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 == 'dog':
        categories.append(1)
    else:
        categories.append(0)

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

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

Out[4]:

	filename	category
0	cat.0.jpg	0
1	cat.1.jpg	0
2	cat.10.jpg	0
3	cat.100.jpg	0
4	cat.101.jpg	0

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

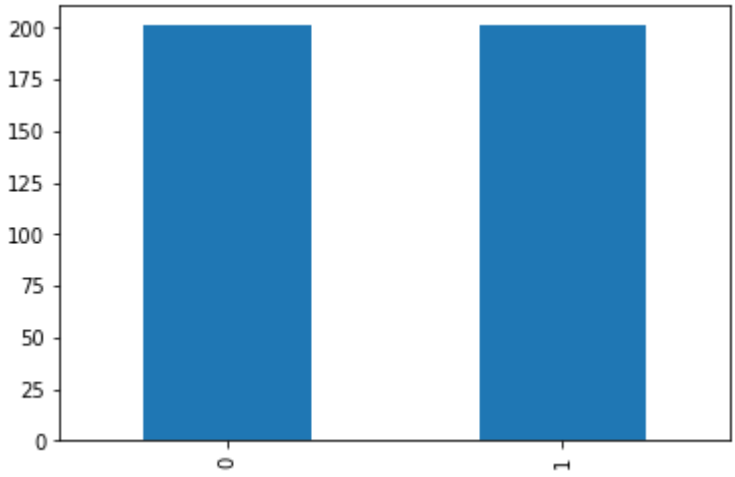
Out[5]:

	filename	category
397	dog.95.jpg	1
398	dog.96.jpg	1
399	dog.97.jpg	1
400	dog.98.jpg	1
401	dog.99.jpg	1

See Total In count

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

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

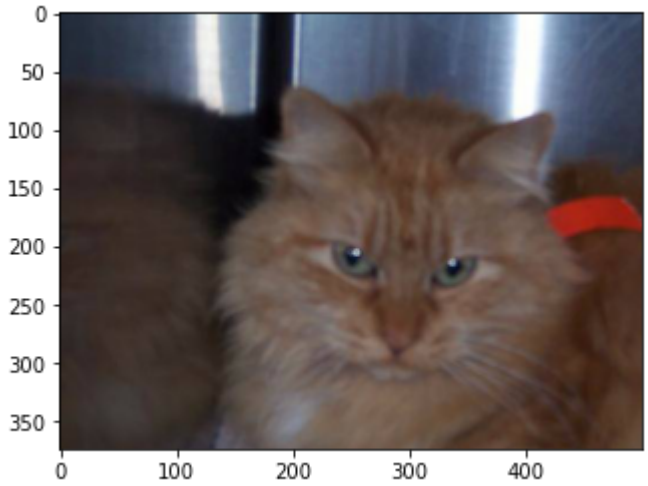


From our data we have 12000 cats and 12000 dogs

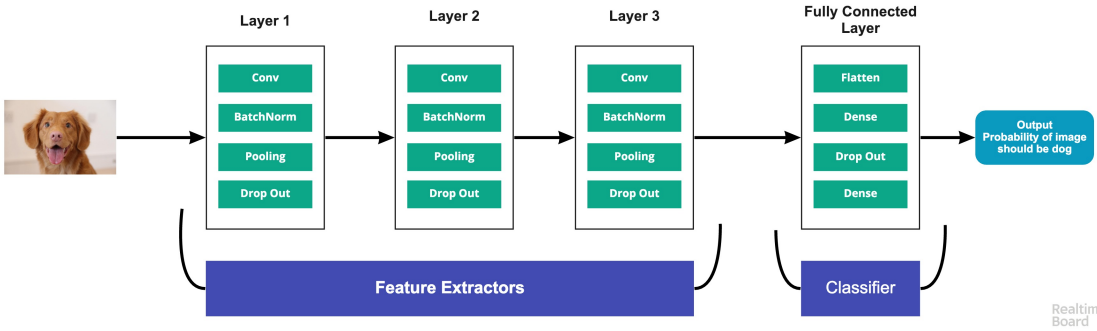
See sample image

In [7]:
```
sample = random.choice(filenames)
image = load_img("C:/Users/DELL/Desktop/Mukund/DATA SCIENCE/Deep Learning/Assigment
plt.imshow(image)
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x2e9602cc8b0>
```



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 [8]:
```
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 [9]:
```
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.

