[Fire No-fire detection]

[LongTimeNoC]

**Data Science Capstone Project-II   
Predictive Modeling Report**

Date:

[03/01/2021]

Team Members:

Name: Himanshu Jat

Name: Smith Amornsaensuk

Name: Chingiz Mardanov

Name: Harshit Geddam

# 1. Problem

Fire detection is a problem that people have been trying to solve for a longtime. It is critical to detect the fire at the earliest stage in order to stop the fire and evacuate people in time. For indoor cases, there are various systems and equipment which are used such as CO2 sensors, smoke detectors, temperature sensors (heat detectors), etc. Sometimes the system alone had a failure rate of 0.32% (Carter, 2008). Nowadays, the use of security cameras has dramatically increased, with an estimation of 85 million cameras in the US in 2021 (Lin & Purnell, 2019). Leveraging the use of a security camera for early detection of fire will potentially help improve the rate of detection. The task can be effectively achieved by using deep learning algorithms to detect fire in images. Furthermore, image fire detection is also beneficial for outdoor cases such as wild fire where fire prevention systems are impossible to be installed. This project will present various deep learning models that are used to classify fire and no fire images

## 1.1 Input

As an input for our model we decided to use formatted images of different environments both containing and not containing fire. We decided to eliminate any of the images that might be ambiguous. We also did not include any images with smoke alone. Originally our data consisted of 2000 images. 1000 with fire and 1000 without fire. They were then split into two different sets i.e. 900 train images as well as 100 test images for each class. In order to provide our models with more data we have decided to also augment our images, by rotating, mirroring, scaling and cropping the images. In order to feed these images into the models we also need to make the images of the same size, all images are resized to be the same.

## 1.2 Data Representation

Our final input format is the numpy array with the following dimensions: (6000,150,150,3). As you can see from the dimensions our images are squares with red, green and blue color channels. All of the images are appended so in the result we obtain an array of images of the length 6000. This numpy array can then be fed into the models.

## 1.3 Output

As mentioned above our goal is to predict the presence of the fire on the images. Thus our output is a binary label where 1 would represent an image with fire, while a 0 label will represent an image without fire. For the purpose of this research we will not build a service that detects fire in the video stream. As an output for most of our model we will actually use a confusion matrix or accuracy for quick comparison. All of the results obtained from our models will be presented in this paper.

# 2. Predictive Models

## 2.1 Multilayer perceptron (MLP)

Multi layer perceptron (MLP) is a type of feed forward neural network. It has three layers, the three layers are input layer, hidden layer and the output layer. The tasks such as predictions and classifications are performed by the Multi layer Perceptron. So, between the input layer and the output layer, we have the hidden layers. As it is a feed forward neural network, the neurons flow from input layer towards the output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. Basically, the MLP is used to solve classification, recognition, prediction problems.

### 2.1.1 MLP Methodology

MLP is the first algorithm that is used in this project. The dataset contains 2,000 images which are 1,000 fire and 1,000 no fire. They are splitted into training, validation and testing sets with 1,600, 200 and 200 images in order. The data is loaded, resized, min-max scaled and labeled according to their classes as the following code.

|  |
| --- |
| def load\_data(file, read\_size=(150,150)):   img = load\_img(file,  target\_size=read\_size,  color\_mode = "rgb",  interpolation="nearest")  return img\_to\_array(img)/255 |

Afterward, by using ImageDataGenerator() module from keras, training data are augmented to be 6,000 images and 600 for testing and validation set. With code below data is passed into a generator and results in a new set of data with desired size.

|  |
| --- |
| def augment\_data(xt,yt,datagen, size):  X = []  Y = []  for x\_batch, y\_batch in datagen.flow(xt, yt):  for x, y in zip(x\_batch,y\_batch):  X.append(x)  Y.append(y)  if len(X) > size:  break  return np.array(X), np.array(Y) |

The images data are flatten then passed into MLP model which consist of 11 layers by the following code.

|  |
| --- |
| model\_mlp = Sequential() model\_mlp.add(Flatten(input\_shape=(150,150,3))) model\_mlp.add(Dense(512,activation="relu")) model\_mlp.add(Dense(256,activation="relu")) model\_mlp.add(Dense(128,activation="relu")) model\_mlp.add(Dense(128,activation="relu")) model\_mlp.add(Dense(256,activation="relu")) model\_mlp.add(Dense(64,activation="relu")) model\_mlp.add(Dense(128,activation="relu")) model\_mlp.add(Dense(64,activation="relu")) model\_mlp.add(Dense(128,activation="relu")) model\_mlp.add(Dense(64,activation="relu")) model\_mlp.add(Dense(1, activation="sigmoid")) |

All hidden layers use rectified linear activation functions with different numbers of nodes. The output layer uses sigmoid as an activation function because of the binary class. The model summary is shown in Figure 1 below.

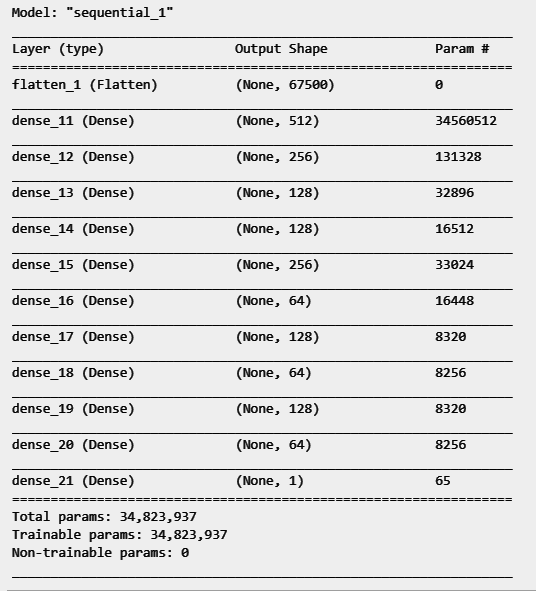


Figure 1: MLP model summary

During training procedure, some functions are utilized in order to facilitate the training process such as save the best model, early stopping and reduce the learning rate.

