



Published in Towards Data Science

You have **2** free member-only stories left this month.

[Sign up for Medium and get an extra one](#)



Dhruvil Shah

[Follow](#)

Jul 5, 2020 · 8 min read · Member-only · Listen

DEEP LEARNING | OPENCV

Early Fire detection system using deep learning and OpenCV

Creating customized InceptionV3 and CNN architectures for indoor and outdoor fire detection.



365



4



Recent advancements in embedded processing have allowed vision-based systems to detect fire using Convolutional Neural Networks during surveillance. In this article, two custom CNN models have been implemented for a cost-effective fire detection CNN architecture for surveillance videos. The first model is a customized basic CNN architecture inspired by AlexNet architecture. We will implement and see its output and limitations and create a customized InceptionV3 model. To balance the efficiency and accuracy, the model is fine-tuned considering the nature of the target problem and fire data. We are going to use three different datasets for training our models. The links for the datasets are available at the end of this article. Let's get to the coding part.

1. Creating the customized CNN architecture

We are going to use TensorFlow API Keras for building our model. Let's first create our ImageDataGenerator for labeling our data. [1] and [2] datasets are used here for training. Finally, we will have 980 images for training and 239 images for validation. We are going to use data augmentation as well.

```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
```

```

TRAINING_DIR = "Train"
training_datagen = ImageDataGenerator(rescale = 1./255,
                                      horizontal_flip=True,
                                      rotation_range=30,
                                      height_shift_range=0.2,
                                      fill_mode='nearest')

VALIDATION_DIR = "Validation"
validation_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                       target_size=(224,224),
                                                       class_mode='categorical',
                                                       batch_size = 64)

validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(224,224),
    class_mode='categorical',
    batch_size= 16)

```

In the above code, 3 data augmentation techniques are applied — horizontal flipping, rotation, and height shifting.

Now, we will create our CNN model. The model contains three Conv2D-MaxPooling2D layers pairs followed by 3 Dense layers. To overcome the problem of overfitting we will also add dropout layers. The last layer is the softmax layer which will give us the probability distribution for both the classes — Fire and Nonfire. *One can also use ‘sigmoid’ activation function at the last layer by changing the number of classes to 1.*

```

from tensorflow.keras.optimizers import Adam
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(96, (11,11), strides=(4,4), activation='relu',

