Import Library

In [1]:
```
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras import utils  # to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import cv2
```

Define Constants

In [2]:
```
FAST_RUN = False
IMAGE_WIDTH=128
IMAGE_HEIGHT=128
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3
```

Prepare Traning Data

In [3]:
```
filenames = os.listdir("C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Ass
categories = []
for filename in filenames:
    category = filename.split('_')[0]
    if category == 'fire':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
```

In [4]:
```
df.head()
```

Out[4]:

	filename	category
0	fire_0001.jpg	1
1	fire_0002.jpg	1
2	fire_0003.jpg	1
3	fire_0004.jpg	1
4	fire_0005.jpg	1

In [5]:
```
df.tail()
```

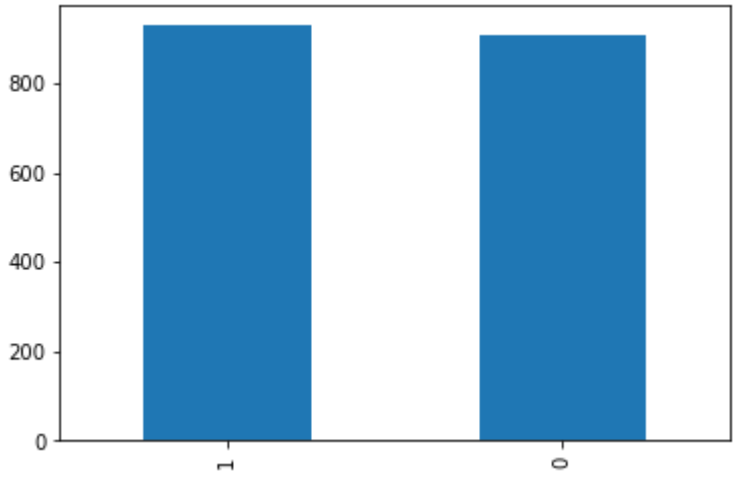
Out[5]:

	filename	category
1828	nofire_334 (95).jpg	0
1829	nofire_334 (96).jpg	0
1830	nofire_334 (97).jpg	0
1831	nofire_334 (98).jpg	0
1832	nofire_334 (99).jpg	0

See Total In count

In [6]:
```
df['category'].value_counts().plot.bar()
```

Out[6]:
```
<AxesSubplot:>
```



In [7]:
```
df['category'].value_counts()
```

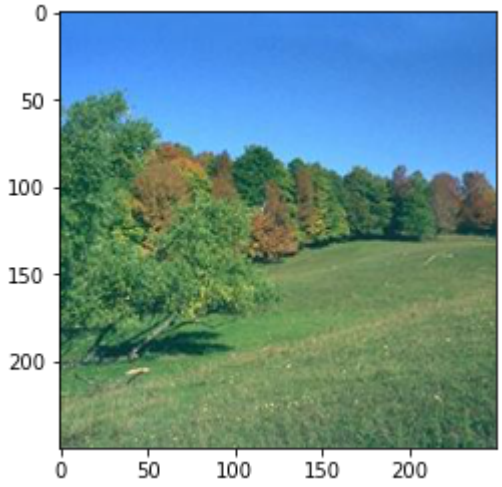
Out[7]:
```
1    928
0    905
Name: category, dtype: int64
```

From our data we have 928 'Fire' and 905 'Nofire'

