FOREST FIRE CLASSIFICATION

Importing Libraries

In [1]:
```
import os, random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.api.models import Sequential
from tensorflow.keras.layers import Input, Convolution2D, MaxPooling2D, Flatten, De
```

Importing Dataset

In [2]:
```
# Get list of file names
_, _, forest_fire_images = next(os.walk('data/train/fire'))
_, _, forest_non_fire_images = next(os.walk('data/train/non_fire'))
```

List of 9 Best Forest Fire Random Images

In [3]:
```
best9_random_fire_imgs = ['F_192.jpg',
                          'F_1069.jpg',
                          'F_9.jpg',
                          'F_1820.jpg',
                          'F_2379.jpg',
                          'F_122.jpg',
                          'F_2017.jpg',
                          'F_2470.jpg',
                          'F_200.jpg']
```

Plotting the images

In [4]:
```
# List of image file names
random_image_files = random.sample(forest_fire_images, 9)
image_files = best9_random_fire_imgs
# Create a figure and get the axes objects
fig = plt.figure(figsize=(10, 10))
axes = [fig.add_subplot(3, 3, i+1) for i in range(9)]

# Loop through the images and display them
for i, ax in enumerate(axes):
    if i < len(image_files):
        img = mpimg.imread('data/train/fire/'+image_files[i])
        ax.imshow(img)
        ax.axis('off')
    else:
        ax.set_visible(False)
```

```python
plt.suptitle('9 Random Forest Fire Images', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

9 Best Forest Non-Fire Random Images

In [5]:
```python
best9_random_non_fire_imgs = ['NF_335.jpg',
                              'NF_262.jpg',
                              'NF_2025.jpg',
                              'NF_1134.jpg',
                              'NF_2380.jpg',
                              'NF_162.jpg',
                              'NF_810.jpg',
                              'NF_1221.jpg',
                              'NF_1139.jpg']
```

Plotting the images

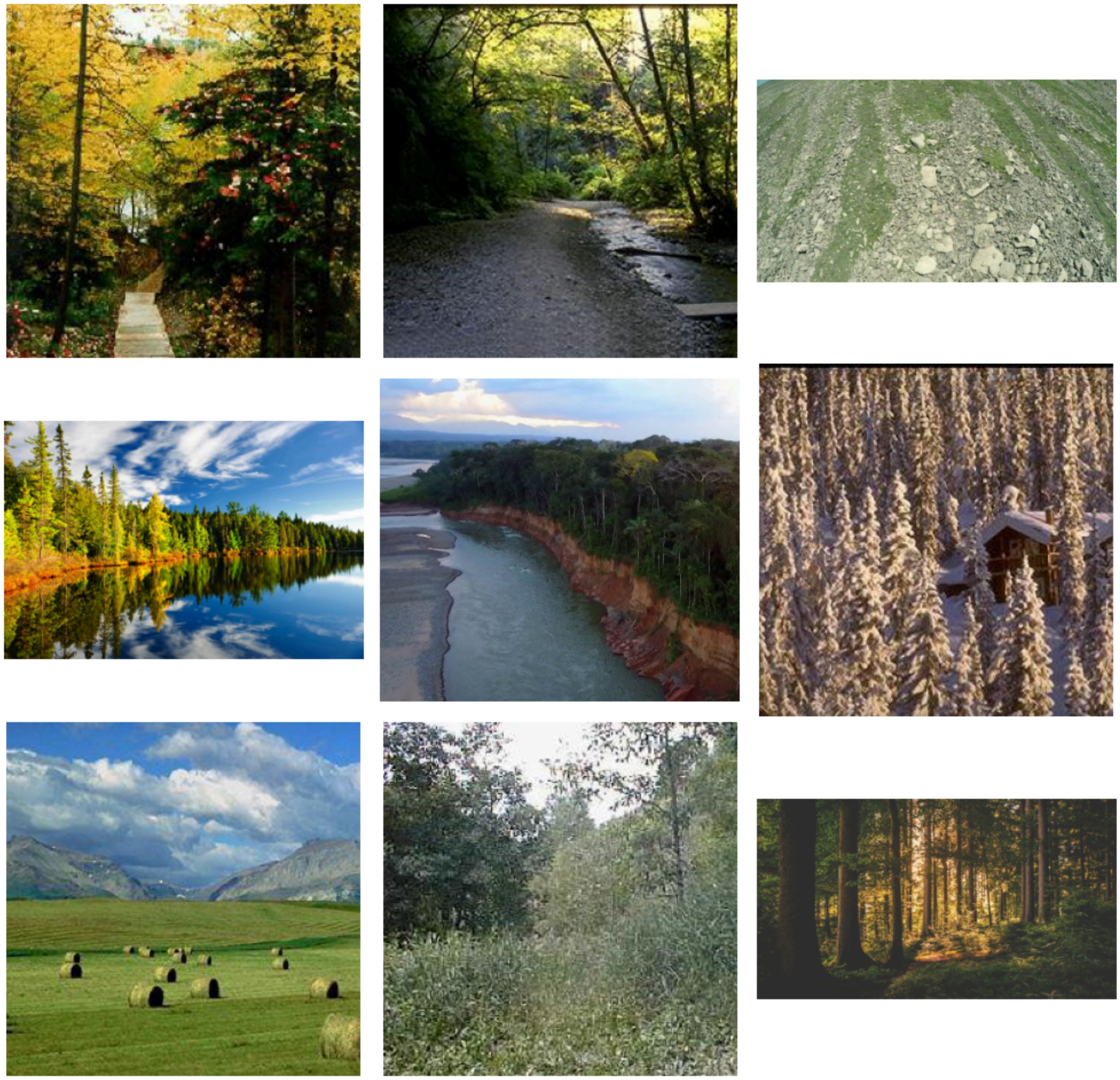
In [6]:

```
# List of image file names
random_image_files = random.sample(forest_non_fire_images, 9)
image_files = best9_random_non_fire_imgs
# image_files = best9_random_fire_imgs
# Create a figure and get the axes objects
fig = plt.figure(figsize=(10, 10))
axes = [fig.add_subplot(3, 3, i+1) for i in range(9)]

# Loop through the images and display them
for i, ax in enumerate(axes):
    if i < len(image_files):
        img = mpimg.imread('data/train/non_fire/'+image_files[i])
        ax.imshow(img)
        ax.axis('off')
    else:
        ax.set_visible(False)

plt.suptitle('9 Random Forest Non Fire Images', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

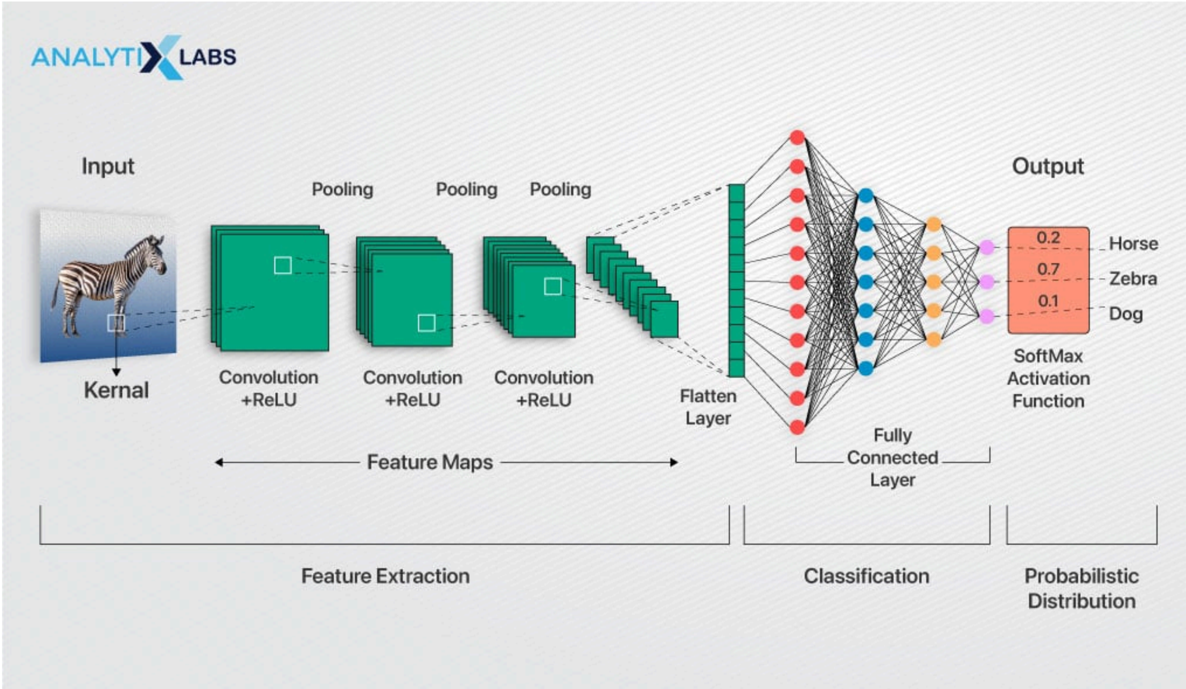
9 Random Forest Non Fire Images



CNN Model

In [7]:
```python
import requests
from IPython.display import Image
url = 'https://www.analytixlabs.co.in/blog/wp-content/uploads/2024/01/7.jpg'
response = requests.get(url)
image_data = response.content
Image(data=image_data)
```

