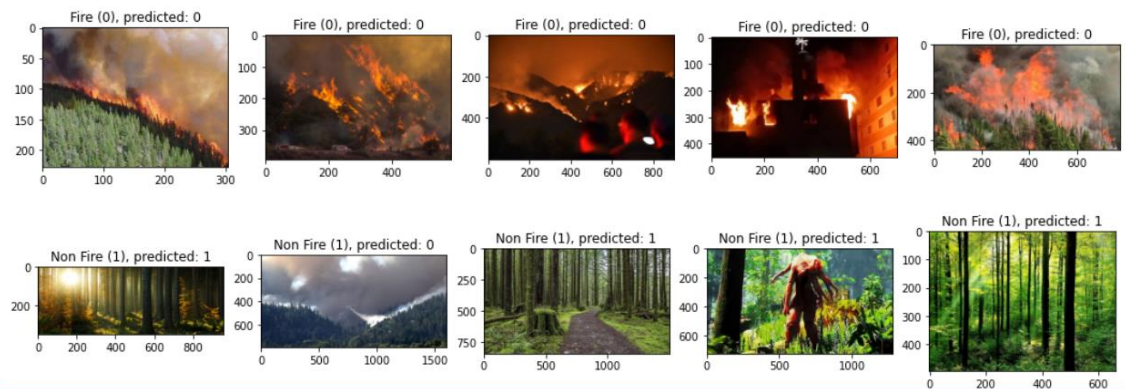
Classification Report:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	76
1	1.00	0.92	0.96	26
accuracy			0.98	102
macro avg	0.99	0.96	0.97	102
weighted avg	0.98	0.98	0.98	102

The first trained model was better than our benchmark model and already have 98% accuracy on the test dataset.

Another great point was the fact that our model classified all the fire images correctly. So we already have a result that is perfect for production.

We can see the plot of the results of the model:



Missclassified one of the smokes images, but get all the fire images!

Let's see the results for the transfer learning and transfer learning + data augmentation:

Test accuracy: 0.9411764705882353

Test Confusion Matrix:

```
[[72  4]
 [ 2 24]]
```

Test Classification Report:

	precision	recall	f1-score	support
0	0.97	0.95	0.96	76
1	0.86	0.92	0.89	26
accuracy			0.94	102
macro avg	0.92	0.94	0.92	102
weighted avg	0.94	0.94	0.94	102

The transfer learning model was worse than the traditional cnn model, misclassifying some of the fire images as nonfire.

For our objective this model was good but not better than the first example.

Last model was the same as the previous but using more data augmentation techniques trying to simulate more variability in the images:

```
Test accuracy: 0.7549019607843137
```

```
Test Confusion Matrix:
```

```
[[52 24]
 [ 1 25]]
```

```
Test Classification Report:
```

	precision	recall	f1-score	support
0	0.98	0.68	0.81	76
1	0.51	0.96	0.67	26
accuracy			0.75	102
macro avg	0.75	0.82	0.74	102
weighted avg	0.86	0.75	0.77	102

The implementation of more data augmentation techniques only dragged down our accuracy. One of the possible problems was the fact that the augmentation doesn't represent the variability present in the test data.

### 3.8 Deploy the best model

After choosing the traditional cnn as the best model, the model was deployed using `.deploy()` method of Sagemaker estimator.

```
predictor = estimator.deploy(initial_instance_count=1, instance_type='ml.p2.xlarge')
```

## 10. Conclusion

Our final model has 98% accuracy, better than the 95% benchmark mark. The model misclassified just 2 out of 24 nonfire images. All the fire images from the test dataset were classified correctly, which was our best scenario hypothesis. The main goal was to construct a pipeline using the Sagemaker platform to build, train and deploy a model and this objective was completed successfully.

As next steps in this project I can say:

- Acquire more data to increase the generability of the model and balance the dataset.
- Try to use the hyperparameter tuning options of the sagemaker.

- Try to use the model endpoint substitution module from sagemaker to alternate between production models.
- Try to implement a combination of API GATEWAY + Lambda to use this model deployed in a web app or in a more complex system.

## 11. Reference

- [1] Kaggle data: <https://www.kaggle.com/phylake1337/fire-dataset>
- [2] [https://en.wikipedia.org/wiki/Computer\\_vision](https://en.wikipedia.org/wiki/Computer_vision)
- [3] [greenpeace.org/international/story/45383/brazil-fire-amazon-pantanal-bolsonaro/](https://greenpeace.org/international/story/45383/brazil-fire-amazon-pantanal-bolsonaro/)
- [4] <https://www.kaggle.com/chemamasamuel/fire-detection-96-accuracy>
- [5] <https://aws.amazon.com/pt/getting-started/hands-on/build-train-deploy-machine-learning-model-sagemaker/>