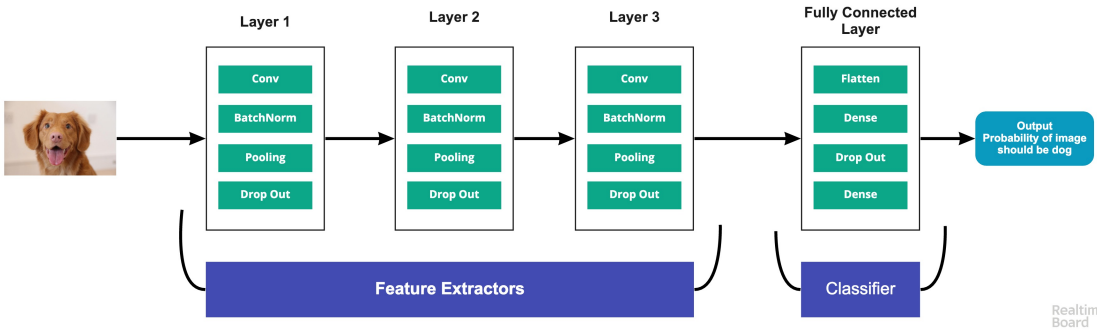
See sample image

In [8]:
```
sample = random.choice(filenames)
image = load_img("C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assigment
plt.imshow(image)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x269de0711c0>
```



Build Model



- **Input Layer**: It represent input image data. It will reshape image into single diminsion array. Example your image is 64x64 = 4096, it will convert to (4096,1) array.
- **Conv Layer**: This layer will extract features from image.
- **Pooling Layer**: This layerreduce the spatial volume of input image after convolution.
- **Fully Connected Layer**: It connect the network from a layer to another layer
- **Output Layer**: It is the predicted values layer.

In [9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense,

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEI
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accur

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 batch_normalization (BatchN  (None, 126, 126, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 dropout (Dropout)           (None, 63, 63, 32)        0

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 batch_normalization_1 (Batc  (None, 61, 61, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 30, 30, 64)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 batch_normalization_2 (Batc  (None, 28, 28, 128)      512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 14, 14, 128)       0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 batch_normalization_3 (Batc  (None, 512)              2048
 hNormalization)

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 12,942,786
Trainable params: 12,941,314