Out[7]:



Constructing CNN Model

In [8]:

```
classifier = Sequential(name='ForestFireClassifierCNN')
classifier.add(Input(shape=(32,32,3)))      # 1st Layer: Input
classifier.add(Convolution2D(filters=32, kernel_size=(3,3), strides=2, padding='sam
classifier.add(MaxPooling2D(pool_size = (2,2)))      # 3rd Layer: MaxPooling2D
classifier.add(Convolution2D(filters=32, kernel_size=(3,3), strides=2, padding='sam
classifier.add(MaxPooling2D(pool_size = (2,2)))      # 5th Layer: MaxPooling2D
classifier.add(Flatten())       # 6th Layer: Flatten
classifier.add(Dense(units = 128, activation = 'relu'))      # 7th Layer: Dense
classifier.add(Dropout(0.5))      # 8th Layer: Dropout
classifier.add(Dense(units = 1, activation = 'sigmoid'))      # 9th Layer: Output
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['ac
classifier.summary()
```

Model: "ForestFireClassifierCNN"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 32)	896
max_pooling2d (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_1 (Conv2D)	(None, 4, 4, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 2, 2, 32)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 128)	16,512
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 26,785 (104.63 KB)

Trainable params: 26,785 (104.63 KB)

Non-trainable params: 0 (0.00 B)

Generating Image Data for Train & Test

In [9]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
training_data_generator = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_
test_data_generator = ImageDataGenerator(rescale=1./255)

train_set = training_data_generator.flow_from_directory('data/train', target_size=(
test_set = test_data_generator.flow_from_directory('data/test', target_size=(32, 32

classifier.fit(train_set, steps_per_epoch=4609//16, epochs=20, validation_data=test
```

```
Found 4609 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
Epoch 1/20
D:\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.p
y:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)`
in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_qu
eue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
288/288 ━━━━━━━━━━━━━━━━━━━━ 74s 242ms/step - accuracy: 0.8057 - loss: 0.4464 - val_
accuracy: 0.8125 - val_loss: 0.4997
Epoch 2/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 396us/step - accuracy: 0.9375 - loss: 0.2028 - val_a
ccuracy: 1.0000 - val_loss: 0.0627
Epoch 3/20
D:\anaconda3\Lib\contextlib.py:158: UserWarning: Your input ran out of data; interru
pting training. Make sure that your dataset or generator can generate at least `step
s_per_epoch * epochs` batches. You may need to use the `.repeat()` function when bui
lding your dataset.
  self.gen.throw(typ, value, traceback)
```