The code below shows how a function for early stopping is initialized.

|  |
| --- |
| early\_stopping = EarlyStopping(monitor='val\_loss',  patience=10,  verbose=0,  mode='auto') |

The function will monitor validation loss and it will stop the training process if validation loss doesn’t improve for 10 epochs.

The code below shows how a function for adaptive learning rate is initialized.

|  |
| --- |
| reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',  factor=0.1,  patience=5,  verbose=1,  min\_delta=1e-4,  mode='min') |

The function will monitor validation loss and it will reduce the learning rate by factor of 0.1 if the learning range improves less than 1e-4 for 5 epochs.

The model is trained with Adam optimizer and using binary cross entropy as a loss function.

## 2.2 Convolutional Neural Network (CNN)

Convolutional neural network is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers, and is designed to automatically and adaptively learn spatial hierarchies of features through a backpropagation algorithm. CNN always contains two operations, convolution and pooling, the [convolution operation](https://www.sciencedirect.com/topics/computer-science/convolution-operation) using multiple filters is able to extract features from the data set, through which their corresponding spatial information can be preserved. The pooling operation, also called subsampling, is used to reduce the [dimensionality](https://www.sciencedirect.com/topics/computer-science/dimensionality) of feature maps from the convolution operation. The below diagram shows the processing of the image data by a convolutional network. The output can be a single class or a probability of classes that best describes the image.

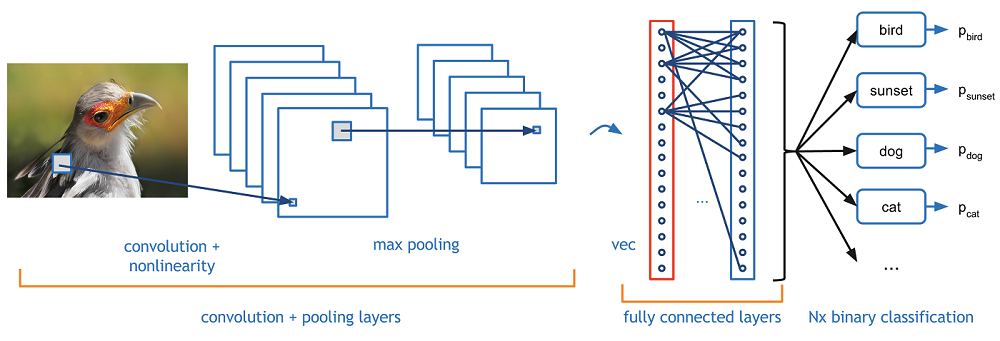


Figure2: Convolutional Neural Network with Fully connected layers

### 2.2.1 CNN Methodology

Initially, our first CNN model was trained with the same dataset as MLP model. However, our first CNN model yielded very good performance. So, we decided to focus on improving CNN models

The model architecture consists of multiple convolutional layers and max pooling. Then smaller representations of images are flattened and passed into multiple dense layers. An example of model architecture is illustrated in Figure 3 below.

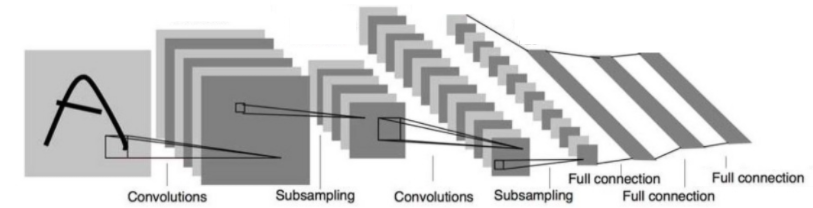


Figure 3: Sample of CNN model architecture

This part we have continued to develop our CNN model from the first presentation. However, there was a problem with data leakage when we trained our model with additional dataset. The problem occurred during up-sampling procedure. We perform upsampling before splitting the testing data which causes some testing data to be in the training set. As a result when we test the model with additional dataset, it gives 95% accuracy as in the first presentation. So, we will focus on hyperparameter optimization methods instead.

Hyperparameter optimization is performed by varying number of convolutional layers, max pooling, kernels, kernels size, stride, dense layers, dense layer nodes and dropout layers. The code below illustrates the parameter optimization process.

|  |
| --- |
| filters = [32,64,128] nodes = [256,512,128] convs = range(1,6) denses = range(4) for c in convs:  for d in denses:  model = Sequential()  n1 = random.choice(filters)  nc.append(str(n1))  model.add(Conv2D(n1, 3, padding="same", activation="relu", input\_shape=(150,150,3)))  model.add(MaxPool2D(pool\_size=(2, 2)))    for i in range(c):  n1 = random.choice(filters)  f1 = random.choice([3,3,3,3,5,5])  st = random.choice([1,2])  info = f'{n1},{f1},{st}'  nc.append(str(info))  model.add(Conv2D(n1, f1,strides=(st, st), padding="same", activation="relu"))  if i < 3:  model.add(MaxPool2D(pool\_size=(2, 2)))  model.add(Flatten())  for i in range(d):  n2 = random.choice(nodes)  nd.append(str(n2))  model.add(Dense(n2,activation="relu"))  if i > 1:  model.add(Dropout(0.2))   model.add(Dense(1, activation="sigmoid")) |

From the code above, it will train 20 models in total with all combinations of 1 tp 5 convolutional layers (exclude input layer) and 1-4 dense layers (exclude flatten and output). Size of convolutional kernels will be randomly picked between 3 and 5 with probability of 0.66 and 0.33 in order. The number of kernels for each convolutional layer will be randomly picked between 32, 64 and 128 with equal probability. The stride of convolution kernels will be randomly picked between 1 and 2. Max pooling layers will be added only at the convolutional layer 1 and 2 to avoid the model running out of pixels. Dense layer nodes will be randomly chosen between 256, 512 and 128 with equal chance. Drop out will be added if there are more than one dense layer to avoid overfitting.

