In [1]:

```
import tensorflow as tf

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

Téléchargez et préparez le jeu de données CIFAR10

L'ensemble de données CIFAR10 contient 60 000 images couleur dans 10 classes, avec 6 000 images dans chaque classe. L'ensemble de données est divisé en 50 000 images d'entraînement et 10 000 images de test. Les classes sont mutuellement exclusives et il n'y a pas de chevauchement entre elles.

In [2]:

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar1
0.load_data()

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

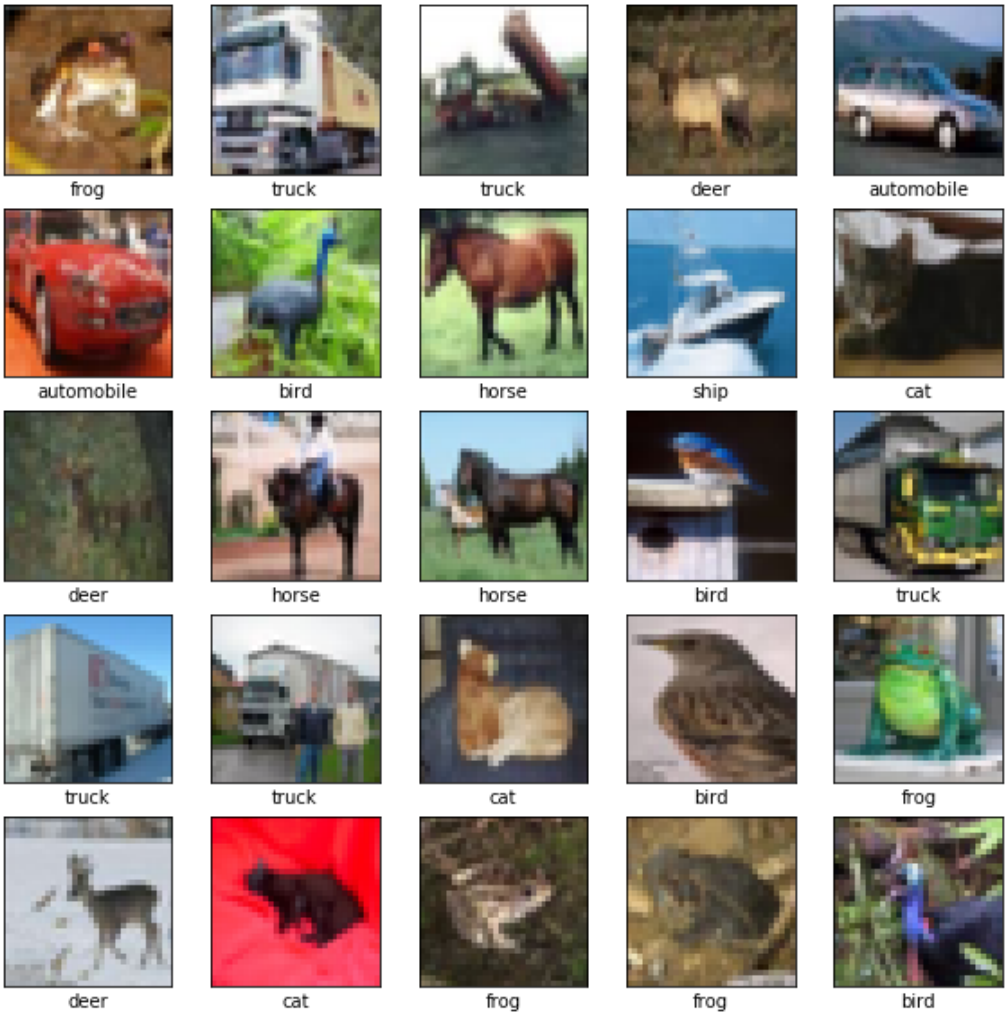
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-
10-python.tar.gz
170500096/170498071 [==============================] - 482s 3
us/step
```

Vérifier les données

In [4]:

```
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()
```



Créer la base convolutive

Les 6 lignes de code ci-dessous définissent la base convolutive en utilisant un modèle commun: une pile de couches Conv2D et MaxPooling2D . En entrée, un CNN prend des tenseurs de forme (image_height, image_width, color_channels), en ignorant la taille du lot. Si vous êtes nouveau dans ces dimensions, color_channels fait référence à (R, V, B). Dans cet exemple, vous allez configurer notre CNN pour traiter les entrées de forme (32, 32, 3), qui est le format des images CIFAR. Vous pouvez le faire en passant l'argument input_shape à notre première couche.

In [5]:

```
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'))
```

Présentons l'architecture de notre modèle jusqu'à présent.