```

```

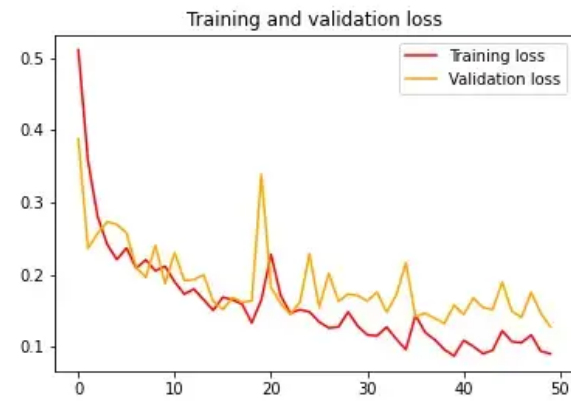
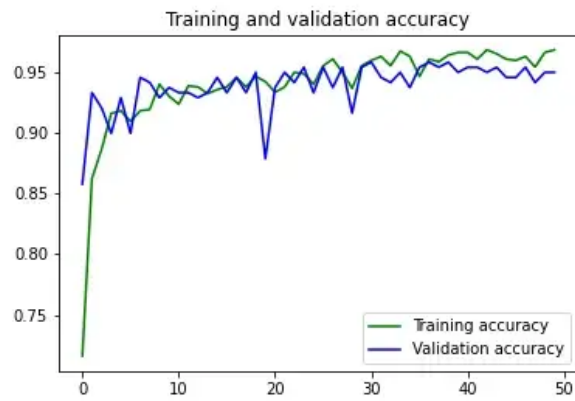
input_shape=(224, 224, 3)), tf.keras.layers.MaxPooling2D(pool_size =
(3,3), strides=(2,2)),
tf.keras.layers.Conv2D(256, (5,5), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),
tf.keras.layers.Conv2D(384, (5,5), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),
tf.keras.layers.Flatten(),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2048, activation='relu'),
tf.keras.layers.Dropout(0.25),
tf.keras.layers.Dense(1024, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2, activation='softmax'))])

model.compile(loss='categorical_crossentropy',
optimizer=Adam(lr=0.0001),
metrics=['acc'])

history = model.fit(
train_generator,
steps_per_epoch = 15,
epochs = 50,
validation_data = validation_generator,
validation_steps = 15
)

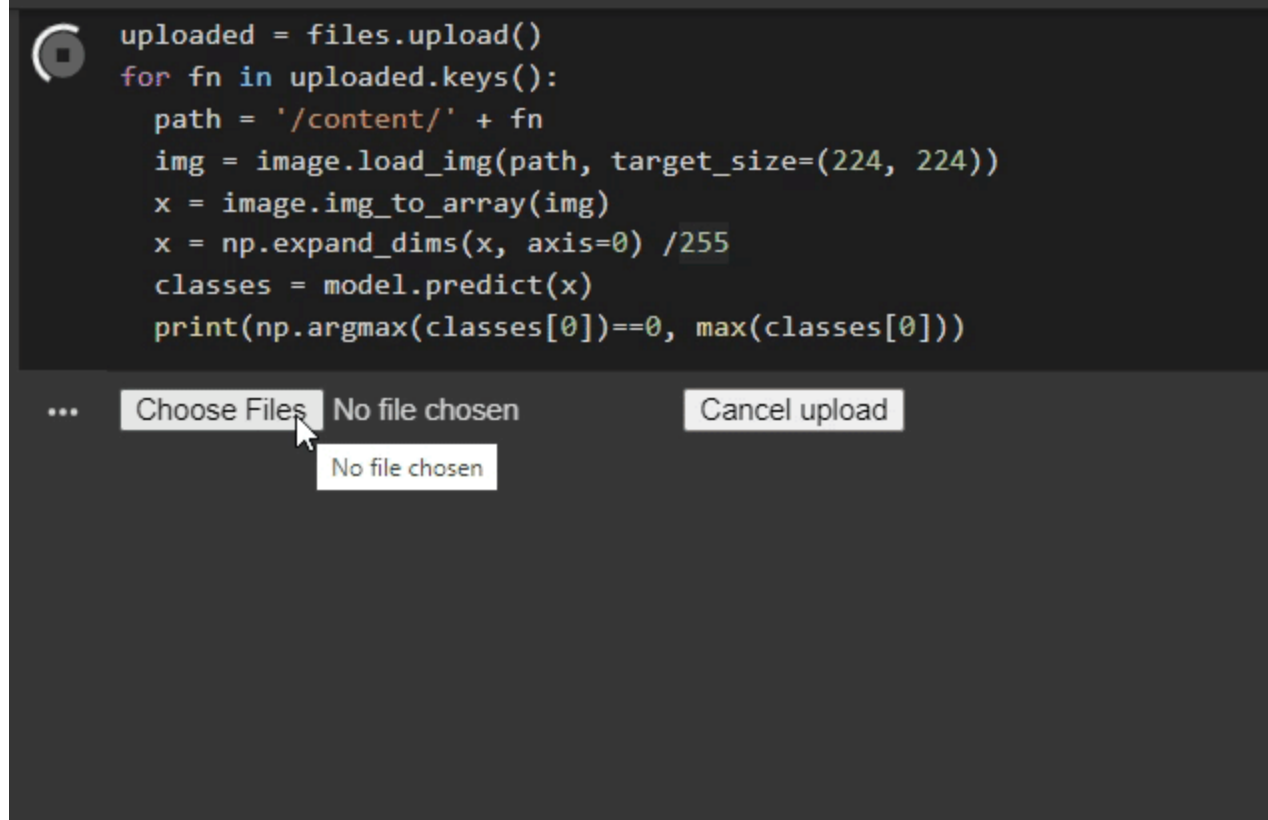
```

We will use Adam as an optimizer with a learning rate of 0.0001. After training for 50 epochs, we get the training accuracy of 96.83 and validation accuracy of 94.98. The training and validation loss is 0.09 and 0.13 respectively.