```
288/288 ━━━━━━━━━━━━━━━━━━━━ 32s 108ms/step - accuracy: 0.9013 - loss: 0.2411 - val_accuracy: 0.8125 - val_loss: 0.4366
Epoch 4/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 160us/step - accuracy: 0.8125 - loss: 0.2910 - val_accuracy: 1.0000 - val_loss: 0.0181
Epoch 5/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 31s 103ms/step - accuracy: 0.9208 - loss: 0.2110 - val_accuracy: 0.8125 - val_loss: 0.5744
Epoch 6/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 198us/step - accuracy: 0.8750 - loss: 0.3212 - val_accuracy: 1.0000 - val_loss: 0.2354
Epoch 7/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 30s 101ms/step - accuracy: 0.9262 - loss: 0.1977 - val_accuracy: 0.8750 - val_loss: 0.3885
Epoch 8/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 222us/step - accuracy: 0.9375 - loss: 0.1572 - val_accuracy: 1.0000 - val_loss: 0.0021
Epoch 9/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 30s 99ms/step - accuracy: 0.9292 - loss: 0.1868 - val_accuracy: 0.8750 - val_loss: 0.4014
Epoch 10/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 188us/step - accuracy: 1.0000 - loss: 0.0181 - val_accuracy: 1.0000 - val_loss: 0.0283
Epoch 11/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 33s 112ms/step - accuracy: 0.9390 - loss: 0.1617 - val_accuracy: 0.8750 - val_loss: 0.4098
Epoch 12/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 281us/step - accuracy: 0.9375 - loss: 0.1752 - val_accuracy: 1.0000 - val_loss: 0.1590
Epoch 13/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 31s 106ms/step - accuracy: 0.9384 - loss: 0.1765 - val_accuracy: 0.8958 - val_loss: 0.3991
Epoch 14/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 160us/step - accuracy: 1.0000 - loss: 0.0944 - val_accuracy: 1.0000 - val_loss: 0.5490
Epoch 15/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 31s 105ms/step - accuracy: 0.9410 - loss: 0.1634 - val_accuracy: 0.7917 - val_loss: 0.4984
Epoch 16/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 139us/step - accuracy: 1.0000 - loss: 0.0389 - val_accuracy: 1.0000 - val_loss: 8.3962e-04
Epoch 17/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 31s 105ms/step - accuracy: 0.9463 - loss: 0.1444 - val_accuracy: 0.9375 - val_loss: 0.3267
Epoch 18/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 149us/step - accuracy: 0.9375 - loss: 0.1810 - val_accuracy: 0.5000 - val_loss: 2.7899
Epoch 19/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 31s 103ms/step - accuracy: 0.9394 - loss: 0.1587 - val_accuracy: 0.8542 - val_loss: 0.5386
Epoch 20/20
288/288 ━━━━━━━━━━━━━━━━━━━━ 0s 254us/step - accuracy: 0.9375 - loss: 0.0977 - val_accuracy: 1.0000 - val_loss: 0.0198
```

Out[9]:
```
<keras.src.callbacks.history.History at 0x288e5a43e10>
```

Metrics Evaluation

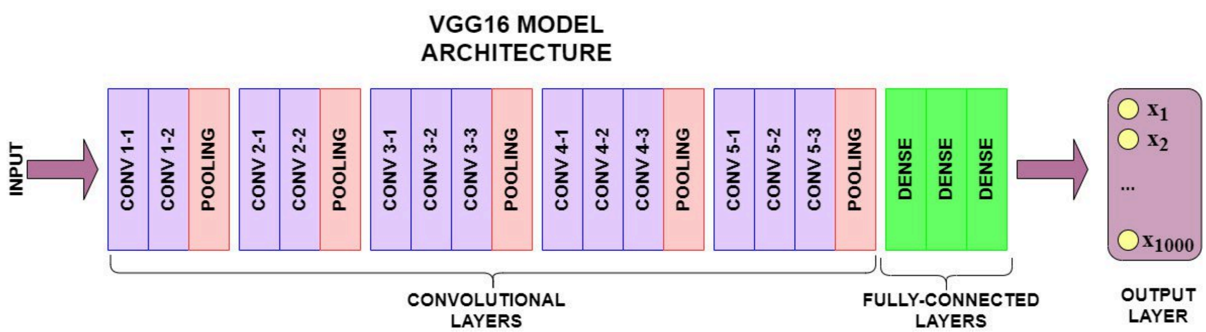
In [10]:
```
score = classifier.evaluate(test_set, steps=len(test_set))
for idx, metric in enumerate(classifier.metrics_names):
    print("{}: {}".format(metric, score[idx]))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 166ms/step - accuracy: 0.8773 - loss: 0.5130
loss: 0.5499006509780884
compile_metrics: 0.8600000143051147
```

Transfer Learning using VGG16

In [11]:
```
url = 'https://storage.googleapis.com/lds-media/images/vgg16-architecture.width-120
response = requests.get(url)
image_data = response.content
Image(data=image_data)
```

Out[11]:



In [12]:
```
from tensorflow.keras.applications.vgg16 import VGG16
```

In [13]:
```
INPUT_SIZE = 128 # Change this to 48 if the code takes too long to run
vgg16 = VGG16(include_top=False, weights='imagenet', input_shape=(INPUT_SIZE,INPUT_
```

> Note that we used `include_top=False when we created a new VGG16 model.
> This

argument tells Keras not to import the fully connected layers at the end of the VGG16 network.

We're now going to freeze the rest of the layers in the VGG16 model, since we're not going to retrain them from scratch

In [14]:
```
for layer in vgg16.layers:
    layer.trainable = False
```

Next, we're going to add a fully connected layer with 1 node right at the end of the neural network. The syntax to do this is slightly different, since the VGG16 model is not a Keras Sequential model that we're used to. In any case, we can add the layers by running the following code:

In [15]:
```
from keras.api.models import Model
input_ = vgg16.input
output_ = vgg16(input_)
```

```python
last_layer = Flatten(name='flatten')(output_)
last_layer = Dense(1, activation='sigmoid')(last_layer)
# model = Model(input=input_, output=last_layer)
model = Model()
```

In [16]:
```python
vgg16.output
```

Out[16]:
```
<KerasTensor shape=(None, 4, 4, 512), dtype=float32, sparse=False, name=keras_tens
or_55>
```

In [16]:
```python

```