Load Cifar10 image dataset and display the first 25 images to verify that loading was successful

In [1]:
```
from __future__ import absolute_import, division, print_function, unicode_lite
rals

import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.lo
ad_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

```
<Figure size 1000x1000 with 25 Axes>
```

Create our Convolutional Neural Network. Note that in Tensorflow, we create our network as a static graph and then run our data through it. We cannot change the structure of the network dynamically and the structure of the network does not depend on the data distribution.

In [2]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3
)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

View our Network structure and the number of trainable parameters on each layer.

In [3]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928     
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 64)                65600     
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650       
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Run the network with our data.

In [4]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

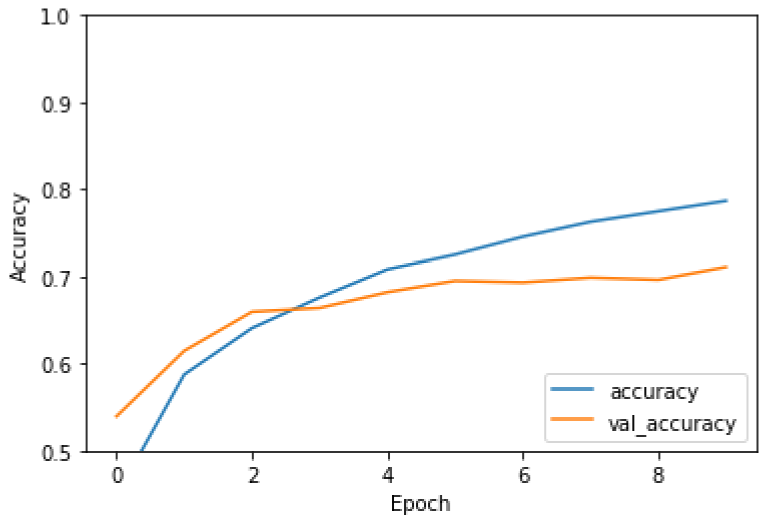
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 17s 349us/sample - loss: 1.515
7 - accuracy: 0.4481 - val_loss: 1.2657 - val_accuracy: 0.5394
Epoch 2/10
50000/50000 [==============================] - 12s 246us/sample - loss: 1.162
9 - accuracy: 0.5875 - val_loss: 1.0821 - val_accuracy: 0.6147
Epoch 3/10
50000/50000 [==============================] - 12s 245us/sample - loss: 1.017
2 - accuracy: 0.6408 - val_loss: 0.9818 - val_accuracy: 0.6596
Epoch 4/10
50000/50000 [==============================] - 12s 238us/sample - loss: 0.916
9 - accuracy: 0.6760 - val_loss: 0.9709 - val_accuracy: 0.6640
Epoch 5/10
50000/50000 [==============================] - 12s 239us/sample - loss: 0.832
9 - accuracy: 0.7081 - val_loss: 0.9215 - val_accuracy: 0.6820
Epoch 6/10
50000/50000 [==============================] - 12s 240us/sample - loss: 0.776
8 - accuracy: 0.7251 - val_loss: 0.8841 - val_accuracy: 0.6950
Epoch 7/10
50000/50000 [==============================] - 14s 285us/sample - loss: 0.720
5 - accuracy: 0.7455 - val_loss: 0.8969 - val_accuracy: 0.6931
Epoch 8/10
50000/50000 [==============================] - 13s 251us/sample - loss: 0.677
1 - accuracy: 0.7626 - val_loss: 0.8878 - val_accuracy: 0.6987
Epoch 9/10
50000/50000 [==============================] - 12s 243us/sample - loss: 0.639
0 - accuracy: 0.7746 - val_loss: 0.9177 - val_accuracy: 0.6962
Epoch 10/10
50000/50000 [==============================] - 12s 245us/sample - loss: 0.601
7 - accuracy: 0.7866 - val_loss: 0.8685 - val_accuracy: 0.7109
```

Plot the train and validation accuracy

In [6]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
10000/1 - 1s - loss: 0.8628 - accuracy: 0.7109
```



Print the final test accuracy

In [7]:
```
print(test_acc)
```

```
0.7109
```

In [8]:
```
#Create the confusion matrix
```

```
  File "<ipython-input-8-ec6ee38ea864>", line 1
    Create the confusion matrix
             ^
SyntaxError: invalid syntax
```

In [9]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

pred=model.predict(train_images)
y_pred = np.argmax(pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(train_labels, y_pred))
#target_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
#                'dog', 'frog', 'horse', 'ship', 'truck']
```

```
Confusion Matrix
[[4161   85  260   41   54    9   12   28  270   80]
 [  32 4707   11   10   12    8   13    3  122   82]
 [ 162   24 3999  138  259  103  129  116   57   13]
 [  59   34  360 3314  241  488  215  162   83   44]
 [  82   12  342  177 3899   72   71  309   25   11]
 [  20   23  290  870  168 3195  110  267   33   24]
 [  19   29  241  168  188   48 4235   20   36   16]
 [  29   15  128   99   99   88   17 4481   17   27]
 [ 135   42   35   24    6    6   12    7 4707   26]
 [  96  461   42   28    7    7   17   32  121 4189]]
```

In [ ]:
```python
#Class 1 mistaken for class 3 the most
#Class 2 mistaken for class 9 the most
#Class 3 mistaken for class 5 the most -> Implies Class 3's features aren't di
stinguishable enough
#Class 4 mistken for class 6 the most
#Class 5 mistaken for class 3 the most
#Class 6 mistaken for class 4 the most
#Class 7 mistaken for class 3 the most
#Class 8 mistaken for class 3 the most
#Class 9 mistaken for class 1 the most
#Class 10 mistaken for class 2 the most
```