The training process of our model

Let's test our model for any image and see if it can guess it right. For testing, I have selected 3 images that include a fire-image, a non-fire image, and a photo of me that contains the fire-like colors and shades.



Here, we can see that our above-created model is making a mistake in classifying my image. The model is 52% sure that the image has fire in it. This is because of the dataset it has been trained on. There are very few images in the dataset that teaches a model about indoor fires. So, the model only knows about outdoor fires and hence it errs when given an indoor fire-like shaded image. Another reason is that our model is not a complex one that can learn complex features of fire.

What we will do next is, use a standard InceptionV3 model and customize it. A complex model is capable of learning the complex features from the images.

2. Creating customized InceptionV3 model

We will use a different dataset [3] this time, the one which contains outdoor as well as indoor fire images. I have trained our previous CNN model in this dataset and the result was that it overfitted, as it could not handle this comparatively larger dataset and learn complex features from the images.

Let's start with creating the ImageDataGenerator for our customized InceptionV3. The dataset contains 3 classes but for this article, we will only use 2 classes. It contains 1800 images for training and 200 images for validation. Also, I added 8 images of my living room to add some noise in the dataset.

```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

TRAINING_DIR = "Train"
training_datagen = ImageDataGenerator(rescale=1./255,
zoom_range=0.15,
horizontal_flip=True,
fill_mode='nearest')

VALIDATION_DIR = "/content/FIRE-SMOKE-DATASET/Test"
validation_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_directory(
TRAINING_DIR,
target_size=(224,224),
shuffle = True,
class_mode='categorical',
batch_size = 128)
```



```
validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(224,224),
    class_mode='categorical',
    shuffle = True,
    batch_size= 14)
```

To make training even more accurate we can use data augmentation techniques. In the above code, 2 data augmentation techniques are applied — horizontal flipping and zooming.

Let's import our InceptionV3 model from the Keras API. We will add our layers at the top of the InceptionV3 model as shown below. We will add a global spatial average pooling layer followed by 2 dense layers and 2 dropout layers to ensure that our model does not overfit. At last, we will add a softmax activated dense layer for 2 classes.

Next, we will first train only the layers that we added and are randomly initialized. We will use RMSprop as an optimizer here.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
Input, Dropout

input_tensor = Input(shape=(224, 224, 3))
base_model = InceptionV3(input_tensor=input_tensor,
weights='imagenet', include_top=False)
```

```

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(2048, activation='relu')(x)
x = Dropout(0.25)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(2, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['acc'])

history = model.fit(
train_generator,
steps_per_epoch = 14,
epochs = 20,
validation_data = validation_generator,
validation_steps = 14)

```

After training our top layers for 20 epochs, we will freeze the first 249 layers of the models and train the rest i.e the top 2 inception blocks. Here, we will use SGD as an optimizer with a learning rate of 0.0001.

```

#To train the top 2 inception blocks, freeze the first 249 layers and
unfreeze the rest.

for layer in model.layers[:249]:
    layer.trainable = False

for layer in model.layers[249:]:
    layer.trainable = True

#Recompile the model for these modifications to take effect

```

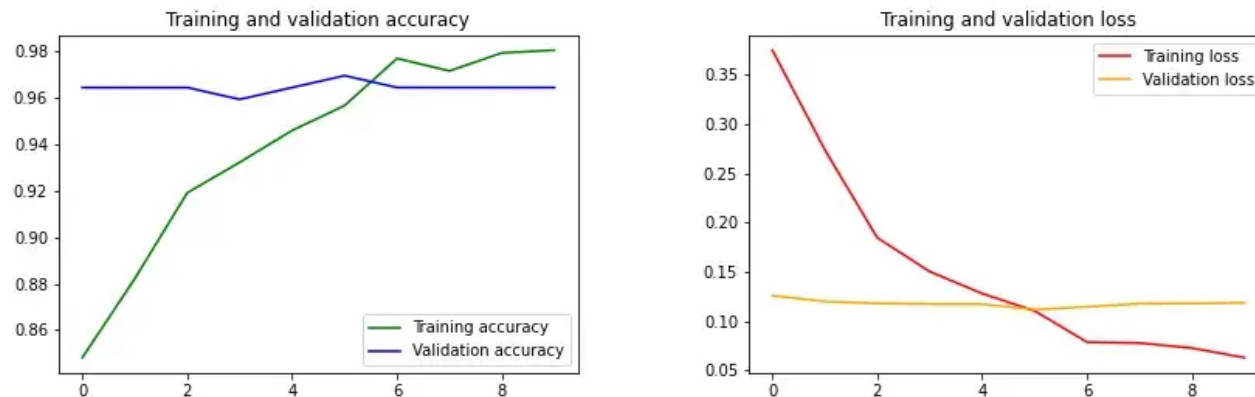
```