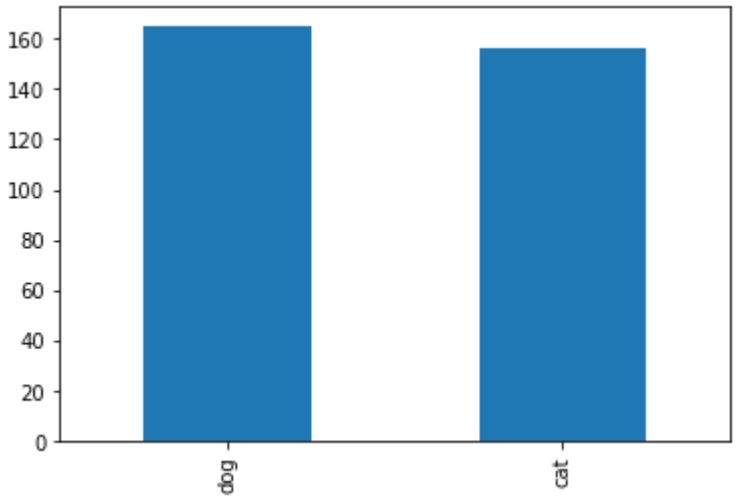
So we will convert 1 to dog and 0 to cat

In [10]:
```
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
```

In [11]:
```
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)
```

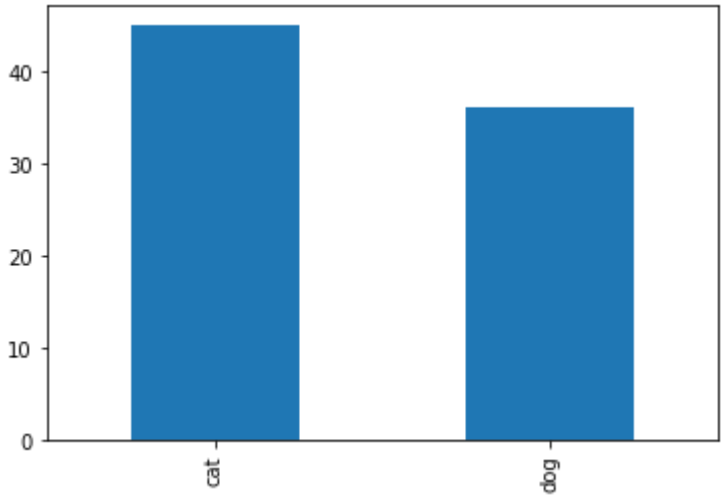
In [12]:
```
train_df['category'].value_counts().plot.bar()
```

Out[12]:
```
<AxesSubplot:>
```



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

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



In [14]:
```
total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
batch_size=15
```

Traning Generator

In [15]:
```
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/16262924/DS
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

```
Found 321 validated image filenames belonging to 2 classes.
```

Validation Generator

In [16]:
```
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/16262924/DS
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