Non-trainable params: 1,472
_________________________________________________________________
```

Callbacks

Early Stop

To prevent over fitting we will stop the learning after 10 epochs and val_loss value not decreased

Learning Rate Reduction

We will reduce the learning rate when then accuracy not increase for 2 steps

In [10]:
```
earlystop = EarlyStopping(patience=10)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

callbacks = [earlystop, learning_rate_reduction]
```

Prepare data

Because we will use image genaretor `with class_mode="categorical"`. We need to convert column category into string. Then imagenerator will convert it one-hot encoding which is good for our classification.

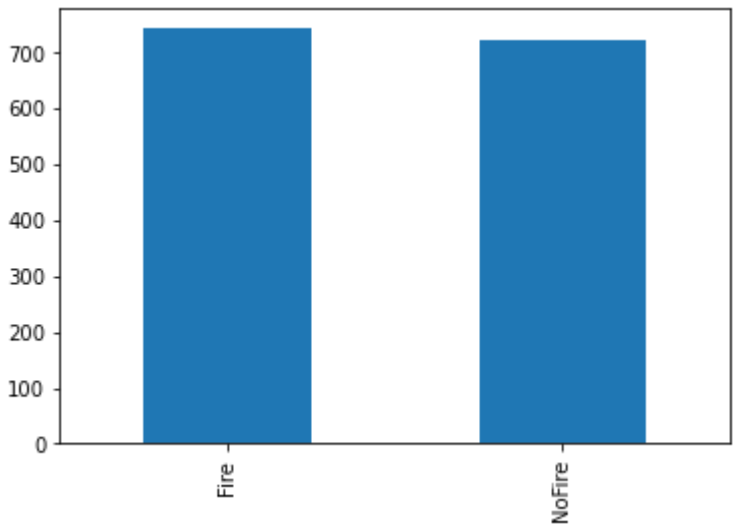
So we will convert 1 to dog and 0 to cat

In [11]:
```
df["category"] = df["category"].replace({0: 'NoFire', 1: 'Fire'})
```

In [12]:
```
train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
```

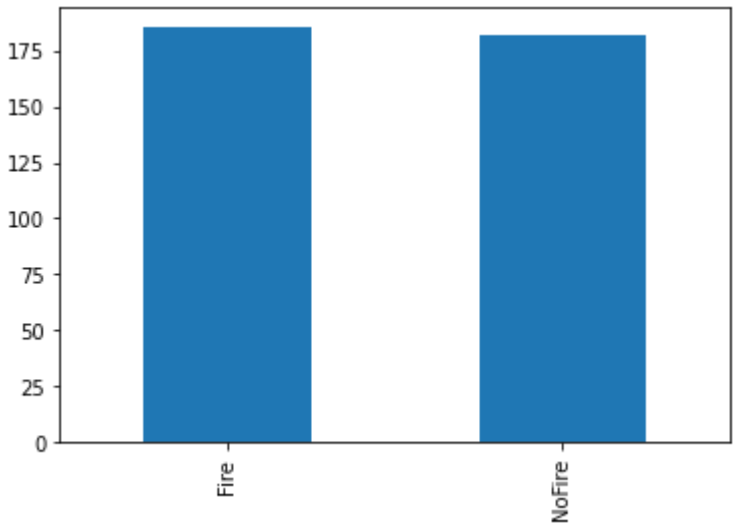
In [13]:
```
train_df['category'].value_counts().plot.bar()
```

Out[13]:
```
<AxesSubplot:>
```



In [14]:
```
validate_df['category'].value_counts().plot.bar()
```

Out[14]: <AxesSubplot:>



In [15]:
```
total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
batch_size=15
```

Traning Generator

In [16]:
```
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assigment/16263469/DS
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

```
Found 1465 validated image filenames belonging to 2 classes.
C:\Users\DELL\AppData\Roaming\Python\Python39\site-packages\keras\preprocessing\im
age.py:1137: UserWarning: Found 1 invalid image filename(s) in x_col="filename". T
hese filename(s) will be ignored.
  warnings.warn(
```