from tensorflow.keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
              loss='categorical_crossentropy', metrics=['acc'])

history = model.fit(
    train_generator,
    steps_per_epoch = 14,
    epochs = 10,
    validation_data = validation_generator,
    validation_steps = 14)


```

After training for 10 epochs, we get a training accuracy of 98.04 and a validation accuracy of 96.43. The training and validation losses are 0.063 and 0.118 respectively.



The training process for the above 10 epochs

Let's test our model for the same images and see if it can guess it right.



```
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()
for fn in uploaded.keys():
    path = '/content/' + fn
    img = image.load_img(path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0) / 255
    classes = model.predict(x)
    print(np.argmax(classes[0])==0, max(classes[0]))
```

...

Choose Files

No file chosen

Cancel upload

No file chosen

This time our model can get all three predictions correct. It is 96% sure that my image does not contain any fire. The other two images that I am using for testing are:



Fire-Nonfire images from the datasets referenced below

Real-time Testing:

Now, our model is ready to be tested with real scenarios. Below is the sample code for using OpenCV to access our webcam and predicting whether each frame contains fire or not. If a frame contains fire in it, we want to change the color of that frame to B&W.

```
import cv2
import numpy as np
from PIL import Image
import tensorflow as tf
from keras.preprocessing import image

#Load the saved model
model = tf.keras.models.load_model('InceptionV3.h5')
video = cv2.VideoCapture(0)

while True:
    _, frame = video.read()
```

```

#Convert the captured frame into RGB
im = Image.fromarray(frame, 'RGB')

#Resizing into 224x224 because we trained the model with this image
size.
im = im.resize((224,224))
img_array = image.img_to_array(im)
img_array = np.expand_dims(img_array, axis=0) / 255
probabilities = model.predict(img_array)[0]
#Calling the predict method on model to predict 'fire' on the
image
prediction = np.argmax(probabilities)
#if prediction is 0, which means there is fire in the frame.
if prediction == 0:
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    print(probabilities[prediction])

cv2.imshow("Capturing", frame)
key=cv2.waitKey(1)
if key == ord('q'):
    break

video.release()
cv2.destroyAllWindows()

```

Below is the live output of the above code.



The Github's link for this project is [here](#). You can find the dataset and all of the code above from there. You can connect with me on LinkedIn from [here](#). If any query arises drop a response here or in my LinkedIn inbox.





Conclusion

Using smart cameras you can identify various suspicious incidents such as collisions, medical emergencies, and fires. Of such, fire is the most dangerous abnormal occurrence, because failure to control it at an early stage can lead to huge disasters, leading to human, ecological and economic losses. Inspired by the great potential of CNNs, we can detect fire from images or videos at an early stage. This article shows two custom models for fire detection. Considering the fair fire detection accuracy of the CNN model, it can be of assistance to disaster management teams in managing fire disasters on time, thus preventing huge losses.

Datasets used for this article:

1. Fire Detection Dataset Label 1 indicates Fire while 0 indicates Normal www.kaggle.com	
2. FIRE Dataset	

Outdoor-fire images and non-fire images for computer vision tasks.

www.kaggle.com

3. DeepQuestAI/Fire-Smoke-Dataset

An image dataset for training fire and frame detection AI Fire-Flame-Dataset is a dataset collected to train...

github.com

Deep Learning

Data Science

Python

Artificial Intelligence


Towards Data Science

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

 [Get this newsletter](#)