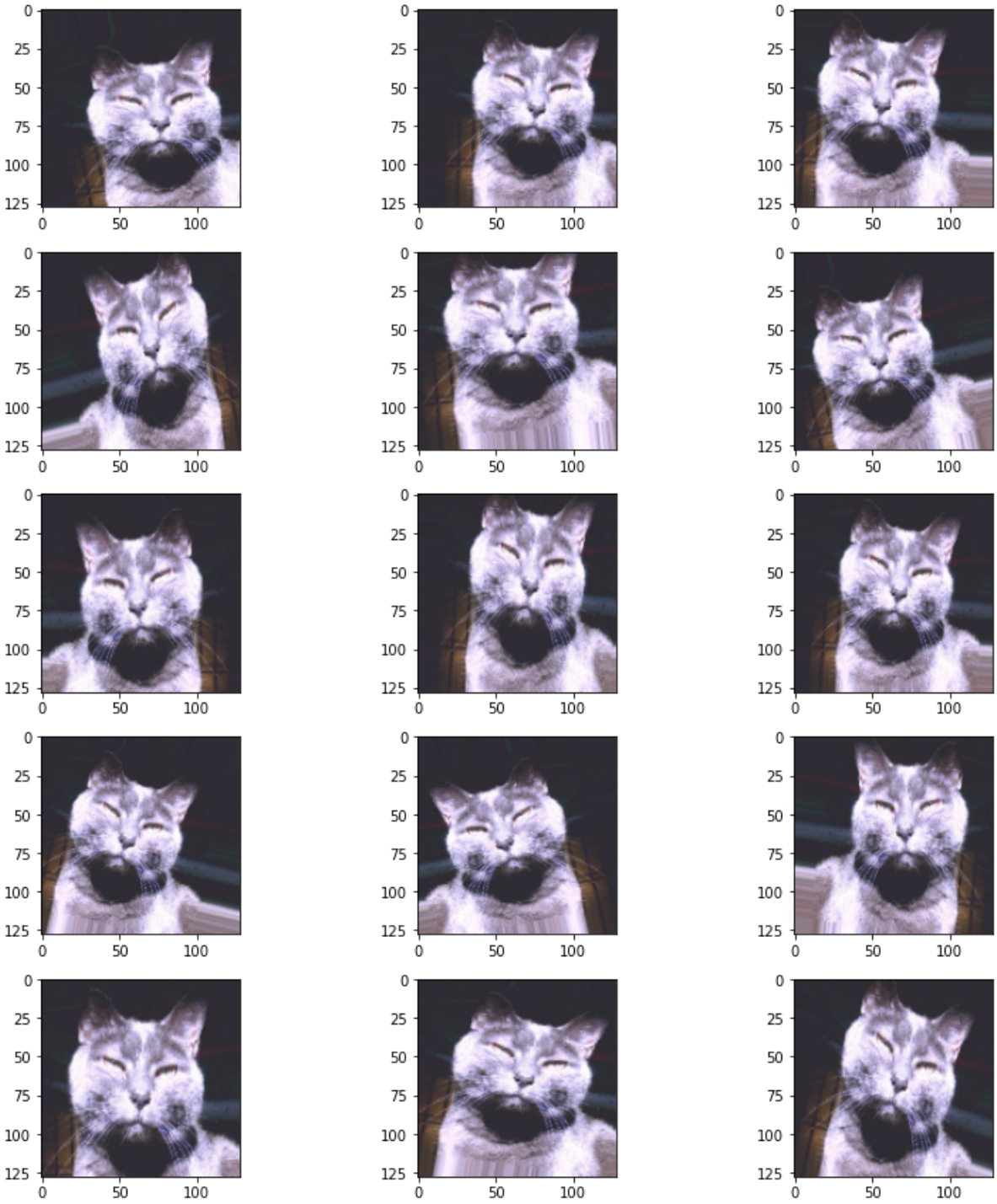
```
Found 81 validated image filenames belonging to 2 classes.
```

See how our generator work

In [17]:
```
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/16262924/DS
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
```