Validation Generator

In [17]:
```
validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_dataframe(
    validate_df,
    "C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assigment/16263469/DS
    x_col='filename',
    y_col='category',
```

```
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

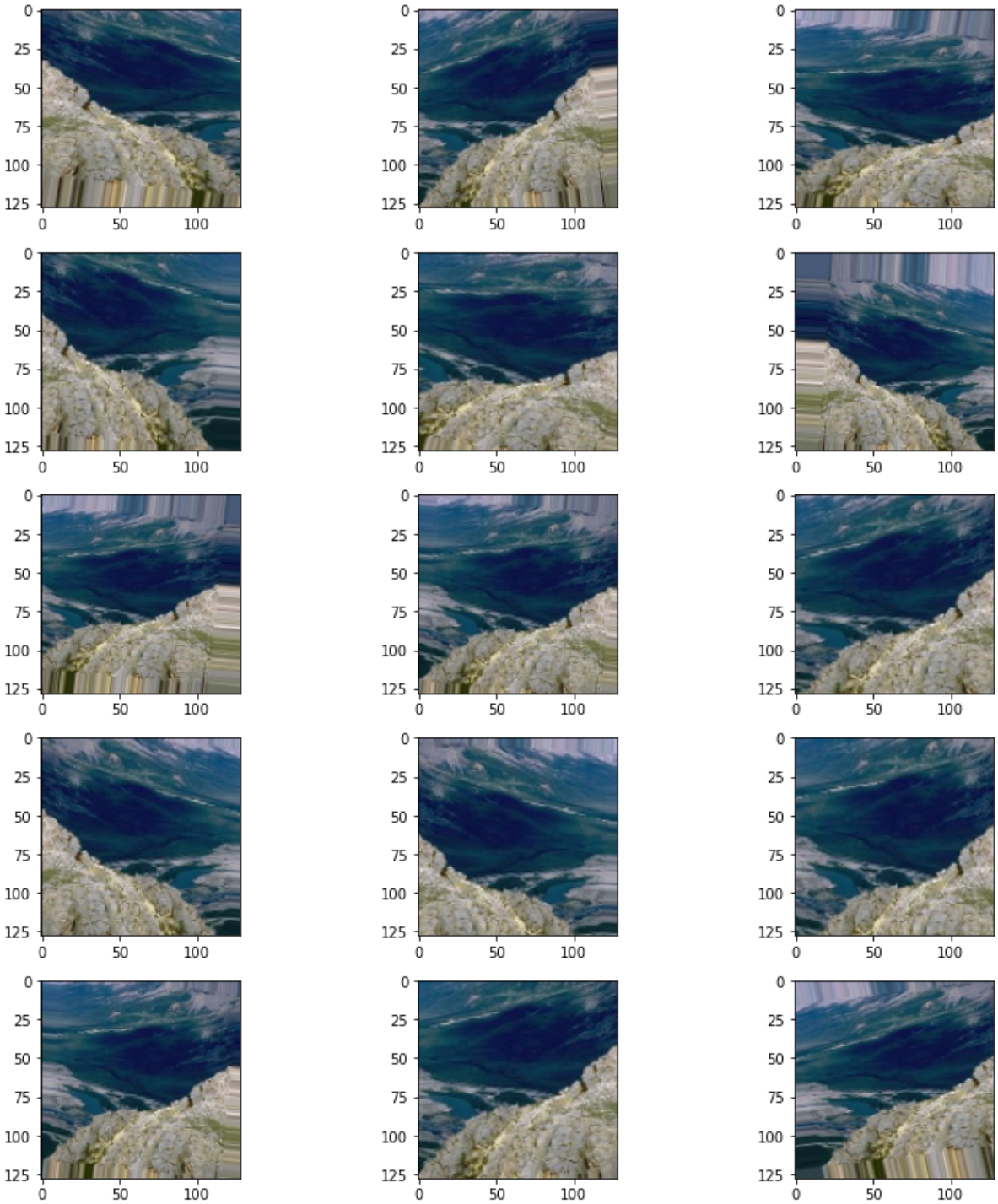
```
Found 367 validated image filenames belonging to 2 classes.
```

See how our generator work

In [18]:
```
example_df = train_df.sample(n=1).reset_index(drop=True)
example_generator = train_datagen.flow_from_dataframe(
    example_df,
    "C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assigment/16263469/DS
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
```