Because there are many models to train and the training data is augmented to 4,000 images, we have to utilize cloud computing to train all the models. In addition, with budget concern, we are not capable of using high performance and high memory cloud computing instances. Our solution is to use a data generator to train our models. The implementation of the data generator is shown below.

|  |
| --- |
| def load\_xy(l\_fname,size,verbose=True):  x = []  y = []  if verbose == True:  for name in tqdm(l\_fname, desc='creating x,y'):  x.append(load\_data(name,size))  y.append(is\_fire(name))  else:  for name in l\_fname:  x.append(load\_data(name,size))  y.append(is\_fire(name))  x = np.array(x)  y = np.array(y)  return x,y |

|  |
| --- |
| def data\_gen(fname,batch\_size=32):  random.shuffle(fname)  while True:  num\_batch = len(fname)//batch\_size  for n in range(num\_batch+1):  x,y = am.load\_xy(fname[n\*batch\_size:(n+1)\*batch\_size],  (150,150), verbose=False)  yield x,y |

The function above will take file names and batch size as arguments. Then it will pass a number of filenames into `load\_xy()` which will return numpy arrays of images as ‘x’ and corresponding labels as ‘y’. This algorithm works very well when we have a very large data set. However, it is slow because it needs to read data for every training step. With our time constraint, instead of reading data for every training step, we read the whole 2,000 images once then augment the data with a datagenerator from keras and pass to the mode. The code is shown below

|  |
| --- |
| model.fit\_generator(generator=datagen\_train.flow(X\_train,y\_train),  steps\_per\_epoch=4000//batch, validation\_data=datagen\_test.flow(X\_test,y\_test),  validation\_steps=len(X\_test)//batch,  callbacks=[tb,model\_save],  epochs=50) |

datagen\_train is an object of keras ImageDatagenerator() which gives the same output as out data\_gen(). We only augmented the model to 4,000 instead of 6,000 as our first CNN model to reduce the training time. The model will be trained for 50 epochs with Adam optimizer with binary cross entropy loss function.

The best parameter will be chosen from the best model to train again with more augmented data and additional dataset.

The additional dataset is imbalanced so we choose to use up-sampling technique to solve this problem. The code is as follows.

|  |  |  |
| --- | --- | --- |
| def up\_sampling(fire\_name, nml\_name):  n\_fire = len(fire\_name)  n\_nml = len(nml\_name)  up = np.abs(n\_fire - n\_nml)  if n\_fire < n\_nml:  up\_time = (up//n\_fire) + 1  up\_remain = up%n\_fire  fire\_name = fire\_name \* up\_time  fire\_name = fire\_name + fire\_name[:up\_remain]  else:  up\_time = (up//n\_nml) + 1  up\_remain = up%n\_nml  nml\_name = nml\_name \* up\_time  nml\_name = nml\_name + nml\_name[:up\_remain]  return fire\_name, nml\_name  During the training process of the final CNN model, some functions are utilized in order to facilitate the training process such as save the best model, early stopping and reduce the learning rate.  The code below shows how a function for early stopping is initialized.   |  | | --- | | early\_stopping = EarlyStopping(monitor='val\_loss',  patience=20,  verbose=0,  mode='auto') |   The function will monitor validation loss and it will stop the training process if validation loss doesn’t improve for 20 epochs.  The code below shows how a function for adaptive learning rate is initialized.   |  | | --- | | reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',  factor=0.1,  patience=5,  verbose=1,  min\_delta=1e-4,  mode='min') |   The function will monitor validation loss and it will reduce the learning rate by factor of 0.1 if the learning range improves less than 1e-4 for 5 epochs. |

## 2.3 Transfer Learning

We are going to use Transfer Learning to re-train a pre-trained MobileNet model. MobileNet is a streamlined architecture that uses depth wise separable convolutions to construct lightweight deep convolutional neural networks and provides an efficient model for mobile and embedded vision applications.

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

We are training only the last layer (final\_training\_ops in the figure below). While all the previous layers retain their already-trained state