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

In [18]:
```
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 [19]:
```
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
21/21 [==============================] - 16s 698ms/step - loss: 1.9752 - accuracy:
0.5784 - val_loss: 1.4809 - val_accuracy: 0.5600 - lr: 0.0010
Epoch 2/50
21/21 [==============================] - 12s 569ms/step - loss: 1.2010 - accuracy:
0.5850 - val_loss: 1.6615 - val_accuracy: 0.5467 - lr: 0.0010
Epoch 3/50
21/21 [==============================] - ETA: 0s - loss: 1.0907 - accuracy: 0.6373
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
21/21 [==============================] - 12s 575ms/step - loss: 1.0907 - accuracy:
0.6373 - val_loss: 1.9526 - val_accuracy: 0.4267 - lr: 0.0010
Epoch 4/50
21/21 [==============================] - 12s 557ms/step - loss: 0.9290 - accuracy:
0.5980 - val_loss: 1.1686 - val_accuracy: 0.5467 - lr: 5.0000e-04
Epoch 5/50
21/21 [==============================] - ETA: 0s - loss: 0.8793 - accuracy: 0.6209
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
21/21 [==============================] - 13s 618ms/step - loss: 0.8793 - accuracy:
0.6209 - val_loss: 3.4507 - val_accuracy: 0.5467 - lr: 5.0000e-04
Epoch 6/50
21/21 [==============================] - 11s 526ms/step - loss: 0.7362 - accuracy:
0.6765 - val_loss: 4.4798 - val_accuracy: 0.5467 - lr: 2.5000e-04
Epoch 7/50
21/21 [==============================] - ETA: 0s - loss: 0.7198 - accuracy: 0.7026
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
21/21 [==============================] - 11s 514ms/step - loss: 0.7198 - accuracy:
0.7026 - val_loss: 4.0511 - val_accuracy: 0.5600 - lr: 2.5000e-04
Epoch 8/50
21/21 [==============================] - 11s 525ms/step - loss: 0.7573 - accuracy:
0.6569 - val_loss: 4.0508 - val_accuracy: 0.5600 - lr: 1.2500e-04
Epoch 9/50
21/21 [==============================] - ETA: 0s - loss: 0.6312 - accuracy: 0.7222
Epoch 9: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
21/21 [==============================] - 11s 536ms/step - loss: 0.6312 - accuracy:
0.7222 - val_loss: 4.1143 - val_accuracy: 0.5600 - lr: 1.2500e-04
Epoch 10/50
21/21 [==============================] - 12s 557ms/step - loss: 0.5183 - accuracy:
0.7549 - val_loss: 4.5486 - val_accuracy: 0.5467 - lr: 6.2500e-05
Epoch 11/50
21/21 [==============================] - ETA: 0s - loss: 0.6980 - accuracy: 0.6928
Epoch 11: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
21/21 [==============================] - 13s 608ms/step - loss: 0.6980 - accuracy:
0.6928 - val_loss: 4.6933 - val_accuracy: 0.5333 - lr: 6.2500e-05
Epoch 12/50
21/21 [==============================] - 12s 551ms/step - loss: 0.6374 - accuracy:
0.7288 - val_loss: 4.7438 - val_accuracy: 0.5467 - lr: 3.1250e-05
Epoch 13/50
21/21 [==============================] - ETA: 0s - loss: 0.6491 - accuracy: 0.7353
Epoch 13: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
21/21 [==============================] - 11s 527ms/step - loss: 0.6491 - accuracy:
0.7353 - val_loss: 4.3648 - val_accuracy: 0.5600 - lr: 3.1250e-05
Epoch 14/50
21/21 [==============================] - 11s 519ms/step - loss: 0.5580 - accuracy:
0.7255 - val_loss: 4.4046 - val_accuracy: 0.5600 - lr: 1.5625e-05
```

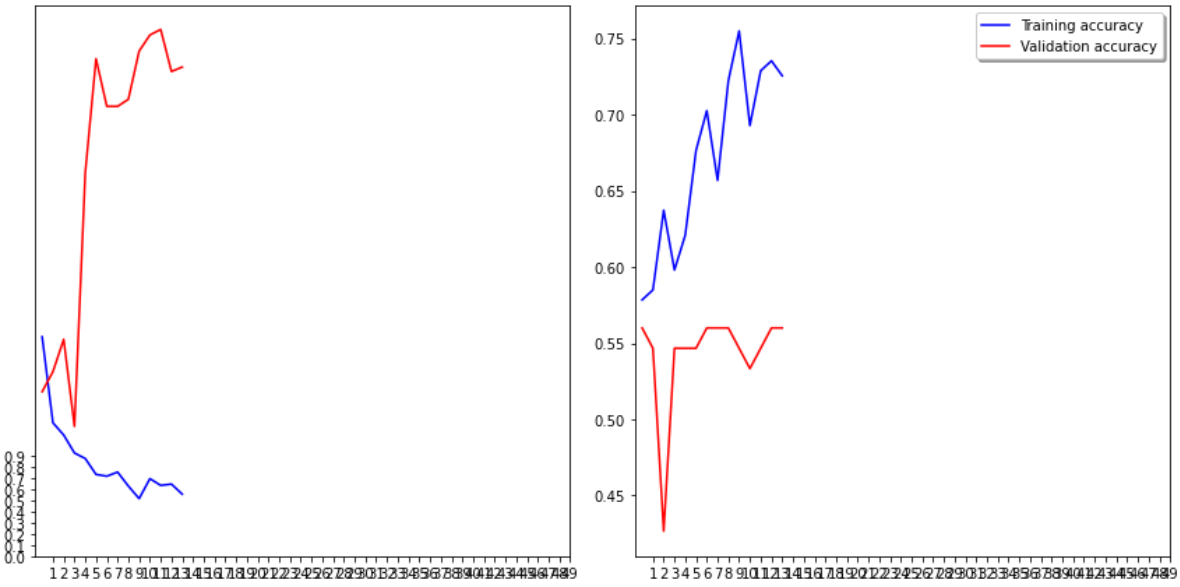
Visualize Training

In [20]:

```
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(['cat','dog'][int(pred[0][0])])
    cv2.imshow('Live predictions',org)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
1/1 [==============================] - 0s 183ms/step
dog
1/1 [==============================] - 0s 55ms/step
dog
1/1 [==============================] - 0s 31ms/step
dog
1/1 [==============================] - 0s 31ms/step
dog
1/1 [==============================] - 0s 47ms/step
dog
1/1 [==============================] - 0s 31ms/step
dog
1/1 [==============================] - 0s 45ms/step
dog
1/1 [==============================] - 0s 34ms/step
dog
```