In [6]:

```
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
=================================================================
====
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
____
```

Ci-dessus, vous pouvez voir que la sortie de chaque couche Conv2D et MaxPooling2D est un tenseur 3D de forme (hauteur, largeur, canaux). Les dimensions de largeur et de hauteur ont tendance à rétrécir au fur et à mesure que vous approfondissez le réseau. Le nombre de canaux de sortie pour chaque couche Conv2D est contrôlé par le premier argument (par exemple, 32 ou 64). En règle générale, à mesure que la largeur et la hauteur diminuent, vous pouvez vous permettre (de manière informatique) d'ajouter plus de canaux de sortie dans chaque couche Conv2D.

Ajouter des couches denses sur le dessus

Pour compléter notre modèle, vous allez alimenter le dernier tenseur de sortie de la base convolutionnelle (de forme (4, 4, 64)) dans une ou plusieurs couches denses pour effectuer la classification. Les couches denses prennent des vecteurs en entrée (qui sont 1D), tandis que la sortie de courant est un tenseur 3D. Tout d'abord, vous allez aplatir (ou dérouler) la sortie 3D en 1D, puis ajouter une ou plusieurs couches Dense sur le dessus. CIFAR a 10 classes de sortie, vous utilisez donc une couche Dense finale avec 10 sorties.

In [7]:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Voici l'architecture complète de notre modèle.

In [8]:

```
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
_________________________________________________________________
```

Comme vous pouvez le voir, nos sorties (4, 4, 64) ont été aplaties en vecteurs de forme (1024) avant de passer par deux couches denses.

Compiler et former le modèle

In [9]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logi
ts=True),
              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 [==============================] - 71s 1ms/sample
- loss: 1.5023 - accuracy: 0.4529 - val_loss: 1.2779 - val_ac
curacy: 0.5325

Epoch 2/10
50000/50000 [==============================] - 52s 1ms/sample
- loss: 1.1312 - accuracy: 0.6015 - val_loss: 1.0767 - val_ac
curacy: 0.6234
Epoch 3/10
50000/50000 [==============================] - 57s 1ms/sample
- loss: 0.9859 - accuracy: 0.6551 - val_loss: 0.9762 - val_ac
curacy: 0.6554
Epoch 4/10
50000/50000 [==============================] - 57s 1ms/sample
- loss: 0.8918 - accuracy: 0.6866 - val_loss: 0.9292 - val_ac
curacy: 0.6730
Epoch 5/10
50000/50000 [==============================] - 56s 1ms/sample
- loss: 0.8175 - accuracy: 0.7144 - val_loss: 0.8925 - val_ac
curacy: 0.6904
Epoch 6/10
50000/50000 [==============================] - 54s 1ms/sample
- loss: 0.7573 - accuracy: 0.7345 - val_loss: 0.8423 - val_ac
curacy: 0.7093
Epoch 7/10
50000/50000 [==============================] - 53s 1ms/sample
- loss: 0.7083 - accuracy: 0.7503 - val_loss: 0.8479 - val_ac
curacy: 0.7093
Epoch 8/10
50000/50000 [==============================] - 55s 1ms/sample
- loss: 0.6607 - accuracy: 0.7666 - val_loss: 0.8528 - val_ac
curacy: 0.7071
Epoch 9/10
50000/50000 [==============================] - 53s 1ms/sample
- loss: 0.6224 - accuracy: 0.7817 - val_loss: 0.8416 - val_ac
curacy: 0.7185
Epoch 10/10
50000/50000 [==============================] - 52s 1ms/sample
- loss: 0.5892 - accuracy: 0.7910 - val_loss: 0.8541 - val_ac
curacy: 0.7154
```

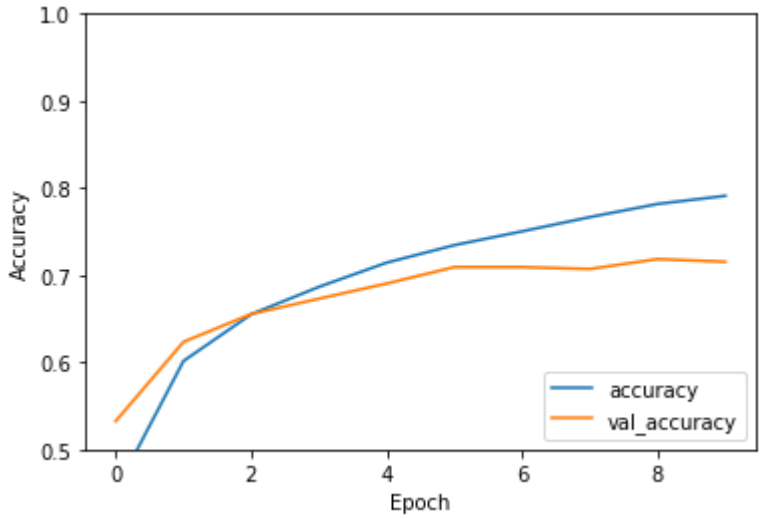
Évaluer le modèle

In [10]:

```
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)
```

```
10000/10000 - 6s - loss: 0.8541 - accuracy: 0.7154
```



In [11]:

```
print(test_acc)
```

```
0.7154
```

In [ ]:

```

```