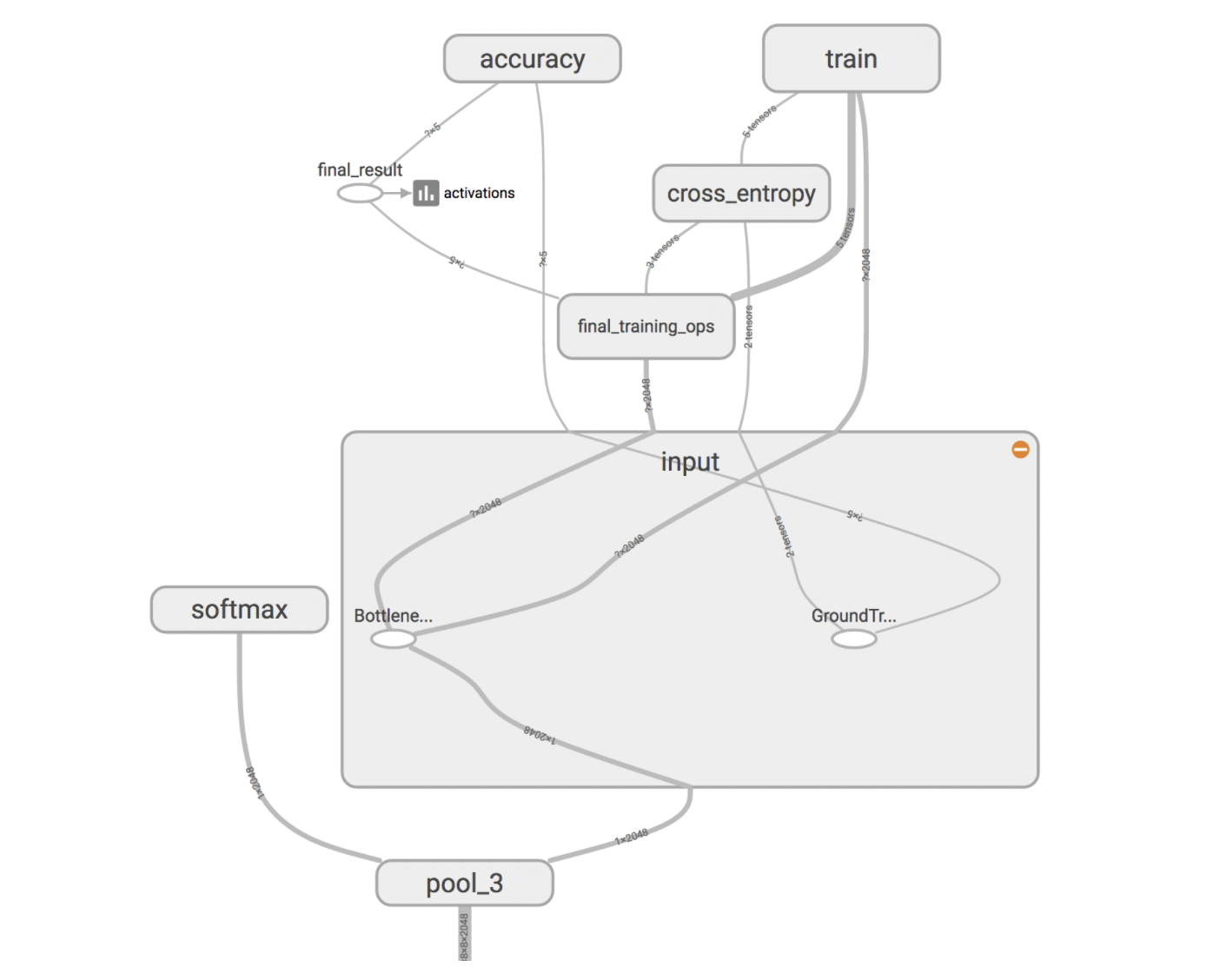


Figure 4: MobileNet Graph

### 2.3.1 Transfer Learning Methodology

In terms of methodology, we are going to use the procedure described in Google [Codelabs](https://kiosk-dot-codelabs-site.appspot.com/codelabs/tensorflow-for-poets/#0) for retraining the MobileNet model.

MobileNet is configurable in 2 ways:

* Input image resolution: 128,160,192, or 224px. Unsurprisingly, feeding in a higher resolution image takes more processing time, but results in better classification accuracy.
* The relative size of the model as a fraction of the largest MobileNet: 1.0, 0.75, 0.50, or 0.25.

For this project we are going to use input resolution of 224px and relative size of 0.5 of the largest MobileNet model. We will set the environment variables as follows:

*IMAGE\_SIZE=224*

*ARCHITECTURE="mobilenet\_0.50\_${IMAGE\_SIZE}"*

We will clone the repository provided in codelabs and add our data(from [this](https://github.com/DeepQuestAI/Fire-Smoke-Dataset) datasource) inside tf\_files/fire\_photos/ . The folder structure should look like this:

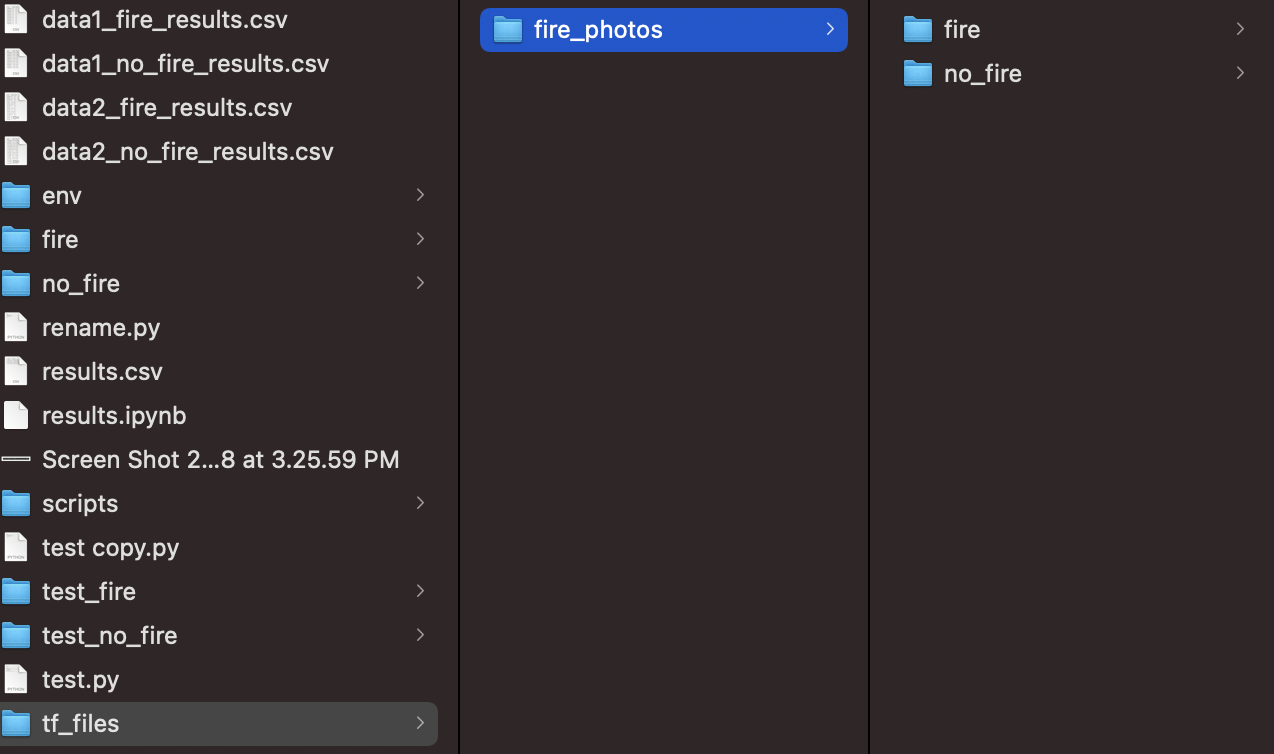


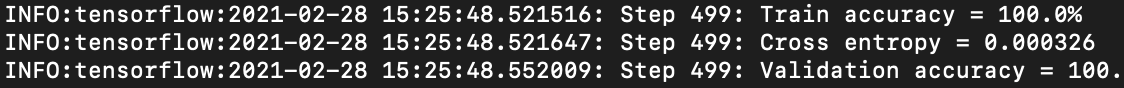
Figure 5: Folder structure

We are keeping 10% data out for testing purposes out of this dataset. Then the next step is to run the training script with the following parameters:

*python -m scripts.retrain \ --bottleneck\_dir=tf\_files/bottlenecks \ --how\_many\_training\_steps=500 \ --model\_dir=tf\_files/models/ \ --summaries\_dir=tf\_files/training\_summaries/"${ARCHITECTURE}" \ --output\_graph=tf\_files/retrained\_graph.pb \ --output\_labels=tf\_files/retrained\_labels.txt \ --architecture="${ARCHITECTURE}" \ --image\_dir=tf\_files/fire\_photos*

In the above command, we are specifying the training steps to be 500, architecture to be replaced with the ARCHITECTURE environment variable that we’ve specified as environment variable and lastly setting the image\_dir to be tf\_files/fire\_photos

After running the command the training will start and it will end after 500 steps.



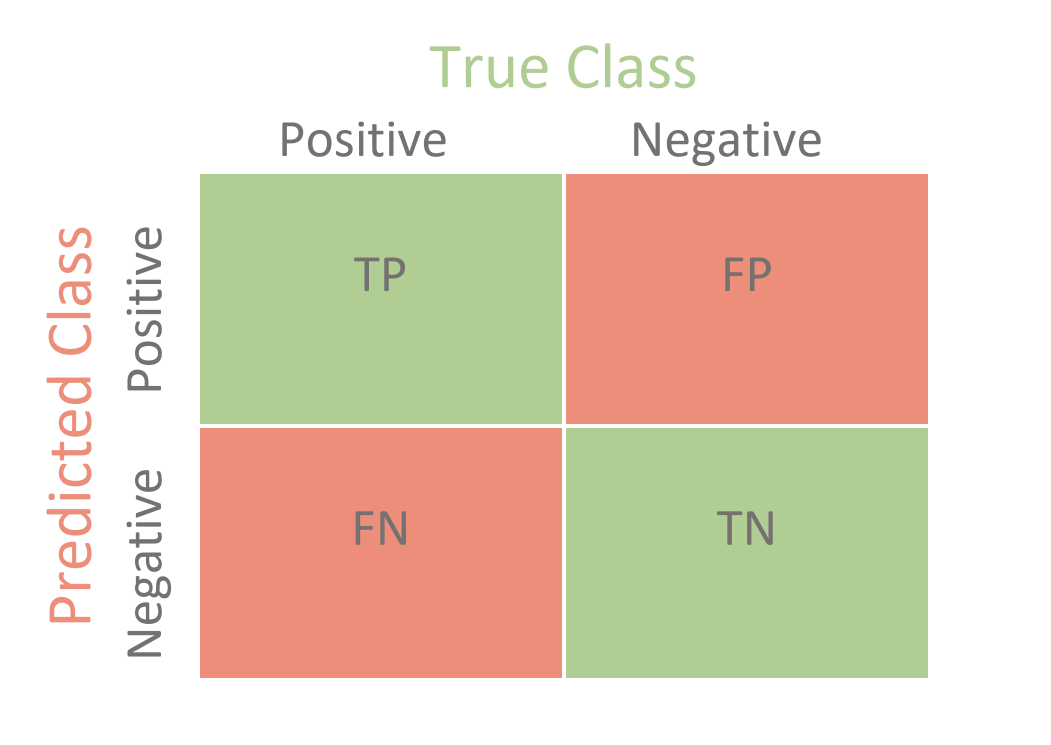
# 3. Evaluation

## 3.1 Evaluation Metric

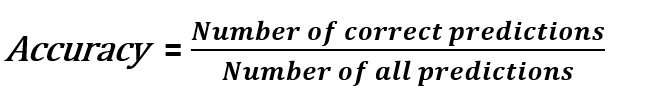
As we are using regression models for modelling, we have several techniques for modelling, such as accuracy, recall, precision, F1. We calculate the accuracy, recall, precision, F1 scores using different formulas. The highest the accuracy, recall, precision, F1 scores, the better the performance of the models. Evaluation of the models are generally done by making up an evaluation matrix. A confusion matrix is not a metric to evaluate a model, but it provides insight into the predictions. We have 4 parameters in the confusion matrix.

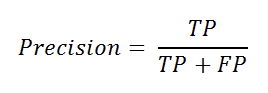
* True positive (TP): Predicting positive class as positive (ok)
* False positive (FP): Predicting negative class as positive (not ok)
* False negative (FN): Predicting positive class as negative (not ok)
* True negative (TN): Predicting negative class as negative (ok)

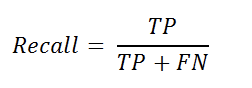
We calculate Precision, recall, Accuracy, F1 scores to evaluate our models. To calculate the scores, we have the formulas respectively.

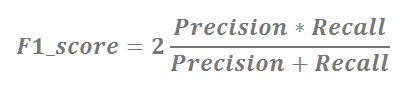


Confusion Matrix









## 3.2 Performance

### 3.2.1 MLP

MLP model stopped training at 27 epochs with around 0.7 validation loss and 0.7 validation accuracy. The plot of loss and accuracy are shown in figures below.

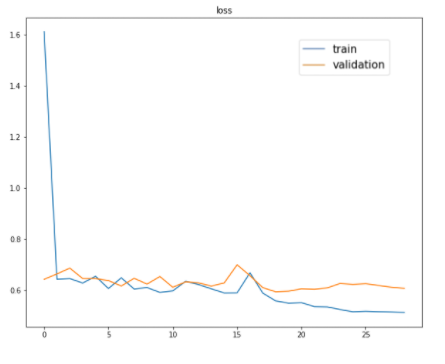
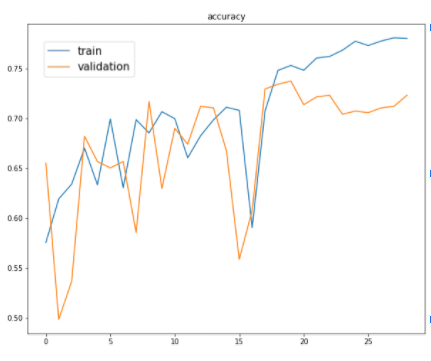


Figure 6: MLP training loss

  
Figure 7: MLP training accuracy

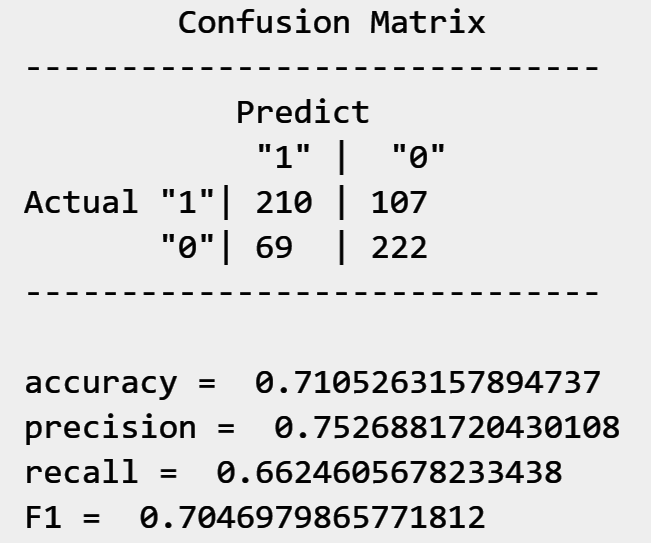


Figure 8: MLP confusion matrix

Figure 8 shows the confusion of MLP models for testing data. As we can see, all the scores are about 70%. With 50% class prior for our training data, the 70% score is almost random guess predictions.

### 3.2.2 CNN

It took about 55 hours to train all 20 models for 50 epochs and the results are as follows.

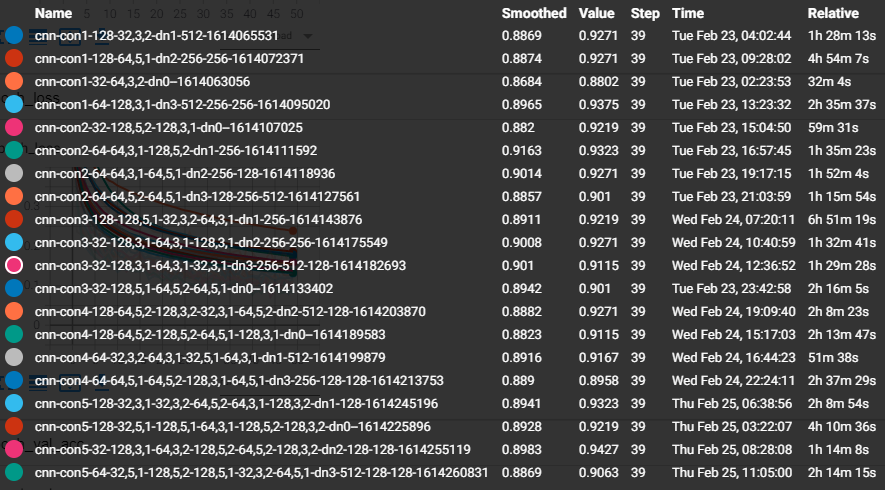


Figure 9: Results summary of all 20 models

Figure 9 illustrates the result of all models at the 39th epoch where “Name” is the name of models. “Smoothed” and “Values” are rolling average and actual accuracy of the models.

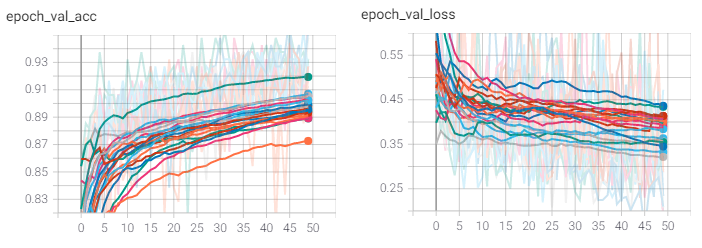


Figure 10: Validation accuracy and loss per epochs

Figure 10: shows validation accuracy and loss from all 20 models. The solid lines represent the smoothed values and the fade lines represent the actual values. As we can see from the graph, after smoothing, the best overall accuracy is the top green lines. In addition the same model also produces good loss. The parameters of the model are as follows.

|  |
| --- |
| layer1: Convolutional, size 3x3, 64 kernels and stride 1, rectified  layer2: Max pooling, size 2x2, stride 2  layer3: Convolutional, size 3x3, 64 kernels and stride 1, rectified  layer4: Max pooling, size 2x2, stride 2  layer5: Convolutional, size 5x5, 128 kernels and stride 2, rectified  layer6: Max pooling, size 2x2, stride 2 layer7: Flatten layer8: Dense, 256 nodes, rectified output: Dense, 1 nodes, sigmoid |

During training, we have used Adam optimizer to help us achieve a high accuracy model. We also tried to prevent overfitting by setting the training to stop when there is no improvement of validation loss. In addition, a dropout layer and regularization are also used. Finally, we use additional adaptive learning rate by monitoring the improvement of validation loss to help the model converge faster. The learning rate will reduce by a factor of 0.1 if the validation loss improves less than 0.0001 for 5 consecutive epochs. The final architecture of our model is as follows.

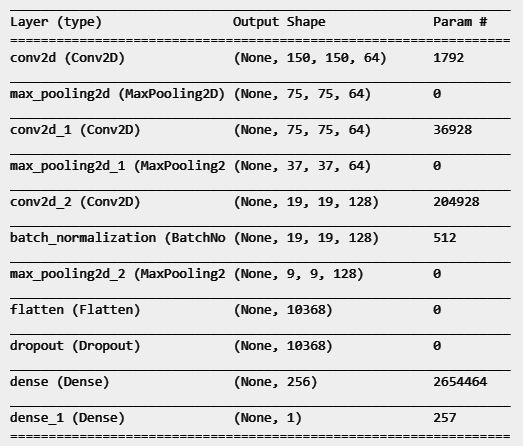
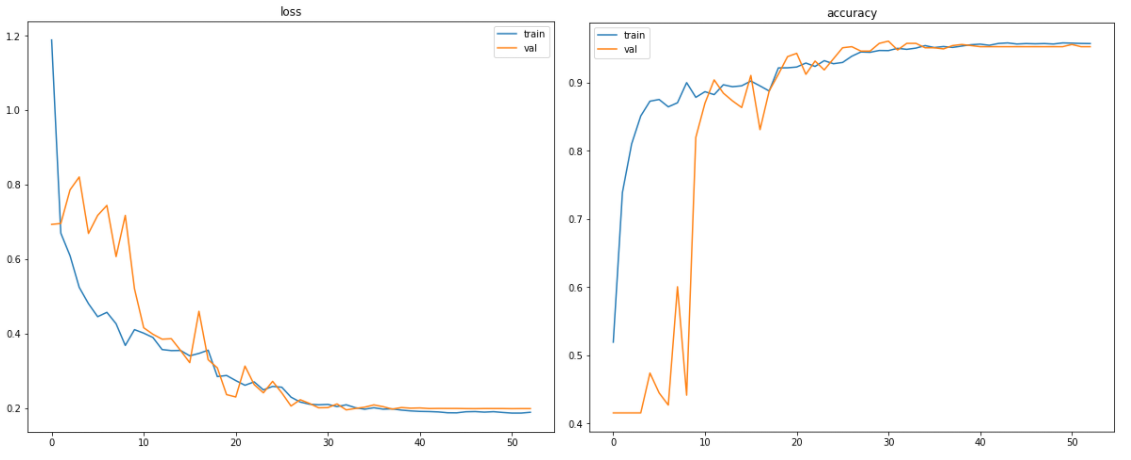
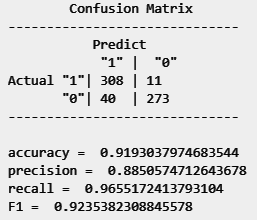


Figure 11: The final CNN model architecture

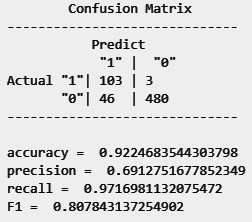
  
Figure 12: Loss(left) and accuracy (right) of the last CNN model

As we can see from Figure 12 left, the model converged around epoch 27th with 0.7 training loss and 0.2 validation loss. On the right, the training accuracy ends at 0.99 and validation accuracy ends at 0.93.

Evaluation on 10% of testing data from 1st dataset:



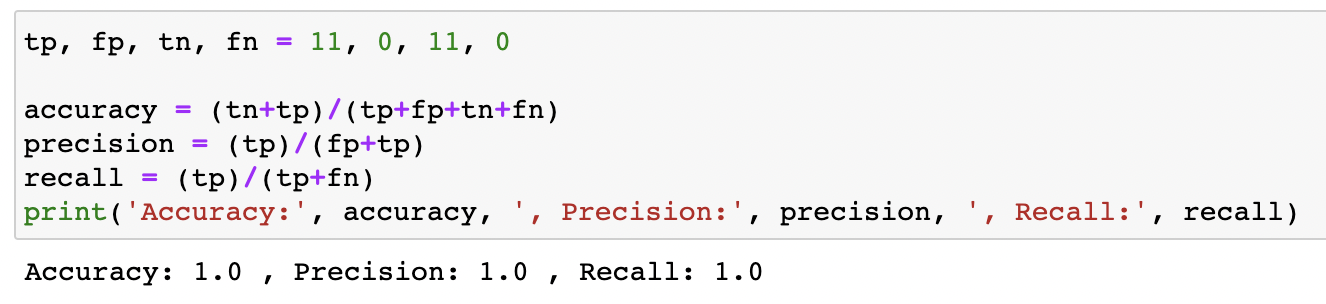
Evaluation on 10% of testing data from new dataset, the 2nd dataset:



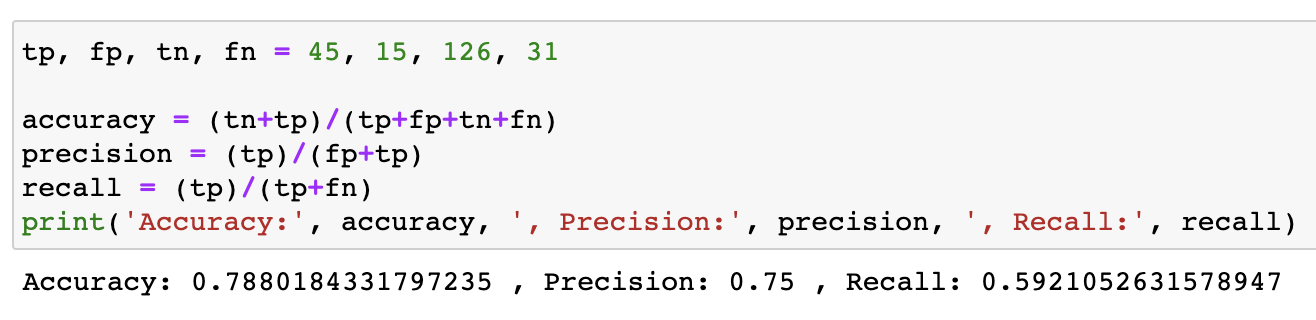
### 3.2.3 Transfer Learning

For the transfer learning model we’re using 2 datasets for testing. 1st will be the 10% of data from the same dataset that we’ve trained this model on and 2nd will be [this](https://github.com/cair/Fire-Detection-Image-Dataset) dataset that we will use just for testing the model on a different type of dataset.

Evaluation on 10% of testing data from 1st dataset:



Evaluation on a completely new dataset, the 2nd dataset:



## 3.3 Limitation

* Our main limitation is the time we have to develop the model. Since images classification with deep learning need
* The computational cost to train images in neural networks is expensive.
* Parameter tuning will be another problem because training each model is time consuming.
* We may need some budgets to use cloud computing services.
* Although we can use a datagenerator to solve limited memory problems. limited time is still our issue.

## 3.4 Models Comparison

The models are evaluated with two datasets. i.e. the main dataset and additional dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accuracy** | **Precision** | **Recall** | **F1** |
| M1: MLP (main) | 0.71 | 0.75 | 0.66 | 0.7 |
| M2: CNN (main) | 0.92 | 0.9 | 0.95 | 0.92 |
| M3: CNN (additional) | 0.94 | 0.73 | 1 | 0.84 |
| M4: Transfer Learning(main) | 1 | 1 | 1 | 1 |
| M5: Transfer Learning (additional) | 0.78 | 0.75 | 0.59 | 0.66 |

Table 1: Models scores comparison

From Table 1, MLP has the worst performance. The reason can be that the data has lost the spatial information of images. That is also the reason why CNN gives very high scores. CNN models give more than 90% accuracy on both dataset. However, testing on the second dataset, the model seems to be better in recall which is 97%. For fire detection, we may want to focus on recall more than precision because false negative may cause a bigger problem. Having said that, we can also reduce the threshold or increase weight to “Fire” class more than “No Fire” inorder to make the model more sensitive to fire images. When we train and test the Transfer Learning model on the main dataset, we get the accuracy of 100% which is the best accuracy among all our models. But when we test this model on an additional dataset then we see a significant drop in accuracy from 100% to 78%.

# 4. Conclusion

We’ve used 3 different types of algorithms, which are MLP, CNN and Transfer Learning, to figure out the best algorithm to detect fire in images. MLP underperformed whereas Transfer Learning overperformed but did not generalize well, so the best model to detect fire in images is CNN written from scratch. CNN gives us a good accuracy of over 90% and does generalize well on additional dataset as well. Furthermore, working with images datasets requires a lot of hardware memory especially when a training dataset is very large, the problem can be solved by using a datagenerator which will fit only a small batch of data in the memory.

# 5. References

<https://www.sciencedirect.com/topics/engineering/convolutional-neural-network>

<https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>

https://kiosk-dot-codelabs-site.appspot.com/codelabs/tensorflow-for-poets/#4

Table of Contributions

The table below identifies contributors to various sections of this document.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Section | Writing | Editing |
| 1 | Predictive Modeling Problem Definition | Chingiz  Harshit | Chingiz  Harshit |
| 2 | Predictive Models | Smith  Himanshu | Smith  Himanshu |
| 3 | Evaluations | Smith | Himanshu |
| 4 | Appendix | Chingiz | Harshit |

Grading

The grade is given on the basis of quality, clarity, presentation, completeness, and writing of each section in the report. This is the grade of the group. Individual grades will be assigned at the end of the term when peer reviews are collected.