```
Found 1 validated image filenames belonging to 1 classes.
```

In [19]:
```
plt.figure(figsize=(12, 12))
for i in range(0, 15):
    plt.subplot(5, 3, i+1)
    for X_batch, Y_batch in example_generator:
        image = X_batch[0]
        plt.imshow(image)
        break
plt.tight_layout()
plt.show()
```



Seem to be nice

Fit Model

In [20]:
```
epochs=3 if FAST_RUN else 50
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks
)
model.save("model.h5")
```

```
Epoch 1/50
97/97 [==============================] - 65s 652ms/step - loss: 0.4135 - accuracy:
0.8841 - val_loss: 1.2322 - val_accuracy: 0.5056 - lr: 0.0010
Epoch 2/50
97/97 [==============================] - 58s 601ms/step - loss: 0.3783 - accuracy:
0.8848 - val_loss: 1.4283 - val_accuracy: 0.5000 - lr: 0.0010
Epoch 3/50
97/97 [==============================] - 56s 576ms/step - loss: 0.2296 - accuracy:
0.9234 - val_loss: 0.6996 - val_accuracy: 0.6056 - lr: 0.0010
Epoch 4/50
97/97 [==============================] - 56s 579ms/step - loss: 0.2219 - accuracy:
0.9166 - val_loss: 0.1247 - val_accuracy: 0.9694 - lr: 0.0010
Epoch 5/50
97/97 [==============================] - 65s 674ms/step - loss: 0.1823 - accuracy:
0.9297 - val_loss: 0.1326 - val_accuracy: 0.9583 - lr: 0.0010
Epoch 6/50
97/97 [==============================] - ETA: 0s - loss: 0.1690 - accuracy: 0.9441
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
97/97 [==============================] - 60s 615ms/step - loss: 0.1690 - accuracy:
0.9441 - val_loss: 0.2477 - val_accuracy: 0.9306 - lr: 0.0010
Epoch 7/50
97/97 [==============================] - 67s 687ms/step - loss: 0.1390 - accuracy:
0.9455 - val_loss: 0.1045 - val_accuracy: 0.9694 - lr: 5.0000e-04
Epoch 8/50
97/97 [==============================] - 65s 672ms/step - loss: 0.1157 - accuracy:
0.9572 - val_loss: 0.0874 - val_accuracy: 0.9722 - lr: 5.0000e-04
Epoch 9/50
97/97 [==============================] - 75s 768ms/step - loss: 0.1323 - accuracy:
0.9503 - val_loss: 0.1264 - val_accuracy: 0.9583 - lr: 5.0000e-04
Epoch 10/50
97/97 [==============================] - 58s 593ms/step - loss: 0.1183 - accuracy:
0.9641 - val_loss: 0.0775 - val_accuracy: 0.9750 - lr: 5.0000e-04
Epoch 11/50
97/97 [==============================] - 60s 612ms/step - loss: 0.1271 - accuracy:
0.9593 - val_loss: 0.1776 - val_accuracy: 0.9361 - lr: 5.0000e-04
Epoch 12/50
97/97 [==============================] - ETA: 0s - loss: 0.0990 - accuracy: 0.9669
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
97/97 [==============================] - 53s 543ms/step - loss: 0.0990 - accuracy:
0.9669 - val_loss: 0.1152 - val_accuracy: 0.9556 - lr: 5.0000e-04
Epoch 13/50
97/97 [==============================] - 53s 547ms/step - loss: 0.1079 - accuracy:
0.9579 - val_loss: 0.1011 - val_accuracy: 0.9639 - lr: 2.5000e-04
Epoch 14/50
97/97 [==============================] - ETA: 0s - loss: 0.1020 - accuracy: 0.9614
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
97/97 [==============================] - 53s 541ms/step - loss: 0.1020 - accuracy:
0.9614 - val_loss: 0.1137 - val_accuracy: 0.9639 - lr: 2.5000e-04
Epoch 15/50
97/97 [==============================] - 53s 549ms/step - loss: 0.0894 - accuracy:
0.9718 - val_loss: 0.0826 - val_accuracy: 0.9722 - lr: 1.2500e-04
Epoch 16/50
97/97 [==============================] - ETA: 0s - loss: 0.0833 - accuracy: 0.9738
Epoch 16: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
97/97 [==============================] - 53s 545ms/step - loss: 0.0833 - accuracy:
0.9738 - val_loss: 0.1124 - val_accuracy: 0.9583 - lr: 1.2500e-04
Epoch 17/50
97/97 [==============================] - 54s 559ms/step - loss: 0.0849 - accuracy:
0.9710 - val_loss: 0.0670 - val_accuracy: 0.9750 - lr: 6.2500e-05
Epoch 18/50
97/97 [==============================] - ETA: 0s - loss: 0.0677 - accuracy: 0.9759
Epoch 18: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
97/97 [==============================] - 54s 558ms/step - loss: 0.0677 - accuracy:
0.9759 - val_loss: 0.0795 - val_accuracy: 0.9722 - lr: 6.2500e-05
```

```
Epoch 19/50
97/97 [==============================] - 53s 547ms/step - loss: 0.0946 - accuracy:
0.9683 - val_loss: 0.0806 - val_accuracy: 0.9750 - lr: 3.1250e-05
Epoch 20/50
97/97 [==============================] - ETA: 0s - loss: 0.0679 - accuracy: 0.9772
Epoch 20: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
97/97 [==============================] - 53s 545ms/step - loss: 0.0679 - accuracy:
0.9772 - val_loss: 0.0821 - val_accuracy: 0.9750 - lr: 3.1250e-05
Epoch 21/50
97/97 [==============================] - 53s 547ms/step - loss: 0.0555 - accuracy:
0.9786 - val_loss: 0.0775 - val_accuracy: 0.9750 - lr: 1.5625e-05
Epoch 22/50
97/97 [==============================] - ETA: 0s - loss: 0.0864 - accuracy: 0.9752
Epoch 22: ReduceLROnPlateau reducing learning rate to 1e-05.
97/97 [==============================] - 53s 544ms/step - loss: 0.0864 - accuracy:
0.9752 - val_loss: 0.0839 - val_accuracy: 0.9722 - lr: 1.5625e-05
Epoch 23/50
97/97 [==============================] - 54s 556ms/step - loss: 0.0762 - accuracy:
0.9786 - val_loss: 0.0806 - val_accuracy: 0.9750 - lr: 1.0000e-05
Epoch 24/50
97/97 [==============================] - 53s 545ms/step - loss: 0.0816 - accuracy:
0.9752 - val_loss: 0.0845 - val_accuracy: 0.9722 - lr: 1.0000e-05
Epoch 25/50
97/97 [==============================] - 53s 542ms/step - loss: 0.0948 - accuracy:
0.9759 - val_loss: 0.0854 - val_accuracy: 0.9722 - lr: 1.0000e-05
Epoch 26/50
97/97 [==============================] - 53s 549ms/step - loss: 0.0712 - accuracy:
0.9766 - val_loss: 0.0843 - val_accuracy: 0.9722 - lr: 1.0000e-05
Epoch 27/50
97/97 [==============================] - 52s 537ms/step - loss: 0.0882 - accuracy:
0.9731 - val_loss: 0.0870 - val_accuracy: 0.9722 - lr: 1.0000e-05
```

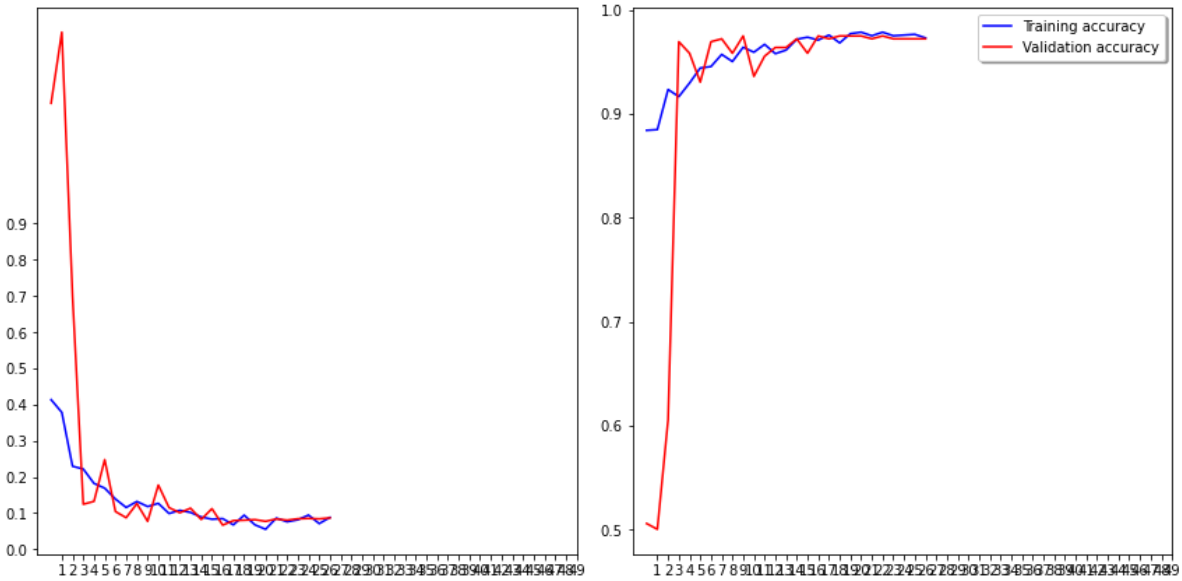
Visualize Training

In [21]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation loss")
ax1.set_xticks(np.arange(1, epochs, 1))
ax1.set_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")
ax2.plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
ax2.set_xticks(np.arange(1, epochs, 1))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```



In [ ]:
```
for i in range(10):
    all_test_images = os.listdir("C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Le
    random_image = random.choice(all_test_images)
    img = cv2.imread(f'C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assi
    img = cv2.resize(img,(IMAGE_HEIGHT,IMAGE_WIDTH))

    org = img.copy()
    img = img.reshape(1,128,128,3)

    pred = model.predict(img)
    print(['NoFire','Fire'][int(pred[0][0])])
    cv2.imshow('Live predictions',org)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
1/1 [==============================] - 0s 157ms/step
Fire
```

In [ ]: